SYBASE®

Data Modeling

Sybase® PowerDesigner®

15.0

Windows

Part number: DC38058-01-1500-01
Last modified: September 2008

Contents

PART I

BUILDING DATA MODELS

This part explains how to use PowerDesigner to build Data Models.

About This Manual

Subject

This book describes the PowerDesigner Conceptual, Logical, and Physical Data Models, including how to create a CDM, LDM, and PDM, build each of the available diagrams, and generate and reverse engineer databases.

Audience

This book assumes that you are an experienced Windows user with some experience with relational databases and SQL.

Documentation primer

For information about the complete documentation set provided with PowerDesigner, see the "Getting Started with PowerDesigner" chapter of the *Core Features Guide* .

Typographic conventions

PowerDesigner documentation uses special typefaces to help you readily identify specific items:

- `monospace text (normal and bold)`

  Used for: Code samples, commands, compiled functions and files, references to variables.

  Example: `declare user_defined...`, the `BeforeInsertTrigger` template.

- **bold text**

  Used for: New terms.

  Example: A **shortcut** has a target object.

- SMALL CAPS

  Used for: Key names.

  Example: Press the ENTER key.

Bibliography

Data Modeling Essentials

Graeme Simsion, Van Nostrand Reinhold, 1994, 310 pages; paperbound; ISBN 1850328773

Information engineering

James Martin, Prentice Hall, 1990, three volumes of 178, 497, and 625 pages respectively; clothbound, ISBN 0-13-464462-X (vol. 1), 0-13-464885-4 (vol. 2), and 0-13-465501-X (vol. 3).

Celko95

Joe Celko, Joe Celko's SQL for Smarties (Morgan Kaufmann Publishers, Inc., 1995), 467 pages; paperbound; ISBN 1-55860-323-9.

CHAPTER 1

Getting Started with Data Modeling

About this chapter

This chapter presents the Conceptual, Logical, and Physical Data Models and provides guidance for data modeling with PowerDesigner.

Contents

Data Modeling with PowerDesigner

A data model is a representation of the information consumed and produced by a system. Data modeling involves analyzing the data objects present in a system and the relationships between them. PowerDesigner provides conceptual, logical, and physical data models to allow you to analyze and model your system at all levels of abstraction.

Conceptual Data Models

A Conceptual Data Model (CDM) represents the logical structure of a data system independent of any software or data storage structure. It gives a formal representation of the data needed to run an enterprise or a business activity, and may contain data objects not yet implemented in a physical database.

A CDM allows you to:

- Represent the organization of data in a graphic format to create Entity Relationship Diagrams (ERD).
- Verify the validity of data design.
- Generate a Logical Data Model (LDM), a Physical Data Model (PDM) or an Object-Oriented Model (OOM), which specifies an object representation of the CDM using the UML standard.

☞ To create a CDM, see "Creating a Data Model" on page 5. For detailed information about conceptual diagrams, see "Conceptual Diagram Basics" in the Building Conceptual and Logical Diagrams chapter.

Logical Data Models

A Logical Data Model helps you design a database structure and perform some database denormalization actions independent of any specific DBMS physical requirements.

You can use a logical model as an intermediary step in the database design process between the conceptual and physical designs:

- Start with a CDM containing entities, attributes, relationships, domains, data items and business rules. If need be, you may develop the CDM in several design steps starting from a high level model to a low level CDM
- Generate an LDM. Create indexes and specify FK column names and other common features

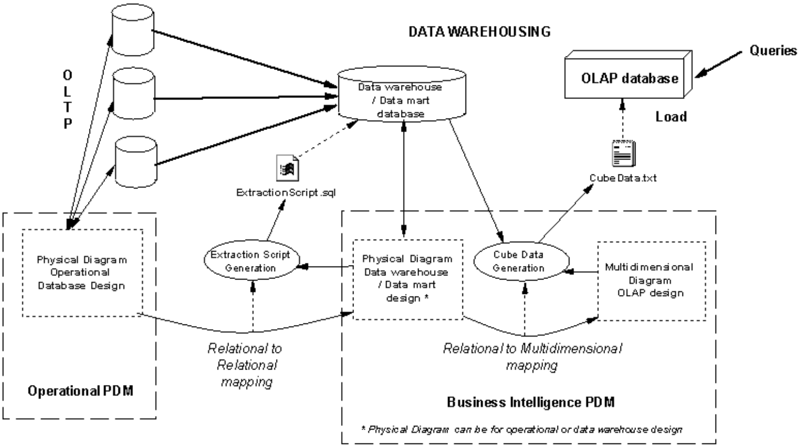
- ♦ Generate one or more PDMs, each targeted to a specific DBMS implementation

This design process allows you to keep everything consistent in a large development effort.

☞ To create an LDM, see “Creating a Data Model” on page 5. For detailed information about logical diagrams, see “Logical Diagram Basics” in the Building Conceptual and Logical Diagrams chapter.

Physical Data Models

A Physical Data Model (PDM) is a database design tool suitable for modeling the implementation of physical structures and data queries in a database.


DATA WAREHOUSING
OLTP
Queries
OLAP database
Load
Data warehouse / Data mart database
ExtractionScript.sql
Cube Data.txt
Physical Diagram Operational Database Design
Extraction Script Generation
Physical Diagram Data warehouse / Data mart design *
Cube Data Generation
Multidimensional Diagram OLAP design
Relational to Relational mapping
Relational to Multidimensional mapping
Operational PDM
Business Intelligence PDM
* Physical Diagram can be for operational or data warehouse design

Depending on the type of database you want to design, you will use different types of diagrams in the PDM:

- ♦ Operational PDM - You use PDM to design the structure of an operational database. Usually, in data modeling, the physical analysis follows the conceptual and/or logical analysis, and addresses the details of the actual physical implementation of data in a database, to suit your performance and physical constraints.
- ♦ Business intelligence PDM - You can use a PDM to design the structure of a data environment, which consists of:
  - **Data warehouse** or **data mart** database – are populated with data transferred from operational databases, and gather together all the information that may be needed in an OLAP database, where queries for business analysis and decision making are performed. The data warehouse database gathers all the data manipulated in a company for

example, whereas the data mart focuses on smaller entities in the company.

You use physical diagrams to design a data warehouse or data mart database. Since these databases usually contain very large amounts of data for storage, you do not need to design them for performance. You may assign types (fact and dimension) to the database tables to have a preview of the multidimensional structure in an OLAP database.

- A multidimensional **OLAP** database - which is generally populated with data that has first been aggregated in a data warehouse or data mart (though sometimes it is transferred directly from operational databases), and in which information is organized to facilitate queries performed by different tools. Business analysts use OLAP databases to send queries and retrieve business information from the different dimensions existing in the database.

  You use PDM multidimensional diagrams to design the different dimensions and cubes within the OLAP database.

☞ To create Physical Diagrams, see the “Building Physical Diagrams” chapter. To create Multidimensional Diagrams, see the “Building Multidimensional Diagrams” chapter.

Creating a Data Model

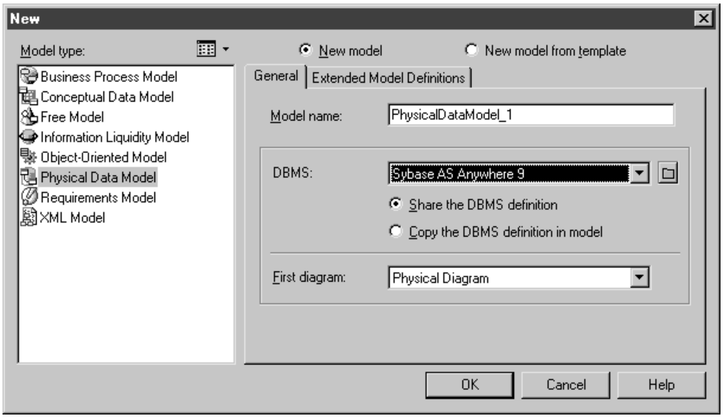
You can create a new CDM from scratch, by importing a Process Analyst Model (.PAM) or an ERwin model (.ERX), or by generating it from a CDM, PDM, or OOM.

You can create a new PDM from scratch, or reverse engineer the model from an existing database.

☞ For information about reverse engineering, see the “Reverse Engineering a Database into a PDM” chapter.

❖ To create a new CDM, LDM, or PDM

1. Select File ➤ New to open the New dialog box.



● New model ○ New model from template

Model name: PhysicalDataModel_1

DBMS: Sybase AS Anywhere 9

● Share the DBMS definition

○ Copy the DBMS definition in model

First diagram: Physical Diagram

2. Select one of the following model types:
   - ♦ Conceptual Data Model
   - ♦ Logical Data Model
   - ♦ Physical Data Model

3. Select one of the following radio buttons:
   - ♦ New model – Creates a new, empty, model.
   - ♦ New model from template – Creates a model from a model template, which can contain pre-configured options, preferences, extensions, and objects. For more information, see “Model Templates” in the Models chapter of the *Core Features Guide* .

4. Enter a model name. The code of the model, which is used for script or code generation, is derived from this name according to the model naming conventions.

5. [PDM only] Select a DBMS, and specify whether to:
    - Share the DBMS definition – use the original DBMS file in the "Resource Files\DBMS" directory. Changes made to the DBMS are shared by all PDMs that share it.
    - Copy the DBMS definition in model – make a copy of the DBMS file. The copied DBMS is saved with the PDM and changes made to it do not impact any other PDMs.

      ☞ For more information on DBMS properties and customizing a DBMS, see the DBMS Resource File Reference chapter of the *Customizing and Extending PowerDesigner* manual.

6. [PDM only] Select the type of the first diagram. The diagram chosen becomes the default for the next time you create a new PDM. You can create as many diagrams as you need in a CDM, LDM, or PDM.

7. [optional] Click the Extended Model Definitions tab, and select one or more extended model definitions to attach to your model.

   ☞ For more information on using extended model definitions, see "Extended Model Definitions" in the Resource Files and the Public Metamodel chapter of the *Customizing and Extending PowerDesigner* manual.

8. Click OK to create the new data model in the current Workspace.

Demo example
Sample data models are available in the Examples directory.

Model properties

To open a model property sheet, double-click its Browser entry.

The General tab contains the following properties:

Property	**Description**
Name	Specifies the name of the model, which should be clear and meaningful, and should convey its purpose to non-technical users.
Code	Specifies the technical name of the item used for generating code or scripts, which may be abbreviated, and should not include spaces.
Comment	Provides descriptive information about the model.

Property	Description
Filename	Specifies the location of the model file. This field is empty if the model has never been saved.
Author	Specifies the author of the model. If you enter nothing, the Author field in diagram title boxes displays the user name from the model property sheet Version Info tab. If you enter a space, the Author field displays nothing.
Version	Specifies the version of the model. You can use this box to display the repository version or a user defined version of the model. This parameter is defined in the Title page of the model display preferences
DBMS	[PDM only] Specifies the DBMS attached to the model. Clicking the Properties tool to the right of this field to open the DBMS file in the Resource Editor.
Database	[PDM only] Specifies the database that is the target for the model. You can create a database in the model by clicking the Create tool to the right of this field. If your DBMS supports multiple databases in a single model (enabled by the EnableManyDatabases entry in the Database category of the DBMS), this field is not present, and is replaced by a list of databases in the Model menu. A Database category is also displayed in the physical options of your database objects.
Default diagram	Specifies the diagram displayed by default when you open the model.

Database properties (PDM)

You can create a database from the General tab of the model property sheet or, if your DBMS supports multiple databases in a single model, from the list of databases in the Model menu.

A database has the following properties:

Property	Description
Name	Name for the database
Code	Code for the database. This code is generated in database scripts
Comment	Descriptive label for the database

Property	Description
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined
DBMS	DBMS for the database
Options	Physical options available in the DBMS
Script	Begin and end scripts that are inserted at the start and end of a database creation script
Rules	Business rules for the database

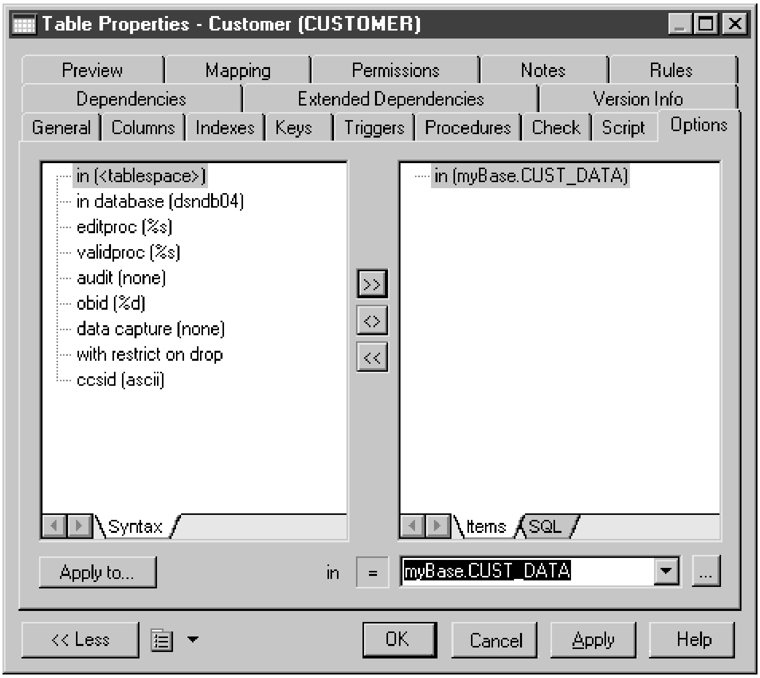
❖ To use a database in a physical option

1. Open the property sheet of an object with physical options.
2. Click the Options tab, select the in database (. . .) option and click the >> button.
3. Select a database from the list below the right pane.
4. Click OK.

When you use the **in [<tablespace>]** physical option, you associate a predefined tablespace with a database using the following syntax:

```
DBname.TBSPCname
```

For example, tablespace CUST_DATA belongs to database myBase. In the following example, table Customer will be created in tablespace CUST_DATA:



You should not define a database together with a tablespace physical option on the same object, this will raise an error during check model.

The database Dependencies tab displays the list of objects that use the current database in their physical options.

Archiving a PDM

Archived models store all constraint names without making a difference between user defined and calculated constraints. These models are used with the modify database feature.

You can archive a PDM with the .apm file extension, using the following methods:

- Save a PDM as an archived model
- Automatically archive PDM after database creation

❖ To archive a PDM

1. Select File ➤ Save As, select Archived PDM (bin) or Archived PDM (xml) in the Save As Type list, and click Save.

   or

Select Database ➤ Generate Database, click the Options tab, select the Automatic Archive check box in the After Generation groupbox, and click OK.

CHAPTER 2

Building Conceptual and Logical Diagrams

About this chapter

This chapter describes how to build conceptual and logical diagrams, and how to create and modify the associated objects.

Contents

Introducing Conceptual and Logical Diagrams

The data models in this chapter allow you to model the semantic and logical structure of your system.

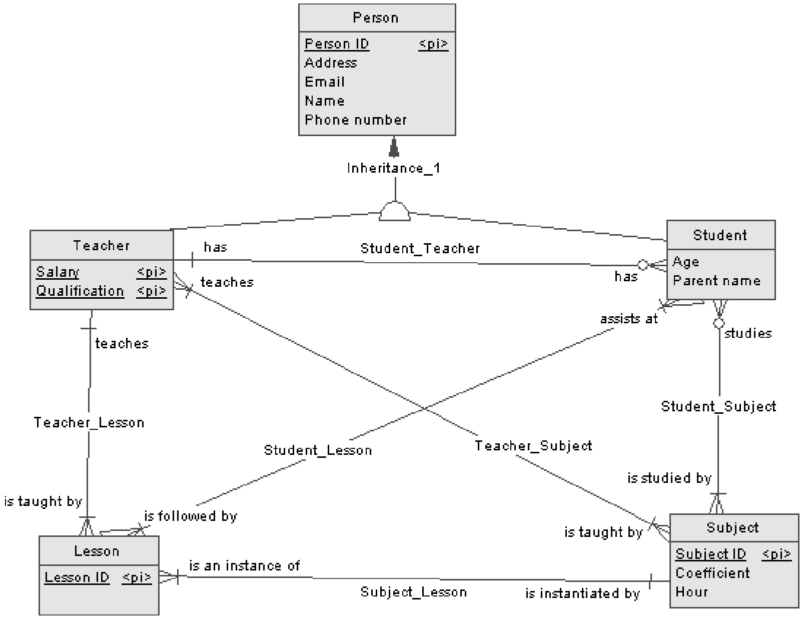
PowerDesigner provides you with a highly flexible environment in which to model your data systems. You can begin with either a CDM (see "Conceptual Diagram Basics" on page 13) or an LDM (see "Logical Diagram Basics" on page 17) to analyze your system and then generate a PDM (see the Building Physical Diagrams chapter) to work out the details of your implementation. Full support for database reverse-engineering allows you to take existing data structures and analyze them at any level of abstraction.

For more information about intermodel generation, see "Generating Other Models from a Data Model" in the Working with Data Models chapter.

Conceptual Diagram Basics

A Conceptual Data Model (CDM) represents the structure of your database, independent of any software or data storage structure. It describes entities (things of significance to a organization) and their identifiers and other attributes, along with the relationships and inheritances that connect them.

In the following conceptual diagram, the Teacher and Student entities inherit attributes from the Person parent entity. The two child entities are linked with a one-to-many relationship (a teacher has several students but each student has only one main teacher).

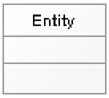
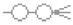
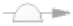


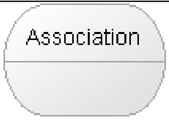
In addition:

- a teacher can teach several subjects and a subject can be taught by several teachers (many-to-many).
- a teacher can teach several lessons and a lesson is taught by only one teacher (one-to-many).
- a student attends multiple lessons and a lesson is followed by multiple students (many-to-many).
- a student studies multiple subjects and a subject can be studied by multiple students (many-to-many).

Conceptual diagram objects

You can create the following objects in a conceptual diagram:

Object	Tool	Symbol	Description
Domain	[none]	[none]	Set of values for which a data item is valid. See “Domains (CDM/LDM/PDM)” in the Building Physical Diagrams chapter.
Data Item	[none]	[none]	Elementary piece of information. See “Data Items (CDM)” on page 21.
Entity	▦		Person, place, thing, or concept that is of interest to the enterprise. See “Entities (CDM/LDM)” on page 24.
Entity Attribute	[none]	[none]	Elementary piece of information attached to an entity. See “Attributes (CDM/LDM)” on page 27.
Identifier	[none]	[none]	One or many entity attributes, whose values uniquely identify each occurrence of the entity. See “Identifiers (CDM/LDM)” on page 30.
Relationship	▦		Named connection or relation between entities (ER modeling methodology). See “Relationships (CDM/LDM)” on page 32.
Inheritance	▦		Relationship that defines an entity as a special case of a more general entity. See “Inheritances (CDM/LDM)” on page 51.

Object	Tool	Symbol	Description
Association			Named connection or association between entities (Merise modeling methodology). See “Associations and Association Links (CDM)” on page 44.
Association Link			Link that connects an association to an entity. See “Associations and Association Links (CDM)” on page 44.

Creating a conceptual diagram

You can create a conceptual diagram in an existing CDM in any of the following ways:

- Right-click the model in the Browser and select New ➤ Conceptual Diagram from the contextual menu
- Right-click the background of any diagram and select Diagram ➤ New Diagram from the contextual menu.

To create a new CDM with a conceptual diagram, select File ➤ New, choose Conceptual Data Model from the Model type list, and click OK.

Opening a v6 PAM into a CDM

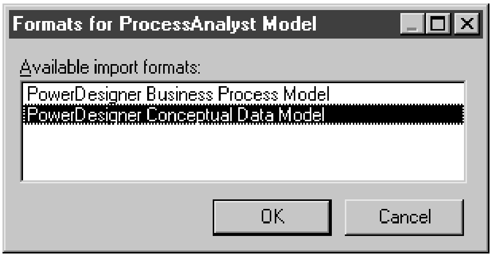
You can open a v6 process analyst model (PAM) into a CDM, to recover process modeling information, as follows:

PAM object	CDM object
Business rule	Business rule
Domain	Domain
Data store	Entity
Data item	Data item

You can recover processes from a PAM by opening it into a BPM (see the Business Process Modeling guide).

❖ To open a PAM into a CDM

1. Select File ➤ Open and select the PAM file.
2. Click Open to display the Formats for ProcessAnalyst Model window.



Formats for ProcessAnalyst Model

Available import formats:

PowerDesigner Business Process Model

PowerDesigner Conceptual Data Model

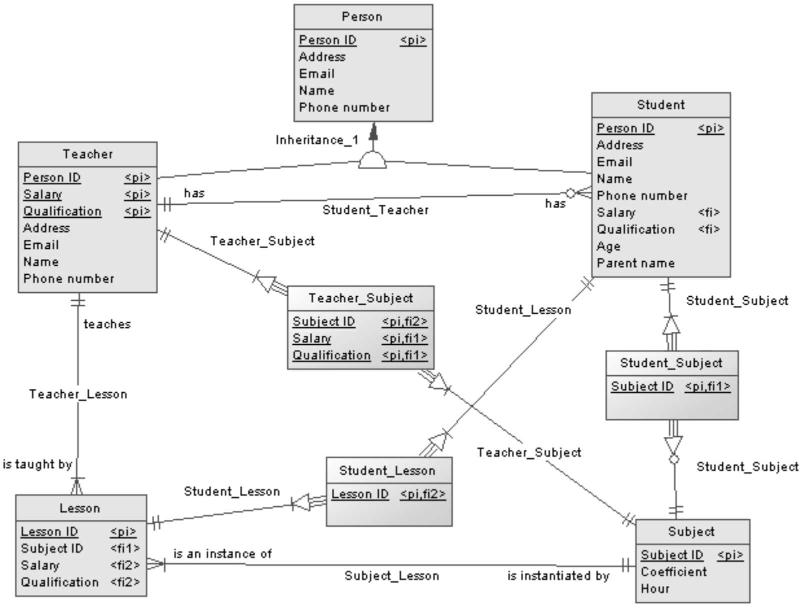
3. Select PowerDesigner Conceptual Data Model and click OK to begin the import.

   The recovered objects are imported into the CDM and appear in a default diagram.

Logical Diagram Basics

A Logical Data Model (LDM) allows you to validate the relationships identified in your CDM. The objects are similar to those in the CDM, but primary identifiers migrate along one-to-many relationships to become foreign identifiers, and many-to-many relationships, which are not permitted in an LDM, are replaced by intermediate entities.

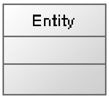
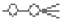
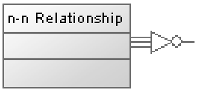
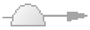
The following logical diagram represent the same system as that in our CDM example (see “Conceptual Diagram Basics” on page 13).



Primary identifiers have migrated along one-to-many relationships to become foreign identifiers, and many-to-many relationships are replaced with an intermediary entity linked with one-to-many relationships to the extremities.

Logical diagram objects

You can create the following objects in a logical diagram:

Object	Tool	Symbol	Description
Domain	[none]	[none]	Set of values for which a data item is valid. See “Domains (CDM/LDM/PDM)” in the Building Physical Diagrams chapter.
Entity			Person, place, thing, or concept that is of interest to the enterprise. See “Entities (CDM/LDM)” on page 24.
Entity Attribute	[none]	[none]	Elementary piece of information attached to an entity. See “Attributes (CDM/LDM)” on page 27.
Identifier	[none]	[none]	One or many entity attributes, whose values uniquely identify each occurrence of the entity. See “Identifiers (CDM/LDM)” on page 30.
Relationship			Named connection or relation between entities (ER modeling methodology). See “Relationships (CDM/LDM)” on page 32.
n-n Relationship			[LDM only] Named cardinality represented with an intermediary entity. See “Relationships (CDM/LDM)” on page 32.
Inheritance			Relationship that defines an entity as a special case of a more general entity. See “Inheritances (CDM/LDM)” on page 51.

Creating a logical diagram

You can create a logical diagram in an existing LDM in any of the following ways:

- Right-click the model in the Browser and select New ➤ Logical Diagram from the contextual menu.
- Right-click the background of any diagram and select Diagram ➤ New Diagram from the contextual menu.

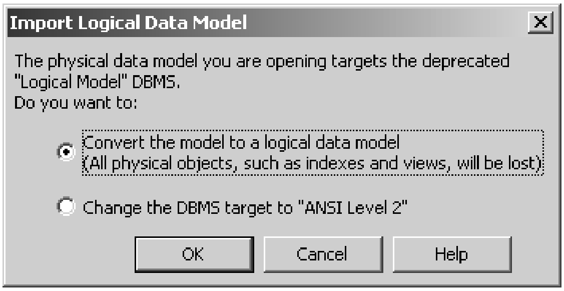
To create a new LDM with a logical diagram, select File ➤ New, choose Logical Data Model from the Model type list, and click OK.

Importing a deprecated PDM logical model

If you have previously created a PDM with the logical model DBMS, you will be invited to migrate to an LDM when you open it.

❖ To open a deprecated PDM logical model

1. Select File ➤ Open and browse to the PDM logical model to open.
2. Click Open to display the Import Logical Data Model dialog:



Import Logical Data Model

The physical data model you are opening targets the deprecated "Logical Model" DBMS.
Do you want to:

● Convert the model to a logical data model (All physical objects, such as indexes and views, will be lost)

○ Change the DBMS target to "ANSI Level 2"

3. Choose one of the following options:
   - Convert the model to a logical data model – Note that only tables, columns, keys and references are preserved
   - Change the DBMS target to “ANSI Level 2” and open it as a PDM
4. Click OK to open the model.

Restoring generation links to a converted LDM

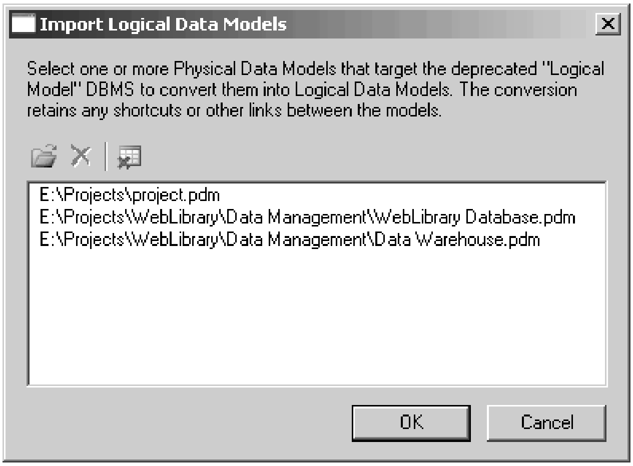
A PDM with the logical model DBMS that had been generated from a CDM will retain its links to the source CDM when you convert it to an LDM. However, for any PDM generated from the old LDM, you will need to restore the generation links by regenerating the PDM from the new LDM, using the Update existing PDM option (see Linking and Synchronizing Models in the *Core Features Guide*).

Importing multiple interconnected PDM logical models

If you have previously created multiple PDMs with the logical model DBMS, and these models are connected by shortcuts and generation or other links, you can convert them en masse to logical data models and preserve their interconnections.

❖ To open multiple deprecated PDM logical models

1. Select File ➤ Import ➤ Legacy Logical Data Models to open the Import Logical Data Models dialog:



Select one or more Physical Data Models that target the deprecated "Logical Model" DBMS to convert them into Logical Data Models. The conversion retains any shortcuts or other links between the models.

E:\Projects\project.pdm
E:\Projects\WebLibrary\Data Management\WebLibrary Database.pdm
E:\Projects\WebLibrary\Data Management\Data Warehouse.pdm

2. Click Open, browse to the legacy PDMs you want to import, select them, and then click OK to add them to the list. You can, if necessary, add multiple PDMs from multiple directories by repeating this step.

3. When you have added all the necessary PDMs to the list, click OK to import them into interconnected LDMs.

Data Items (CDM)

A **data item** is an elementary piece of information, which represents a fact or a definition in an information system, and which may or may not have any eventual existence as a modeled object.

You can attach a data item to an entity (see "Entities (CDM/LDM)" on page 24) in order to create an entity attribute (see "Attributes (CDM/LDM)" on page 27), which is associated with the data item.

There is no requirement to attach a data item to an entity. It remains defined in the model and can be attached to an entity at any time.

Data items are not generated when you generate an LDM or PDM.

Example

In the information system for a publishing company, the last names for authors and customers are both important pieces of business information. The data item LAST NAME is created to represent this information. It is attached to the entities AUTHOR and CUSTOMER, and becomes entity attributes of those entities.

Another piece of information is the date of birth of each author. The data item BIRTH DATE is created but, as there is no immediate need for this information in the model, it is not attached to any entity.

Creating a data item

You can create a data item in any of the following ways:

- ♦ Select Model ➤ Data Items to access the List of Data Items, and click the Add a Row tool.
- ♦ Create an entity attribute (see "Attributes (CDM/LDM)" on page 27). A data item will be automatically created.
- ♦ Right-click the model or package in the Browser, and select New ➤ Data Item.

☞ For general information about creating objects, see the Objects chapter in the *Core Features Guide* .

Data item properties

You can modify an object's properties from its property sheet. To open a data item property sheet, double-click its Browser entry in the Data Items folder.

The General tab contains the following properties:

Property	Description
Name	Specifies the name of the item, which should be clear and meaningful, and should convey the item's purpose to non-technical users.
Code	Specifies the technical name of the object, which is used for generating code or scripts, which may be abbreviated, and should not generally include spaces.
Comment	Specifies a descriptive label for the data item.
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined.
Data type	Specifies the code indicating the data format, such as N for numeric or A for alphanumeric, followed by the number of characters.
Length	Specifies the maximum number of characters.
Precision	Specifies the number of places after the decimal point, for data values that can take a decimal point.
Domain	Specifies the name of the associated domain (See "Domains (CDM/LDM/PDM)" in the Building Physical Diagrams chapter). If you attach a data item to a domain, the domain supplies a data type to the data item, and can also apply length, decimal precision, and check parameters.

The following tabs are also commonly used:

- Standard Checks - contains checks which control the values permitted for the data item (see "Check parameters (CDM/LDM/PDM)" in the Building Physical Diagrams chapter).
- Additional Checks - allows you to specify additional constraints (not defined by standard check parameters) for the data item.
- Rules - lists the business rules associated with the data item (see "Business Rules (CDM/LDM/PDM)" in the Building Physical Diagrams chapter).

Controlling uniqueness and reuse of data items

The following model options allow you to control naming restraints and reuse for data items:

Option	When selected	When cleared
Unique code	Each data item must have a unique code.	Multiple data items can have the same code.
Allow reuse	One data item can be an entity attribute for multiple entities.	Each data item can be an entity attribute for only one entity

If you do not select Unique Code, two data items can have the same code, and you differentiate them by the entities that use them. The entities are listed in the Used By column of the list of data items.

Item not visible in list
To make an item visible in a list, click the Customize Columns and Filter tool in the list toolbar, select the appropriate check box from the list of filter options that is displayed, and click OK.

❖ To define code and reuse options for data items

1. Select Tools ➤ Model Options to open the Model Options dialog box:
2. Select or clear the Unique Code and Allow Reuse check boxes in the Data Item groupbox, and then click OK to return to the model.

Error message

The following error message is displayed if you select the Unique Code option, when data items are already sharing a name in the CDM:

Error message	Solution
Unique Code option could not be selected because two data items have the same code: *data_item_code*.	Assign unique codes to all data items

Entities (CDM/LDM)

An entity represents an object about which you want to store information. For example, in a model of a major corporation, the entities created may include Employee and Division.

When you generate a PDM from a CDM or LDM, entities are generated as tables.

Creating an entity

You can create an entity in any of the following ways:

- Use the Entity tool in the diagram Palette.
- Select Model ➤ Entities to access the List of Entities, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤ Entity.

☞ For general information about creating objects, see the Objects chapter in the *Core Features Guide* .

Entity properties

You can modify an object’s properties from its property sheet. To open an entity property sheet, double-click its diagram symbol or its Browser entry in the Entities folder.

The General tab contains the following properties:

Property	Description
Name	Specifies the name of the item, which should be clear and meaningful, and should convey the item's purpose to non-technical users.
Code	Specifies the technical name of the object, which is used for generating code or scripts, which may be abbreviated, and should not generally include spaces.
Comment	Specifies a descriptive label for the entity.
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined.
Number	Specifies the estimated number of occurrences in the physical database for the entity (the number of records).
Generate	Specifies that the entity will generate a table in a PDM.
Parent Entity	[read-only] Specifies the parent entity. Click the Properties tool at the right of the field to open the parent property sheet.

The following tabs are also available:

- Attributes - lists the attributes associated with the entity (see "Attributes (CDM/LDM)" on page 27).
- Identifiers - lists the attributes associated with the entity (see "Identifiers (CDM/LDM)" on page 30).
- Rules - lists the business rules associated with the entity (see "Business Rules (CDM/LDM/PDM)" in the Building Physical Diagrams chapter).
- Subtypes – [Barker only] lists the subtypes that inherit from the entity.

Copying an entity

You can make a copy of an entity within the same model or between models.

The following rules apply to copied entities. The indicated selections for Unique code and Allow reuse apply to the model that receives the copied entity:

Data item options selected	Result of copying an entity
Unique Code Allow Reuse	New entity with new name and code New identifier with new name code Reuses other entity attributes
Unique Code only	New entity with new name and code New identifier with new name and code New attributes with new names and codes
Allow Reuse only	New entity with new name and code New identifier with same name and code Reuses other entity attributes
None	New entity with new name and code New identifier with same name code New entity attributes with same names and codes

❖ To copy an entity within a model

1. Select an entity in the CDM/LDM, and then select Edit ➤ Copy and Edit ➤ Paste.
2. [alternatively] Press CTRL and drag the entity to a new position in the diagram.

   The entity is copied and the new entity is displayed in the Browser and diagram.

❖ To copy an entity to a different model

1. Select an entity in the CDM/LDM, and then select Edit ➤ Copy
2. Select the new diagram or model and then select Edit ➤ Paste.

   The entity is copied and the new entity is displayed in the Browser and diagram.

Attributes (CDM/LDM)

In a CDM, **attributes** are data items attached to an entity, association, or inheritance. In an LDM, there are no data items, and so attributes exist in entities without a conceptual origin.

When you generate a PDM from a CDM or LDM, entity attributes are generated as table columns.

Creating an attribute

You can create an entity attribute using the following tools, available on the Attributes tab in the property sheet of an entity, association, or inheritance:

Tool	Description
▦	Add a Row – Creates a new attribute and associated data item (with the same name and code). If you have enabled the Allow Reuse model option, the new data item can be used as an attribute for other objects. If you have enabled the Allow Reuse and Unique Code model options and you type the name of an existing data item, it will be automatically reused.
▦	Add Data Item (CDM)/Add Attributes (LDM) - Opens a Selection window listing all the data items/attributes available in the model. Select one or more data items/attributes in the list and then click OK to make them attributes to the object. If the data item/attribute has not yet been used, it will be linked to the object. If the data item/attribute has already been used, it will be copied (with a modified name if you have enabled the Unique code model option) and the copy attached to the object.

☞ For general information about creating objects, see the Objects chapter in the *Core Features Guide* .

Attribute properties

You can modify an object’s properties from its property sheet. To open an attribute property sheet, double-click its Browser entry in the Attributes folder within an entity, association, or inheritance.

The General tab contains the following properties:

Property	Description
Name	Specifies the name of the item, which should be clear and meaningful, and should convey the item's purpose to non-technical users.
Code	Specifies the technical name of the object, which is used for generating code or scripts, which may be abbreviated, and should not generally include spaces.
Comment	Specifies a descriptive label for the attribute.
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined.
Entity/ Association/ Inheritance	[read-only] Specifies the parent object. Click the tool to the right of the field to open its property sheet.
Data Item	[CDM only, read-only] Specifies the related data item. Click the tool to the right of the field to open its property sheet.
Inherited from	[LDM only, read-only] Specifies the parent entity from which the attribute is migrated through an inheritance.
Data type	Specifies the data type of the attribute, such as numeric, alphanumeric, boolean, or others. Click the question mark button to open the list of data types (see "Domains (CDM/LDM/PDM)" in the Building Physical Diagrams chapter).
Length	Specifies the maximum length of the data type.
Precision	Specifies the maximum number of places after the decimal point.
Domain	Specifies the name of the associated domain (See "Domains (CDM/LDM/PDM)" in the Building Physical Diagrams chapter). If you attach an attribute to a domain, the domain supplies a data type to the attribute, and can also apply length, decimal precision, and check parameters.
Primary Identifier	[entity attributes only] Indicates whether or not the attribute is the primary identifier of the entity.
Displayed	[entity and association attributes only] Displays the attribute in the object symbol.

Property	Description
Mandatory	Specifies that every object occurrence must assign a value to the attribute. Identifiers (see "Identifiers (CDM/LDM)" on page 30) are always mandatory.
Foreign identifier	[LDM only, read-only] Indicates whether or not the attribute is the foreign identifier of the entity.

The following tabs are also available:

- Standard Checks - contains checks which control the values permitted for the attribute (see "Check parameters (CDM/LDM/PDM) in the Building Physical Diagrams chapter).
- Additional Checks - allows you to specify additional constraints (not defined by standard check parameters) for the attribute.
- Rules - lists the business rules associated with the attribute (see "Business Rules (CDM/LDM/PDM)" in the Building Physical Diagrams chapter).

Deleting attributes (CDM)

When you delete an attribute, model options determine whether or not the corresponding data items are also deleted:

Model options selected	Result of deleting an attribute
Unique Code and Allow Reuse	Does not delete corresponding data item
Unique Code only	Does not delete corresponding data item
Allow Reuse only	Deletes corresponding data item if it is not used by another entity
None	Deletes corresponding data item

Identifiers (CDM/LDM)

An **identifier** is one or many entity attributes, whose values uniquely identify each occurrence of the entity.

Each entity must have at least one identifier. If an entity has only one identifier, it is designated by default as the primary identifier.

When you generate a PDM from a CDM or LDM, identifiers are generated as primary or alternate keys.

Creating an identifier

You can create an entity in any of the following ways:

- Open the Attributes tab in the property sheet of an entity, select one or more attributes, and click the Create Identifier tool. The selected attributes are associated with the identifier and are listed on the attributes tab of its property sheet.
- Open the Identifiers tab in the property sheet of an entity, and click the Add a Row tool.

☞ For general information about creating objects, see the Objects chapter in the *Core Features Guide* .

Identifier properties

You can modify an object’s properties from its property sheet. To open an identifier property sheet, double-click its Browser entry in the Identifiers folder beneath an entity.

The General tab contains the following properties:

Property	Description
Name	The name of the identifier which should be clear and meaningful, and should convey its purpose to non-technical users.
Code	The technical name of the identifier used for generating code or scripts, which may be abbreviated, and should not generally include spaces.
Comment	Specifies a descriptive label for the identifier.
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined.
Entity	Specifies the name of the entity to which the identifier belongs.
Primary identifier	Specifies that the identifier is a primary identifier.

The following tabs are also available:

- Attributes - lists the attributes (see "Attributes (CDM/LDM)" on page 27) associated with the identifier: Click the Add Attributes tool to add an attribute.

Relationships (CDM/LDM)

A relationship is a link between entities. For example, in a model that manages human resources, the "Member" relationship links the entities Employee and Team and expresses that each employee works in a team, and each team has employees.

An occurrence of a relationship corresponds to one instance of each of the two entities involved in the relationship. For example, *the employee Martin working in the Marketing team* is one occurrence of the relationship Member.

When you generate a PDM from a CDM or LDM, relationships are generated as references.

Relationships and associations

Relationships are used in the Entity Relationship (ER), Barker, and IDEF1X modeling methodologies. In the Merise methodology associations (see "Associations and Association Links (CDM)" on page 44) are used to link entities. PowerDesigner lets you use either relationships or associations exclusively, or combine the two methodologies in the same model.

This section analyzes relationships in the Entity Relationship methodology, for more information on IDEF1X, see "Setting CDM/LDM Model Options" in the Working with Data Models chapter.

Creating a relationship

You can create a relationship in any of the following ways:

- Use the Relationship tool in the diagram Palette. Click inside the first entity to be linked and, while continuing to hold down the mouse button, drag the cursor to the second entity. Release the mouse button inside the second entity.
- Select Model ➤ Relationships to access the List of Relationships, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤ Relationship.

☞ For general information about creating objects, see the Objects chapter in the *Core Features Guide* .

Relationship properties

You can modify an object's properties from its property sheet. To open a relationship property sheet, double-click its diagram symbol or its Browser entry in the Relationships folder.

The General tab contains the following properties:

Property	Description
Name	Specifies the name of the item, which should be clear and meaningful, and should convey the item's purpose to non-technical users.
Code	Specifies the technical name of the object, which is used for generating code or scripts, which may be abbreviated, and should not generally include spaces.
Comment	Specifies a descriptive label for the relationship.
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined.
Entity1 Entity2	Specifies the two entities linked by the relationship. You can use the tools to the right of the lists to create an object, browse the complete tree of available objects or view the properties of the currently selected object.
Generate	Specifies that the relationship should be generated as a reference when you generate a PDM.
Cardinalities	Contains data about cardinality as the number of instances of one entity in relation to another entity.

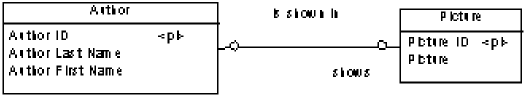
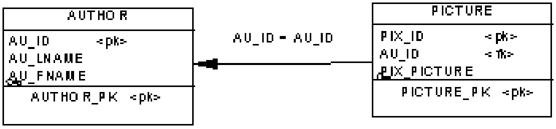
Relationship property sheet Cardinalities tab

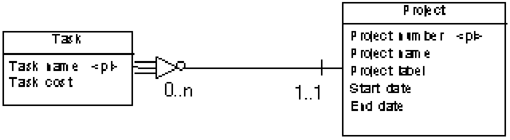
The Cardinalities tab contains the following properties:

Property	Description
Cardinality	Specifies the number of instances (none, one, or many) of an entity in relation to another entity. You can choose from the following values: ♦ One-to-one (symbol: <1..1>) - One instance of entity A can correspond to only one instance of entity B. ♦ One-to-many (symbol: <1..n>) - One instance of entity A can correspond to more than one instance of entity B. ♦ Many-to-one (symbol: <n..1>) - More than one instance of entity A can correspond to the same one instance of entity B. ♦ Many-to-many (symbol: <n..n>) - More than one instance of entity A can correspond to more than one instance of entity B. To use n..n relationships in an LDM, see "Enabling many-to-many relationships in an LDM" on page 38.

☞ For information about the termination points of the relationships in each of the supported notations, see "Supported CDM/LDM notations" in the Working with Data Models chapter.

In addition, this tab contains a groupbox for both directions of the relationship, each of which contains the following properties:

Property	Description
Dominant role	In a one-to-one relationship, you can define one direction of the relationship as dominant. If you define a dominant direction, the one-to-one relationship generates one reference in the PDM. The dominant entity becomes the parent table. If you do not define a dominant direction, the one-to-one relationship generates two references. The relationship pictured here shows the one-to-one relationship.  Author Author ID <pk> Author Last Name Author First Name is shown in shows Picture Picture ID <pk> Picture In a PDM, this relationship generates the following reference: Author is the parent table, and its primary key migrates to the Picture table as foreign key.  AUTHOR AU_ID <pk> AU_LNAME AU_FNAME AUTHOR_PK <pk> AU_ID = AU_ID PICTURE PIX_ID <pk> AU_ID <fk> PIX_PICTURE PICTURE_PK <pk>
Role name	Text that describes the relationship of EntityA to EntityB.

Property	Description
Dependent	In a dependent relationship, one entity is partially identified by another. Each entity must have an identifier. In some cases, however, the attributes of an entity are not sufficient to identify an occurrence of the entity. For these entities, their identifiers incorporate the identifier of another entity with which they have a dependent relationship. For example, an entity named Task has two entity attributes, TASK NAME and TASK COST. A task may be performed in many different projects and the task cost will vary with each project. To identify each occurrence of TASK COST the unique Task entity identifier is the compound of its *Task name* entity attribute and the *Project number* identifier from the Project entity. A many-to-many relationship cannot be a dependent relationship. The relationship pictured here expresses this dependency.  Task Task name <pi> Task cost 0..n 1..1 Project Project number <pi> Project name Project label Start date End date The circle at the top of the triangle indicates that occurrences of the Project entity do not require an occurrence of the Task entity. But an occurrence of the Task entity requires an occurrence of the Project entity on which it depends.
Mandatory	Indicates that the relationship between entities is mandatory. You define options from the point of view of the both entities in the relationship. For example, the subcontract relationship is optional from customer to project, but mandatory from project to customer. Each project must have a customer, but each customer does not have to have a project.
Cardinality	Specifies the maximum and minimum number of instances of EntityA in relation to EntityB (if mandatory, at least 1). You can choose from the following values: ♦ 0..1 – Zero to one instances ♦ 0..n – Zero to many instances ♦ 1..1 – Exactly one instance ♦ 1..n – one to many instances

Relationship property sheet Joins tab (LDM)

A **join** is a link between an attribute in a parent entity and an attribute in a child entity (attribute pair) that is defined within a relationship.

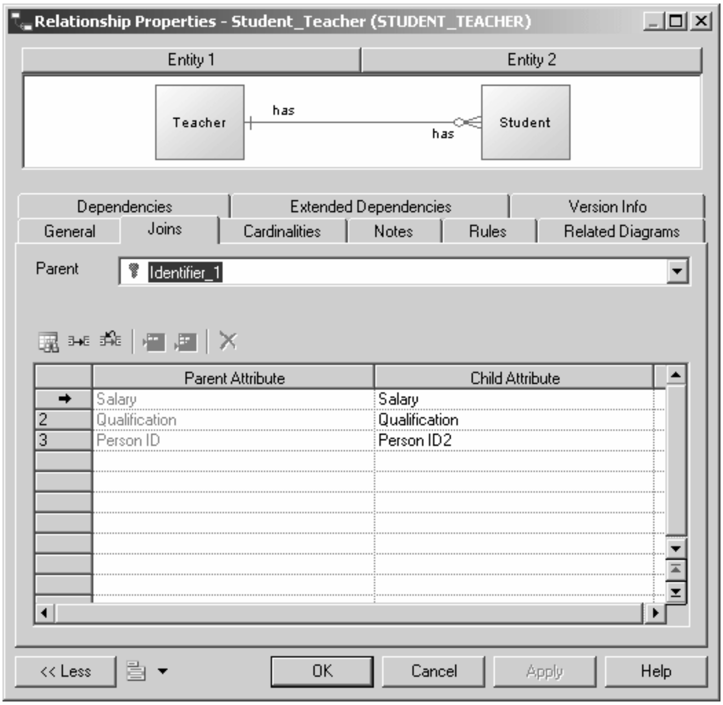
A join can link primary, alternate or foreign identifiers, or user-specified attributes in the parent and child entities that are independent of identifier attributes.

❖ To define joins in a relationship

1. Double-click a relationship in the diagram to open its property sheet and then click the Joins tab.

2. Select a key in the Parent Identifier list to create joins on its attributes. If you select <NONE>, the attribute lists are empty and you must specify your own attributes to join.

   The attributes linked by the joins are listed in the Parent Attribute and Child Attribute columns.



Parent: Identifier_1

	Parent Attribute	Child Attribute
➜	Salary	Salary
2	Qualification	Qualification
3	Person ID	Person ID2

Changing a foreign identifier attribute linked by a join

You can change the foreign identifier attribute linked by a join by clicking the attribute in the Child Entity list, and selecting another attribute from the list.

3. [optional] If you selected <NONE> from the Parent Identifier list, click the Parent Attribute column and select an attribute from the list, then click the Child Attribute column and select a child attribute.

4. [optional] Select the Auto arrange join order check box to sort the list by the identifier attribute order. If this option is not selected, you can re-arrange the attributes using the arrow buttons.

5. Click OK.

Linking attributes in a primary or alternate identifier

For any relationship you can choose to link a primary or alternate identifier, to a corresponding foreign identifier. When you select an identifier from the Joins tab of the relationship property sheet, all the identifier attributes are linked to matching foreign identifier attributes in the child entity.

Changing a foreign identifier attribute link

A foreign identifier attribute can be changed to link to another parent entity attribute, either within the identifier relationship, or independent of it.

Reuse and Migration option for a selected relationship

You can use the following buttons on the Joins tab to reuse or migrate attributes linked by joins.

Tool	Description
	Reuse Attributes - Reuse existing child attributes with same code as parent entity attributes.
	Migrate Attributes - Migrate identifier attributes to foreign identifier attributes. If attributes do not exist they are created.
	Cancel Migration - Delete any migrated attributes in child entity.

Enabling many-to-many relationships in an LDM

In an LDM, many-to-many relationships are, by default, not permitted and are represented with an intermediary entity. If you allow many-to-many relationships, you can select the many-to-many value in the cardinalities tab.

❖ To display the many-to-many value in a LDM cardinalities tab

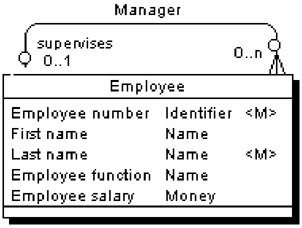
1. Select Tools ➤ Model Options to open the Model Options dialog box.

2. Select the Allow n-n relationships check box in the Relationship groupbox, and then click OK to return to the model.

Creating a reflexive relationship

A reflexive relationship is a relationship between an entity and itself.

In the following example, the reflexive relationship *Supervise* expresses that an employee (Manager) can supervise other employees.



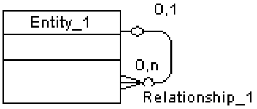
Getting neat relationship lines
To obtain clean lines with rounded corners when you create a reflexive relationship, select Display Preferences ➤ Format ➤ Relationship and modify the Line Style with the appropriate type from the Corners list.

❖ To create a reflexive relationship

1. Click the Relationship tool in the Palette.

2. Click inside the entity symbol and, while continuing to hold down the mouse button, drag the cursor a short distance within the symbol, before releasing the button.

   A relationship symbol loops back to the same entity.



Entity dependencies
In the Dependencies page of the entity, you can see two identical occurrences of the relationship, this is to indicate that the relationship is reflexive and serves as origin and destination for the link

Defining a code option for relationships

You can control naming restraints for relationships so that each relationship must have a unique code.

If you do not select Unique Code, two relationships can have the same code, and you differentiate them by the entities they link.

❖ To define the Unique Code model option for relationships

1. Select Tools ➤ Model Options to open the Model Options dialog box:
2. Select or clear the Unique Code check box in the Relationship groupbox, and then click OK to return to the model.

Error message

The following error message is displayed when the option you choose is incompatible with the current CDM:

Error message	Solution
Unique Code option could not be selected because at least two relationships have the same code: *relationship_code*.	Change the code of one relationship

Changing a relationship into an associative entity

You can transform a relationship into an associative entity (see "Associations and Association Links (CDM)" on page 44) linked by two relationships, and then attach entity attributes to the associative entity, that you could not attach to the relationship.

The associative entity retains the name and code of the relationship, and the two new relationships handle cardinality properties.

❖ To change a relationship directly into an associative entity

1. Right-click a relationship symbol and select Change to Entity ➤ Standard from the contextual menu.

   An associative entity with two relationships replaces the relationship. The associative entity takes the name of the original relationship.

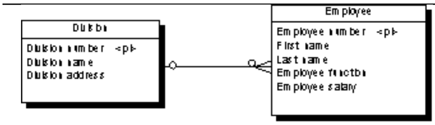
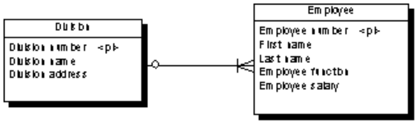
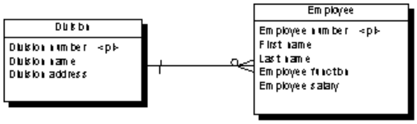
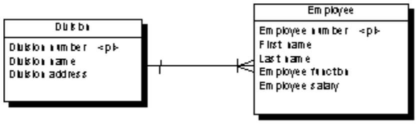
❖ To change a relationship into an associative entity using the Change to Entity Wizard

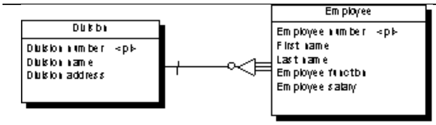
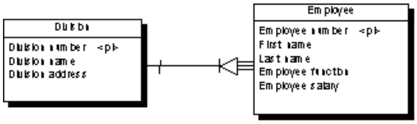
1. Right-click a relationship symbol and select Change to Entity ➤ Wizard from the contextual menu to open the Change to Entity Wizard.
2. On the Customizing Entity page, type an entity name and code, and then click Next.
3. On the first Customizing Relationship page, complete the details for the relationship that will be created between the first entity and the new entity, and then click Next.
4. On the second Customizing Relationship page, complete the details for the relationship that will be created between the new entity and the second entity, and then click Finish.

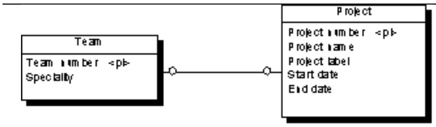
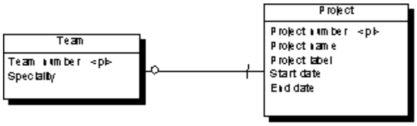
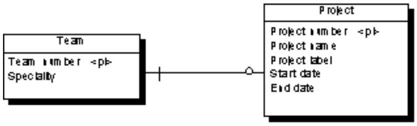
   The associative entity with two relationships replaces the relationship.

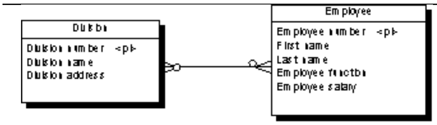
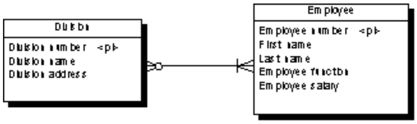
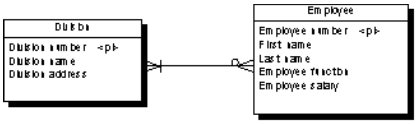
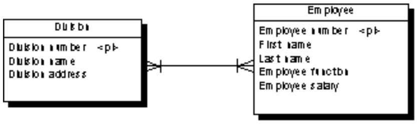
Relationship examples

This section shows the graphic representation of various relationship properties.

One-to-many relationship	Description
 Division Division number <pi> Division name Division address Employee Employee number <pi> First name Last name Employee function Employee salary	Each division may have zero or more employees Each employee may belong to zero or one division
 Division Division number <pi> Division name Division address Employee Employee number <pi> First name Last name Employee function Employee salary	Each division must have one or more employees Each employee may belong to zero or one division
 Division Division number <pi> Division name Division address Employee Employee number <pi> First name Last name Employee function Employee salary	Each division may have zero or more employees Each employee must belong to one and only one division
 Division Division number <pi> Division name Division address Employee Employee number <pi> First name Last name Employee function Employee salary	Each division must have one or more employees Each employee must belong to one and only one division

One-to-many relationship	**Description**
 Division Division number <pi> Division name Division address Employee Employee number <pi> First name Last name Employee function Employee salary	Each division may have zero or more employees Each employee must belong to one and only one division Each employee is identified uniquely by division number and employee number
 Division Division number <pi> Division name Division address Employee Employee number <pi> First name Last name Employee function Employee salary	Each division must have one or more employees Each employee must belong to one and only one division Each employee is identified uniquely by division number and employee number

One-to-one relationship	**Description**
 Team Team number <pi> Speciality Project Project number <pi> Project name Project label Start date End date	Each team works on zero or one project Each project is managed by zero or one team
 Team Team number <pi> Speciality Project Project number <pi> Project name Project label Start date End date	Each team works on one and one project only Each project is managed by zero or one team
 Team Team number <pi> Speciality Project Project number <pi> Project name Project label Start date End date	Each team works on zero or one project Each project is managed by one and one team only

Many-to-many relationship	Description
 Division Division number <pi> Division name Division address Employee Employee number <pi> First name Last name Employee function Employee salary	Each division may have zero or more employees Each employee may belong to zero or more divisions
 Division Division number <pi> Division name Division address Employee Employee number <pi> First name Last name Employee function Employee salary	Each division must have one or more employees Each employee may belong to zero or more divisions
 Division Division number <pi> Division name Division address Employee Employee number <pi> First name Last name Employee function Employee salary	Each division may have zero or more employees Each employee must belong to one or more divisions
 Division Division number <pi> Division name Division address Employee Employee number <pi> First name Last name Employee function Employee salary	Each division must have one or more employees Each employee must belong to one or more divisions

Identifier Migration along Relationships

Migrations are made instantaneously in an LDM or during generation if you generate a PDM from a CDM.

Relationship type	Migration
Dependent one-to-many	Foreign identifiers become attributes of the primary identifier of the child entity.
Many-to-many	No attributes are migrated.
Dominant one-to-one	Primary identifier migrate from the dominant attribute.
Mandatory one-to-many	If the child to parent role is mandatory, migrated attributes are mandatory.

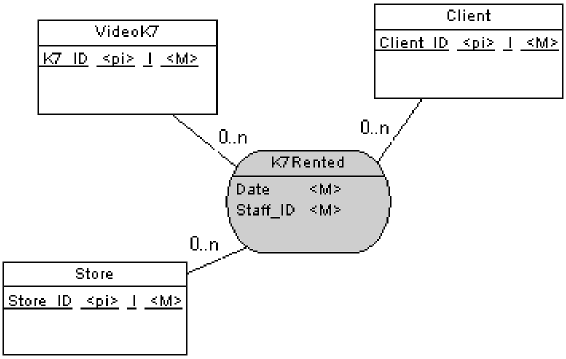
Associations and Association Links (CDM)

In the Merise modeling methodology an **association** is used to connect several entities that each represents clearly defined objects, but are linked by an event, which may not be so clearly represented by another entity.

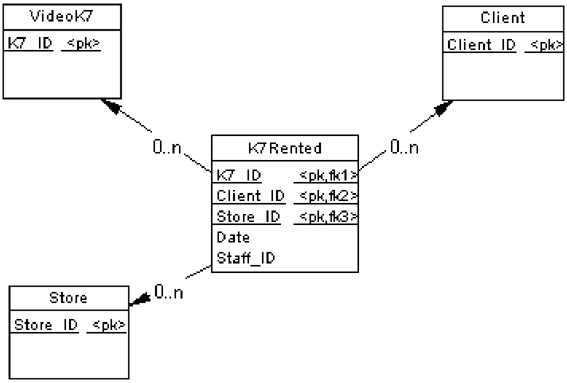
Each instance of an association corresponds to an instance of each entity linked to the association.

When you generate a PDM from a CDM, associations are generated as tables or references.

In the following example, three entities VIDEOK7, CLIENT, and STORE contain video cassette, client, and store information. They are linked by an association which represents a video cassette rental (K7RENTAL). The K7RENTAL association also contains the attributes DATE and STAFF_ID, which give the date of the rental, and the identity of the staff member who rented out the video cassette.



When you generate a PDM, K7RENTED is generated as a table with five columns, STORE_ID,K7_ID, CLIENT_ID, DATE, and STAFF_ID.



You can use associations exclusively in your CDM, or use both associations and relationships.

Association links

An association is connected to an entity by an association link, which symbolizes the role and the cardinality between an association and an entity.

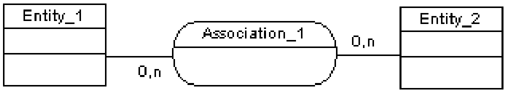
Creating an association with links

The easiest way to create an association between entities is to use the Association Link tool, which will create the association and the necessary links as well.

❖ To create an association with links

1. Click the Association Link tool in the Palette.
2. Click inside the first entity and while continuing to hold down the mouse button, drag the cursor to a second entity. Release the mouse button.

   An association symbol is created between the two entities.



Creating an association without links

You can create an association without links in any of the following ways:

- ♦ Use the Association tool in the diagram Palette.
- ♦ Select Model ➤ Associations to access the List of Associations, and click the Add a Row tool.

- Right-click the model or package in the Browser, and select New ➤ Association.

Once you have created the association, you can link it to the relevant entities by using the Association Link tool.

☞ For general information about creating objects, see the Objects chapter in the *Core Features Guide* .

Association properties

You can modify an object's properties from its property sheet. To open an association property sheet, double-click its diagram symbol or its Browser entry in the Associations folder.

The General tab contains the following properties:

Property	Description
Name	Specifies the name of the item, which should be clear and meaningful, and should convey the item's purpose to non-technical users.
Code	Specifies the technical name of the object, which is used for generating code or scripts, which may be abbreviated, and should not generally include spaces.
Comment	Specifies a descriptive label for the association.
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined.
Number	Specifies the estimated number of occurrences in the physical database for the association (the number of records).
Generate	Specifies that the association will generate a table in a PDM.
Attributes	Specifies the data item attached to an association.
Rules	Specifies the business rules associated with the association.

Association link properties

You can modify an object's properties from its property sheet. To open an association link property sheet, double-click its diagram symbol or its Browser entry in the Association Links folder.

The General tab contains the following properties:

Property	Description
Entity	Specifies the entity connected by the association link. You can use the tools to the right of the list to create an object, browse the complete tree of available objects or view the properties of the currently selected object.
Association	Specifies the association connected by the association link.
Role	Specifies the label indicating the role of the association link.
Identifier	Indicates if the entity is dependent on the other entity.

Cardinality	Specifies the number of occurrences (one or many) that one entity has relative to another. You define the cardinality for each association link between the association and the entity. You can choose between:
	♦ 0,1 - There can be zero or one occurrence of the association in relation to one instance of the entity. The association is not mandatory
	♦ 0,n - There can be zero or many occurrences of the association in relation to one instance of the entity. The association is not mandatory
	♦ 1,1 - One occurrence of the entity can be related to only one occurrence of the association. The association is mandatory
	♦ 1,n - One occurrence of the entity can be related to one or many occurrences of the association. The association is mandatory
	You can change the default format of cardinalities from the registry:
	`HKEY_CURRENT_USER\Software\Sybase\PowerDesigner <version>\ModelOptions\Conceptual Options`
	`CardinalityNotation=1 (0..1) or 2 (0,1)`

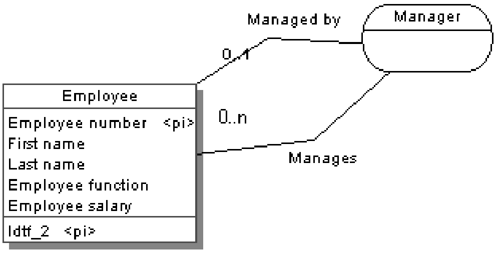
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined.

Creating a reflexive association

A reflexive association is a relationship between an entity and itself.

❖ To create a reflexive association

1. Click the Association Link tool in the Palette.
2. Click inside the entity symbol and, while continuing to hold down the mouse button, drag the cursor a short distance within the symbol, before releasing the button.
3. Drag the resulting association symbol away from entity to make clear its two links to the entity:



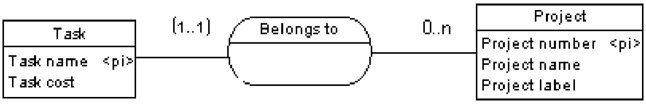
In the example above, the reflexive association Manager expresses that an employee (Manager) can manage other employees.

Defining a dependent association

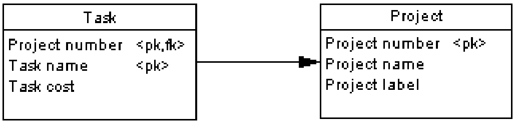
In a dependent association, one entity is partially identified by another. Each entity must have an identifier. In some cases, however, the attributes of an entity are not sufficient to identify an occurrence of the entity. For these entities, their identifiers incorporate the identifier of another entity with which they have a dependent association.

Example

An entity named Task has two entity attributes, TASK NAME and TASK COST. A task may be performed in many different projects and the task cost will vary with each project. To identify each occurrence of TASK COST the unique Task entity identifier is the compound of its *Task name* entity attribute and the *Project number* identifier from the Project entity.



When you generate a PDM, the TASK table contains the PROJECT NUMBER column as a foreign key, which is also a primary key column. The primary key therefore consists of both PROJECT NUMBER and TASK NAME columns.



Association link identifiers and associations
The same association can not have two identifier association links.

❖ To define a dependent association

1. Double-click an association link symbol to display the association link property sheet.

2. Select the Identifier check box and then click OK to return to the model..

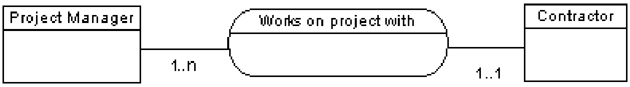
   The cardinality of the association link is enclosed in parenthesis to indicate that the association link is an identifier.

Changing an association into an associative entity

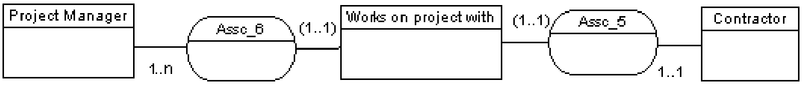
You can transform an association into an associative entity linked by two associations. The associative entity gets the name and code of the association. The two new associations handle cardinality properties.

Example

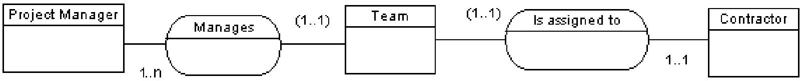
Two entities PROJECT MANAGER and CONTRACTOR are linked by the association WORKS ON PROJECT WITH:



You can represent this association with an associative entity:



The two new associations can be represented as follows:



❖ To change an association into an associative entity

1. Right-click an association symbol, and select Change to Entity from the contextual menu.

   An associative entity that is linked to two associations replaces the original association. The associative entity takes the name of the original association.

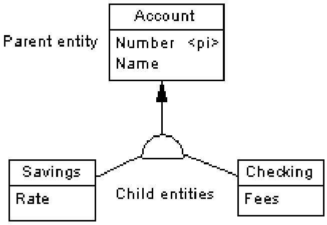
Creating an association attribute

The tools used for creating association attributes on this tab are the same as those for creating entity attributes. For more information, see "Creating an attribute" on page 27.

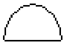
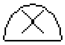
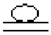
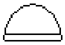
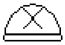
Inheritances (CDM/LDM)

An **inheritance** allows you to define an entity as a special case of a more general entity. The general, or supertype (or parent) entity contains all of the common characteristics, and the subtype (or child) entity contains only the particular characteristics.

In the example below, the Account entity represents all the bank accounts in the information system. There are two subtypes: checking accounts and savings accounts.



The inheritance symbol displays the inheritance status:

IDEF1X	E/R and Merise	Description
		Standard
—		Mutually exclusive inheritance
		Complete inheritance
—		Mutually exclusive and complete inheritance

Inheritances in the Barker notation

There is no separate inheritance object in the Barker notation. You represent an inheritance by placing one entity symbol on top of another. Barker inheritances are always complete and mutually exclusive. The supertype lists its subtypes on the Subtypes tab (see “Entity properties” on page 24).

Creating an inheritance

You can create an inheritance in any of the following ways:

- Use the Inheritance tool in the diagram Palette (see “Creating an inheritance with the Inheritance tool” on page 52).
- Select Model ➤ Inheritances to access the List of Inheritances, and click the Add a Row tool. You will be required to specify a parent entity.
- Right-click the model or package in the Browser, and select New ➤ Inheritance. You will be required to specify a parent entity.

☞ For general information about creating objects, see the Objects chapter in the *Core Features Guide* .

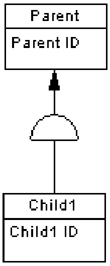
Creating an inheritance with the Inheritance tool

You can use the inheritance tool to create inheritances between entities and to join additional children to an inheritance.

❖ To create an inheritance link using the Inheritance tool

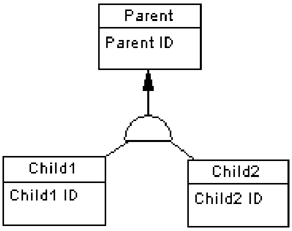
1. Select the Inheritance tool in the Palette.
2. Click the child entity and, while continuing to hold down the mouse button, drag the cursor to the parent entity. Release the mouse button inside the child entity.

   The link is displayed between the two entities and has a half-circle in the middle and an arrowhead that points to the parent entity.



3. [optional] To create additional child entities for the same link, drag and drop an inheritance link from the half-circle to the additional child entity.

   The symbol links all the child entities to the parent.



Dragging an inheritance link to a different child entity
You can change the child entity at the end of an inheritance link by clicking the inheritance link and drag one of its attach points to a different entity.

4. [optional] Double-click the new link in the diagram to open the inheritance property sheet, and enter any appropriate properties.

Inheritance properties

You can modify an object's properties from its property sheet. To open an inheritance property sheet, double-click its diagram symbol or its Browser entry in the Inheritances folder.

The General tab contains the following properties:

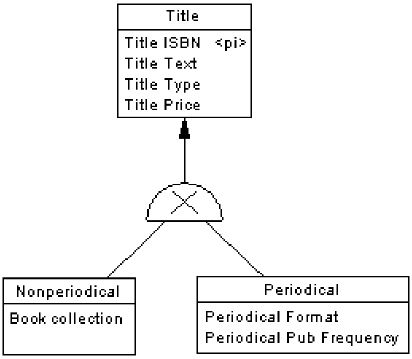
Property	Description
Name	Specifies the name of the item, which should be clear and meaningful, and should convey the item's purpose to non-technical users.
Code	Specifies the technical name of the object, which is used for generating code or scripts, which may be abbreviated, and should not generally include spaces.
Comment	Specifies a descriptive label for the inheritance link.
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined.
Parent	Specifies the name of the parent entity. You can use the tools to the right of the list to create an object, browse the complete tree of available objects or view the properties of the currently selected object.
Mutually exclusive children	Specifies that only one child can exist for one occurrence of the parent entity.
Complete	Specifies that all instances of the parent entity (surtype) must belong to one of the children (subtypes). For example, entity Person has 2 sub-types Male and Female; each instance of entity Person is either a male or a female.

Inheritance property sheet Generation tab

The generation mode defines the physical implementation of an inheritance structure by specifying which entities in an inheritance structure should be generated as tables in the PDM.

The Generation tab contains the following properties:

Property	Description
Generation Mode	Specifies which parts of the inheritance will be generated. You can specify one or both of the following: ♦ Generate parent - Generates a table corresponding to the parent entity, which: • Inherits entity attributes of each child entity • Is affected by child entity relationships • Contains reference to any table which has a many-to-one relationship with child entity ♦ Generate children - Generates a table corresponding to each child entity. The primary key of each child table is the concatenation of the child entity identifier and the parent entity identifier. When this option is selected, you must choose between: • Inherit all attributes – Each table inherits all the entity attributes of the parent entity • Inherit only primary attributes - Each table inherits only the identifier of the parent entity Note that, if you do not select Generate Children, you can control the generation of individual child tables using the option Generate in the property sheet of each child entity.

Property	Description
Specifying attributes	In the case of parent-only generation, you can choose to define a **specifying attribute**, an entity attribute that is defined for a parent entity which differentiates occurrences of each child. In the example below, the TITLE entity has two children, NONPERIODICAL and PERIODICAL. As only the parent table TITLE will be generated in a PDM, a specifying entity attribute PERIODICAL is defined for the inheritance link to differentiate between the two child entities. In the PDM, each of the child entity attributes will generate columns in the table TITLE, and the specifying entity attribute PERIODICAL will generate a corresponding column PERIODICAL. The values of this column indicate whether an instance of TITLE is a periodical or not.  Title Title ISBN <pi> Title Text Title Type Title Price Nonperiodical Book collection Periodical Periodical Format Periodical Pub Frequency The tools available on this tab for creating specifying attributes are the same as those for creating entity attributes. For more information, see "Creating an attribute" on page 27.

Inheritance property sheet Children tab

The Children tab list the child entities attached to the inheritance.

❖ To add a child entity to an inheritance link

1. Open an inheritance property sheet and click the Children tab:
2. Click the Add Children tool to open a selection window listing all the entities available in the model.
3. Select one or more entities, and then click OK to return to the inheritance property sheet.

   The new entity is added to the list of child entities.

4. Click OK to return to the diagram.

   The new child entity is linked to the inheritance symbol in the diagram.

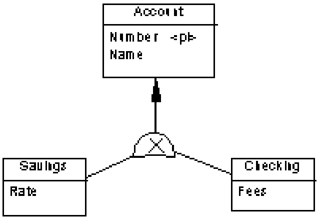
Making inheritance links mutually exclusive

When an inheritance link is mutually exclusive, one occurrence of the parent entity cannot be matched to more than one child entity. This information is for documentation only and has no impact in generating the PDM.

To make an inheritance link mutually exclusive, open the inheritance property sheet and select the Mutually Exclusive Children check box. Then click OK to return to the diagram.

The mutually exclusive inheritance link displays an X on its half-circle symbol.

In the diagram below, the inheritance link is mutually exclusive, meaning that an account is either checking or savings, but never both.



Identifier Migration through an Inheritance (LDM)

In an LDM, primary identifiers of a parent entity always migrate to the child entity. The migration of other attributes of the parent entity depends on which inheritance option is selected.

If a child entity is not generated, its attributes migrate to its parent entity.

CHAPTER 3

Building Physical Diagrams

About this chapter

This chapter describes how to build physical diagrams, and how to create and modify the associated objects.

Contents

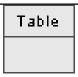
Physical Diagram Basics

A physical diagram allows you to define a database structure from the physical implementation point of view. It takes into account the physical resources: DBMS, data storage structures and software, to describe the structure of the database.

You build a physical diagram at the end of the data analysis process, before you start the software programming. The physical diagram allows you to define how data from conceptual model are implemented in the database.

Physical diagram objects

You can create the following objects in a physical diagram:

Object	Tool	Symbol	Description
Table	▦		Collection of rows (records) that have associated columns (fields). See "Tables (PDM)" on page 62.
Column	[none]	[none]	Data structure that contains an individual data item within a row (record), model equivalent of a database field. See "Columns (PDM)" on page 86.
Primary key	[none]	[none]	Column or columns whose values uniquely identify each row in a table, and are designated as the primary identifier of each row in the table. See "Keys (PDM)" on page 95.
Alternate key	[none]	[none]	Column or columns whose values uniquely identify each row in a table, and which is not a primary key. See "Keys (PDM)" on page 95.
Foreign key	[none]	[none]	Column or columns whose values depend on and migrate from a primary or alternate key in another table. See "Keys (PDM)" on page 95.
Index	[none]	[none]	Data structure associated with one or more columns in a table, in which the column values are ordered in such a way as to speed up access to data. See "Indexes (PDM)" on page 103.

Object	Tool	Symbol	Description
Default	[none]	[none]	[certain DBMSs] A default value for a column. See "Defaults (PDM)" on page 111.
Domain	[none]	[none]	Defines valid values for a column. See "Domains (CDM/LDM/PDM)" on page 115.
Sequence	[none]	[none]	[certain DBMSs] Defines the form of incrementation for a column. See "Sequences (PDM)" on page 124.
Abstract data type	[none]	[none]	[certain DBMSs] User-defined data type. See "Abstract Data Types (PDM)" on page 129.
Reference	⊶	→	Link between a primary or an alternate key in a parent table, and a foreign key of a child table. Depending on its selected properties, a reference can also link columns that are independent of primary or alternate key columns. See "References (PDM)" on page 138.
View	▦	View	Data structure that results from a SQL query and that is built from data in one or more tables. See "Views (PDM)" on page 152.
View Reference	⊶	→	Link between a table and a view. See "View References (PDM)" on page 169.
Trigger	[none]	[none]	A segment of SQL code associated with a table or a view. See "Trigger Overview in the Building Triggers and Procedures chapter.
Procedure	⚙	⚙ Proc	Precompiled collection of SQL statements stored under a name in the database and processed as a unit. See "Stored Procedures and Functions" in the Building Triggers and Procedures chapter.

Object	Tool	Symbol	Description
Database	[none]	[none]	The database of which the PDM is a representation. See “Creating a Database” in the Getting Started with Data Modeling chapter.
Storage	[none]	[none]	A partition on a storage device. See “Tables spaces and Storages” in the Generating a Database from a PDM chapter.
Tablespace	[none]	[none]	A partition in a database. See “Tables spaces and Storages” in the Generating a Database from a PDM chapter.
User	[none]	[none]	A person who can log in or connect to the database. See “Users (PDM)” in the Building a Database Access Structure chapter.
Role	[none]	[none]	A predefined user profile. See “Roles (PDM)” in the Building a Database Access Structure chapter.
Group	[none]	[none]	Defines privileges and permissions for a set of users. See “Groups (PDM)” in the Building a Database Access Structure chapter.
Synonym	[none]	[none]	An alternative name for various types of objects. See “Synonyms (PDM)” in the Building a Database Access Structure chapter.
Web service	[none]	[none]	Collection of SQL statements stored in a database to retrieve relational data in HTML, XML, WSDL or plain text format, through HTTP or SOAP requests. See “Web Services (PDM)” in the Building Web Services chapter.
Web operation	[none]	[none]	Sub-object of a Web service containing a SQL statement and displaying Web parameters and result columns. See “Web Service Operations (PDM)” in the Building Web Services chapter.

Creating a physical diagram

You can create a physical diagram in an existing PDM in any of the following ways:

- Right-click the model in the Browser and select New ➤ Physical Diagram from the contextual menu

- Right-click the background of any diagram and select Diagram ➤ New Diagram ➤ Physical Diagram from the contextual menu.

To create a new PDM with a physical diagram, select File ➤ New, choose Physical Data Model from the Model type list, choose Physical Diagram as the first diagram, and click OK.

Tables (PDM)

A table represents a collection of data arranged in columns and rows. Tables may contain any of the following objects:

- ♦ Columns are named properties of a table that describe its characteristics (see "Columns (PDM)" on page 86).
- ♦ Indexes are data structures associated with a table that are logically ordered by key values (see "Indexes (PDM)" on page 103).
- ♦ Keys are columns, or combinations of columns, that uniquely identify rows in a table. Each key can generate a unique index or a unique constraint in a target database (see "Keys (PDM)" on page 95).
- ♦ Triggers are segments of SQL code associated with tables, and stored in a database. They are invoked automatically whenever there is an attempt to modify data in associated tables (see the Building Triggers and Procedures chapter).

You can use database-specific physical options to specify physical parameters for tables and many other objects (see "Physical Options" on page 188).

Creating a table

You can create a table in any of the following ways:

- ♦ Use the Table tool in the diagram Palette
- ♦ Select Model ➤ Tables to access the List of Tables, and click the Add a Row tool
- ♦ Right-click the model or package in the Browser, and select New ➤ Table

☞ For general information about creating objects, see the Objects chapter in the *Core Features Guide* .

Table properties

You can modify an object's properties from its property sheet. To open a table property sheet, double-click its diagram symbol or its Browser entry in the Tables folder. The following sections detail the property sheet tabs that contain the properties most commonly entered for tables.

The General tab contains the following properties:

Property	Description
Name	Specifies the name of the item which should be clear and meaningful, and should convey the item's purpose to non-technical users
Code	Specifies the technical name of the item used for generating code or scripts, which may be abbreviated, and should not generally include spaces
Comment	Specifies a descriptive label for the table
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined
Owner	Specifies the name of the table owner. You choose an owner from a list of users. A table can only have one owner at a time. This is normally the table creator
Number	Specifies the estimated number of records in the table in the physical database, which is used to estimate database size. This box is automatically populated during reverse engineering if you select the Statistics check box in the Reverse Engineering dialog box (see "Reverse Engineering from a Live Database" in the "Reverse Engineering a Database into a PDM" chapter). You can enter your own value in this field, or refresh its statistics (along with those for all of the table's columns) at any time by right-clicking the table symbol or its entry in the Browser and selecting Update Statistics from the contextual menu. You can also update the statistics for all tables by selecting Tools ➤ Update Statistics (see "Reverse Engineering Database Statistics" in the "Reverse Engineering a Database into a PDM" chapter).
Generate	Specifies that the table is generated in the database
Dimensional type	Specifies the multidimensional type of the table. You can choose between: ♦ Dimension - see "Dimensions (PDM)" in the Building Multidimensional Diagrams chapter ♦ Fact - see "Facts (PDM)" in the Building Multidimensional Diagrams chapter

Property	Description
Type	Specifies the type of the table. You can choose between: ♦ Relational ♦ Object - for abstract data types ♦ XML - for storing and retrieving data using an XML format. For more information, see "Creating an XML table or view" on page 65

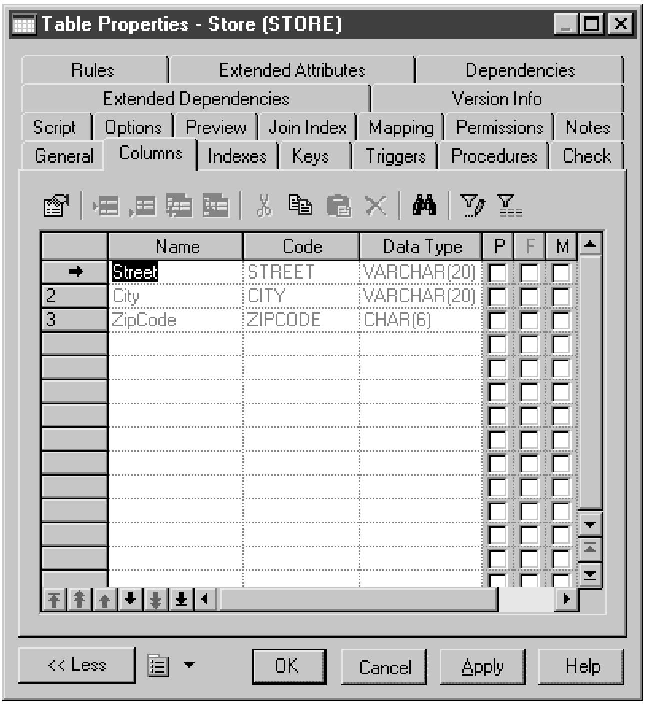
The following tabs are also available:

- Columns - lists the columns associated with the table (see "Columns (PDM)" on page 86).
- Indexes - lists the indexes associated with the table (see "Indexes (PDM)" on page 103).
- Keys - lists the keys associated with the table (see "Keys (PDM)" on page 95).
- Triggers - lists the triggers associated with the table (see The Building Triggers and Procedures chapter).
- Procedures - lists the procedures associated with the table (see the Building Triggers and Procedures chapter).
- Physical Options - list the physical options associated with the table (see "Physical Options" on page 188).
- Preview - displays the SQL code associated with the table (see "Previewing SQL statements" in the Working with Data Models chapter).

Linking a table to an abstract data type

Some DBMS like Oracle or DB2 Common Server support tables based on abstract data types (ADT). A table based on an ADT uses the properties of the ADT and the ADT attributes become table columns.

To link a table to an ADT you have to use the Based On list to select an abstract data type. Not all ADT can be used, only ADT of type Object in Oracle, or Structured in DB2 Common Server appear in the Based On list.



	Name	Code	Data Type	P	F	M
→	Street	STREET	VARCHAR(20)	□	□	□
2	City	CITY	VARCHAR(20)	□	□	□
3	ZipCode	ZIPCODE	CHAR(6)	□	□	□

☞ For more information on abstract data types, see "Abstract Data Types (PDM)" on page 129.

Creating an XML table or view

Some DBMS support tables and views of XML type.

An XML table is used to store an XML document, it does not contain columns. It is possible to associate this table with an XML schema registered in a relational database, in this case the schema is used to validate the XML document stored in the table.

If you select the XML type in the Type list, the Column tab disappears and the following additional properties appear in the table property sheet:

Property	Description
Schema	Allows you to enter the target namespace or the name of an XML model. The schema must be registered in the database to be used for validating XML documents. You can: ♦ type a user-defined schema name ♦ click the Select a registered schema button to connect to a database and select a registered schema If you select an element from an XML model open in the PowerDesigner workspace, the Schema property is automatically initialized with the XML model target namespace. Note that this schema must also be registered in the database to be used for validating XML documents
Element	Allows you to select a root element in the XML document. You can: ♦ type a user-defined element name ♦ click the Select an element button to select an element from the XML models open in the workspace or from the schema registered in the database

❖ To create an XML table

1. Right-click the Table category in the Browser and select New.

   The property sheet of a new table is displayed.

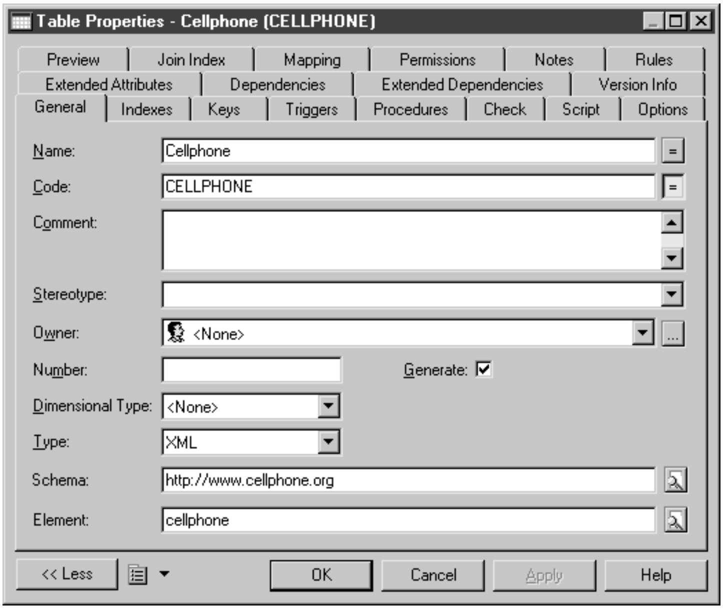
2. Type a table name and a table code.

3. In the Type list, select XML.

   The Columns tab disappears and the Schema and Element boxes appear in the lower part of the General tab.

4. In the Schema box, type the target namespace or name of an XML model or use the Select a registered schema button to select among the registered schema in a selected database.

5. In the Element box, type the name of the root element of the selected schema.



Name: Cellphone
Code: CELLPHONE
Comment:
Stereotype:
Owner: <None>
Number:
Generate: ☑
Dimensional Type: <None>
Type: XML
Schema: http://www.cellphone.org
Element: cellphone

6. Click OK.

Naming a table constraint

A table constraint is a named check that enforces data requirements of check parameters.

Whenever you place a data restriction on a table, a constraint is created automatically. You have the option of specifying a name for the constraint. If you do not, the database creates a default constraint name automatically.

This name helps you to identify and customize a table constraint in scripts for database creation and modification.

❖ To name a table constraint

1. Double-click a table in the diagram to display its property sheet, and click the Check tab.

2. Click the User Defined button to the right of the Constraint Name box, and type changes to the constraint name in the Constraint Name box.

> **Undo changes to a constraint name**
> You can always return to the default constraint name by re-clicking the User-Defined button.

3. Click OK.

☞ For more information, see “Check Parameters (CDM/LDM/PDM)” on page 174.

Creating external tables

You can create external tables when you need to access data in a remote table. The external table has all the properties of the remote table but it does not contain any data locally.

External tables are metamodel extensions defined in the profile category of an extended model definition attached to a PDM. For example, in Sybase ASA, external tables are called proxy tables and a specific extended model definition is delivered to let you design these tables.

☞ For more information on designing proxy tables in Sybase ASA, see section Working with proxy tables in Sybase ASA in chapter DBMS-specific Features.

☞ For more information on designing proxy tables in Sybase ASE, see section Working with proxy tables in Sybase ASE in chapter DBMS-specific Features.

Denormalizing Tables and Columns

Database normalization consists in eliminating redundancy and inconsistent dependencies between tables. While normalization is generally considered the goal of database design, denormalization, the deliberate duplication of certain data in order to speed data retrieval, may be appropriate in certain cases:

- Critical queries rely upon data from more than one table
- Many calculations need to be applied to one or many columns before queries can be successfully answered
- Tables need to be accessed in different ways by different users during the same timeframe
- Certain columns are queried a large percentage of the time

When deciding whether to denormalize, you need to analyze the data access requirements of the applications in your environment and their actual performance characteristics. Often, good indexing and other solutions solve many performance problems rather than denormalization.

Denormalization may be accomplished in several ways:

- **Horizontal partitioning** is used to divide a table into multiple tables containing the same columns but fewer rows

- **Vertical partitioning** is used to divide a table into multiple tables containing the same number of rows but fewer columns
- **Table collapsing** is used to merge tables in order to eliminate the join between them
- **Column denormalization** is used to repeat a column in tables in order to avoid creating a join between tables

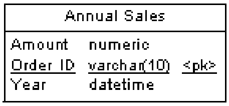
The following sections explain how to implement these denormalization techniques in PowerDesigner.

Creating horizontal partitions

Horizontal partitioning consists in segmenting a table into multiple tables each containing a subset of rows and the same columns as the partitioned table in order to optimize data retrieval. You can use any column, including primary keys, as partitioning criteria.

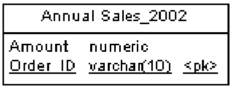
Example

In this example, the table Annual_Sales contains the following columns:

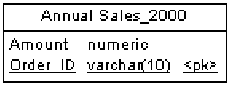


Annual Sales		
Amount	numeric	
Order ID	varchar(10)	<pk>
Year	datetime	

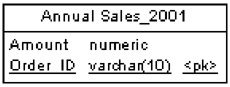
This table may contain a very large amount of data. You could optimize data retrieval by creating horizontal partitions by year. The result is as follows:



Annual Sales_2002		
Amount	numeric	
Order ID	varchar(10)	<pk>



Annual Sales_2000		
Amount	numeric	
Order ID	varchar(10)	<pk>



Annual Sales_2001		
Amount	numeric	
Order ID	varchar(10)	<pk>

Horizontal partitioning has the following pros and cons:

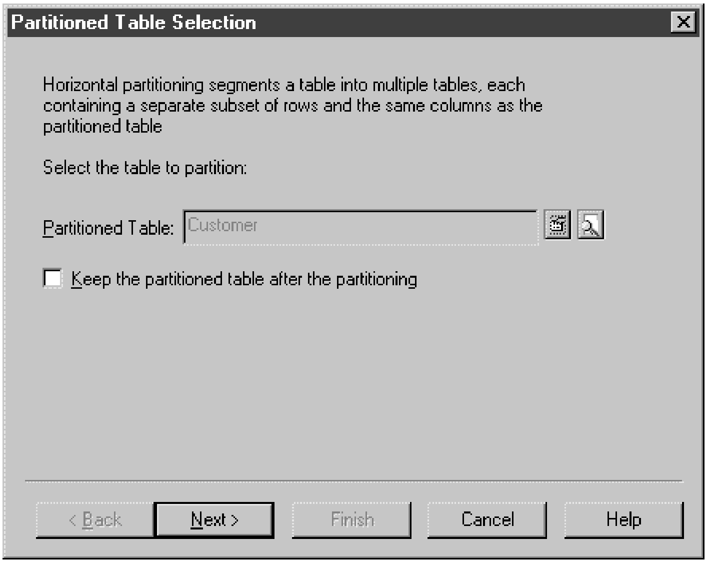
Pros	Cons
Improve the query response time	Requires additional joins and unions to retrieve data from multiple tables
Accelerate incremental data backup and recovery	Requires more intelligent queries to determine which table contains the requested data
Decrease time required to load into indexed tables	Requires additional metadata to describe the partitioned table

Horizontal Partitioning Wizard

You can partition tables horizontally using the Horizontal Partitioning Wizard.

❖ To partition a table with the Horizontal Partitioning Wizard

1. Select Tools ➤ Denormalization ➤ Horizontal Partitioning, or right-click a table in the diagram and select Horizontal Partitioning from the contextual menu, in order to open the Horizontal Partitioning Wizard:



Partitioned Table Selection

Horizontal partitioning segments a table into multiple tables, each containing a separate subset of rows and the same columns as the partitioned table

Select the table to partition:

Partitioned Table: Customer

□ Keep the partitioned table after the partitioning

2. Select the table to partition and select the check box if you want to keep the original table after partitioning. Then click Next to go to the Partition Definition page.

3. The Partition Definition page allows you to create as many partitions as you need with the Insert and Add a row tools. The name of each partition must be unique in the model. A table will be created for each partition

you specify, and will take the name of the relevant partition. Then click Next to go to the Discriminant Column Selection page.

4. The Discriminant Column Selection page allows you to specify the columns that will be used as partition criteria using the Add Columns tool. These columns will not be included in the partitions. Then click Next to go to the Partitioning Information page.

5. The Partitioning Information page allows you to specify a name and code for the transformation object that will be created together with the partitions. Then click Finish.

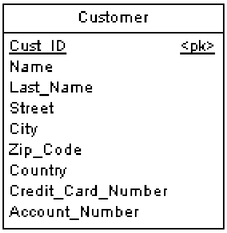
   The table is partitioned, a horizontal partitioning object is created, and all references to the original table are created on each partition table.

Creating vertical partitions

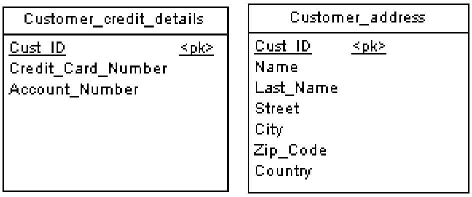
Vertical partitioning consists in segmenting a table into multiple tables each containing a subset of columns and the same number of rows as the partitioned table. The partition tables share the same primary key.

Example

The table Customer contains the following columns:



This table can be divided in two tables corresponding to different aspects of the table. You can use the Vertical Partitioning Wizard to split the table as follows:



Vertical partitioning has the following pros and cons:

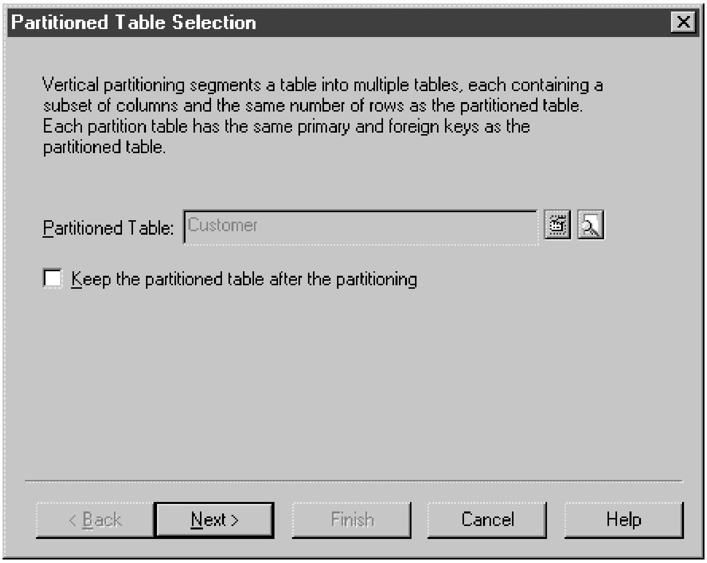
Pros	Cons
Improve the query response time	Requires additional joins and unions to retrieve data from multiple tables
Allows you to split data requiring different levels of protection, you can store confidential information in a special partition	Requires more intelligent queries to determine which table contains the requested data
	Requires additional metadata to describe the partitioned table

Vertical Partitioning Wizard

You can partition tables vertically using the Vertical Partitioning Wizard. The key columns of the partitioned table are duplicated whereas the other columns are distributed among the partition tables. PowerDesigner verifies that all the columns of the partitioned table are used in the partition tables.

❖ To partition a table with the Vertical Partitioning Wizard

1. Select Tools ➤ Denormalization ➤ Vertical Partitioning, or right-click a table in the diagram and select Vertical Partitioning from the contextual menu, in order to open the Vertical Partitioning Wizard:



Partitioned Table Selection

Vertical partitioning segments a table into multiple tables, each containing a subset of columns and the same number of rows as the partitioned table. Each partition table has the same primary and foreign keys as the partitioned table.

Partitioned Table: Customer

□ Keep the partitioned table after the partitioning

2. Select the table to partition and select the check box if you want to keep the original table after partitioning. Then click Next to go to the Partition Definition page.

3. The Partition Definition page allows you to create as many partitions as

you need with the Insert and Add a row tools. The name of each partition must be unique in the model. A table will be created for each partition you specify, and will take the name of the relevant partition. Then click Next to go to the Discriminant Column Selection page.

4. The Discriminant Column Selection page allows you to specify which columns will be included in each partition table. Drag columns from the Available columns pane, and drop them onto the appropriate partition table in the Columns distribution pane, or use the Add and Remove buttons at the bottom of each pane. When all your columns are allocated, click Next to go to the Partitioning Information page.

5. The Partitioning Information page allows you to specify a name and code for the transformation object that will be created together with the partitions. Then click Finish.

   The table is partitioned, a vertical partitioning object is created, and all references to the original table are created on each partition table.

Creating table collapsings

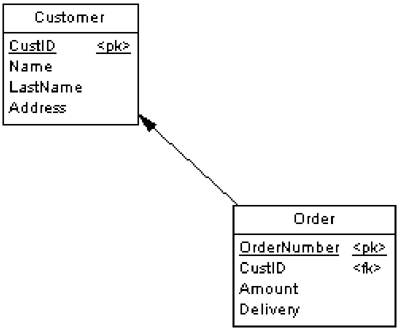
Table collapsing consists in merging tables into a single table in order to eliminate joins and to improve query performance.

The generated table gathers the columns of the merged tables. All incoming and outgoing references to the input tables are preserved in the resulting table. When the collapsed tables are related by references, the following occurs:

- The parent column of the join is no longer needed, thus removed
- The columns of the parent table are duplicated
- The foreign keys of the children are removed, but their columns are preserved in the resulting table

Example

Tables Customer and Order are linked together.



To optimize data retrieval in the database, you collapse both tables into a single table to eliminate the join. The result is a single table (with 2 synonym symbols) with the primary key of the child table:

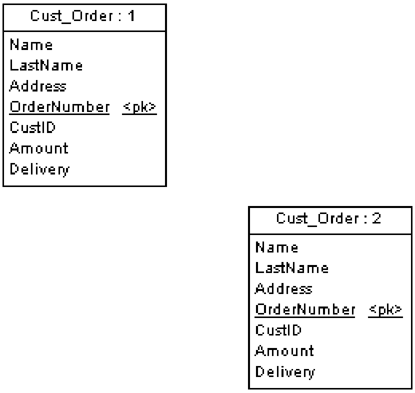


Table Collapsing Wizard

The Table Collapsing Wizard lets you merge multiple tables into a single table. You can collapse tables related to each other with a reference or tables with identical primary keys.

❖ To combine multiple tables with the Table Collapsing Wizard

1. Select Tools ➤ Denormalization ➤ Table Collapsing, or right-click a reference between the tables to collapse and select Table Collapsing from the contextual menu, in order to open the Table Collapsing Wizard:

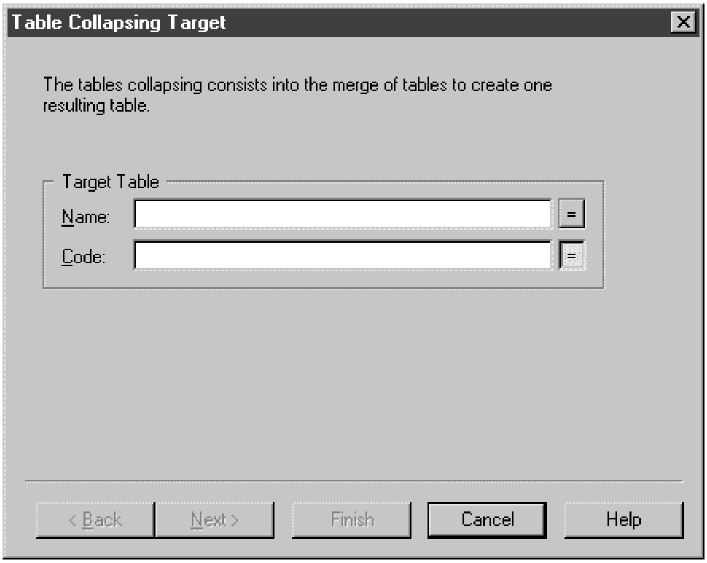


Table Collapsing Target

The tables collapsing consists into the merge of tables to create one resulting table.

Target Table

Name:

Code:

2. Specify a name and code for the target table to be created, and then click Next to go to the Input Table Selection page.

3. The Input Table Selection page allows you to select the tables to collapse with the Add Tables tool. Select the check box if you want to keep the original tables after collapsing, and then click Next to go to the Table Collapsing Information page.

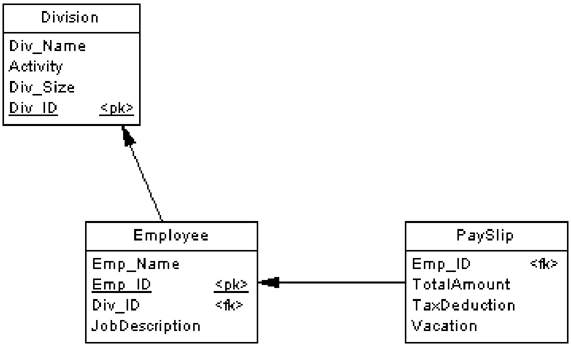
4. The Table Collapsing Information page allows you to specify a name and code for the transformation object that will be created together with the table collapsing. Then click Finish.

   The selected tables are collapsed, and a table collapsing object is created.

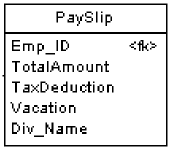
Denormalizing columns

You can denormalize columns to eliminate frequent joins using column denormalization.

Example



In this example, you want to have the division name printed on the pay slip of each employee, however, you do not want to create a join between those tables. You can denormalize columns in order to have column Div_Name in table PaySlip:



Column denormalization eliminates joins for many queries, however it requires more maintenance and disk space.

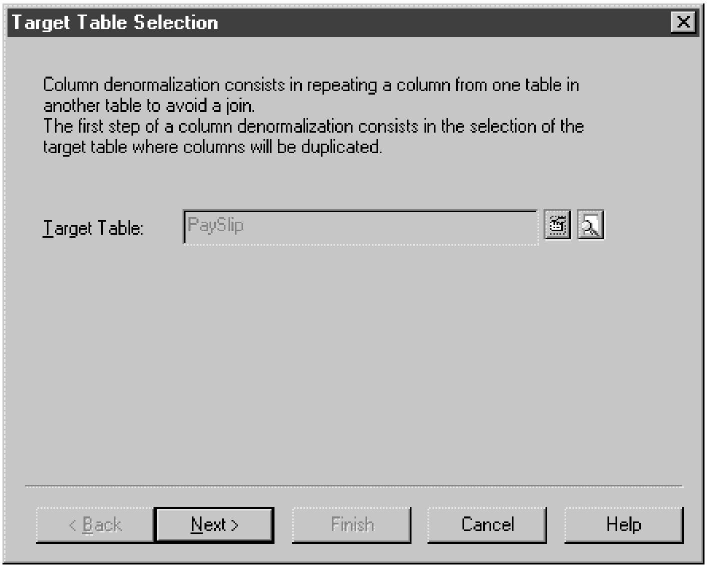
Column Denormalization Wizard

The Column Denormalization Wizard lets you duplicate columns in a selected table. The result is a replica of the original column in the target table.

☞ For more information about object replicas, see the Shortcuts and Object Replications chapter in the *Core Features Guide* .

❖ To denormalize a column with the Column Denormalization Wizard

1. Select Tools ➤ Denormalization ➤ Column Denormalization, or right-click a table and select Column Denormalization from the contextual menu, in order to open the Column Denormalization Wizard:



Target Table Selection

Column denormalization consists in repeating a column from one table in another table to avoid a join.
The first step of a column denormalization consists in the selection of the target table where columns will be duplicated.

Target Table: PaySlip

2. Select the table in which you want the denormalized columns to be added, and then click Next to go to the Column Selection page.

3. The Column Selection page allows you to select the columns to replicate. Select one or more columns to replicate, and then click Finish.

   A replication is created for each selected column. You can display the list of replicas from the menu command Model ➤ Replications. Each replica has its own property sheet. For more information about object replicas, see the Shortcuts and Object Replications chapter in the *Core Features Guide* .

Removing a denormalized column

You can move and paste a denormalized column into another model or package in the standard way.

Reverting a denormalized column

You can revert a column denormalization by deleting the duplicated column from the target table property sheet. This automatically removes the column replica. Note that you cannot revert a column denormalization by deleting a column replica from the list of replications.

Denormalization object properties

A denormalization transformation object is automatically created when you partition a table using the Horizontal or Vertical Partitioning Wizard or collapse tables with the Table Collapsing Wizard.

To access this object, select Model ➤ Transformations to open the List of Transformations, select the appropriate object, and then click the Properties tool.

The following properties are available on the General tab:

Property	Description
Name	Specifies the name of the partitioning object. It is recommended to provide a clear name in the Partitioning Wizard
Code	Specifies the code of the partitioning object
Comment	Specifies additional information about the partitioning object.
Partitioned table	[partitionings only] Specifies the name of the table used to create the table partitions.
Discriminant Columns	[horizontal partitionings only] Specifies the name and code of the columns used as partition criteria
Target table	[table collapsings only] Specifies the name of the table resulting from the collapsing of selected tables

Partitions tab

The Partitions tab is only available for partitionings, and lists the tables associated with the partitioning. The following actions can be performed on this tab:

- Open the property sheets of the partition tables.
- Add more partitions and edit the properties of the corresponding tables.
- Add comments to identify the different partitions
- Delete partitions and their corresponding tables. When you delete a partition, you are prompted to specify whether you want to delete the corresponding table. You can delete a partition and keep its table, but you cannot delete a table and keep an empty partition

Partition Columns tab

The Partition Columns tab is only available for vertical partitionings, and displays the distribution of columns between the partition tables. You can drag and drop columns to reallocate them between tables.

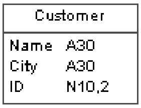
Source Tables tab

The Source Tables tab is only available for table collapsings, and displays the tables that were collapsed. These tables will no longer exist unless you selected to keep the original tables in the Table Collapsing Wizard.

Example: Intermodel generation and horizontal partitions

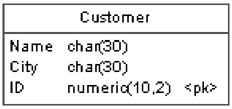
When you update a PDM generated from another model, any horizontal partitioning is preserved.

For example, the Sales CDM contains the entity Customer:



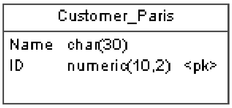
Customer	
Name	A30
City	A30
ID	N10,2

You generate the Sales PDM from the CDM, and the Customer entity is generated to the Customer table:

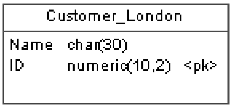


Customer		
Name	char(30)	
City	char(30)	
ID	numeric(10,2)	<pk>

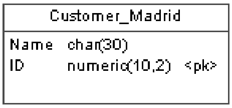
You partition this table using City as the criterion. The City column is excluded from the partition tables:



Customer_Paris		
Name	char(30)	
ID	numeric(10,2)	<pk>

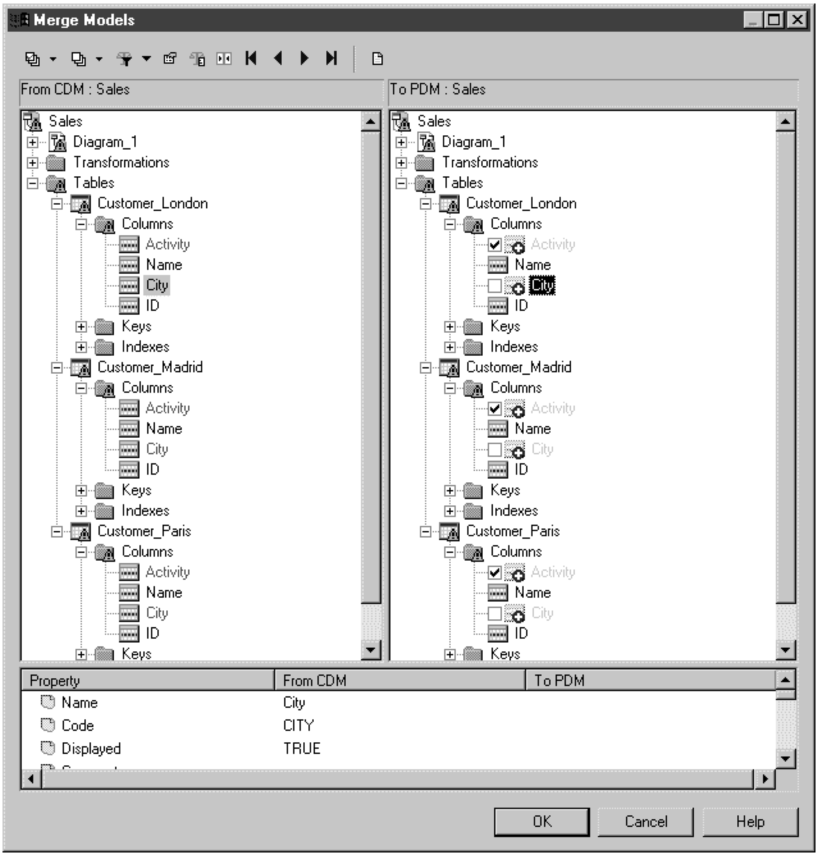


Customer_London		
Name	char(30)	
ID	numeric(10,2)	<pk>



Customer_Madrid		
Name	char(30)	
ID	numeric(10,2)	<pk>

You modify the CDM by adding an Activity attribute to the Customer entity, and regenerate the PDM in update mode. The partitions are taken into account in the merge dialog box: The new Activity attributes are selected by default, while the City criteria columns are not selected.

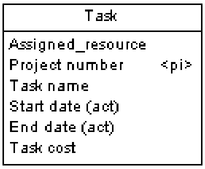


Property	From CDM	To PDM
Name	City	
Code	CITY	
Displayed	TRUE	

Example: Intermodel generation and vertical partitions

When you generate in update mode a PDM from a PDM, a CDM or an OOM, vertical partitioning is preserved.

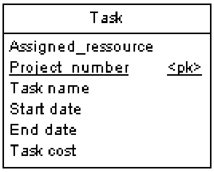
For example, you build a CDM to design the project management process, this model contains entity Task:



Task

Assigned_resource
Project number <pi>
Task name
Start date (act)
End date (act)
Task cost

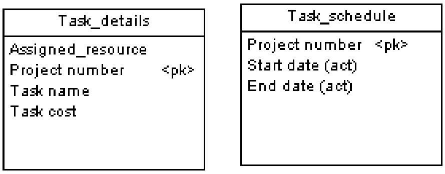
The CDM is generated in a PDM, entity Task becomes table Task:



Task

Assigned_ressource
Project number <pk>
Task name
Start date
End date
Task cost

You decide to split the table in two table partitions: one table contains the details about the task, the other table contains the task schedule:



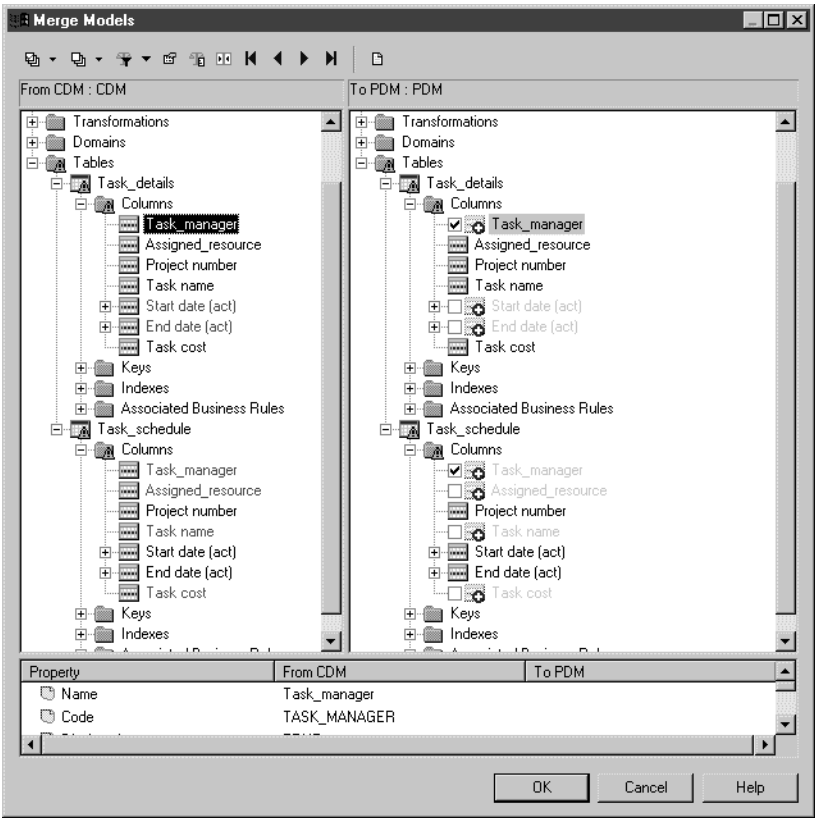
Task_details

Assigned_resource
Project number <pk>
Task name
Task cost

Task_schedule

Project number <pk>
Start date (act)
End date (act)

You modify the CDM and regenerate the PDM in update mode. The partitions are taken into account in the merge dialog box as you can see in the following dialog box: CDM changes (creation of the Task_Manager attribute) are selected by default, and column modifications related to partition creation are not selected.



Property	From CDM	To PDM
Name	Task_manager	
Code	TASK_MANAGER	

Removing partitionings and table collapsings

You can remove partitionings or table collapsings and either keep or remove the associated tables.

❖ To remove a denormalization and keep the associated tables

1. Select Model ➤ Transformations to open the List of Transformations.
2. Select the denormalization object, and then click the Delete tool.

   The denormalization object (and, for partitionings, its partitions) are deleted, but the corresponding tables are retained, and become independent from each other.

❖ To remove a denormalization as well as the associated tables

1. Select Model ➤ Transformations to open the List of Transformations.
2. Select the denormalization object, and then click the Cancel tool.

Cancel Transformation tool
The Cancel Transformation tool is only available if the selected denormalization object is based upon a table generated from another model. You can recover the original table by regenerating it from the source model.

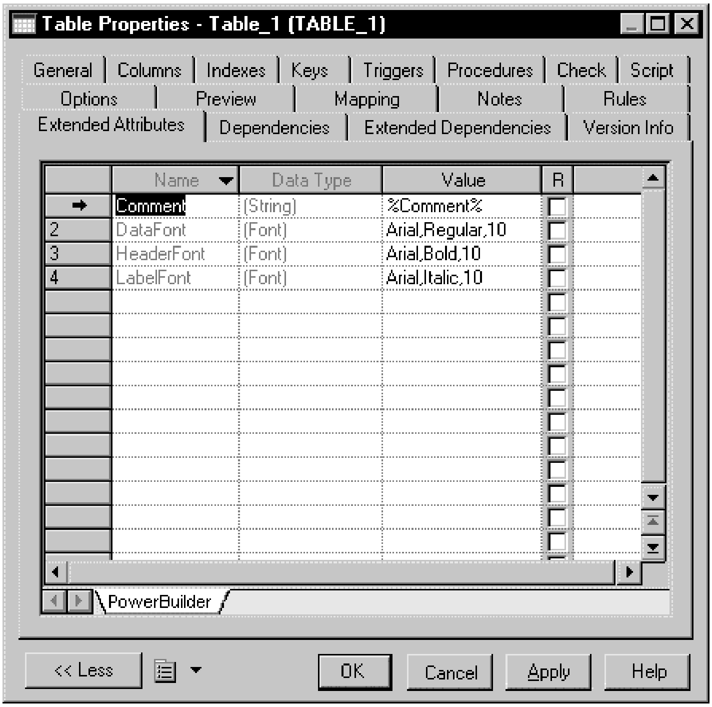
The denormalization object (and, for partitionings, its partitions) are deleted, as well as the corresponding tables.

Moving denormalizations
You cannot move or paste a denormalization object to another model or package.

Using PowerBuilder Extended Attributes

When designing tables to be used in a PowerBuilder DataWindow, you can manage the extended attributes which PowerBuilder uses to store application-based information, such as label and heading text for columns, validation rules, display formats, and edit styles.

PowerDesigner supports certain columns for two PowerBuilder system tables, **PBCatTbl** (for tables) and **PBCatCol** (for columns), on the Extended Attributes tab of tables and columns available in a model to which the PowerBuilder extended model definition is attached:



	Name	Data Type	Value	R
➔	Comment	(String)	%Comment%	☐
2	DataFont	(Font)	Arial,Regular,10	☐
3	HeaderFont	(Font)	Arial,Bold,10	☐
4	LabelFont	(Font)	Arial,Italic,10	☐

Attaching the PowerBuilder XEM

In order to use the PowerBuilder extended attributes, you must first attach the PowerBuilder extended model definition to your PDM:

❖ To attach the PowerBuilder extended model definition

1. In your PDM, select Model ➤ Extended Model Definition to open the List of Extended Model Definitions.

2. Click the Import an Extended Model Definition tool to open the Extended Model Definition Selection dialog box.

3. Select PowerBuilder and specify whether you want to share the XEM or copy it to your model.

4. Click OK to attach the XEM and click OK to close the List of Extended Model Definitions and return to your model.

   The attributes are now available for editing on the PowerBuilder tab of tables and columns.

Generating PowerBuilder extended attributes

You can update the PowerBuilder extended attribute system tables by performing a PowerBuilder extended attribute generation.

During generation, certain extended attributes may contain variables in their values, which are translated during generation, for example to access object properties. The following object properties are translated during generation:

Object	Property
Table	Comment
Column	Comment Label Header Initial value

This automated process uses the PowerDesigner generation template language (see the Customizing Generation with GTL chapter in *Customizing and Extending PowerDesigner*).

❖ To generate PowerBuilder extended attributes

1. Select Tools ➤ PowerBuilder ➤ Generate Extended Attributes to open the PowerBuilder Extended Attributes Generation dialog box.

2. Click the Connect to a Data Source tool to open the Connect to a Data Source window.

3. Select a machine or file data source and click Connect.

   The selected data source is displayed in the Data Source box in the upper part of the PowerBuilder Extended Attributes Generation dialog box.

4. Select the tables you want to generate.

5. Click OK to start generation.

   The Output window displays the generation messages.

Reverse engineering PowerBuilder extended attributes

The reverse engineering feature reads the PowerBuilder extended attributes contained in a database and writes them into the appropriate tables and columns in a PDM.

During reverse engineering, certain reversed extended attributes are compared with the translated default values in the PowerBuilder extended model definition. It these attributes match, the reversed value is replaced by the default value from the extended model definition.

☞ For more information about objects translated during reverse engineering, see “Generating PowerBuilder extended attributes” on page 83.

This automated process uses the PowerDesigner generation template language (see the Customizing Generation with GTL chapter in *Customizing and Extending PowerDesigner*).

❖ To reverse engineer PowerBuilder extended attributes

1. Select Tools ➤ PowerBuilder ➤ Reverse Extended Attributes.

   The PowerBuilder Extended Attributes Reverse Engineering dialog box is displayed.

2. Click the Connect to a Data Source tool to open the Connect to a Data Source dialog box.

3. Select a machine or file data source and click Connect.

   The selected data source is displayed in the Data Source box in the upper part of the PowerBuilder Extended Attributes Reverse Engineering dialog box.

4. Select the tables you want to reverse engineer.

5. Click OK to start reverse engineering.

   The Output window displays the reverse engineering messages.

Columns (PDM)

A column contains an individual data item within a row. It is the model equivalent of a database column. A column is always defined for a table. When you create a column, it must be assigned a name and code. You can also select a data type for the column. This can be done directly from a list of available data types, or by attaching the column to a domain.

Creating a column

You can create a column in any of the following ways:

♦ Open the Columns tab in the property sheet of a table, and click the Add a Row tool

♦ Right-click a table in the Browser, and select New ➤ Column

☞ For general information about creating objects, see the Objects chapter in the *Core Features Guide* .

Column properties

You can modify an object's properties from its property sheet. To open a column property sheet, double-click its row in the Columns tab of a table or its Browser entry. The following sections detail the property sheet tabs that contain the properties most commonly entered for columns.

The General tab contains the following properties:

Property	Description
Name	Specifies the name of the item, which should be clear and meaningful, and should convey the item's purpose to non-technical users
Code	Specifies the technical name of the object, which is used for generating code or scripts, which may be abbreviated, and should not generally include spaces
Comment	Specifies a descriptive label for the column
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined
Table	Specifies the table which contains column
Data type	Specifies the form of the data corresponding to the column, such as numeric, alphanumeric, boolean, or others

Property	Description
Displayed	When selected, allows the display of the selected column in the table symbol
Length	Specifies the maximum length of the data type
Precision	Specifies the maximum number of places after the decimal point
Identity	When selected, indicates that the data is auto-incremented (not available for all DBMS)
Domain	Specifies the name of the associated domain. You can use the tools to the right of the list to create an object, browse the complete tree of available objects or view the properties of the currently selected object
Computed	When selected, designates that the column is computed from an expression using values from other columns in the table (not available for all DBMS)
Primary key	When selected, designates a column whose values uniquely identify a row in the table
Foreign key	When selected, designates a column that depends on and migrates from a primary key column in another table
Mandatory	When selected, indicates a column that must be assigned a not null value
With default	When selected, indicates if a default value is assigned to the column when a Null value is inserted (not available for all DBMS)

Column property sheet Detail tab

The Detail tab contains the following properties:

Property	Description
Null Values	Specifies the number (or percentage) of column entries which contain null values. You can enter a number or percentage in this field or derive its value from database statistics (see "Updating Column Statistics" on page 88).

Property	Description
Distinct Values	Specifies the number (or percentage) of column entries which contain distinct values. You can enter a number or percentage in this field or derive its value from database statistics (see "Updating Column Statistics" on page 88). For example, you generate a table with 2 columns and 10 rows. You set the percentage of distinct values to 100 % for Column 1 and to 80% for Column 2. This implies that 10 rows will have distinct values in Column 1, and 8 rows in Column 2. When you apply a test data profile with a list generation source to a column with a given percentage of distinct values, PowerDesigner uses the values from the test data profile list. If there are not enough values declared in the list, a warning message is displayed in the Output window to inform you that the distinct value parameter cannot be enforced due to lack of distinct values in the list of values.
Average Length	Specifies the average length of a value. You can enter a number in this field or derive its value from database statistics (see "Updating Column Statistics" on page 88).
Profile	Test data profile selected from the list. Profiles can use characters, numbers or date/time data types. ☞ For more information, see "Using test data" section in the Generating a Database from a PDM chapter.
Computed Expression	Computed expression typed directly in the Computed Expression pane or defined with the SQL Editor (accessed with the Edit tool) which helps you define more complex expressions. ☞ For more information, see "Creating a computed column" on page 89.

Updating Column Statistics

You can enter values in the Null Values, Distinct Values and Average Length fields. Alternatively, you can automatically populate them during reverse engineering by selecting the Statistics check box in the Reverse Engineering dialog box (see "Reverse Engineering from a Live Database" in the "Reverse Engineering a Database into a PDM" chapter).

You can refresh the value of these fields (along with those for all of the table's columns) at any time by right-clicking the table symbol or its entry in the Browser and selecting Update Statistics from the contextual menu. You can also update the statistics for all tables by selecting Tools ➤ Update Statistics (see “Reverse Engineering Database Statistics” in the “Reverse Engineering a Database into a PDM” chapter).

Creating a computed column

A computed column is a column whose content is computed from an expression using values from other columns in the table. The computed column is then filled with the results.

Simple computed expressions can be entered directly in the Computed Expression pane on the Detail tab of the column property sheet. For more complex expressions, use the SQL Editor available through the Edit tool found on the same tab.

Computed columns are not available in all DBMS.

Example

Assume that you want to automatically fill a column with the total sales of widgets. To do this, you can create a computed column that will use the number of widgets multiplied by the widget price:

Column name	Contents	Action on data
Number of widgets	Number of widgets sold	—
Widget price	Price of widgets when sold	—
Widget sales	Total widget sales	Computed by multiplying the first two columns

While our example is very simple, the SQL Editor allows you to define very complex computed column expressions.

☞ For more information on the SQL Editor, see Using SQL tools in the Working with Data Models chapter.

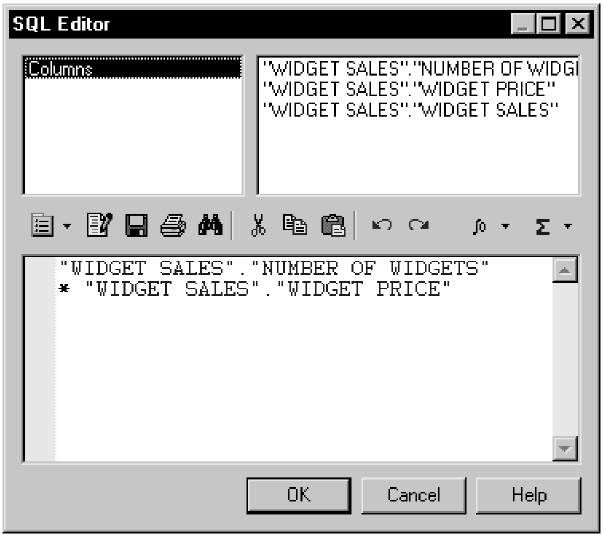
❖ To create a computed column

1. Double-click a table to open its property sheet, and click the Columns tab.

2. Click the Add a Row tool, and then click the Properties tool to open the property sheet for the new column.

3. On the General tab, select the Computed checkbox, and then click the Detail tab.

4. Enter an expression in the Computed Expression box to define the computed column. Alternatively,you can click the Edit with SQL Editor tool to use the SQL Editor.



```
"WIDGET SALES"."NUMBER OF WIDGI
"WIDGET SALES"."WIDGET PRICE"
"WIDGET SALES"."WIDGET SALES"
```

```
"WIDGET SALES"."NUMBER OF WIDGETS"
* "WIDGET SALES"."WIDGET PRICE"
```

In our example, we use the asterisk (*) as an arithmetic operator to multiply the number of widgets by their price.

5. Click OK to return to the column property sheet.

   The expression is displayed in the Computed Expression pane.

6. Click OK in each of the dialog boxes.

Selecting a data type for a column

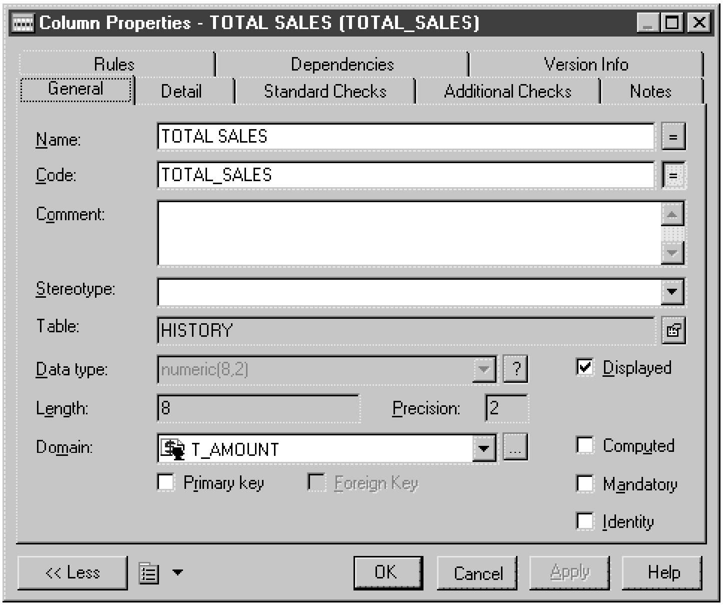
There are two ways to select a data type for a column:

- **Attach the column to a domain** - The domain dictates a data type, a length, and a level of precision, as well as optional check parameters
- **Manually select a data type** - You select a data type along with a length, a level of precision, and optional check parameters

About check parameters
Check parameters indicate data ranges and validation rules. You can set check parameters for domains, tables, and columns.

❖ To select a data type for a column

1. Double-click a table to open its property sheet, and click the Columns tab.
2. Click the required column entry and then click the Properties tool to open its property sheet.



Name: TOTAL SALES

Code: TOTAL_SALES

Comment:

Stereotype:

Table: HISTORY

Data type: numeric(8,2) ☑ Displayed

Length: 8 Precision: 2

Domain: T_AMOUNT □ Computed

□ Primary key □ Foreign Key □ Mandatory

□ Identity

3. Select a data type from the Data Type list or click the Question mark button to open and choose a data type from the Standard Data Types dialog box.

4. If required, enter a data type length and precision.

Undefined data type
If you do not want to select a data type immediately, you can choose the <undefined> data type. When you generate the database, this data type is replaced by the default data type for your database, as defined in the DBMS.

5. Click OK in each of the dialog boxes.

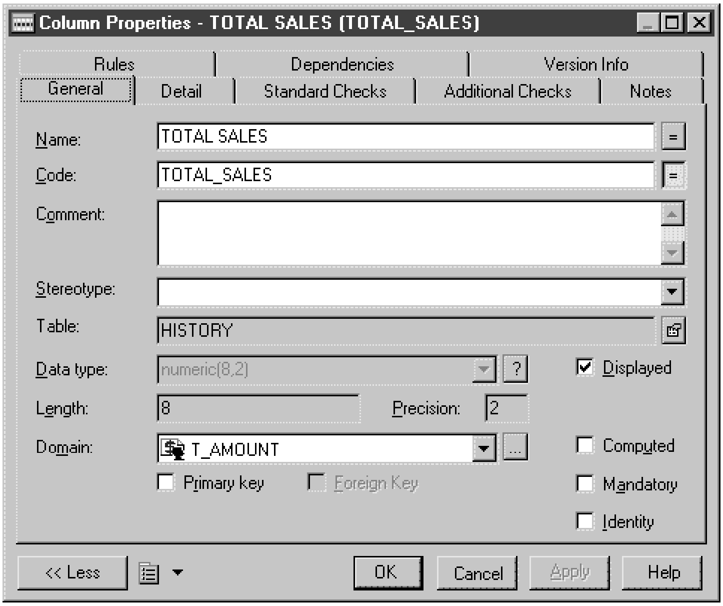
Attaching a column to a domain

If you attach a column to a domain, the domain supplies the data type and related data characteristics. It may also indicate check parameters, and business rules.

❖ To attach a column to a domain

1. Double-click a table to open its property sheet, and click the Columns tab.

2. Click the required column entry and then click the Properties tool to open its property sheet.



Column Properties - TOTAL SALES (TOTAL_SALES)

Name: TOTAL SALES

Code: TOTAL_SALES

Comment:

Stereotype:

Table: HISTORY

Data type: numeric(8,2)

Length: 8

Precision: 2

Domain: T_AMOUNT

☑ Displayed

□ Computed

□ Mandatory

□ Identity

□ Primary key

□ Foreign Key

3. Select a domain from the Domain list and then click OK.

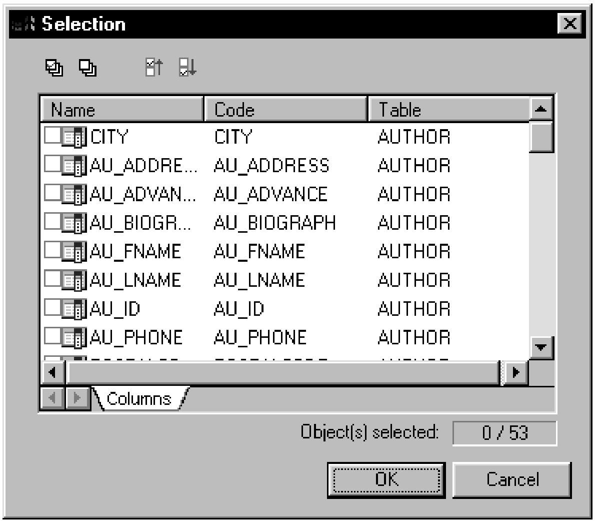
Copying a column to another table

You can copy a column from one table and add it to another table. If the table already contains a column with the same name or code as the copied column, the copied column is renamed. For example, the column PUB_ID is renamed PUB_ID2 when it is copied to a table which already contains a column PUB_ID.

❖ To copy a column to another table

1. Double-click a table to open its property sheet, and click the Columns tab.

2. Click the Add Columns tool to open a selection box listing the columns attached to all other tables in the model.



Name	Code	Table
CITY	CITY	AUTHOR
AU_ADDRE...	AU_ADDRESS	AUTHOR
AU_ADVAN...	AU_ADVANCE	AUTHOR
AU_BIOGR...	AU_BIOGRAPH	AUTHOR
AU_FNAME	AU_FNAME	AUTHOR
AU_LNAME	AU_LNAME	AUTHOR
AU_ID	AU_ID	AUTHOR
AU_PHONE	AU_PHONE	AUTHOR

Object(s) selected: 0 / 53

3. Select one or more columns in the list and then click OK.

   The copied columns appear in the list of columns for the current table.

4. Click OK.

Naming a column constraint

A column constraint is a named check that enforces data requirements of check parameters.

Whenever you place a data restriction on a column, it generates a constraint automatically. You have the option of specifying a name for the constraint. If you do not specify a name for the constraint, PowerDesigner creates a default constraint name automatically.

This name helps you to identify and customize a column constraint in scripts for database creation and modification.

❖ To name a column constraint

1. Open the property sheet of a column and click the Additional Checks tab.

2. Type changes to the constraint name in the Constraint Name box.

The User-Defined button at the end of the box is pressed automatically.

Undo changes to a constraint name
You can always return to the default constraint name by clicking the User-Defined button.

3. Click OK in each of the dialog boxes.

Configuring the display of the list of columns

You can sort the columns in the list in two ways:

- By any property that is displayed in the title bar of the property lists
- By alphabetical or reverse alphabetical order

The listed order is indicated by an arrow head that is displayed at the end of the title bar of the property column. Each time you click a title bar, you change the listed order for that column, according to the displayed arrow.

Each arrow type corresponds to the following list orders:

Arrow type	Listed order
Down arrow	Alphabetically
Up arrow	Reverse alphabetically

For example, when you click the title bar Name, the columns are listed by column name alphabetically when the down arrow is indicated, and in reverse order when the up arrow is indicated.

❖ To configure the display of the list of columns

1. Select Model ➤ Columns to open the List of Columns.
2. Click a property title bar to sort the list by the indicated property.
3. Click OK.

Keys (PDM)

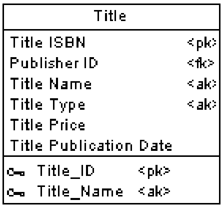
A **key** is a column, or a combination of columns, that uniquely identifies a row in a table. Each key can generate a unique index or a unique constraint in a target database.

The physical diagram supports the following types of keys:

- Primary - Column or combination of columns whose values uniquely identify every row in a table. A table can have only one primary key
- Alternate - Column or combination of columns (not the same column or combination of columns as for a primary key) whose values uniquely identify every row in a table
- Foreign - Column or combination of columns whose values are required to match a primary key, or alternate key, in some other table

Example

The TITLE table shown below has a primary, alternate and foreign key:



Title

Title ISBN <pk>
Publisher ID <fk>
Title Name <ak>
Title Type <ak>
Title Price
Title Publication Date

Title_ID <pk>
Title_Name <ak>

- TITLE_ID is the primary key and consists of the column TITLE ISBN which identifies each book title in the table
- TITLE_NAME is an alternate key containing the columns TITLE NAME and TITLE TYPE. It allows each title to be identified by its name and type, The fact that it is an alternate key indicates that there is a constraint that no two titles of the same type can have the same name

The TITLE table also contains the foreign key column PUBLISHER ID. This column references the primary key column in the Publisher table.

Creating a Key

The method for creating a key depends on the type of key. See the appropriate section:

- "Primary keys" on page 96
- "Alternate keys" on page 98
- "Foreign keys" on page 99

Key properties

You can modify an object's properties from its property sheet. To open a key property sheet, go to the Keys tab of its parent table, or double-click its Browser entry.

The General tab contains the following properties:

Property	Description
Name	Specifies the name of the item, which should be clear and meaningful, and should convey the item's purpose to non-technical users
Code	Specifies the technical name of the object, which is used for generating code or scripts, which may be abbreviated, and should not generally include spaces
Comment	Specifies the descriptive comment for the key
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined
Table	Specifies the name of the table where the key is defined
Constraint name	Specifies the name of the key constraint. PowerDesigner automatically creates a constraint name for a key, it is used during database creation and modification. You can modify the default name, you can also cancel these changes and go back to the default name
Primary key	Indicates if the key is the primary key of the current table. There must be only one primary key in a table, if you select the Primary Key check box in a key property sheet, it replaces an already existing primary key
Cluster	Indicates whether the key constraint is a clustered constraint (for those DBMS that support clustered indexes)

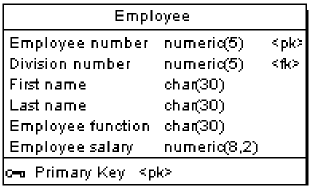
Primary keys

A primary key is the primary identifier for a table, and is attached to one or more columns whose values uniquely identify every row in the table.

Every table must have a primary key, composed of one or more of its columns.

Example

Employee number is the primary key for the table Employee. This means that each employee must have one unique employee number.



Employee		
Employee number	numeric(5)	<pk>
Division number	numeric(5)	<fk>
First name	char(30)	
Last name	char(30)	
Employee function	char(30)	
Employee salary	numeric(8,2)	
Primary Key <pk>		

You can define one or more columns as the primary key of a table from the list of columns.

❖ To designate a primary key

1. Double-click a table in the diagram to open its property sheet.
2. Click the Columns tab, and select the check box in the P column for one or more columns in the list.
3. [optional] Click the Keys tab and rename the key or select it and click the Properties tool to access its property sheet.
4. Click OK.

Rebuilding primary keys

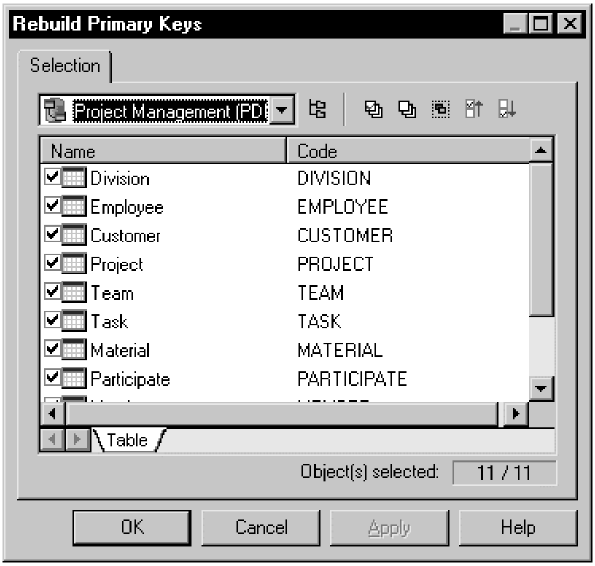
Rebuilding primary keys in a physical diagram updates primary keys for tables.

Rebuilding primary keys is useful following the reverse engineering of a database in which all of the primary keys could not be reverse engineered, or if you did not select the rebuild option for primary keys when you reverse engineered the database. The rebuild option for primary keys creates primary keys for tables that have no key and a single unique index.

You can choose to rebuild all primary keys in your model, or select the tables for which you want to rebuild the primary keys.

❖ To rebuild primary keys

1. Select Tools ➤ Rebuild Objects ➤ Rebuild Primary Keys to open the Rebuild Primary Keys dialog box, which lists all the tables in the current model.



Project Management (PD

Name	Code
☑ Division	DIVISION
☑ Employee	EMPLOYEE
☑ Customer	CUSTOMER
☑ Project	PROJECT
☑ Team	TEAM
☑ Task	TASK
☑ Material	MATERIAL
☑ Participate	PARTICIPATE

Object(s) selected: 11 / 11

Rebuilding primary keys in a package
To rebuild the primary keys in package, select the package from the list at the top of the tab.

To rebuild the primary keys in a sub-package, select the Include Sub-Packages icon next to the list, and then select a sub-package from the dropdown list.

2. Select the tables containing the primary keys that you want to rebuild.

Selecting or clearing all check boxes
You can select all check boxes, or clear all check boxes, by selecting the Select All tool, or Clear All tool, from the toolbar at the top of the tab.

3. Click OK.

Alternate keys

An alternate key is a key attached to one or more columns whose values uniquely identify every row in the table, but which is not a primary key. An alternate key can also be a foreign key. Each alternate key can generate a unique index or a unique constraint in a target database.

Create key tool

You can also select one or several columns and use the Create Key tool in the Columns tab of the table property sheet.

❖ To designate an alternate key

1. Double-click a table in the diagram to open its property sheet.

2. Click the Keys tab, which lists all the keys defined for the table.

3. Click the Add a Row tool, and type a name for the newly created key.

Alternate key naming convention
The naming convention for an alternate key is AK followed by the number of the key column code; for example AK1_CUSNAME.

4. [optional] Type a constraint name in the Constraint Name column.If you do not specify a constraint name, PowerDesigner creates a default constraint name automatically.

Displaying additional property columns
If you do not see the Constraint Name column, display it with the Customize Columns and Filter tool. For more information, see "Customizing object list columns and filtering lists" section in the Objets chapter of the *Core Features Guide* .

5. Click the property tool and confirm the object creation in order to open the property sheet for the new key.

6. Click the columns tab, which lists all the columns to which the key is attached. At key creation, the list is empty.

7. Click the Add Columns tool to open a selection box listing all the columns in the table (except those attached to the primary key).

8. Select one or more columns and click OK in each of the dialog boxes.

Foreign keys

A foreign key is a primary key, or an alternate key, that migrates from another table. Depending on selected model options, a primary key can be automatically migrated to a child table as a foreign key at reference creation.

The columns that are defined in a foreign key can also be user-specified at creation and changed at any time from the Joins tab of the reference property sheet.

☞ For information about the auto-migration of a foreign key, see "Automatic reuse and migration of columns " on page 139.

☞ For more information on defining references, see “References (PDM)” on page 138.

Adding parent table columns to a key

You can add additional columns from the parent table to a primary key or an alternate key.

❖ To add parent table columns to a key

1. Open the key’s property sheet and click the Columns tab.
2. Click the Add Columns tool to open a selection box listing all the columns in the table (except those already attached to the primary key).
3. Select one or more columns and click OK in each of the dialog boxes.

Naming key constraints

Naming key constraints helps you to identify and customize key constraints in scripts for database creation and modification. The constraint name gives you greater flexibility for modifying keys in subsequent database generations.

If you do not specify a constraint name, PowerDesigner creates a default constraint name automatically.

Naming a primary key constraint

A primary key constraint is a named check that enforces the uniqueness and the presence of values in a primary key column.

You can use the following variable in the name of a primary key constraint:

Variable	Description
%TABLE%	Code of the table

☞ For a full list of all variables that you can use in PowerDesigner, see the appendix Variables in PowerDesigner.

❖ To name a primary key constraint

1. Double-click a table in the diagram to open its property sheet.
2. Click the Keys tab, select the primary key, and click the Properties tool to open its property sheet.

3. Enter the required name in the Constraint Name box.

   The User-Defined button to the right of the Constraint box is pressed automatically. You can return to the default constraint name by re-clicking the User-Defined button.

4. Click OK in each of the dialog boxes.

Naming an alternate key constraint

You can use the following variable in the name of a alternate key constraint:

Variable	Description
%AK%	Code of the alternate key
%AKNAME%	Name of the alternate key
%TABLE%	Code of the table

☞ For a full list of all variables that you can use in PowerDesigner, see the appendix Variables in PowerDesigner.

❖ To name an alternate key constraint

1. Double-click a table in the diagram to open its property sheet.

2. Click the Keys tab, select an alternate key, and click the Properties tool to open its property sheet.

3. Enter the required name in the Constraint Name box.

   The User-Defined button to the right of the Constraint box is pressed automatically. You can return to the default constraint name by re-clicking the User-Defined button.

4. Click OK in each of the dialog boxes.

Naming a foreign key constraint

You can use the following variable in the name of a foreign key constraint:

Variable	Description
%REFR-NAME%	Name of the reference
%REFR-CODE%	Code of the reference
%PARENT%	Code of the parent table
%CHILD%	Code of the child table

☞ For a full list of all variables that you can use in PowerDesigner, see the appendix Variables in PowerDesigner.

❖ To name a foreign key constraint

1. Double-click a reference in the diagram to open its property sheet.
2. Click the Integrity tab and enter the required name in the Constraint Name box.

   The User-Defined button to the right of the Constraint box is pressed automatically. You can return to the default constraint name by re-clicking the User-Defined button.
3. Click OK in each of the dialog boxes.

Indexes (PDM)

An index is a data structure associated with a table that is logically ordered by the values of a key. It improves database performance and access speed.

You normally create indexes for columns that you access regularly, and where response time is important. Indexes are most effective when they are used on columns that contain mostly unique values.

Example

In a table called Author, you create indexes for the primary key Author ID and the column Author name, but not for the column City. The values for city, are not likely to be unique, nor searched regularly, and do not help reduce query time.

Creating an index

You can create the following types of index:

- A user-defined index - Associated with one or more columns
- An index linked to a key - Automatically updated when the key column or columns are modified. An index linked to a key is unique because it uses the same unique set of columns as the key.
- A function-based index - [if supported by the DBMS] Precomputes the value of a function or expression based on one or more columns and stores it in the index. The function or the expression will replace the index column in the index definition. Function-based indexes provide an efficient mechanism for evaluating statements that contain functions in their WHERE clauses.

Index naming conventions

Use the following naming conventions for indexes:

Index	Naming convention
Primary key	Table code followed by PK; for example EMPLOYEE _PK
Foreign key	Table code followed by FK; for example PROJECT _ FK
Alternate key	Table code followed by AK; for example EMPLOYEE _ AK

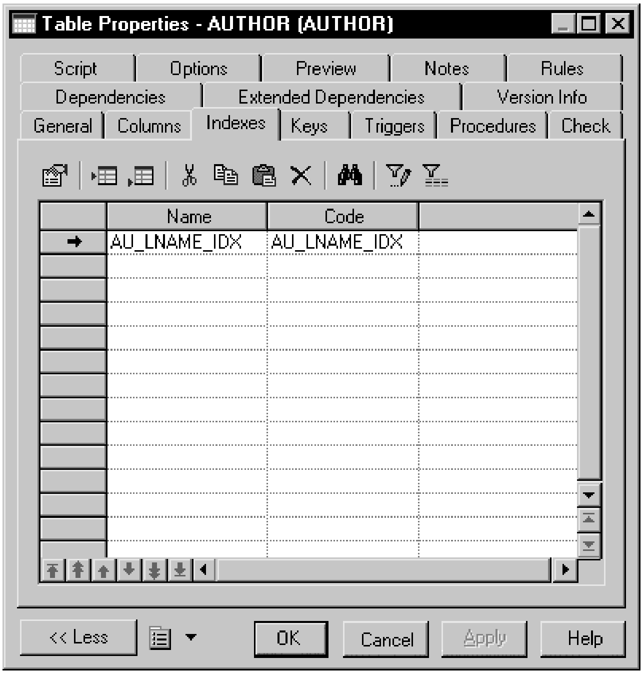
Example

A table contains a compound primary key. This is a primary key designated to more than one column in a table. You create an index and link it to the primary key. If one of the primary key columns is deleted, the corresponding index associated with the column is also deleted.

❖ To create an index

1. Double-click a table symbol to display its property sheet and click the Indexes tab.

2. Click the Add a Row tool and enter an index name and an index code.



	Name	Code
➔	AU_LNAME_IDX	AU_LNAME_IDX

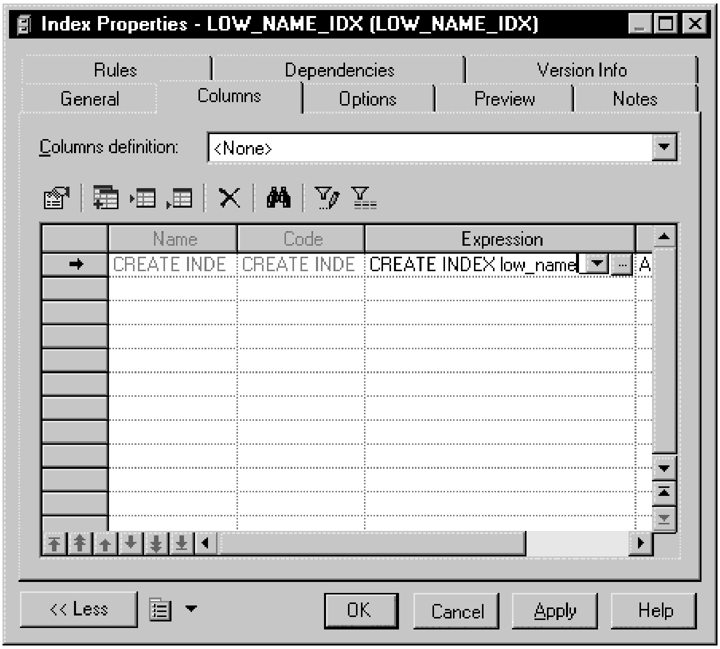
3. Click the Properties tool to open the property sheet of the new index.

4. Type or select any appropriate index properties, and then click the Columns tab.

5. **To create a user defined index:** click the Add Columns tool, select one or more columns from the list, and then click OK

   To create an index linked to a key: select the primary key, an alternate key, or foreign key from the Columns definition list

   To create a function-based index [if supported by the DBMS]: click the Add a Row tool, then click in the Expression column and select the ellipsis button to open the SQL Editor. Enter an expression in the editor and then click OK



Columns definition: <None>

Name	Code	Expression	
CREATE INDE	CREATE INDE	CREATE INDEX low_name	A

6. Select Ascending or Descending in the Sort column.

7. Click OK in each of the dialog boxes.

Reverse engineering function-based index

An index column with an expression has a LONG data type that cannot be concatenated in a string statement during reverse engineering. The only way to bypass this limitation and concatenate this value is to use variables in the query executed to retrieve the adequate information.

In the Oracle 8i and Oracle 8i2 DBMS, the query `SqlListQuery` defined in the Index category contains the following variable used to recover the index expression in a column with the LONG data type.

```
'%SqlExpression.Xpr'||i.table_name||i.index_name||c.column_
        position||'%'
```

☞ For more information on the use of variables in reverse engineering queries, see section Extension mechanism for live database reverse engineering queries, in the DBMS Resource File Reference chapter of the *Customizing and Extending PowerDesigner* manual.

Example

Function-based indexes defined on UPPER(column_name) or LOWER(column_name) can facilitate case-insensitive searches.

You want to define an index that will put all names in lowercase on the table EMPLOYEE in order to ease search. You can define the following index (syntax for Oracle 8i):

```
CREATE INDEX low_name_idx ON EMPLOYEE (LOWER(EMPLNAM))
```

Then the DBMS can use it when processing queries such as:

```
SELECT * FROM EMPLOYEE WHERE LOWER(EMPLNAM)="brown"
```

☞ For general information about creating objects, see the Objects chapter in the *Core Features Guide* .

Index properties

You can modify an object's properties from its property sheet. To open an index property sheet, double-click its diagram symbol or its Browser entry. The following sections detail the property sheet tabs that contain the properties most commonly entered for indexes.

The General tab contains the following properties:

Property	**Description**
Name	Specifies the name of the item, which should be clear and meaningful, and should convey the item's purpose to non-technical users
Code	Specifies the technical name of the object, which is used for generating code or scripts, which may be abbreviated, and should not generally include spaces
Comment	Specifies a descriptive label for the index
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined
Owner	Specifies the name of index owner. You choose an owner from a list of users, the index and table owners can be identical or different. An index can only have one owner at a time. This is normally the index creator. Some DBMS allow you to define an index owner, either identical or different from the table owner. If the DBMS of the current model does not support index owners, the table owner will be automatically assigned to the index after switching to a DBMS that supports index owners.
Table	Specifies the table to index

Property	Description
Type	[Sybase IQ, and Oracle only] Specifies the type of index. You can choose between: ♦ Bitmap – [Oracle] In a bitmap index, a bitmap for each key value is used instead of a list of row Ids ♦ HG – [Sybase IQ] HighGroup indexes are used for GROUP BY, COUNT(DISTINCT) and SELECT DISTINCT statements when data has more than 1000 unique values ♦ HNG – [Sybase IQ] HighNonGroup indexes make equality comparisons, SUM and AVG calculations very fast when data has more than 1000 unique values. Nonequality comparisons can also be done ♦ LF – [Sybase IQ] LowFast indexes are used for columns that have a very low number of unique values. This index also facilitates join index processing. It is one of the two indexes allowed for columns used in join relationships ♦ CMP – [Sybase IQ] Compare indexes are used for columns that store the binary comparison (<, >, or =) of any two distinct columns with identical data types, precision, and scale ♦ WD – [Sybase IQ] Is used to index keywords by treating the contents of a CHAR or VARCHAR column as a delimited list
Unique	Specifies whether an index is a unique index
Cluster	Specifies that the index is a clustered index. A table cannot have more than one clustered index. Note that clusters in Oracle 11 and higher are modeled as extended objects with a <<Cluster>> stereotype. For more information, see the DBMS-Specific Features chapter.

The following tabs are also available:

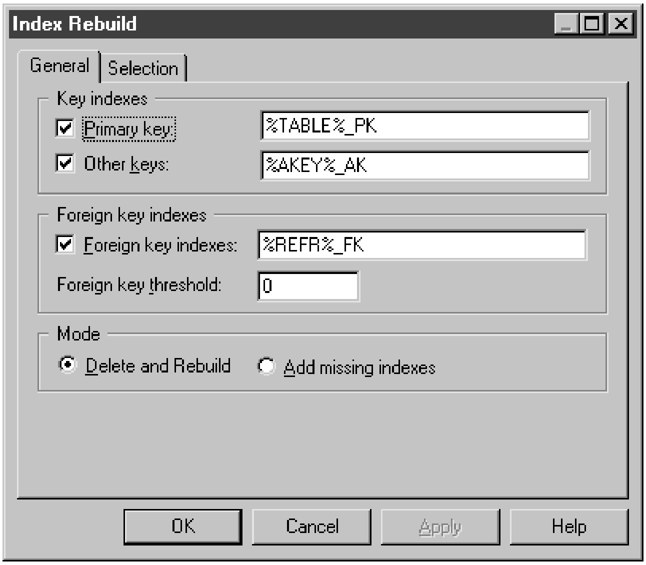
- Columns - lists the columns with which the index is associated (see "Creating an index" on page 103).

Rebuilding indexes

Rebuilding indexes in a physical diagram automatically updates any changes that you have made to primary keys, foreign keys, or alternate keys in your model.

❖ To rebuild indexes

1. Select Tools ➤ Rebuild Objects ➤ Rebuild Indexes to open the Rebuild Indexes dialog box.



Key indexes

☑ Primary key: %TABLE%_PK

☑ Other keys: %AKEY%_AK

Foreign key indexes

☑ Foreign key indexes: %REFR%_FK

Foreign key threshold: 0

Mode

● Delete and Rebuild ○ Add missing indexes

2. Set the appropriate options.

3. [optional] Click the Selection tab to specify which tables you want to rebuild indexes for.

4. Click OK. If you selected the Delete and Rebuild mode, a confirmation box asks you to confirm your choice. Click Yes to confirm the deletion and rebuild of the selected references.

Rebuilding index options

The following options are available when rebuilding indexes:

Option	Description
Primary key	Rebuilds primary key indexes. The text box shows the naming convention for primary keys. By default this is %TABLE%_PK
Other keys	Rebuilds alternate key indexes. The text box shows the naming convention for alternate keys. By default this is %AKEY%_AK
Foreign key indexes	Rebuilds foreign key indexes. The text box shows the naming convention for foreign keys. By default this is %REFR%_FK

Option	Description
Foreign key threshold	Specifies the minimum number of estimated records in a table that are necessary before a foreign key index can be created. The estimated number of records is defined in the Number box in the table property sheet. If the table has no specified number of occurrences, the foreign key indexes are generated by default
Mode	Specifies the extent of the rebuild. You can select: ♦ Delete and Rebuild – deletes and rebuilds all indexes presently attached to primary, alternate, and foreign keys ♦ Add missing indexes – preserves all indexes presently attached to primary, alternate, and foreign keys and adds any that are missing

PK index name variables

You can use the following variables in the PK index names fields:

Variable	Value
%TABLE%	Generated code of the table. This is the table code generated in the database. It may be truncated if the code contains characters not supported by the DBMS
%TNAME%	Table name
%TCODE%	Table code
%TLABL%	Table comment

FK index name variables

You can use the following variables in the FK index name field. The generated code of a variable is the code defined in the object property sheet, it may be truncated when generated if the code contains characters not supported by the DBMS

Variable	Value
%REFR%	Generated code of the reference
%PARENT%	Generated code of the parent table
%PNAME%	Parent table name
%PCODE%	Parent table code
%CHILD%	Generated code of the child
%CNAME%	Child table name

Variable	Value
%CCODE%	Child table code
%PQUALIFIER%	Parent table qualifier
%CQUALIFIER%	Child table qualifier
%REFRNAME%	Reference name
%REFRCODE%	Reference code

Indexes in query tables

You can create an index associated with the columns of a query table, which is a special type of view available in Oracle and DB2. These indexes are called view indexes. Query table indexes behave like indexes defined on tables, they are data structures that improve database performance and access speed. You normally create indexes for columns that you access regularly, and where response time is important.

For more information about query tables, see "Creating a query table" on page 163.

Defaults (PDM)

A default is a value that can be assigned to a column or a domain in the DBMS of the Sybase Adaptive Server Enterprise and Microsoft SQL Server families.

You select a default from the Default list in the Check Parameters tab of a column or domain property sheet.

Example

The default object **citydflt** is used to assign the same default value to all columns of type city.

Creating a default

You can create a default in any of the following ways:

- Select Model ➤ Defaults to access the List of Defaults, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤ Default.

☞ For general information about creating objects, see the Objects chapter in the *Core Features Guide* .

Default properties

You can modify an object's properties from its property sheet. To open a default property sheet, double-click its diagram symbol or its Browser entry in the Defaults folder. The following sections detail the property sheet tabs that contain the properties most commonly entered for defaults.

The General tab contains the following properties:

Property	Description
Name	Specifies the name of the item, which should be clear and meaningful, and should convey the item's purpose to non-technical users
Code	Specifies the technical name of the object, which is used for generating code or scripts, which may be abbreviated, and should not generally include spaces
Comment	Specifies the descriptive label
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined
Owner	Specifies the name of default owner. You choose an owner from a list of users
Value	Specifies the value of default object that will be generated

Default property sheet Preview tab

You can view the default creation order in the Preview tab of the default property sheet.

```
create default CITYDFLT
    as 'Dublin'
```

Assigning a default to a column or a domain

You can select a default from the list of defaults and assign it to a column or a domain from the Standard Checks tab of the column or domain property sheet.

☞ For more information on check parameters, see "Check Parameters (CDM/LDM/PDM)" on page 174.

If you only type a value in the Default list of a domain property sheet, it will not be generated as a default object in the database. It is highly recommended to use the Rebuild Default feature to create the default object corresponding to this value.

☞ For more information, see section "Rebuilding defaults" on page 113.

❖ To assign a default to a column or a domain

1. Open the property sheet of a column or a domain, and click the Standard Checks tab.

2. Select a default in the Default list in the Value groupbox.

   Alternatively, you can type a default value in the listbox; this does not create a default object in the model, it only assigns a default value for the current column or domain. If you type a name that already exists in the list, the default object is attached to the column or domain.

3. Click OK in each of the dialog boxes.

Rebuilding defaults

You can generate defaults from domains and columns having default values. The Default Rebuild feature uses the default values to create default objects and attaches them to the appropriate domains and/or columns.

Upgrading models
When you open a model containing domains with default values and saved in a previous version of PowerDesigner, default objects corresponding to the default values are created in the model.

Default objects are also created when you change the DBMS of a model containing domains with default values, to a DBMS that supports default objects. The opposite process occurs when you switch to a DBMS that does not support default objects: default objects are converted into default values.

Default name template

You can define a template for the generated default names. This template has the D_%.U:VALUE% value and supports the following variables:

- DOMAIN for the code of the domain using the default
- COLUMN for the code of the column using the default
- TABLE for the code of the table that contains the column with a default

You can define one template for domain defaults and one for column defaults.

❖ To rebuild defaults

1. Select Tools ➤ Rebuild Objects ➤ Rebuild Defaults to open the Default Rebuild dialog box.

2. Specify a default name template in the Domain and Column boxes.

3. [optional] Select the Reuse default with identical value check box – this option will reuse default objects with identical value among columns and domains. If you do not select this option, rebuild creates one default per object.

4. [optional] Select the Delete and rebuild check box – this option detaches the default objects attached to selected objects and deletes them if they are not used. If you select all objects, this option allows you to clean up the model from all existing defaults and recreate new default objects.
5. [optional] Click the Selection tab to specify domains and tables for default generation.
6. Click OK.

   The defaults are automatically created and attached to the domains and/or columns.

Domains (CDM/LDM/PDM)

Domains help you identify the types of information in your model. They define the set of values for which a column/entity attribute is valid. Applying domains to columns/entity attributes makes it easier to standardize data characteristics for columns/entity attributes in different tables/entities.

In a diagram, you can associate the following information with a domain:

- Data type, length, and precision
- Check parameters
- Business rules
- Mandatory property

Creating a domain

You can create a domain as follows:

- Select Model ➤ Domains to access the List of Domains, and click the Add a Row tool
- Right-click the model or package in the Browser, and select New ➤ Domain

☞ For general information about creating objects, see the Objects chapter in the *Core Features Guide* .

Domain properties

You can modify an object’s properties from its property sheet. To open a domain property sheet, double-click its Browser entry in the Domains folder.

The General tab contains the following properties:

Property	Description
Name	Specifies the name of the item, which should be clear and meaningful, and should convey the item’s purpose to non-technical users
Code	Specifies the technical name of the object, which is used for generating code or scripts, which may be abbreviated, and should not generally include spaces
Comment	Specifies a descriptive label for the domain

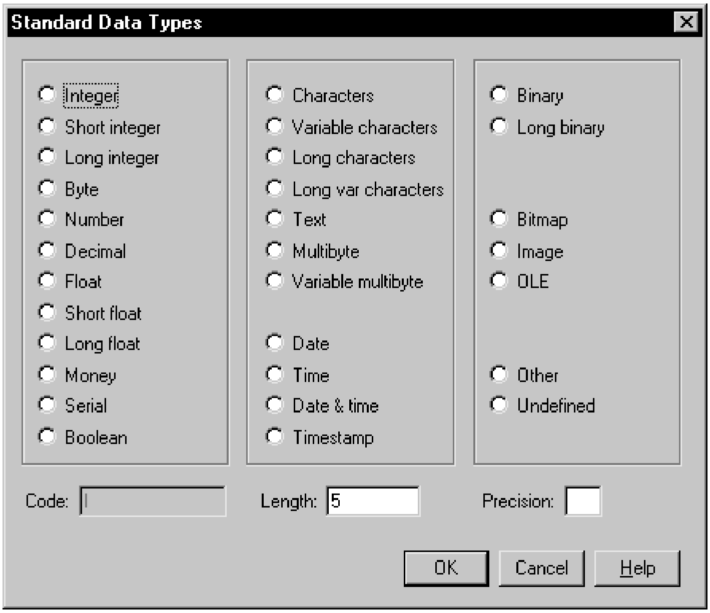
Property	Description
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined
Owner	[PDM only] Specifies the name of domain owner. You choose an owner from a list of users. A domain can only have one owner at a time. This is normally the domain creator
Data type	Specifies the form of the data corresponding to the domain, such as numeric, alphanumeric, Boolean, or others. The <undefined> data type indicates a domain without a data type. If an <undefined> data type is present when you generate your database, it is replaced by the default data type for your database
Length	[where appropriate] Specifies the maximum number of characters. In the PhysDataType list of available data types (select Database ➤ Edit current database ➤ Script ➤ DataType ➤ PhysDataType), a variable indicates where you have to type a length or precision, as follows: ♦ %n - length ♦ %s - length with precision ♦ %p - decimal precision For example, if you are using Sybase Adaptive Server Anywhere and you choose the data type `char(%n)`, you can choose a length of ten by typing `char(10)`.
Precision	[where appropriate] Specifies the number of places after the decimal point, for data values that can take a decimal point
Mandatory	Specifies that domain values are mandatory for all columns/entity attributes using that domain
Identity	(For Adaptive Server Enterprise, MS SQL Server and those DBMS that support it). When selected, indicates that the data is auto-incremented for columns using that domain
With default	[PDM only] (For those DBMS that support it). When selected, indicates if a default value is assigned to a column using the domain, when a Null value is inserted
Profile	[PDM only] Specifies the test Data profile assigned to the domain

The following tabs are also available:

- Standard Checks - contains checks which control the values permitted for the column/entity attribute (see “Check Parameters (CDM/LDM/PDM)” on page 174)
- Additional Checks - allows you to specify additional constraints (not defined by standard check parameters) for the column/entity attribute.
- Rules - lists the business rules associated with the column/entity attribute (see “Business Rules (CDM/LDM/PDM)” on page 179).

List of standard data types

You can open the list of Standard Data Types by clicking the question mark button to the left of the list of Data Types on the General Tab of a domain property sheet.



Code:

Length: 5

Precision:

Numeric data types

The following numeric data types are available:

Standard data type	DBMS-specific physical data type	Content	Length
Integer	int / INTEGER	32-bit integer	—
Short Integer	smallint / SMALL-INT	16-bit integer	—

Standard data type	**DBMS-specific physical data type**	**Content**	**Length**
Long Integer	int / INTEGER	32-bit integer	—
Byte	tinyint / SMALLINT	256 values	—
Number	numeric / NUMBER	Numbers with a fixed decimal point	Fixed
Decimal	decimal / NUMBER	Numbers with a fixed decimal point	Fixed
Float	float / FLOAT	32-bit floating point numbers	Fixed
Short Float	real / FLOAT	Less than 32-bit point decimal number	—
Long Float	double precision / BINARY DOUBLE	64-bit floating point numbers	—
Money	money / NUMBER	Numbers with a fixed decimal point	Fixed
Serial	numeric / NUMBER	Automatically incremented numbers	Fixed
Boolean	bit / SMALLINT	Two opposing values (true/false; yes/no; 1/0)	—

Character data types

The following character data types are available:

Standard data type	DBMS-specific physical data type	Content	Length
Characters	char / CHAR	Character strings	Fixed
Variable Characters	varchar / VARCHAR2	Character strings	Maximum
Long Characters	varchar / CLOB	Character strings	Maximum
Long Var Characters	text / CLOB	Character strings	Maximum
Text	text / CLOB	Character strings	Maximum
Multibyte	nchar / NCHAR	Multibyte character strings	Fixed
Variable Multibyte	nvarchar / NVARCHAR2	Multibyte character strings	Maximum

Time data types

The following time data types are available:

Standard data type	DBMS-specific physical data type	Content	Length
Date	date / DATE	Day, month, year	—
Time	time / DATE	Hour, minute, and second	—
Date & Time	datetime / DATE	Date and time	—
Timestamp	timestamp / TIMESTAMP	System date and time	—

Other data types

The following other data types are available:

Standard data type	DBMS-specific physical data type	Content	Length
Binary	binary / RAW	Binary strings	Maximum
Long Binary	image / BLOB	Binary strings	Maximum
Bitmap	image / BLOB	Images in bitmap format (BMP)	Maximum
Image	image / BLOB	Images	Maximum
OLE	image / BLOB	OLE links	Maximum
Other	—	User-defined data type	—
Undefined	undefined	Undefined. Replaced by the default data type at generation.	—

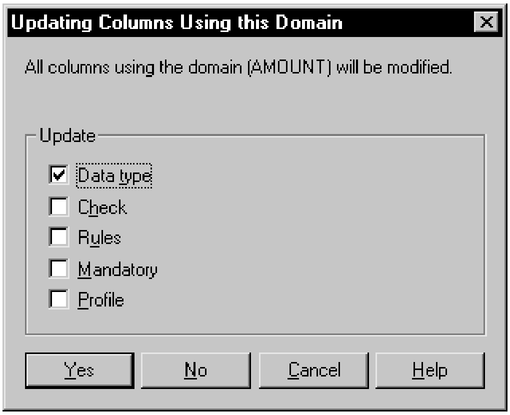
Cascading updates to columns/entity attributes associated with the domain

When you modify data types associated with a domain, an update confirmation box is displayed asking if you want to modify the columns/entity attributes currently using the domain.

❖ To modify domain properties in a data model

1. Open the property sheet of a domain and edit its properties as required.
2. Click OK.

   If the domain is used by one or more columns/entity attributes, an update confirmation box is displayed asking if you want to modify domain properties for the columns/entity attributes using the domain.



All columns using the domain (AMOUNT) will be modified.

Update

☑ Data type
□ Check
□ Rules
□ Mandatory
□ Profile

The Data Type check box is selected or not according to the options set to enforce non-divergence from a domain (see "Enforcing non-divergence from a domain in a data model" on page 121).

3. Select any other properties that you want to update (Check, Rules, Mandatory, Profile) for all columns/entity attributes using the domain.

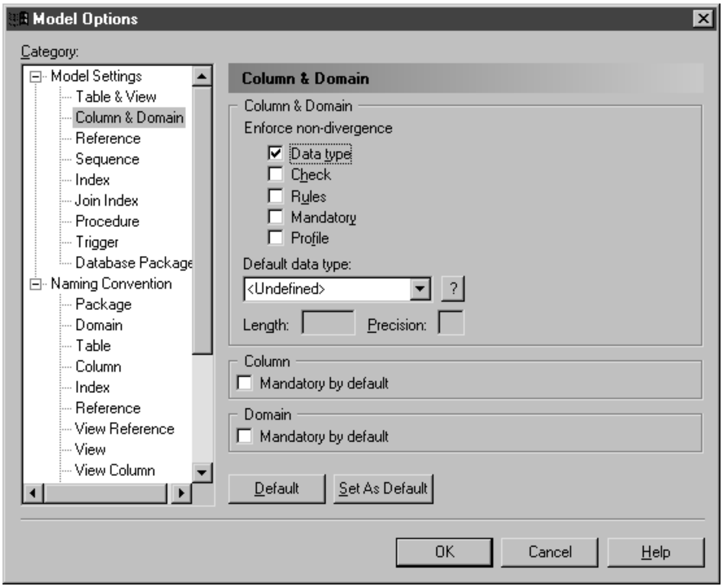
4. Click one of the following buttons:
   - Yes - The columns/entity attributes currently using the domain are modified according to the update
   - No - The columns/entity attributes currently using the domain are not modified according to the update but the current modification is accepted if domain divergence is allowed in the model options (see "Enforcing non-divergence from a domain in a data model" on page 121).
   - Cancel - The update is cancelled and nothing is changed

Enforcing non-divergence from a domain in a data model

You can enforce non-divergence between a domain and the columns/entity attributes that use the domain.

❖ To enforce domain non-divergence in a data model

1. Select Tools ➤ Model Options to open the Model Options dialog box. In a PDM, you have to click the Column and Domain sub-category in the left-hand Category pane to display the Enforce non-divergence option:



Column & Domain

Enforce non-divergence

☑ Data type
☐ Check
☐ Rules
☐ Mandatory
☐ Profile

Default data type: <Undefined>

Length: Precision:

Column

☐ Mandatory by default

Domain

☐ Mandatory by default

2. Select the check boxes of the column/entity attribute properties that are not permitted to diverge from the domain definition. You can specify any or all of:

   - Data type - data type, length, and precision
   - Check - check parameters such as minimum and maximum values
   - Rules – business rules
   - Mandatory – mandatory property of the column
   - [PDM only] Profile - test data profile

   If you subsequently modify in your domain any of the properties specified as non-divergent here, then the corresponding properties of the columns/entity attributes attached to that domain are automatically updated.

   Column/entity attribute properties specified as non-divergent appear dimmed and are non-editable in the List of Columns/Entity attributes and Column/Entity attribute property sheets. If you want to modify a non-divergent column/entity attribute property, you must detach the column/entity attribute from its domain.

3. Click OK to close the Model Options dialog box.

4. When you set the Enforce non-divergence options, you are asked if you want to apply domain properties to columns/entity attributes currently attached to the domain. If you click OK, the column/entity attribute properties are modified in order to be consistent with the properties of the domain to which they belong.

Sequences (PDM)

If your DBMS supports sequences, then you can create a sequence for a column.

A sequence is like an advanced form of an auto-incremented column. Where the latter is a column whose values automatically increment by 1, sequences allow you to define a more complex list of numbers. For example, you could define a list of numbers ranging between two values with an increment by any number (integer) you want.

Once you define a sequence, you can apply and enable it to a column. The data type for the column receiving the sequence must be a numeric data type. Such auto-incremented columns can be used in a key for a PDM table.

☞ For more information on data types, see “Selecting a data type for a column” on page 90.

Example

Assume that you want to create a column listing the months of the year when quarterly reports are published: March, June, September, and December. The first report is published on the third month, the second on the sixth, the third on the ninth and the last on the twelfth.

You can define the proper sequence by typing the following values for sequence option parameters:

Parameter name	Description	Value
`Start with`	March is the third month of the year	3
`Increment by`	Look three months ahead to identify the next month in the list	3
`Maxvalue`	Stop when you have reached the last month of the year	12

The sequence created with these parameter settings allows you to automatically create the list of months in a year when quarterly reports are published.

Creating a Sequence

There are two steps to using sequences:

- Create a sequence (including defining sequence options)
- Apply and enable a sequence to a column

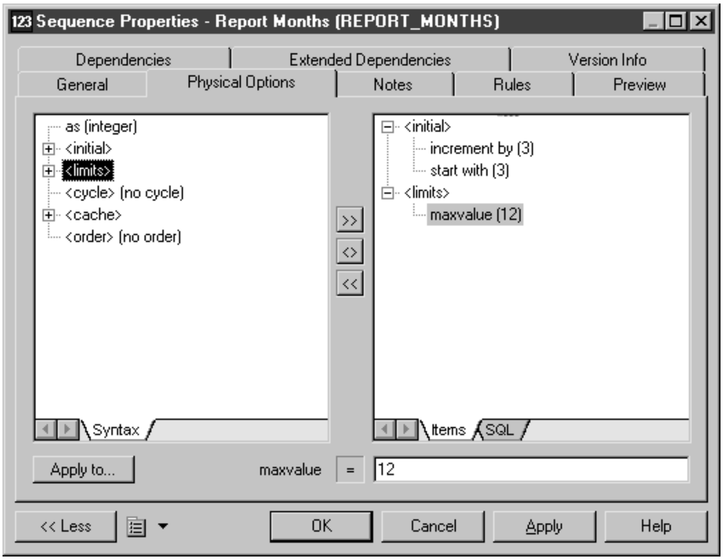
❖ To create a sequence

1. Select Model ➤ Sequences to open the List of Sequences.

2. Click the Add a Row tool and type a name for the new sequence.

3. Double-click the arrow to the left of the new sequence to display its property sheet.

4. Click the Physical Options tab and enter any appropriate parameters. These options are DBMS-specific. For more information on using this tab, see "Physical Options" on page 188.



Sequence Properties - Report Months (REPORT_MONTHS)

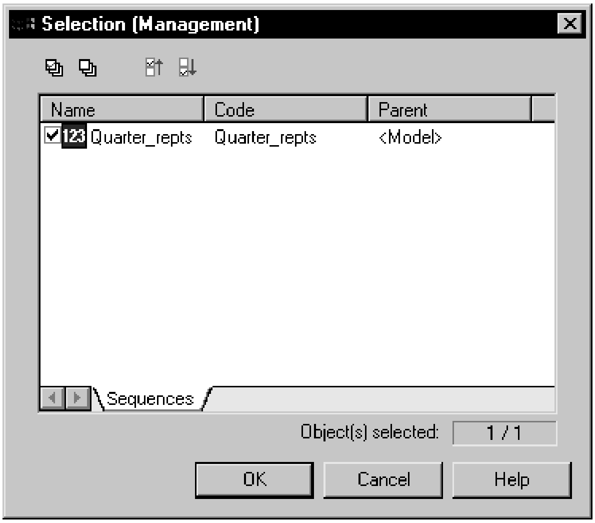
as (integer)
<initial>
<limits>
<cycle> (no cycle)
<cache>
<order> (no order)

<initial>
increment by (3)
start with (3)
<limits>
maxvalue (12)

maxvalue = 12

The above example shows the options and values to create a sequence of months in a year when quarterly reports are published.

5. [optional] Click the Apply To button to open a selection list and specify other sequences to which these same options will apply.



Name	Code	Parent
☑ Quarter_repts	Quarter_repts	<Model>

Sequences

Object(s) selected: 1 / 1

6. Click OK in each of the dialog boxes.

❖ To apply and enable a sequence on a column

1. Open the property sheet of the column to which you want to apply the sequence.
2. On the General tab, select a sequence from the Sequence list.
3. Click OK to close the property sheet.
4. Select Tools ➤ Rebuild Objects ➤ Rebuild Triggers to open the Rebuild Triggers dialog box.
5. Click the Selection tab and select the table or tables containing the column to which you want to attach a sequence.
6. Click OK.

   The triggers are rebuilt and the sequence is enabled on the column.

☞ For more information on rebuilding triggers, see the Building Triggers and Procedures chapter.

Sequence properties

You can modify an object's properties from its property sheet. To open a sequence property sheet, double-click its Browser entry in the Sequences folder or its line in the List of Sequences. The following sections detail the property sheet tabs that contain the properties most commonly entered for sequences.

The General tab contains the following properties:

Property	Description
Name	Specifies the name of the item, which should be clear and meaningful, and should convey the item's purpose to non-technical users
Code	Specifies the technical name of the object, which is used for generating code or scripts, which may be abbreviated, and should not generally include spaces
Comment	Specifies a descriptive label for the sequence
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined
Owner	Specifies the name of sequence owner. You choose an owner from a list of users. A column can only have one owner at a time. This is normally the column creator

The following tabs are also available:

- Physical Options - lists the physical options associated with the sequence (see "Physical Options" on page 188). For information about these options, see your DBMS documentation.

Changing the DBMS of a model which contains sequences and auto-incremented columns

If you create a sequence attached to a column in a DBMS supporting sequences, such as Oracle 8 and higher, Interbase or PostgreSQL, or create an auto-incremented column in a DBMS supporting this feature, and then decide to change the target DBMS, the following effects occur:

DBMS change	Defined in original DBMS	Effect on sequence objects and auto-incremented columns
DBMS supporting sequences to a DBMS supporting auto-incremented columns	Sequence attached to a column	The sequence disappears and the column to which it was attached becomes an auto-incremented column in the DBMS
DBMS supporting auto-incremented columns to a DBMS supporting sequences	Auto-incremented column	The auto-incremented column is deleted and replaced by a sequence object called S_TABLENAME which is attached to the original column

Sequences and intermodel generation

When a CDM or an OOM is generated from a PDM, the data type of a table column attached to a sequence is translated to a numeric data type in the new model:

PDM generated to	Sequence is converted to
CDM	A serial data type for an entity property. The data type has the format `NO%n` where `%n` is a number indicating the length of the data type
OOM	A serial data type for a class attribute. The data type has the format `NO%n`, where `%n` is a number indicating the length of the data type

Abstract Data Types (PDM)

An **abstract data type (ADT)** is a user-defined data type which can encapsulate a range of data values and functions. The functions can be both defined on, and operate on the set of values.

Abstract data types can be used in the following ways in a Physical diagram:

Abstract data type is	**Description**
Created	You can create an abstract data type of any kind supported by your DBMS. If you create an abstract data type of type JAVA, you can link it to a Java class in an OOM to access the Java class properties (see "Linking an abstract data type to a Java class" on page 133).
Reverse engineered	An abstract data type in a database can be reverse engineered into a PDM. If you also reverse engineer the JAVA classes into an OOM, then the abstract data types of the type JAVA in the PDM are automatically linked to the Java classes in the OOM (see "Reverse-engineering a PDM linked to an OOM" on page 137)

☞ For more information on reverse engineering a database into a PDM, see chapter Reverse Engineering.

☞ For more information on creating and reverse engineering Java classes into a PowerDesigner Object-Oriented Model, see the *Object-Oriented* Modeling guide.

Depending on the current DBMS, the following kinds of abstract data types can be created in PowerDesigner:

Type	Description	Example
Array	Fixed length collection of elements	VARRAY (Oracle 8 and higher)
List	Unfixed length collection of objects	TABLE (Oracle 8 and higher)
Java	Java class	JAVA (Adaptive Server Anywhere, and Adaptive Server Enterprise)
Object	Contains a list of attributes and a list of procedures	OBJECT (Oracle 8 and higher)
SQLJ Object	Contains a list of attributes and a list of procedures	SQLJ OBJECT (Oracle 9i and higher)
Structured	Contains a list of attributes	NAMED ROW TYPE (Informix 9.x, and IBM DB2 5.2)

Example

An abstract data type for the Gregorian calendar which has functions defined to do the following:

- Read and write roman numerals
- Convert dates from the Julian calendar to the Gregorian calendar
- Convert dates from the Gregorian calendar to the Julian calendar

Creating an abstract data type

You can create an abstract data type in any of the following ways:

- Select Model ➤ Abstract Data Types to access the List of Abstract Data Types, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤ Abstract Data Type.

☞ See also “Creating object and SQLJ object abstract data types” on page 132.

☞ For general information about creating objects, see the Objects chapter in the *Core Features Guide* .

Abstract data type properties

You can modify an object’s properties from its property sheet. To open an abstract data type property sheet, double-click its Browser entry in the Abstract Data Types folder.

The General tab contains the following properties:

Property	Description
Name	Specifies the name of the item, which should be clear and meaningful, and should convey the item's purpose to non-technical users
Code	Specifies the technical name of the object, which is used for generating code or scripts, which may be abbreviated, and should not generally include spaces
Comment	Specifies the descriptive label
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined
Owner	Specifies the name of abstract data type owner. You choose an owner from a list of users
Type	Specifies the defining group that includes the abstract data type

Depending on its type, an abstract data type definition can also include the following properties:

- Data type, Length, and Precision
- Size (for arrays)
- Linked class name (for Java types)
- File name and path, which contains the declaration of the class
- Authorization (for objects): Invoker Right attribute used for DDL generation
- Supertype (for objects): Parent abstract data type from which the current abstract data type can inherit the procedures
- Final and Abstract (for objects): When Final is checked, the current abstract data type cannot be used as supertype by another abstract data type. When Abstract is checked, the current abstract data type cannot be instantiated. Final and Abstract are mutually exclusive
- Java class (for SQLJ objects): Name of an external Java class to which the SQLJ object points. Beside the Java class box, there is a list to select a mapping interface (CustomDatum, OraData or SQLData)

Creating object and SQLJ object abstract data types

If you select the OBJECT (or SQLJ OBJECT) type for an abstract data type, two additional tabs are displayed in the property sheet:

- The Attributes tab allows you to specify an object (or SQLJ object) with a number of attributes to which are assigned appropriate data types
- The Procedures tab allows you to specify an object (or SQLJ object) with a number of procedures to which are assigned appropriate parameters

Procedure inheritance

An object abstract data type with a supertype can inherit non-final procedures. You can use the Inherit Procedure tool in the Procedures tab of the abstract data type to select a non-final procedure from a parent abstract data type. Inheritance only applies to non-finale procedures.

Object example

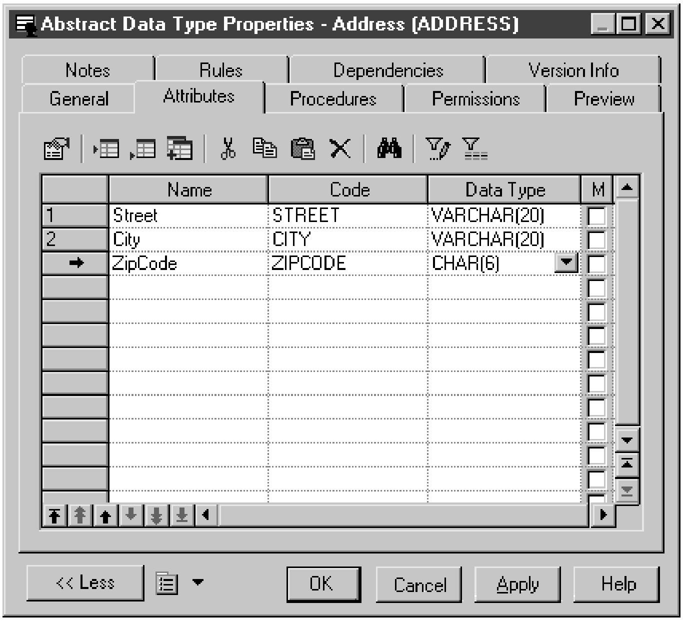
For example, you want to create an Address object with Street, City, and ZipCode attributes, and a Location procedure.

❖ To specify attributes and procedures for an object (or SQLJ object) abstract data type

1. Open the property sheet of the abstract data type and select either OBJECT or SQLJ_OBJECT from the type list.

   The Attributes and Procedures tabs are displayed.

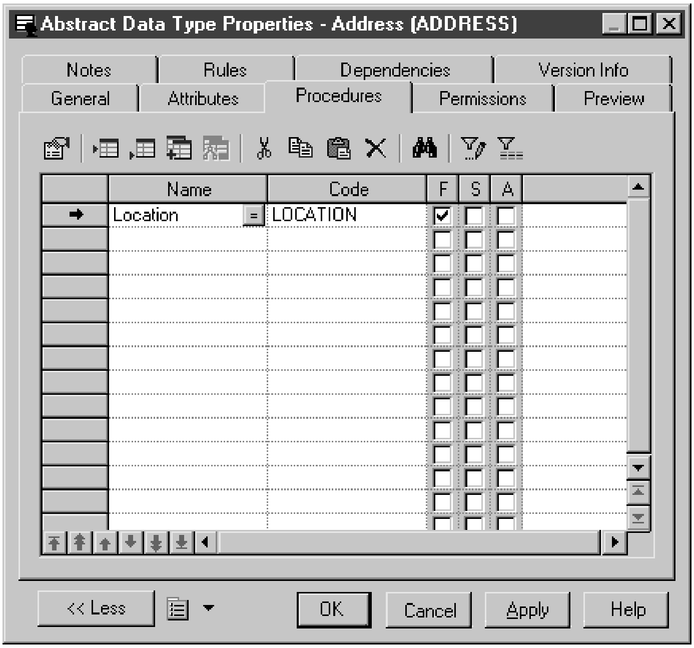
2. Click the Attributes tab.



	Name	Code	Data Type	M
1	Street	STREET	VARCHAR(20)	□
2	City	CITY	VARCHAR(20)	□
➜	ZipCode	ZIPCODE	CHAR(6)	□

3. For each attribute, click the Add a Row tool, and:
    - enter a Name and Code
    - select a data type class from the Data Type list
    - [optional] Select the Mandatory (M) checkbox

4. Click the Procedures tab:



	Name	Code	F	S	A
➜	Location	LOCATION	☑	□	□

5. For each procedure, click the Add a Row tool, and:
    - enter a Name and Code
    - [optional] Select the Final (F), Static (S) and/or Abstract (A) columns

6. Click OK in each of the dialog boxes.

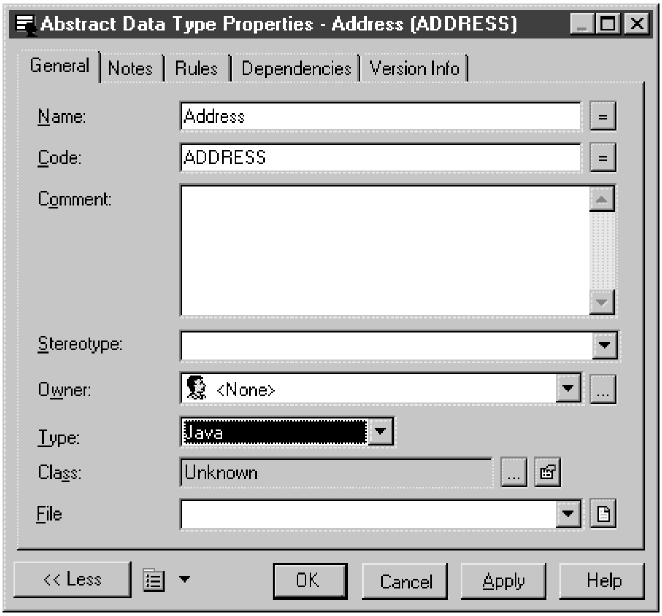
Linking an abstract data type to a Java class

You can specify a Java class in the PDM, and then link it to a Java class in an OOM. The OOM must be open in the current Workspace to be available for linking.

When you link an abstract data type to a Java class, a shortcut is created which allows you to access the properties of the Java class from within the PDM.

❖ To link an abstract data type to a Java class

1. Create an abstract data type and select Java from the Type list on the General tab of its property sheet.



Abstract Data Type Properties - Address (ADDRESS)

General | Notes | Rules | Dependencies | Version Info

Name: Address

Code: ADDRESS

Comment:

Stereotype:

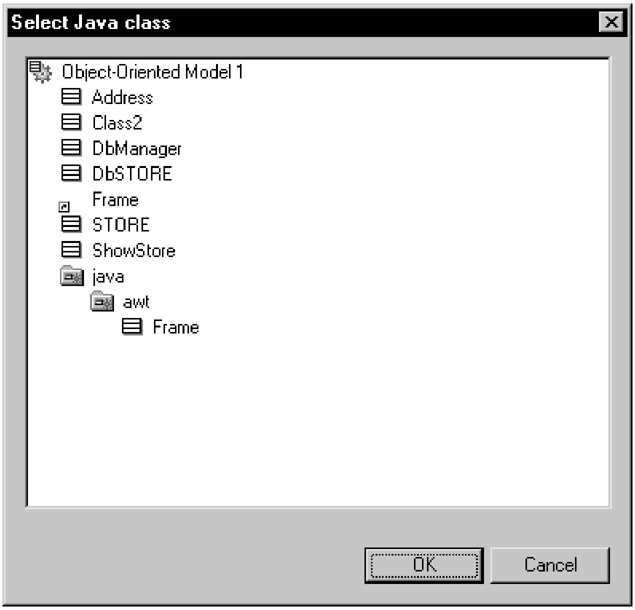
Owner: <None>

Type: Java

Class: Unknown

File

2. Click the Ellipsis button to the right of the Class box to open a Java class selection box, which lists all the Java classes that are available in the OOMs currently open in the Workspace.

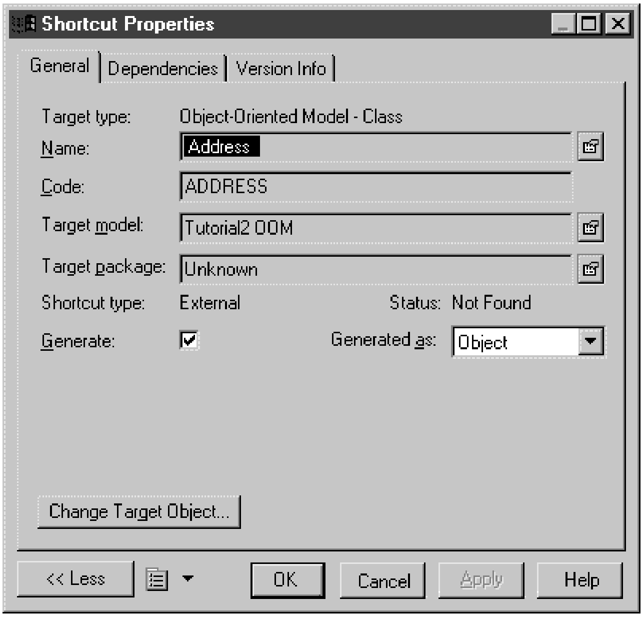


3. Select a Java class and click OK.

   The abstract data type is now linked to the Java class, and the class name is displayed in the Class box.

4. Click the Properties button at the end of the Class box to open the property sheet of the Java class.

   If the related OOM is closed, then a shortcut property sheet for the Java class is displayed, and you must click the Properties button to the right of the Name box to display its actual property sheet..



Target type: Object-Oriented Model - Class

Name: Address

Code: ADDRESS

Target model: Tutorial2 OOM

Target package: Unknown

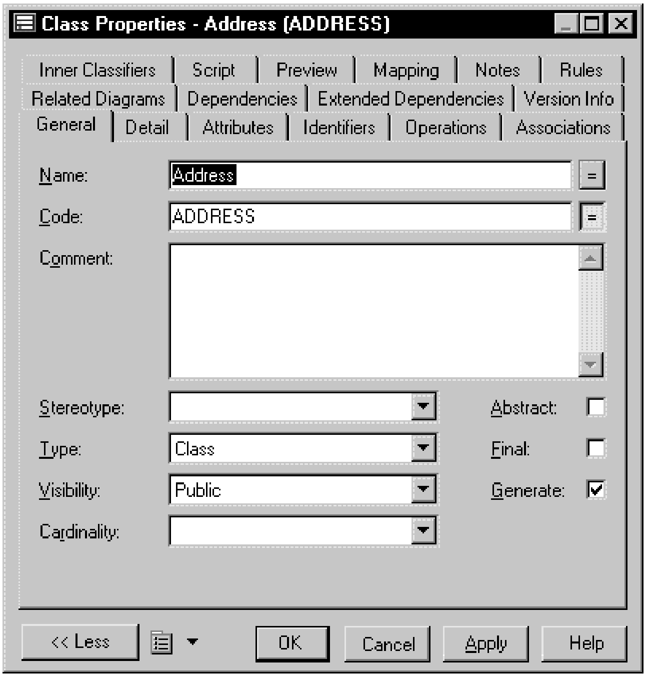
Shortcut type: External

Status: Not Found

Generate: ☑

Generated as: Object

If the related OOM is open, then the class property sheet is opened directly:



Name: Address

Code: ADDRESS

Comment:

Stereotype:

Type: Class

Visibility: Public

Cardinality:

Abstract: □

Final: □

Generate: ☑

5. Click OK in each of the dialog boxes.

Reverse-engineering a PDM linked to an OOM

You can reverse engineer a PDM from a database that contains Java classes and also reverse the Java classes into an OOM. The Java abstract data types in the PDM are automatically linked to the Java classes in the OOM as follows:

- You should reverse engineer the Java classes in the database that are used as data types for the columns and domains in an OOM
- Then reverse engineer the database into a PDM
- PowerDesigner automatically searches the open OOM for the Java classes that correspond to the JAVA abstract data types in the PDM and makes the corresponding links

The Java classes that are reverse engineered into the PDM are created automatically as abstract data types of type JAVA.

You can access the properties of these Java classes from the property sheets of the corresponding abstract data types in the PDM. For more information, see "Linking an abstract data type to a Java class" on page 133.

References (PDM)

A **reference** is a link between a parent table and a child table. It defines a referential integrity constraint between column pairs for a primary key, or alternate key, and a foreign key, or between user specified columns in both tables.

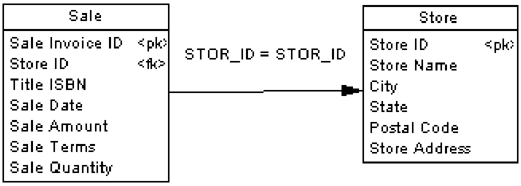
When column pairs are linked by a reference, each value in the child table column refers to an equivalent value in the parent table column.

Within a reference, each column pair is linked by a **join**. Depending on the number of columns in the primary key, or alternate key, or the number of specified columns, a reference can contain one or more joins.

A reference normally links primary key, or alternate key, columns to foreign key columns.

Example

The two tables SALE and STORE are linked by a reference. STORE is the parent table and SALE is the child table. The reference contains a join which links the primary key column STORE ID (the referenced column) to the foreign key column STORE ID (the referencing column).



Creating a reference

You can create a reference that links a primary key, or alternate key, to a foreign key, or user-specified columns in both parent and child tables.

Depending on its properties, a reference can link a parent table and a child table in one of two ways:

Reference links	Description
Primary key, alternate key and foreign keys	Primary or alternate key in the parent table is linked to a foreign key in the child table
User specified columns	One or more columns in the parent table are linked to corresponding columns in the child table. The linked columns in both tables are specified by the user, and are linked independently of primary key, alternate key, and foreign key columns

You can create a reference in any of the following ways:

- ♦ Use the Reference tool in the diagram Palette.
- ♦ Select Model ➤ References to access the List of References, and click the Add a Row tool.
- ♦ Right-click the model or package in the Browser, and select New ➤ Reference.

☞ For general information about creating objects, see the Objects chapter in the *Core Features Guide* .

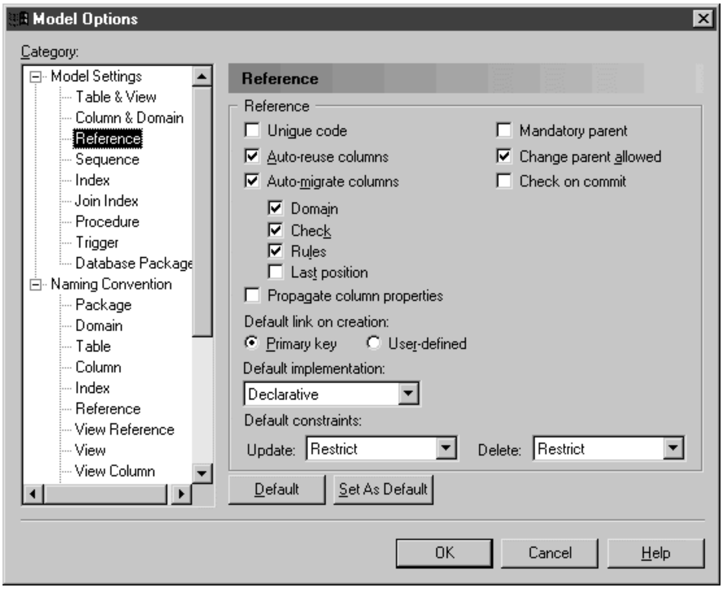
Automatic reuse and migration of columns

When you create a reference, PowerDesigner can automatically:

- ♦ Reuse an appropriate existing column in the child table as the foreign key column
- ♦ Migrate the primary key column in the parent table to create a foreign key column in the child table

❖ To auto-reuse and/or auto-migrate columns

1. Select Tools ➤ Model Options to open the Model Options dialog box.
2. Select the Reference sub-category in the left-hand Category pane to display the Reference tab.



Reference

☐ Unique code
☑ Auto-reuse columns
☑ Auto-migrate columns
☑ Domain
☑ Check
☑ Rules
☐ Last position
☐ Propagate column properties
☐ Mandatory parent
☑ Change parent allowed
☐ Check on commit

Default link on creation:
● Primary key ○ User-defined

Default implementation:
Declarative

Default constraints:
Update: Restrict Delete: Restrict

3. Specify your choices for column reuse and migration as follows:
   - **To auto-reuse existing columns in child tables as foreign key columns when creating references** - select the Auto-reuse columns check box. Note that the column in the child table must have the same code as the migrating primary key column, and cannot already be a foreign key column for it to be suitable for reuse. If you want to reuse a child table column that is already a foreign key column, you must do this manually from the Joins tab of the reference property sheet.
   - **To auto-migrate primary key columns in parent tables for use as foreign key columns in child tables** - select the Auto-migrate columns check box. This will also enable the column properties check boxes, allowing you to specify which of the parent column properties to migrate.
   - **To auto-migrate the properties of parent table primary key columns** - select the appropriate check boxes:
     - Domains
     - Check (check parameters)
     - Rules (business rules)
     - Last position (migrated columns should be added at the end of the table column list. If the Last position option is not selected, migrated columns are inserted between key columns and other columns which implies that a child table must be dropped and recreated each time you add a reference and modify an existing database.)

   Note that, during intermodel generation, whether or not the Auto-migrate columns check box is selected, any selected column property is migrated from the PK to the FK.

4. Ensure that the Default link on creation option is set to Primary key.

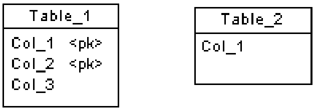
5. Click OK to close the Model Options dialog box.

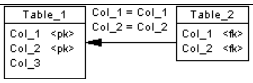
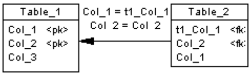
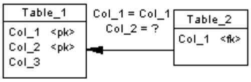
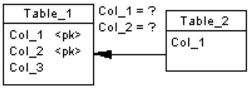
Examples

The following examples illustrate how using the auto-reuse columns and auto-migrate columns options affects the creation of references.

Matching child table column exists

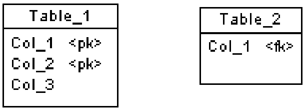
The following table shows the results of migrating primary key columns to a child table that contains a matching column for one of the primary key columns. The original two tables are also shown below:

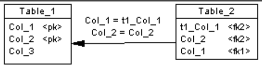
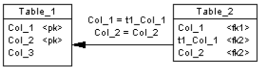
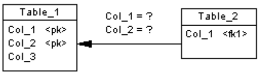
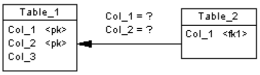


Auto-reuse	Auto-migrate	Result	Description of child table
Selected	Selected	 Table_1 Col_1 <pk> Col_2 <pk> Col_3 Col_1 = Col_1 Col_2 = Col_2 Table_2 Col_1 <fk> Col_2 <fk>	Col_1 is reused and Col_2 is created
Not selected	Selected	 Table_1 Col_1 <pk> Col_2 <pk> Col_3 Col_1 = t1_Col_1 Col 2 = Col 2 Table_2 t1_Col_1 <fk: Col_2 <fk: Col_1	T1_Col_1 is created and Col_2 is created
Selected	Not selected	 Table_1 Col_1 <pk> Col_2 <pk> Col_3 Col_1 = Col_1 Col_2 = ? Table_2 Col_1 <fk>	Col_1 is reused and Col_2 is not created
Not selected	Not selected	 Table_1 Col_1 <pk> Col_2 <pk> Col_3 Col_1 = ? Col_2 = ? Table_2 Col_1	No column is reused and no column is created

Matching child table column is already a FK column

The following table shows the results of migrating primary key columns to a child table that contains a matching child table column that is already a foreign key column for another table. The original two tables are also shown below:



Auto-reuse	Auto-migrate	Result	Description of child table
Selected	Selected	 Table_1 Col_1 <pk> Col_2 <pk> Col_3 Col_1 = t1_Col_1 Col_2 = Col_2 Table_2 t1_Col_1 <fk2> Col_2 <fk2> Col_1 <fk1>	T1_Col_1 is created and Col_2 is created
Not selected	Selected	 Table_1 Col_1 <pk> Col_2 <pk> Col_3 Col_1 = t1_Col_1 Col_2 = Col_2 Table_2 Col_1 <fk1> t1_Col_1 <fk2> Col_2 <fk2>	T1_Col_1 is created and Col_2 is created
Selected	Not selected	 Table_1 Col_1 <pk> Col_2 <pk> Col_3 Col_1 = ? Col_2 = ? Table_2 Col_1 <fk1>	No columns are reused or created
Not selected	Not selected	 Table_1 Col_1 <pk> Col_2 <pk> Col_3 Col_1 = ? Col_2 = ? Table_2 Col_1 <fk1>	No columns are reused or created

Notes:

- By default, only the properties of the primary key column are migrated to the foreign key. If the primary key column is attached to a domain, the domain will not be migrated to the new foreign key column unless the Enforce non-divergence option model option is selected (see "Enforcing non-divergence from a domain in a data model" on page 121).
- The following table shows the results of changing references when you have selected the auto-migrate columns option:

Action	Result
Modify reference attach point	Migrate primary key in parent table to foreign key in child table Delete unused foreign key columns Modify reference join
Delete primary key	Delete corresponding foreign key and reference join

Migrate primary key in parent table to foreign key in child table

Delete unused foreign key columns

Modify reference join

Action	Result
Modify reference attach point	Migrate primary key in parent table to foreign key in child table Delete unused foreign key columns Modify reference join
Delete primary key	Delete corresponding foreign key and reference join

☞ For more information on other model options for references, see "Setting PDM Model Options" section in the "Customizing the PDM Environment" chapter.

Reference properties

You can modify an object's properties from its property sheet. To open a reference property sheet, double-click its diagram symbol or its Browser entry in the References folder. The following sections detail the property sheet tabs that contain the properties most commonly entered for references.

The General tab contains the following properties:

Property	Description
Name	Specifies the name of the item, which should be clear and meaningful, and should convey the item's purpose to non-technical users
Code	Specifies the technical name of the object, which is used for generating code or scripts, which may be abbreviated, and should not generally include spaces
Comment	Specifies a descriptive label for the reference
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined
Parent table	Specifies the parent table of the reference. This table contains the primary key, or alternate key, linked by the reference. You can use the tools to the right of the list to create an object, browse the complete tree of available objects or view the properties of the currently selected object
Parent role	Specifies the role of the parent table in the reference. The text is displayed in the diagram, near the parent table
Child table	Specifies the child table of the reference. This table contains the foreign key linked by the reference
Child role	Specifies the role of the child table in the reference. The text is displayed in the diagram, near the child table
Generate	When selected, indicates to generate the reference in the database

Reference property sheet Joins tab

A **join** is a link between a column in a parent table and a column in a child table (column pair) that is defined within a reference.

A join can link primary, alternate or foreign key, or user-specified columns in the parent and child tables that are independent of key columns.

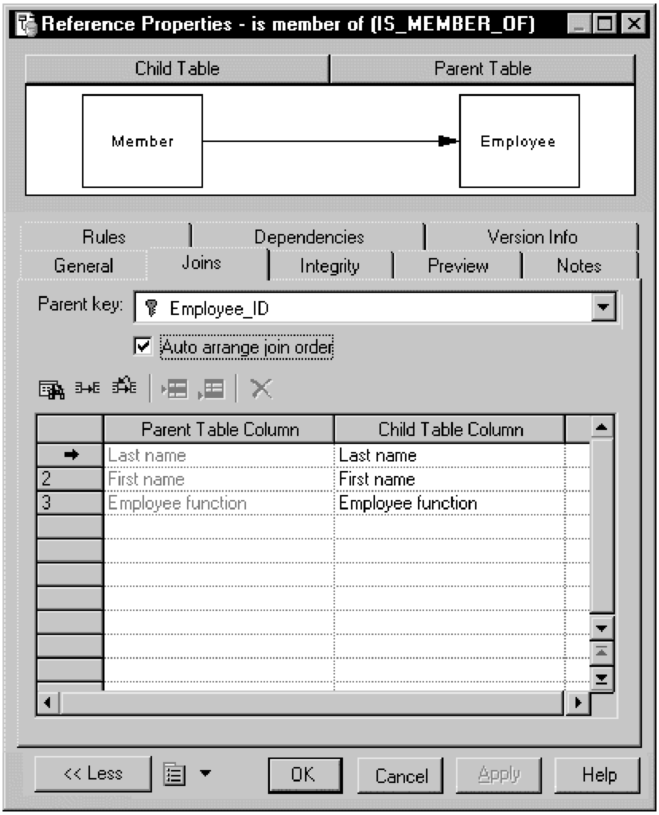
❖ To define joins in a reference

1. Double-click a reference in the diagram to open its property sheet, and then click the Joins tab.

2. Select a key in the Parent Key list to create joins on its columns. If you

select <NONE>, the column lists are empty and you must specify your own columns to join.

The columns linked by the joins are listed in the Parent Table and Child Table columns.



Parent key: Employee_ID

☑ Auto arrange join order

	Parent Table Column	Child Table Column
➡	Last name	Last name
2	First name	First name
3	Employee function	Employee function

Changing a foreign key column linked by a join
You can change the foreign key column linked by a join by clicking the column in the Child Table list, and selecting another column from the list.

3. [optional] If you selected <NONE> from the Parent Key list, click the Parent Table Column and select a column from the list, then click the Child Table Column and select a child column.

4. [optional] Select the Auto arrange join order check box to sort the list by the key column order. If this option is not selected, you can re-arrange the columns using the arrow buttons.

5. Click OK.

Enabling the Auto arrange join order check box
To enable this check box, add an `EnableChangeJoinOrder` item to the Reference category in the DBMS definition file and set the value to YES. See the DBMS Resource File Reference chapter of the *Customizing and Extending PowerDesigner* manual.

Default joins at reference creation

Join creation is determined by the following Model Options:

Default Link on Creation	Auto-migrate Columns	Result
Primary Key	Selected	Joins created between primary and foreign key columns.
	Not selected	Joins created and linked to primary key columns, but are incomplete. Foreign key columns must be specified manually.
User-defined	Selected	No joins created. Parent and child table column pairs must be specified manually.
	Not selected	No joins created. Parent and child table column pairs must be specified manually.

Linking columns in a primary or alternate key

For any reference you can choose to link a primary key, or alternate key, to a corresponding foreign key. When you select a key from the Joins tab of the reference property sheet, all the key columns are linked to matching foreign key columns in the child table.

Changing a foreign key column link
A foreign key column can be changed to link to another parent table column, either within the key relationship, or independent of it.

Reuse and Migration option for a selected reference

You can use the following buttons on the Joins tab to reuse or migrate columns linked by joins.

Tool	Description
	Reuse Columns - Reuse existing child columns with same code as parent table columns
	Migrate Columns - Migrate key columns to foreign key columns. If columns do not exist they are created
	Cancel Migration - Delete any migrated columns in child table

Reference property sheet Integrity tab

Referential integrity is a collection of rules that govern data consistency between primary keys, alternate keys and foreign keys. It dictates what happens when you update or delete a value in a referenced column in the parent table, and when you delete a row containing a referenced column from the parent table.

The Integrity tab contains the following properties:

Property	Description
Constraint name	Name of the referential integrity constraint. Maximum length is 254 characters
Implementation	Specifies how referential integrity will be implemented. You can choose between: ♦ Declarative- Referential integrity constraints are defined for particular references. When the reference is generated the target DBMS evaluates the reference validity and generates appropriate error messages ♦ Trigger - Referential integrity constraints are implemented by triggers based on the integrity constraints defined in the reference property sheet. The trigger evaluates reference validity and generates appropriate user-defined error messages

Property	Description
Cardinality	Indicates the minimum and maximum number of instances in a child table permitted for each corresponding instance in the parent table. The following values are available by default: ♦ 0..* - A parent can have zero or more children. There is no maximum. ♦ 0..1 - A parent can have zero or one children. ♦ 1..* - A parent can have one or more children. There is no maximum. ♦ 1..1 – A parent must have exactly one child Alternately, you can enter your own integer values in one of the following formats: ♦ x..y - A parent can have between x and y children. ♦ x - A parent can have exactly x children. ♦ x..y, a..b - A parent can have between x and y or between a and b children. You can use * or n to represent no limit. Examples: ♦ 2..n – There must be at least 2 children. ♦ 10 - There must be exactly 10 children. ♦ 1..2, 4..n – There must be one, two, four or more children.
User-defined	Indicates a user-defined constraint name

Property	Description
Update constraint	How updating a key value, in the parent table affects the foreign key value in the child table. Depending on the implementation and DBMS, you can choose between: ♦ None - Update or deletion of a value in the parent table has no effect on the child table. ♦ Restrict - A value in the parent table cannot be updated or deleted if one or more matching child values exists ♦ Cascade - Update or deletion of a value in the parent table causes an update or delete of matching values in the child table ♦ Set null - Update or deletion of a value in the parent table sets matching values in the child table to NULL ♦ Set default - Update or deletion of a value in the parent table sets matching values in the child table to the default value
Delete constraint	How deleting a row in the parent table affects the child table
Mandatory parent	Each foreign key value in the child table must have a corresponding key value, in the parent table
Change parent allowed	A foreign key value can change to select another value in the referenced key in the parent table
Check on commit	[Sybase SQL Anywhere 5.0 and 5.5 only] Verifies referential integrity only on the commit, instead of verifying it after row insertion. You can use this feature to control circular dependencies
Cluster	Indicates whether the reference constraint is a clustered constraint (for those DBMS that support clustered indexes)

Rebuilding references

You can rebuild references to create default references between PK columns in one table and columns with identical code and data type in another table. Note that rebuilding is not possible between two tables with PK columns.

Rebuilding references is useful following the reverse engineering of a database in which all of the references could not be reverse engineered.

❖ To rebuild references

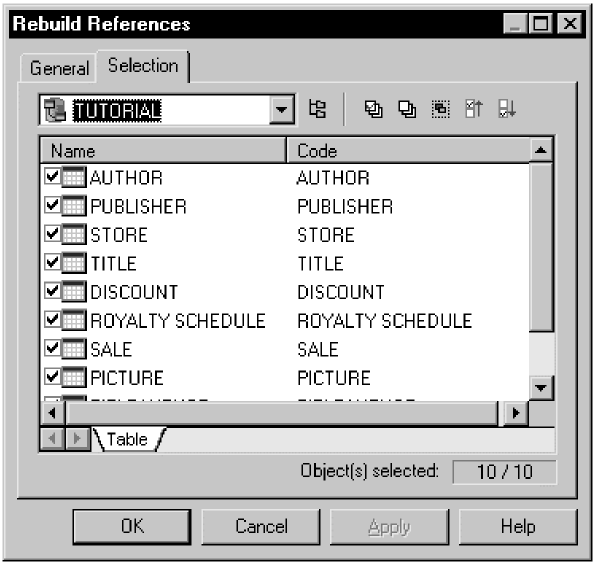
1. Select Tools ➤ Rebuild Objects ➤ Rebuild References to open the Rebuild References dialog box.

2. Select a mode:

   - Delete and Rebuild - All existing references are deleted, and new references built based on matching key columns

   - Preserve - All existing references are kept, and new references are built based on new matching key columns

3. [optional] Click the Selection tab and specify the tables for which you want to rebuild references. By default, all tables are selected.



TUTORIAL

Name	Code
☑ AUTHOR	AUTHOR
☑ PUBLISHER	PUBLISHER
☑ STORE	STORE
☑ TITLE	TITLE
☑ DISCOUNT	DISCOUNT
☑ ROYALTY SCHEDULE	ROYALTY SCHEDULE
☑ SALE	SALE
☑ PICTURE	PICTURE

Object(s) selected: 10 / 10

Rebuilding references in a package

To rebuild references between tables in a package, select the package from the list at the top of the tab.

To rebuild references between tables in a sub-package, select the Include Sub-Packages icon next to the list, and then select a sub-package from the dropdown list.

4. Click OK. If you selected the Delete and Rebuild mode, a confirmation box asks you to confirm your choice. Click Yes to confirm the deletion and rebuild of the selected references.

Changing a table at either end of a reference

After reference creation you can change one table, or both of the tables, linked by a reference, using one of the following methods:

- Click the reference and hold down CTRL as you drag one of its attach points to a different table.
- Double-click a reference in the diagram to display its property sheet and choose a different parent or child table from the lists.
- Select Model ➤ References to open the list of references, and choose a different parent or child table from the lists.

Modifying a reference graphically

You can modify a reference symbol as follows:

❖ To bend a reference symbol

1. Select Symbol ➤ Format to open the Symbol Format dialog box.
2. Click the Line Style tab and select the jagged line symbol from the Corners list.
3. Click OK to return to the diagram.
4. Press and hold CTRL while clicking the point on the reference where you want to insert an angle.

   A handle is added to the reference at the point clicked.
5. Release CTRL. You can now drag the handle to create the desired angle.

❖ To straighten a reference symbol

1. Click a reference symbol that has one or more angles to make its handles appear.
2. Press and hold CTRL while clicking a handle to remove the handle and angle.

❖ To drag a reference to a different table

1. Click a reference symbol.
2. Press and hold CTRL while dragging one of the symbol ends to a new table.

You can set the global display mode for references by clicking Tools ➤ Model Options and selecting a notation from the list. PowerDesigner supports Relational, CODASYL, Conceptual and IDEF1X notations.

Views (PDM)

A view is an alternative way of looking at the data in one or more tables. It is made up of a subset of columns from one or more tables.

You define a SQL query for each view.

Creating a view

You can create a view in any of the following ways:

- Use the View tool in the diagram Palette.
- Select Model ➤ Views to access the List of Views, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤ View.
- Select Tools ➤ Create View. For more details, see "Creating a view from the Tools menu" on page 156. You can, optionally, pre-select one or more tables and views in the diagram to create a view automatically populated with their columns.

☞ For general information about creating objects, see the Objects chapter in the *Core Features Guide* .

View properties

You can modify an object's properties from its property sheet. To open a view property sheet, double-click its diagram symbol or its Browser entry in the Views folder. The following sections detail the property sheet tabs that contain the properties most commonly entered for views.

The General tab contains the following properties:

Property	Description
Name	Specifies the name of the item, which should be clear and meaningful, and should convey the item's purpose to non-technical users
Code	Specifies the technical name of the object, which is used for generating code or scripts, which may be abbreviated, and should not generally include spaces
Comment	Specifies a descriptive label for the view

Property	Description
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined
Owner	Specifies the name of view owner. You choose an owner from a list of users. A view can only have one owner at a time. This is normally the view creator
Usage	Specifies the use of the view: Query only defines a view for consultation only, view cannot update tables; Updatable defines a view for consultation and update, view can update tables; and With Check options implements controls on view insertions
Dimensional type	Specifies the multidimensional type of the view, that is Dimension or Fact
Type	For those DBMS that support it, allows you to define the type of a view. You can select materialized query table, materialized view, summary table, or XML
Generate	Includes view generation as part of database generation script
User-defined	When selected, makes sure the view query is not parsed by PowerDesigner internal parser. This protects the view query from any update using model objects and keeps its syntax as defined by user. Otherwise, the view query is parsed and modified according to model values

☞ For more information on materialized views, materialized query tables, and summary tables, see “Creating a query table” on page 163.

☞ For more information on XML views, see “Creating an XML table or view” on page 65.

View property sheet Columns tab

The Columns tab in a view property sheet displays the list of columns in the view. This list of columns reflects the SELECT orders from the queries of the current view. The only way to add or remove columns from this list is to modify the query of the view.

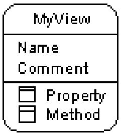
☞ For more information on how to define the query of a view, see “Creating, editing, and deleting queries associated with views” on page 157.

If the view was created from one or several tables or views The name, code, description and data type of the view column are those of the corresponding column in the linked table or view.

If the view is user-defined It implies the view is not linked to another object. The name and code of the view column comes from the column name in the first query in the view definition. For example, MyView is defined by the following queries:

```
select Name, Comment
from Property
union
select Signature, Body
from Method
```

Only the two columns of the first query are used to create the corresponding view columns:



In this case, if you modify the view column code, the view creation script will reflect the change. In our example, if you rename Name in ClientName, the view creation script is the following:

```
create view MYVIEW (ClientName, "Comment") as
select Name, Comment
from Property
```

View column properties

When you select a column in the list of view columns and click the Properties tool in the toolbar, the view column property sheet is displayed. You can define the following properties from the view column property sheet:

Property	Description
Name	Specifies the name of the view column. This name is automatically calculated. If you choose to modify the default name, the User-defined button is selected and the Custom Name column displays the user-defined name. You can recover the default name by clicking again the User-defined button
Code	Specifies the code of the view column. The code is automatically calculated. If you choose to modify the default code, the User-defined button is selected and the Custom Code column displays the user-defined code. You can recover the default code by clicking again the User-defined button
Comment	Specifies the comment of the view column. This comment is automatically calculated from the column comment in the original table or view. If you choose to modify the default comment, the User-defined button is selected. You can recover the default comment by clicking again the User-defined button
Stereotype	View column stereotype
Data Type	View column data type. This data type is automatically calculated from the column data type in the original table or view. If you choose to modify the default data type, the User-defined button is selected. You can recover the default data type by clicking again the User-defined button
Length	Maximum length of the data type
Precision	Maximum number of places after the decimal point

You can also define notes and business rules on a view column.

User-defined name or code

You can modify the name and the code of a view column from the list. If you need to recover the name or code default value, you have to clear the corresponding cell in the list, the default name or code is automatically restored.

View property sheet SQL Query tab

The SQL Query tab displays the SQL code for all the queries associated with the view. You can edit this code directly in this tab or access the property sheets of individual queries. For more information, see “Creating, editing, and deleting queries associated with views” on page 157.

View property sheet Triggers tab

The Triggers tab is only displayed if your DBMS supports triggers on views.

View triggers can make the view behave like a table. You can define a trigger to fire when one or more attributes of a table view column are modified.

❖ To create a trigger on a view

1. Open the view property sheet and click the Triggers tab.
2. Click the Add a Row tool, and then click the Properties tool to open the property sheet of the newly-created trigger.
3. Click the Definition tab. The trigger time type will be set to “instead of”.
4. Write a trigger from scratch or select a trigger template. For more details about writing triggers, see the chapter “Building Triggers and Procedures”

View property sheet Preview tab

The Preview tab displays the SQL code associated with the view. For more information, see “Previewing SQL statements” in the Working with Data Models chapter.

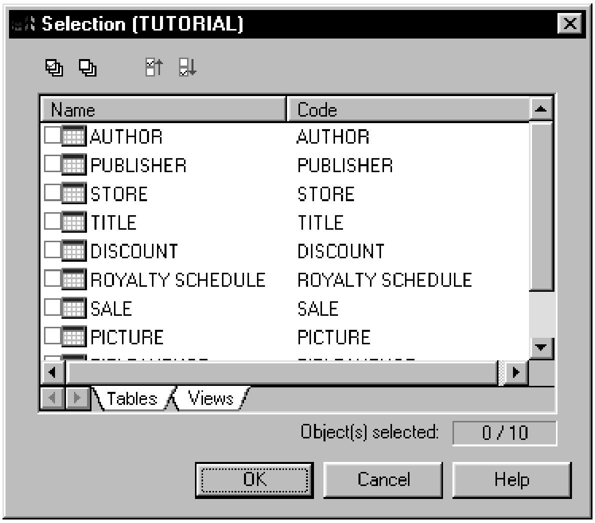
Creating a view from the Tools menu

You can create a view from the Tools menu. This method allows you to automatically populate the view with columns from tables and other views

❖ To create a view from the Tools menu

1. [optional] Select one or more tables and views in the diagram. You can select multiple objects by holding down the SHIFT key while you select them.
2. Select Tools ➤ Create View.

   If you have not selected any tables or views, then a selection box opens, allowing you to select the objects to be included in the view.

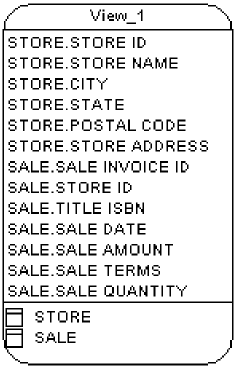


Name	Code
AUTHOR	AUTHOR
PUBLISHER	PUBLISHER
STORE	STORE
TITLE	TITLE
DISCOUNT	DISCOUNT
ROYALTY SCHEDULE	ROYALTY SCHEDULE
SALE	SALE
PICTURE	PICTURE

Object(s) selected: 0 / 10

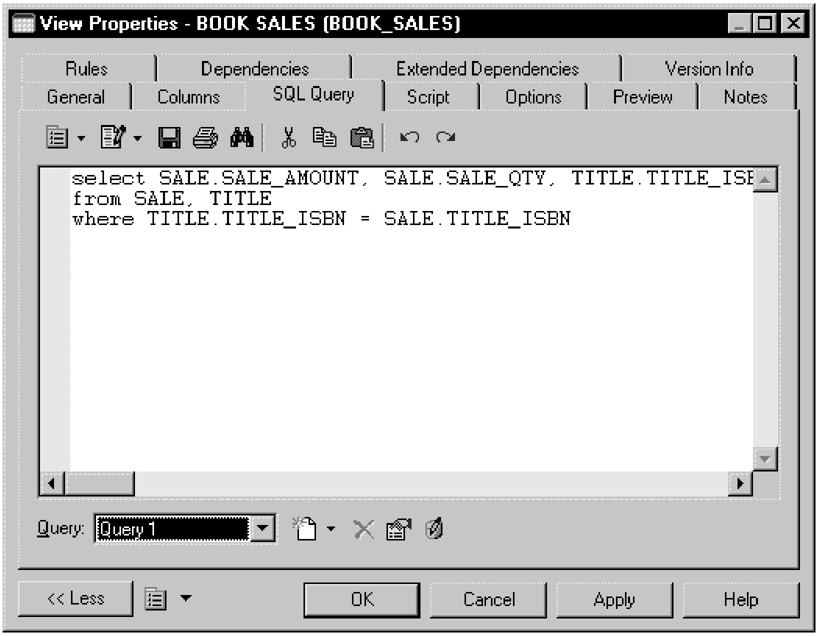
Select the appropriate objects and then click OK.

A view symbol is displayed in the diagram. It displays all the columns in each of the tables and views selected for the view. The names for the tables and views appear at the bottom of the view symbol.



Creating, editing, and deleting queries associated with views

You can edit queries associated with a view from the SQL Query tab of the view property sheet.



```
select SALE.SALE_AMOUNT, SALE.SALE_QTY, TITLE.TITLE_ISE
from SALE, TITLE
where TITLE.TITLE_ISBN = SALE.TITLE_ISBN
```

Any number of queries may be associated with a view, and the totality of their SQL statements is shown in this tab, linked by any of the standard SQL constructs, such as Union, etc.

Editing query code in the SQL Query tab

You can edit the code shown in the SQL Query tab in any of the following ways:

- Edit the code directly in the tab
- Click the Edit with SQL Editor tool to open the code in PowerDesigner's built-in SQL Editor. The SQL Editor provides a more complete query definition environment than the SQL query tab, including access to standard SQL constructs and syntax tools for functions and operators (see "Defining queries with the SQL Editor" in the Working with Data Models chapter).
- Click the Edit with tool (CTRL+E) to open the code in your favorite editor

Any edits you make in the SQL Query tab will propagate to the property sheets of the associated individual queries (see "Opening the property sheet of a query" on page 159).

Query list

The individual queries associated with the view are available from the Query list at the bottom of the SQL Query tab. You can create and delete queries using the tools at the bottom of the tab.

❖ To create a new query in the Query list

1. Click the Add a query tool to the right of the Query list (you can specify the way in which the new query will be linked with the other queries by clicking the arrow to the right of the tool and selecting one of the constructs listed below).

   The new query's property sheet opens.

2. Enter the query code and click OK.

   The new query is added to the Query list in the View property sheet SQL Query tab, and its code is added to the tab.

The following SQL constructs are available for linking queries in PowerDesigner (depending on your DBMS):

Construct	Result	Example

Union [default]	Displays all the data retrieved by both the queries, except where results are repeated.	SELECT 1: ABC
		SELECT 2: BCD
		Result: ABCD

Union All	Displays all the data retrieved by both the queries, including repeated results.	SELECT 1: ABC
		SELECT 2: BCD
		Result: ABCBCD

Intersect	Displays only the data retrieved by both the queries.	SELECT 1: ABC
		SELECT 2: BCD
		Result: BC

Minus	Displays only the data retrieved by one or other of the queries, but not by both.	SELECT 1: ABC
		SELECT 2: BCD
		Result: AD

❖ To delete a query from the Query list

1. Select the query in the Query list.

2. Click the Delete tool to the right of the Query list.

   The query is removed from the Query list and its code deleted from the tab.

Opening the property sheet of a query

Each query associated with a view has its own property sheet, which contains the following tabs:

- SQL (see "Query property sheet SQL tab" on page 160)
- Tables (see "Query property sheet Tables tab" on page 160)
- Columns (see "Query property sheet Columns tab" on page 161)
- Where (see "Query property sheet Where tab" on page 161)
- Group By (see "Query property sheet Group By tab" on page 162)
- Having (see "Query property sheet Having tab" on page 162)
- Order By (see "Query property sheet Order By tab" on page 163)

You can manipulate each of these clauses using the lists in these tabs, and any changes you make will propagate to the other tabs and the SQL Query tab of the parent view (see "Editing query code in the SQL Query tab" on page 158).

❖ To open a query's property sheet from the Query list

1. Select the query in the Query list.
2. Click the Properties tool to the right of the Query list.

Query property sheet SQL tab

This tab displays the SQL code for the query.

You can edit the code of an individual query in its SQL tab in any of the following ways:

- Edit the code directly in the tab
- Click the Edit with SQL Editor tool to open the code in PowerDesigner's built-in SQL Editor (see "Defining queries with the SQL Editor" in the Working with Data Models chapter).
- Click the Edit with tool (CTRL+E) to open the code in your favorite editor

Any edits you make in the SQL tab will propagate to the query's "clause" tabs and the SQL Query tab of the parent view (see "Editing query code in the SQL Query tab" on page 158).

Query property sheet Tables tab

This tab lists the tables in the FROM clause, which specify where the query data will be drawn from.

You can add or delete tables as appropriate, and reorder the tables in the list using the arrows at the bottom of the tab.Any changes you make will propagate to the query's SQL tab and to the SQL Query tab of the parent view.

❖ To add a table

1. Click in the first empty row in the list.
2. Select a table from the list. To enter a more complex expression via the SQL Editor, click the ellipsis button to the right of the list.
3. [optional] Enter an alias for the table in the Alias column.

Query property sheet Columns tab

This tab lists the columns in the SELECT clause, which specify what data will be displayed in the query.

You can add or delete columns as appropriate, and reorder the columns in the list using the arrows at the bottom of the tab. Any changes you make will propagate to the query's SQL tab and to the SQL Query tab of the parent view.

❖ To add a column

1. Click in the first empty row in the list.
2. Select a column from the list. You can add all the columns in a table by selecting a list entry with the table name followed by an asterisk. To enter a more complex expression via the SQL Editor, click the ellipsis button to the right of the list.
3. [optional] Enter an alias for the column in the Alias column.

Query property sheet Where tab

This tab lists the expressions in the WHERE clause, which restrict the data retrieved by the query.

You can add or delete expressions as appropriate, and reorder the expressions in the list using the arrows at the bottom of the tab. Any changes you make will propagate to the query's SQL tab and to the SQL Query tab of the parent view.

❖ To add an expression

1. Click in the first empty row in the list.
2. [optional] Enter a prefix in the Prefix column.
3. In the first Expression column, select a column from the list. To enter a more complex expression via the SQL Editor, click the ellipsis button to the right of the list.
4. In the Operator column, select an operator from the list.
5. In the second Expression column, select a column from the list. To enter a more complex expression via the SQL Editor, click the ellipsis button to the right of the list.
6. [optional] Enter a suffix in the Suffix column.

Query property sheet Group By tab

This tab lists the columns in the GROUP BY clause, which control how the data retrieved by the query will be grouped.

You can add or delete columns as appropriate, and reorder the columns in the list using the arrows at the bottom of the tab. Any changes you make will propagate to the query's SQL tab and to the SQL Query tab of the parent view.

❖ To add a column

1. Click in the first empty row in the list.
2. Select a column from the list. To enter a more complex expression via the SQL Editor, click the ellipsis button to the right of the list.

Query property sheet Having tab

This tab lists the expressions in the HAVING clause, which restrict the data returned by a query with a GROUP BY clause.

You can add or delete expressions as appropriate, and reorder the expressions in the list using the arrows at the bottom of the tab. Any changes you make will propagate to the query's SQL tab and to the SQL Query tab of the parent view.

❖ To add an expression

1. Click in the first empty row in the list.
2. [optional] Enter a prefix in the Prefix column.
3. In the first Expression column, select a column from the list. To enter a more complex expression via the SQL Editor, click the ellipsis button to the right of the list.
4. In the Operator column, select an operator from the list.
5. In the second Expression column, select a column from the list. To enter a more complex expression via the SQL Editor, click the ellipsis button to the right of the list.
6. [optional] Enter a suffix in the Suffix column.

Query property sheet Order By tab

This tab lists the columns in the ORDER BY clause, which control the way in which the data retrieved by the query will be sorted.

You can add or delete columns as appropriate, and reorder the columns in the list using the arrows at the bottom of the tab. Any changes you make will propagate to the query's SQL tab and to the SQL Query tab of the parent view.

❖ To add a column

1. Click in the first empty row in the list.
2. In the Column column, select a column from the list. To enter a more complex expression via the SQL Editor, click the ellipsis button to the right of the list.
3. In the Sort Direction column, select either ASC or DESC.

Creating a query table

A query table is a table whose data proceeds from the result of a query. In PowerDesigner, you design a query table using a view with a specific type, depending on the DBMS.

DB2

In DB2 CS7, you design a query table using a view with the **summary table** type. In later versions of DB2, you should use the **materialized query table** type.

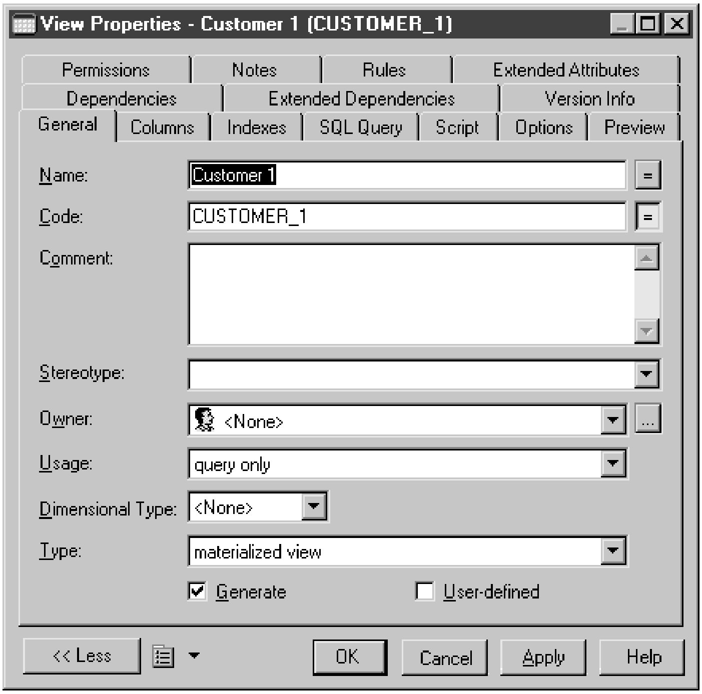
☞ For more information on summary tables and materialized views, see DB2 documentation.

Oracle

In Oracle, you design a query table using a view with the **materialized view** type. Materialized view is the new recommended name for **snapshots** that were used before version 8i.

☞ For more information on materialized views and snapshots, see Oracle documentation.

You define a view as a query table by selecting the query table or materialized view type in the view property sheet.



Name: Customer 1

Code: CUSTOMER_1

Comment:

Stereotype:

Owner: <None>

Usage: query only

Dimensional Type: <None>

Type: materialized view

☑ Generate ☐ User-defined

The query creation order of DB2 summary table is of type:

```
create summary table VIEW_1 as
...
```

The query creation order of DB2 materialized query table is of type:

```
create table VIEW_1 as
...
```

The query creation order of Oracle materialized view is of type:

```
create materialized view VIEW_1 as
...
```

The query creation order of an Oracle snapshot is of type:

```
create snapshot VIEW_1 as
...
```

If you change the DBMS of a model containing query tables, these are converted into regular views.

Physical options

Query tables support physical options. When you select a query table type in the view property sheet, the Options tab automatically is displayed to let you define physical options for view generation.

Using extended dependencies for views

Extended dependencies are links between physical diagram objects. These links help to make object relationships clearer between model objects but are not interpreted and checked by PowerDesigner as they are meant to be used for documentation purposes only.

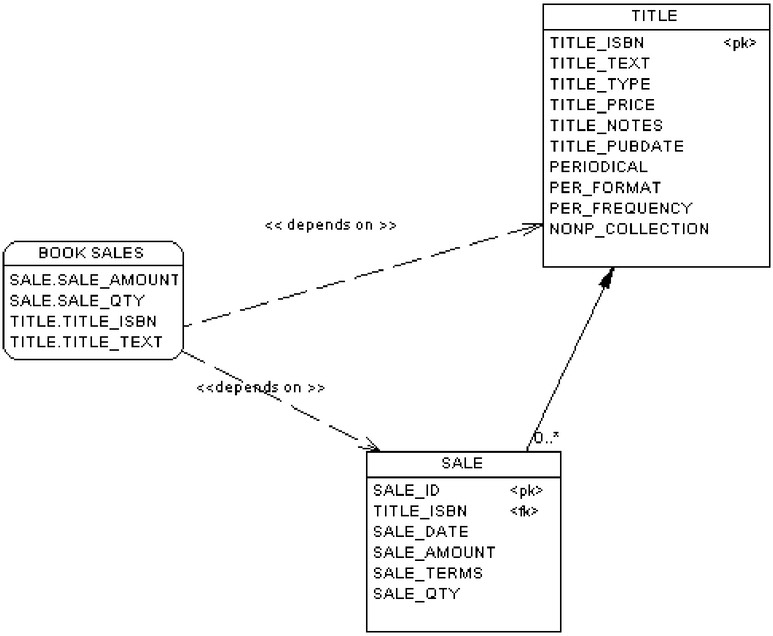
You can complement these links by applying stereotypes.

You can type stereotypes directly in the Stereotype column of the object property sheet or select a value from the list if you have previously defined stereotypes in an embedded or imported extended model definition (.XEM).

You can use extended dependencies between a view and tables links in the model.

Example

A view identified as Book Sales can have two extended dependencies indicating that the view depends on the Title and Sale tables. The diagram displays their extended dependencies and stereotypes.



For more information on extended model definitions, see Extended Model Definitions in the Working with Data Models chapter.

☞ For more information on extended dependencies, see “Using extended dependencies”, in the Objects chapter of the *Core Features Guide*.

Defining a generation order for views

You can use extended dependencies to define an order in the generation of views.

Extended dependencies are free links between PDM objects. These links help to make object relationships clearer between model objects. Usually, these links are not interpreted and checked by PowerDesigner as they are meant to be used for documentation purposes only. However, if you assign the <<**DBCreateAfter**>> stereotype to an extended dependency between views, it will be analyzed during generation.

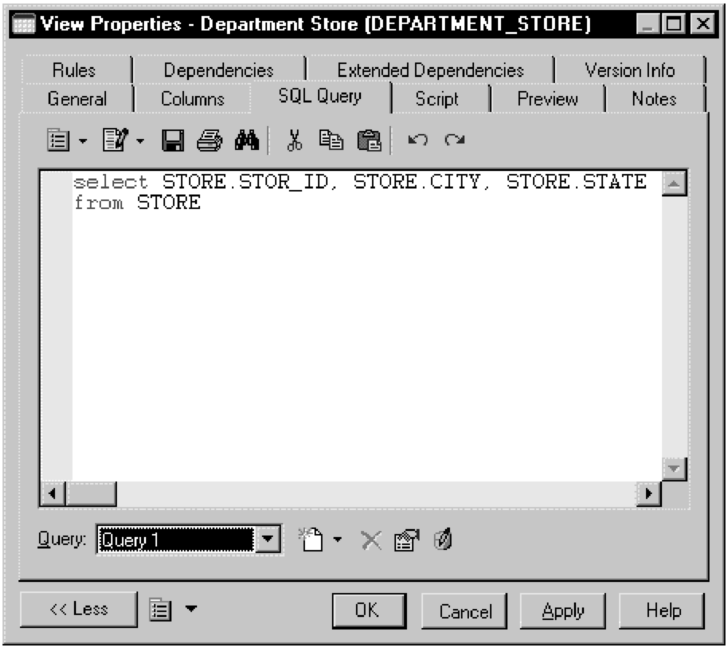
The view from which you start the extended dependency is the dependent view and the view at the other end of the link is the influent view. The influent view will be generated before the dependent view.

Circular extended dependencies

If you create a reflexive and/or circular set of extended dependencies with the <<**DBCreateAfter**>> stereotype, an error message is displayed during the check model. If you choose to ignore this error, the views will be generated in alphabetical order, without taking into account the generation order, which could cause errors in the creation of views in the database.

Example

You create the view DEPARTMENT STORE from the table STORE. The view retrieves information from the table as you can check in the SQL Query tab of the view property sheet:

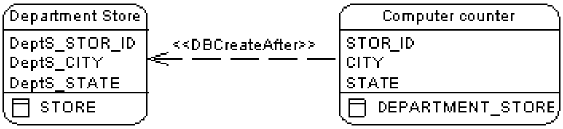


```
select STORE.STOR_ID, STORE.CITY, STORE.STATE
from STORE
```

Query: Query 1

You decide to create another view called COMPUTER COUNTER to show only part of the department store offer. This view is created from the view DEPARTMENT STORE, and retrieves information from it.

By default views are generated in alphabetical order, so the generation of COMPUTER COUNTER will fail since the view DEPARTMENT STORE from which it depends is not generated. To bypass this problem, you can create a extended dependency with the <<**DBCreateAfter**>> stereotype from COMPUTER COUNTER to DEPARTMENT STORE.



This allows you to set an order in the generation of views: DEPARTMENT STORE will be generated before COMPUTER COUNTER.

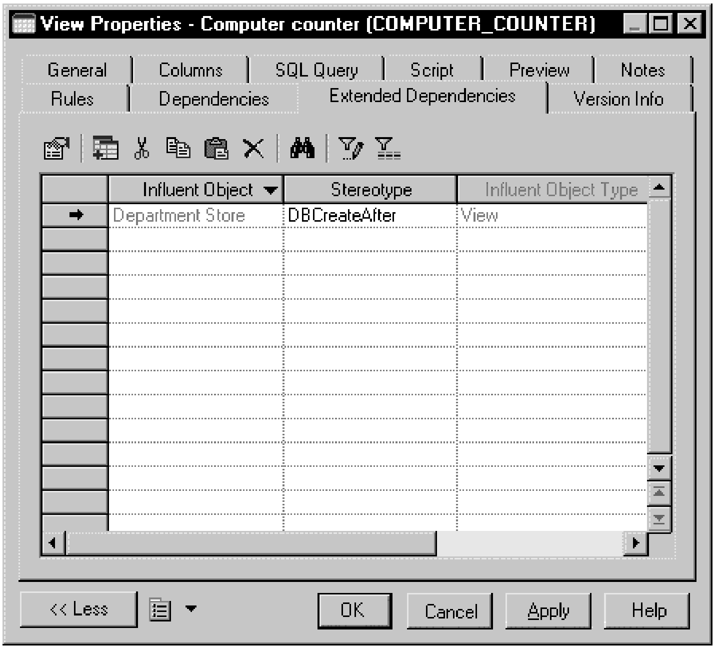
You can create an extended dependency between views from the list of views or from the diagram.

☞ For more information on how to create views from a list, see “Defining a generation order for stored procedures” section in the Building Triggers and Procedures chapter.

❖ To define a generation order for views from the diagram

1. Select the Extended Dependencies tool in the palette.
2. Click inside the dependent view and while holding down the mouse button, drag the cursor into the influent view. Release the mouse button.
3. Double-click the extended dependency link.

   The dependent view property sheet opens to the Extended Dependencies tab. you can check that the influent view is displayed in the Influent Object column of the list of extended dependencies.
4. Click inside the Stereotype column, click the down arrow and select <<**DBCreateAfter**>> in the list.



	Influent Object	Stereotype	Influent Object Type
➡	Department Store	DBCreateAfter	View

5. Click OK.

☞ For more information on extended dependencies, see “Using extended dependencies”, in the Objects chapter of the *Core Features Guide* .

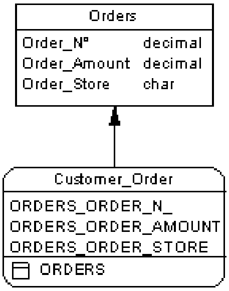
View References (PDM)

A **view reference** is a link between a parent table or view and a child table or view. It is used to define a set of predefined joins between the columns of the parent and the child table or view.

View references are not generated in the database.

Example

Table Orders is the parent of view Customer_Order.



Creating a view reference

You can create a view reference between two views or between a table and a view. A view reference cannot link two tables.

You can create a view reference in any of the following ways:

- Use the Reference tool in the diagram Palette.
- Select Model ➤ View References to access the List of View References, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤ View Reference.

☞ For general information about creating objects, see the Objects chapter in the *Core Features Guide* .

☞ For more information about manipulating view references, see “Changing a table at either end of a reference” on page 150 and “Modifying a reference graphically” on page 150.

View reference properties

You can modify an object’s properties from its property sheet. To open a view reference property sheet, double-click its diagram symbol or its Browser entry in the View References folder.

The General tab contains the following properties:

Property	Description
Name	Specifies the name of the item, which should be clear and meaningful, and should convey the item's purpose to non-technical users
Code	Specifies the technical name of the object, which is used for generating code or scripts, which may be abbreviated, and should not generally include spaces
Comment	Specifies a descriptive label for the view reference
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined
Parent	Specifies the parent table or view of the view reference. You can use the tools to the right of the list to create an object, browse the complete tree of available objects or view the properties of the currently selected object
Parent role	Specifies the role of the parent table or view in the view reference. The text is displayed in the diagram, near the parent table or view
Child	Specifies the child table or view of the view reference. You can use the tools to the right of the list to create an object, browse the complete tree of available objects or view the properties of the currently selected object
Child role	Specifies the role of the child table or view in the view reference. The text is displayed in the diagram, near the child table or view

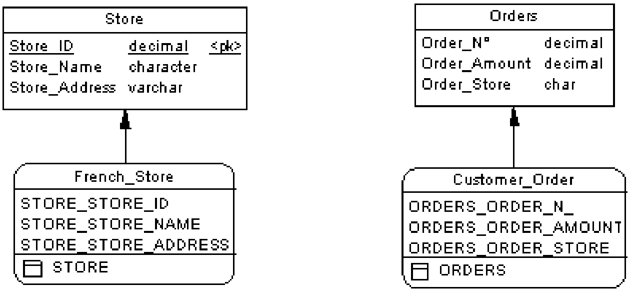
A view reference also includes joins, that are links between parent columns and child columns.

Defining view reference joins

A **join** is a link between a column in a parent table or view and a column in a child table or view that is defined within a view reference.

If you create a new view from existing views, the joins defined on these views influence the WHERE statement in the SQL query of the new view.

Example

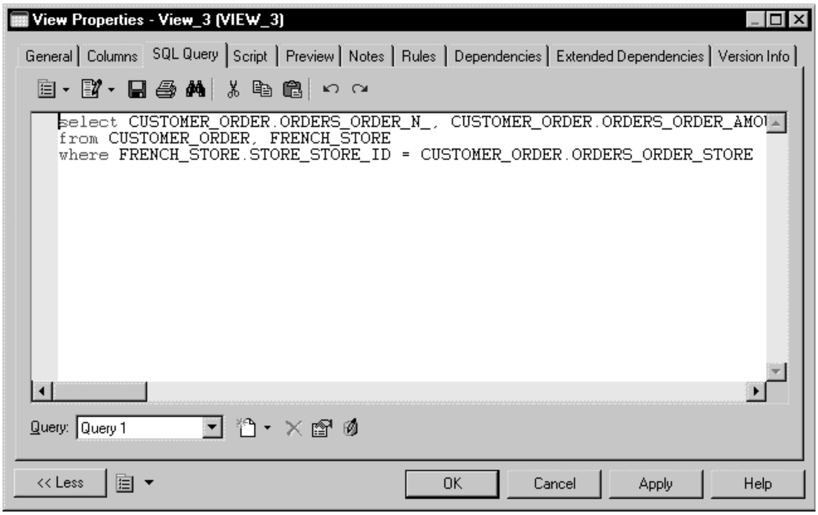


French_Store is a view of table Store. You define a join between Store_ID in the table and STORE_STORE_ID in the view.

Customer_Orders is a view of table Orders. You define a join between Order_No in the table and ORDER_ORDER_N in the view.

You create a view reference between French_Store and Customer_Order in which you define a join between ORDER_ORDER_STORE and STORE_STORE_ID. This is to establish a correspondence between the store ID and the store where the order is sent.

If you create a view from French_Store and Customer_Orders, you can check in the SQL query tab of the view that the SELECT order takes into account the join defined between the views. The SELECT statement will retrieve orders sent to French stores only.



```
select CUSTOMER_ORDER.ORDERS_ORDER_N_, CUSTOMER_ORDER.ORDERS_ORDER_AMO
from CUSTOMER_ORDER, FRENCH_STORE
where FRENCH_STORE.STORE_STORE_ID = CUSTOMER_ORDER.ORDERS_ORDER_STORE
```

In the Joins tab of a view reference property sheet, you can use the Reuse Columns tool to reuse existing child columns with same code as parent columns.

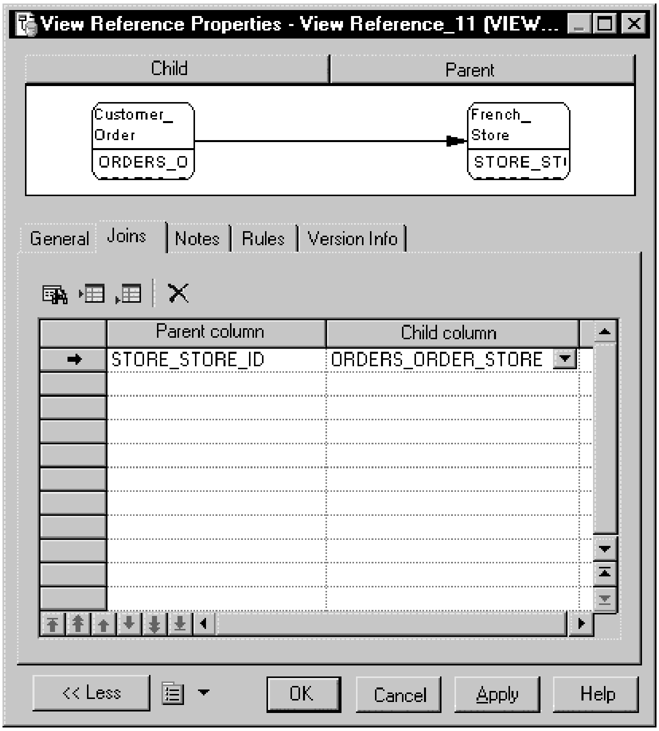
❖ To define joins in a view reference

1. Double-click a view reference in the diagram to display the view reference property sheet.
2. Click the Joins tab to display the Joins tab.
3. Click the Reuse Columns tool to reuse existing child columns with same code as parent columns.

   or

   Click the Add a Row tool.

   A join is created but you have to define the parent and child columns.
4. Click in the Parent Column column and select a column in the list.
5. Click in the Child Column column and select a column in the list.



Parent column	Child column
STORE_STORE_ID	ORDERS_ORDER_STORE

6. Click OK.

Check Parameters (CDM/LDM/PDM)

Check parameters are set of conditions which data must satisfy to remain valid.

There are three types of check parameters:

Parameter type	Description	Can be attached to
Standard check parameters	Common data constraints which define a data range. For example minimum and maximum values for a column	Columns/Entity attributes Domains
Additional check parameters	SQL expression defining a data constraint using the %MINMAX%, %LISTVAL%, and %RULES% variables that are instantiated with standard parameter values	Columns/Entity attributes Domains
Validation rule	Business rule that is defined as a server expression, and is attached to one of the following listed objects	Tables/Entities Columns/Entity attributes Domains

Setting standard check parameters for objects in a data model

Standard parameters indicate common data constraints. The following table lists standard parameters:

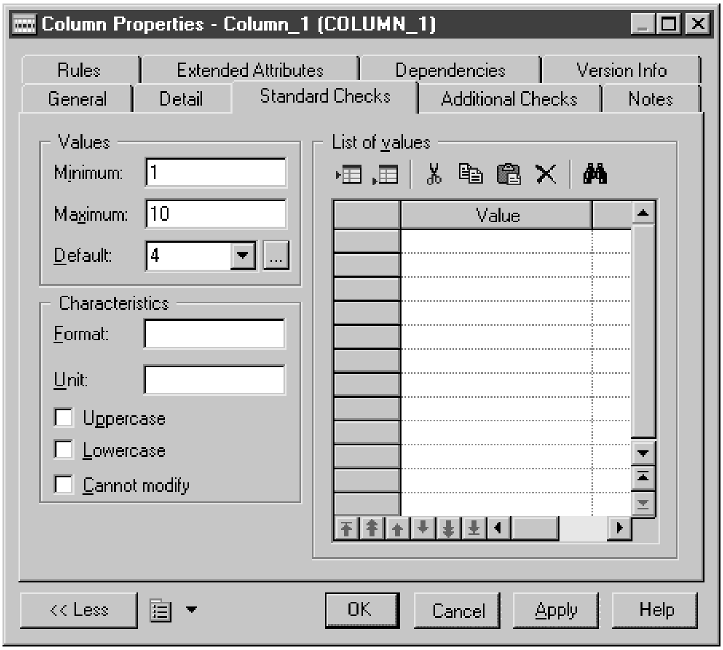
Parameter	Description
Minimum	Lowest acceptable numeric value
Maximum	Highest acceptable numeric value
Default	Value selected from a list of default values or typed in the listbox. The list of values is defined in the Script\Keywords\ReservedDefault entry of the DBMS definition file
Format	Data format (for example, 9999.99)
Unit	Standard measure
Lowercase	Forces all alphabetical characters to lowercase
Uppercase	Forces all alphabetical characters to uppercase

Parameter	Description
Cannot Modify	Protects from changes, results in a non-modifiable column/entity attribute in the table when generated in the database
List of Values	Authorized values
Label	String that identifies an authorized value in the list

☞ For more information on defaults, see “Defaults (PDM)” on page 111.

❖ To set standard parameters

1. Open the property sheet of a domain or column/entity attribute and click the Standard Checks tab.



Column Properties - Column_1 (COLUMN_1)

Values
Minimum: 1
Maximum: 10
Default: 4

Characteristics
Format:
Unit:
☐ Uppercase
☐ Lowercase
☐ Cannot modify

2. Enter the appropriate standard parameters, and then click OK.

☞ See also “Managing quotation marks around values (PDM)” on page 178.

Defining additional check parameters for objects in a data model

You define additional check parameters for data constraints where standard check parameters are not sufficient.

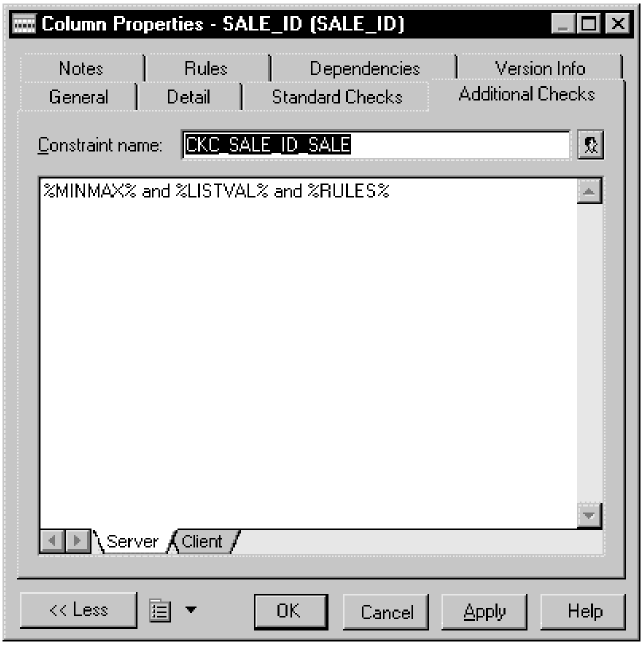
Example

A table/entity in a data model for a clothing shop may contain check parameters defined for a column/entity attribute SIZE, which depend on the check parameters defined on another column/entity attribute CLOTHING TYPE, as clothing size for a skirt in one country may be different from the same size in another country.

In this case an expression is required to create a constraint which uses check parameters defined for both columns/entity attributes.

❖ To define additional check parameters

1. Open the property sheet of a domain or column/entity attribute and click the Additional Checks tab.



Constraint name: CKC_SALE_ID_SALE

```
%MINMAX% and %LISTVAL% and %RULES%
```

2. Type a SQL expression, which may include any of the following variables:
   - %MINMAX% - Minimum and maximum values defined in Values groupbox on Standard Checks tab
   - %LISTVAL% - Customized values defined in List Values groupbox on Standard Checks tab
   - %RULES% - Validation rule expression defined on Expression tab of the Rules property sheet
3. Click OK to return to the model diagram.

Using a validation rule in check parameters in a data model

A validation rule is a rule that validates data based on a corresponding business rule. A validation rule can be generated as a check parameter when the following conditions apply:

- Validation rule is attached to a table/entity, column/entity attribute, or domain
- Validation rule is defined as a server expression

At generation, validation rule variables are instantiated with the following values:

Variable	Value
%COLUMN%	Code of the column to which the business rule applies
%DOMAIN%	Code of the domain to which the business rule applies
%TABLE%	Code of the table to which the business rule applies
%MINMAX%	Minimum and maximum values for the column or domain
%LISTVAL%	List values for the column or domain
%RULES%	Server validation rules for the column or domain

☞ For more information on defining business rules, see "Business Rules (CDM/LDM/PDM)" on page 179.

❖ To use a validation rule in check parameters

1. Open the property sheet of a table, domain, or column, and click the Rules tab.
2. Click the Add Objects button to open a selection box, and select a business rule in the list.
3. Click OK in each of the dialog boxes.

Validation rule expressions
You must click the Rules button to modify the expression attached to a validation rule. You can also modify validation rule expressions from the list of business rules, by clicking the Define button.

Managing quotation marks around values (PDM)

For a domain or a column, standard parameters can indicate minimum, maximum, and default values as well as a list of values.

In general, if the data type of domain or column is a string data type, quotation marks surround its values in the generated script. The generation of single or double quotation marks depends on the target DBMS.

However, quotation marks are not generated in the following cases:

- You define a data type that is not recognized as a string data type by PowerDesigner
- Value is surrounded by tilde characters
- Value is a keyword defined in the DBMS (for example, NULL)

In addition, if the value is already surrounded by quotation marks additional quotation marks are not generated.

The following table shows the way a value for a string data type is generated in a script:

Value for string data type	Result in script
Active	‘Active’
‘Active’	‘Active’
”Active”	‘”Active”’
~Active~	Active
NULL	NULL

Business Rules (CDM/LDM/PDM)

A business rule is a rule that your business follows. This can be a government-imposed law, a customer requirement, or an internal guideline.

Business rules often start as simple observations. For example, “customers call toll-free numbers to place orders”. During the design process they develop into more detailed expressions. For example, what information a customer supplies when placing an order or how much a customer can spend based on a credit limit.

Business rules guide and document the creation of a model. For example, the rule “an employee belongs to only one division” can help you graphically build the link between an employee and a division.

Business rules complement model graphics with information that is not easily represented graphically. For example, some rules specify physical concerns in the form of formulas and validation rules. These technical expressions do not have a graphical representation.

During intermodel generation the business rules transfer directly into the generated model where you can further specify them.

There are three ways to use business rules in a PDM:

- You can apply a business rule to an object in the PDM
- You can create a server expression for a business rule which can be generated to a database
- A business rule expression can also be inserted in a trigger or stored procedure (see the Building Triggers and Procedures chapter)

Before you create business rules, formulate your rules by asking yourself the following questions:

- What business problems do I want to address?
- Are there any mandatory procedures for my system?
- Do any specifications set the scope of my project?
- Do any constraints limit my options?
- How do I describe each of these procedures, specifications, and constraints?
- How do I classify these descriptions: as definitions, facts, formulas, or validation rules?

Creating a business rule

You can create a business rule in any of the following ways:

- ♦ Select Model ➤ Business Rules to access the List of Business Rules, and click the Add a Row tool
- ♦ Right-click the model or package in the Browser, and select New ➤ Business Rule
- ♦ Open the property sheet of the object to which you want to apply the rule, click the Rules tab, and click the Create an Object tool

For general information about creating objects, see the “Creating Objects” chapter in the *Core Features Guide* .

Business rule properties

You can modify an object’s properties from its property sheet. To open a business rule property sheet, double-click its Browser entry in the Business Rules folder.

The General tab contains the following properties:

Property	Description
Name	The name of the item which should be clear and meaningful, and should convey the item’s purpose to non-technical users
Code	The technical name of the item used for generating code or scripts, which may be abbreviated, and should not generally include spaces
Comment	Descriptive label for the rule
Stereotype	Sub-classification used to extend the semantics of an object.

Property	Description
Type	Specifies the nature of the business rule. You can choose between: ♦ Constraint – a check constraint on a value. In a PDM, constraint business rules can be generated in the database. For example, “The start date should be inferior to the end date of a project.” ♦ Definition – a property of the element in the system. For example; “A customer is a person identified by a name and an address”. ♦ Fact – a certainty in the system. For example, “A client may place one or more orders”. ♦ Formula – a calculation. For example, “The total order is the sum of all the order line costs”. ♦ Requirement – a functional specification. For example, “The model is designed so that total losses do not exceed 10% of total sales”. ♦ Validation – a constraint on a value. For example, “The sum of all orders for a client must not be greater than that client’s allowance”.

Attaching an expression to a business rule in a PDM

A business rule typically starts out as a description. As you develop your model and analyze your business problem, you can complete a rule by adding a technical expression. The syntax of expressions depends on the target database.

Each business rule can include two types of expression:

- Server - can be generated to a database. You can generate server expressions as check parameters if they are attached to tables, domains, or columns
- Client - used mainly for documentation purposes. However, you can insert both types of expression into a trigger or a stored procedure

❖ To attach an expression to a business rule in a PDM

1. Open the business rule property sheet and click the Expression tab.
2. Enter a server or client expression on the appropriate sub-tab.

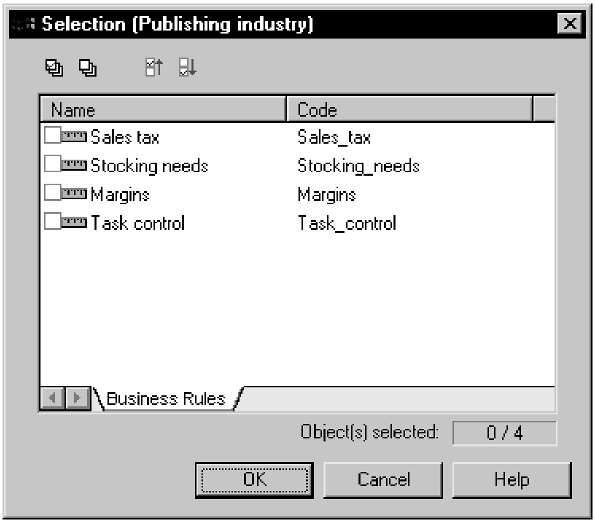
3. Click OK in each of the dialog boxes.

Applying a business rule to a model object

You can apply a business rule to a model object from the object’s property sheet.

❖ To apply a business rule to a model object

1. Open the model object’s property sheet and click the Rules tab.
2. Click the Add Objects tool to open a list of available business rules.



Name	Code
□ Sales tax	Sales_tax
□ Stocking needs	Stocking_needs
□ Margins	Margins
□ Task control	Task_control

Object(s) selected: 0 / 4

3. Select one or more business rules and click OK.

   The business rules are added to the object and appear in the list of business rules for the object.

4. Click OK to return to the model diagram.

U Column in the List of business rules
When you apply a business rule to an object, the U (Used) column beside this business rule is automatically checked in the List of business rules to indicate that the business rule is used by at least one object in the model. The U column allows you to visualize unused business rules, you can then delete them if necessary.

Constraint business rules

A constraint business rule is used to manage multiple constraints on tables and columns in a database for the DBMS that support this feature.

During generation, the variable %RULES% is evaluated in order to generate a single constraint in the SQL script. This constraint is a concatenation of all check parameters defined in the Check tab of a table property sheet, and all validation business rules applied to the current table. You can preview the result of the concatenation of checks and validation business rules in the Preview page of a table property sheet.

☞ For more information on the check parameters of a table, see "Naming a table constraint" on page 67.

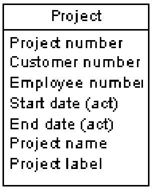
If you want to define distinct check constraints on tables and columns, you have to define constraint business rules and attach them to objects. This type of business rule allows to generate multiple check constraints on an object in the database.

This feature is only available for DBMS supporting multiple check constraints, and provided the **EnableMultiCheck** entry in the General category of the DBMS is set to Yes.

☞ For more information on DBMS entries, see the DBMS Resource File Reference chapter in the *Customizing and Extending PowerDesigner* manual.

Example

Let's consider the table Project:



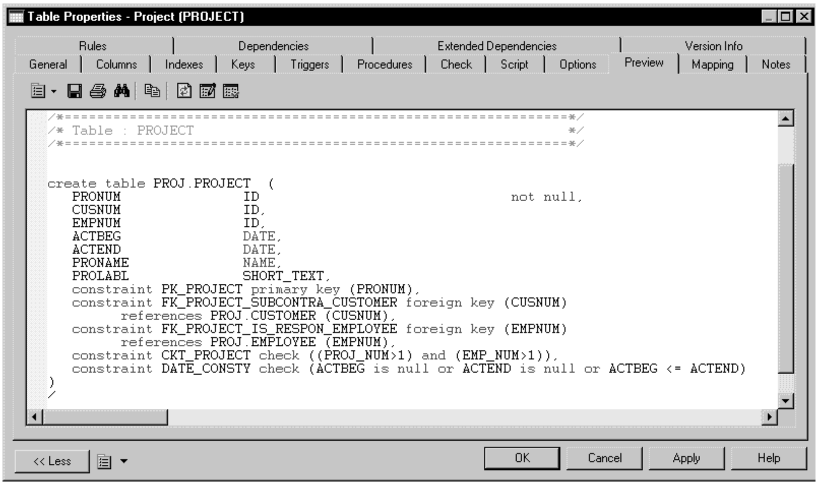
Project

Project number
Customer number
Employee numbei
Start date (act)
End date (act)
Project name
Project label

☞ The following checks are defined for this table:

Check	Description
Check parameter (in the Check page of the table)	This check verifies that the customer number is different from the employee number
Validation business rule	PROJ_NUM to check that the column project number is not null EMP_NUM to check that the employee number is not null
Constraint business rule	DATE_CONSTY to check that the start date of the project is inferior to the end date of the project

☞ When you display the table code preview, you can verify that the check parameters and validation business rules are concatenated into a single constraint, whereas the constraint business rule is displayed as a different constraint in the script.



```
/*==============================================================*/
/* Table : PROJECT                                              */
/*==============================================================*/


create table PROJ.PROJECT  (
   PRONUM               ID                              not null,
   CUSNUM               ID,
   EMPNUM               ID,
   ACTBEG               DATE,
   ACTEND               DATE,
   PRONAME              NAME,
   PROLABL              SHORT_TEXT,
   constraint PK_PROJECT primary key (PRONUM),
   constraint FK_PROJECT_SUBCONTRA_CUSTOMER foreign key (CUSNUM)
         references PROJ.CUSTOMER (CUSNUM),
   constraint FK_PROJECT_IS_RESPON_EMPLOYEE foreign key (EMPNUM)
         references PROJ.EMPLOYEE (EMPNUM),
   constraint CKT_PROJECT check ((PROJ_NUM>1) and (EMP_NUM>1)),
   constraint DATE_CONSTY check (ACTBEG is null or ACTEND is null or ACTBEG <= ACTEND)
)
/
```

Creating a constraint business rule

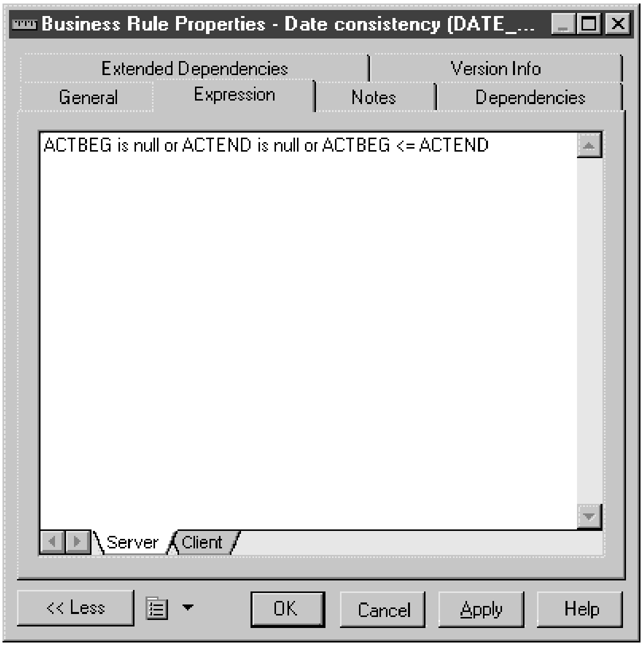
❖ To create a constraint business rule in a PDM

1. Select Model ➤ Business Rules to open the List of Business Rules, and click the Add a Row tool to create a new rule.

2. Enter a name and a code for the new rule, and click the Properties tool to open its property sheet:

3. Select Constraint in the Type list, and then click the Expression tab.

4. Enter an expression in the Server page.



ACTBEG is null or ACTEND is null or ACTBEG <= ACTEND

5. Click OK to save your changes and return to the model diagram.

Applying a constraint business rule to a table or a column

You can attach a constraint business rule to a table or a column. You cannot reuse a constraint business rule between different objects, so you must create as many as needed for your model objects.

When you attach a constraint business rule to an object, the code of the business rule will be used as the constraint name. If you wish to enforce code uniqueness for constraints in your model, you have to set the **UniqueConstName** entry in the General category of the DBMS to Yes. In such case, the code of the constraint generated from validation business rules and the code of the constraint business rules must be unique in the model.

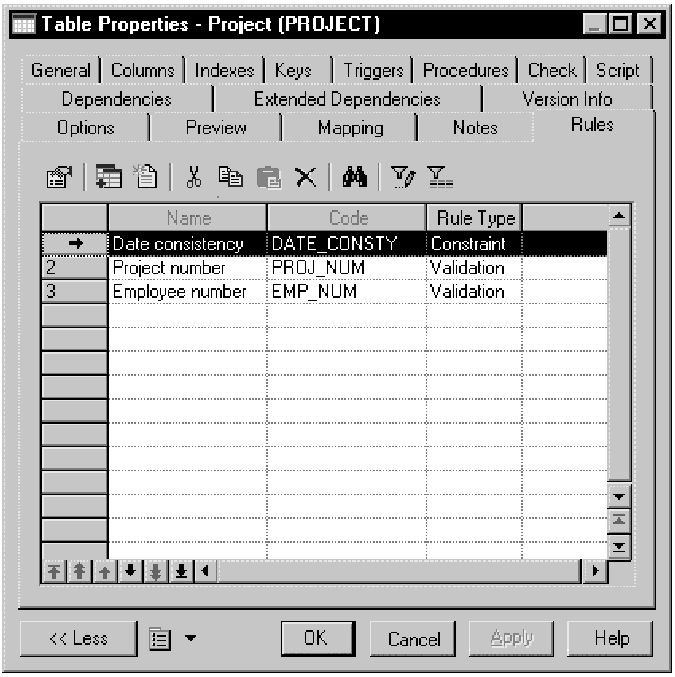
☞ For more information on DBMS entries, see the DBMS Resource File Reference chapter in the *Customizing and Extending PowerDesigner* manual.

❖ To attach a constraint business rule to a table or column

1. Open the table or column’s property sheet and click the Rules tab.

2. Click the Add Objects tool to open a list of available business rules.

3. Select a constraint business rule from the selection list and click OK to attach it to the object.



	Name	Code	Rule Type
→	Date consistency	DATE_CONSTY	Constraint
2	Project number	PROJ_NUM	Validation
3	Employee number	EMP_NUM	Validation

4. [optional] Click Apply to confirm the attachment of the rule and then click the Preview tab to verify that the constraint has been created in the script:

5. Click OK to return to the diagram.

Generating a constraint business rule

Depending on the type of DBMS, the following is generated regarding check constraints:

Database type	Generation result
Database does not support any check constraints	No constraint is generated
Database does not support multiple check constraints	Constraint business rules, check parameters, and validation business rules are concatenated into a single constraint expression
Database supports multiple check constraints	First, the check parameters and validation business rules are generated into a single constraint, then the constraint business rules are generated into different constraints. Constraints are thus ordered

Reverse engineering a constraint business rule

During reverse engineering, the constraint order is taken into account:

- The first constraint is retrieved as check parameters (in the Check page of the table property sheet)
- Each constraint following the initial constraint is retrieved as a constraint business rule

Physical Options

A physical option is a parameter (included at the end of a Create statement) that defines how an object is optimized or stored in a database. Physical options are not supported by all databases, and vary by DBMS. In ASA 6, for example, you can define physical options for tables, columns, indexes, tablespaces, and databases. Other DBMSs provide options for keys, storages, and sequences.

The syntax for a physical option depends on the DBMS. For example:

DBMS	Tablespace option syntax
Oracle 6.0 or higher	Tablespace
Sybase Adaptive Server Anywhere	In

☞ For more information about the syntax of physical options and how they are controlled in the DBMS resource file, see “Physical Options” in the DBMS Resource File Reference chapter of the *Customizing and Extending PowerDesigner* manual.

When you change DBMS, the physical option settings are applied as far as possible to the new DBMS. If a specific physical option was selected in the model, the default value is preserved for the option in the new DBMS. Unselected physical options are reset with the new DBMS default values.

PowerDesigner allows you to set:

- ♦ Default physical options for all the objects in the model (see “Defining default physical options” on page 188)
- ♦ Physical options for a specific object, which override the default physical option values (see “Defining physical options for individual objects” on page 191)

Defining default physical options

Default physical options define physical options for all the objects of a particular type in the model. The default physical options are stored in the DBMS definition file (see the Working with Data Models chapter).

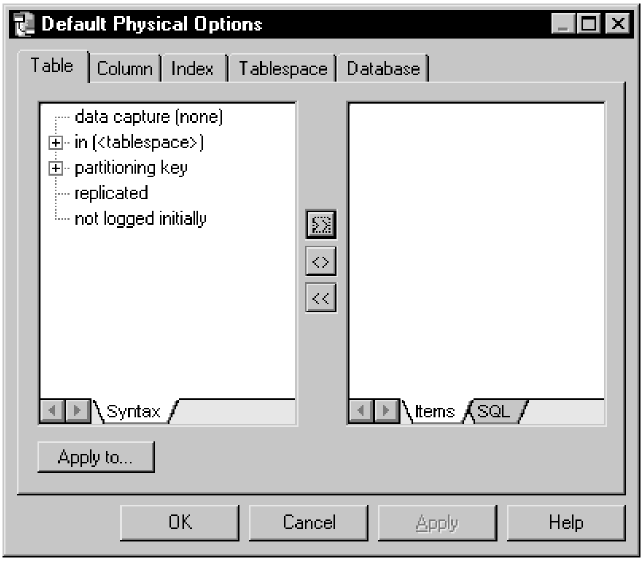
The values you assign for default physical options can be included in a database creation script that you generate from a PDM, or can be directly generated in the target database. You select the appropriate default physical options from the Selection tab of the Database Generation dialog box.

Physical option default values can also be reverse engineered from a database into a PDM.

You can view the script for a physical option in the Preview tab for the object for which it is defined.

❖ To define default physical options

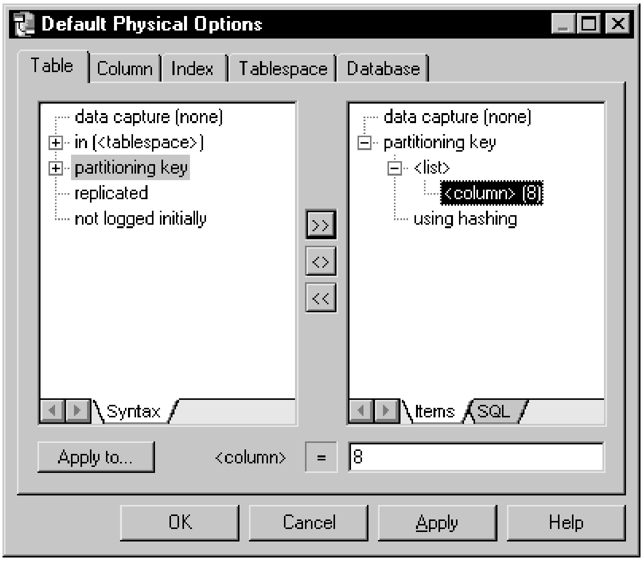
1. Select Database ➤ Default Physical Options to display the Default Physical Options dialog box. There is a tab for each kind of object that supports physical options. The Table tab opens by default:



2. The Syntax tab in the left pane lists the physical options available in the DBMS, and the right pane contains the physical options that have been selected for the object.

3. To add an option for the object, select it in the list in the Syntax pane and click the Add tool between the panes to copy it to the Items pane. To add only a sub-parameter for the option, expand the option in the Syntax pane, select the required parameter and then click the Add tool.

4. To set a value for a physical option parameter, select the parameter in the list in Items pane and enter or select the appropriate value in the field that is displayed below the pane. The entered value will then be displayed against the parameter in the Items list.



5. Repeat the above steps as many times as necessary to specified all your required physical options. By default, these options will be applied to all tables in the model. To specify that the options should apply to only certain of the existing tables, click the Apply to button to display a selection dialog. Select the tables that you want to apply the options to from the list and then click OK.

6. Select the other tabs to specify physical options for other option types. (Note that the Apply to button is not available on the Database tab).

7. Click OK to close the Default Physical Options dialog box.

The following tools are available for adding and removing physical options to an object:

Tool	**Action when clicked**
>>	Adds physical option selected in Syntax tab (left pane) to Items tab (right pane)
<>	Aligns a selected physical option in the Items tab with the corresponding physical option in the Syntax tab
<<	Removes physical option selected in Items tab

Defining physical options for individual objects

☞ You can set physical options for selected objects to override the default physical options (see "Defining default physical options" on page 188).

☞ There are two different interfaces for specifying physical options for individual objects, both of which are accessible through tabs on the object's property sheet:

♦ **Physical Options (Common)** – this tab is displayed by default (along with the **Partition** tab, if applicable), and lists the most commonly-used physical options in the format of a standard property sheet tab. You can enter values for the necessary option parameters and click OK

♦ **Physical Options (All)** – this tab is hidden by default, and lists all the available physical options for the object in a tree format. To display this tab, click the Property Sheet Menu button and select Customize Favorite Tabs ➤ Physical Options (All). You should follow the procedure in "Defining default physical options" on page 188, to modify the appropriate options.

☞ Changes made on either of these tabs will be reflected on the other.

Tablespace and storage options

You can choose a tablespace or storage already defined in the model as a value for a tablespace or storage that you define for a table. These are listed in a list below the right pane in the Options tab. In DB2 OS/390, tablespaces are, by default, prefixed by the name of the database attached to the model (see the DBMS-Specific Features chapter). For more information on defining tablespaces and storages, see the Generating a Database from a PDM chapter.

DBMS-specific information

The following table lists the objects for which physical options are available when working with a particular DBMS:

Database	Objects supporting physical options
Sybase AS Anywhere	Table, Index, Database, Tablespace
Sybase AS Enterprise	Table, Index, Key, Database
IBM DB2 for Common Server	Table, Column, Index, View, Storage, Tablespace
IBM DB2 for OS/390	Table, Column, Index, Database, Storage, Tablespace
Microsoft SQL Server	Table, Index, Key, Database
Oracle	Table, Key, Index, View, Join Index, Sequence, Database, Tablespace, Storage

For information about these options, see your database documentation.

CHAPTER 4

Building Multidimensional Diagrams

About this chapter

This chapter describes how to build multidimensional diagrams, and how to create and modify the associated objects.

.

Contents

Multidimensional Diagram Basics

A **multidimensional** diagram is a model of business activities in terms of cubes and dimensions. It organizes business data into one or other of these categories. Numeric values or measures such as sales total, budget limits, are the facts of the business. The area covered by a business, in terms of geography, time, or products are the dimensions of the business.

The multidimensional diagram is used to design the cubes in an OLAP engine, together with the different analysis dimensions.

Business analysts use OLAP databases to send queries and retrieve business information from the different dimensions existing in the database.

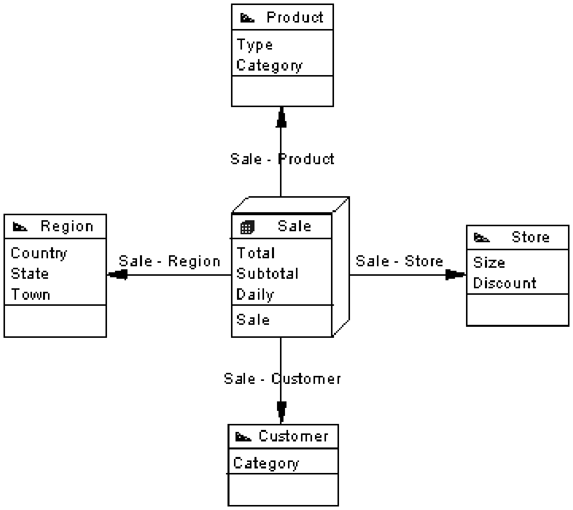
OLAP databases are populated with data from a data warehouse or data mart database. This data transfer is implemented via a relational to multidimensional mapping, the data warehouse or data mart database being the data source of the OLAP database. The OLAP cube is designed to support multidimensional analysis queries, it is organized according to user-defined dimensions.

Multidimensional analysis queries usually involve calculated lists, such as growth and decline rates and time-based comparisons. They aim at detecting trends and relationships. These types of queries are supported by an OLAP database, but not by an operational database.

☞ For more information on the relational to multidimensional mapping, see “Defining a relational to relational mapping” section in the Working with Data Models chapter.

Example

Sales data can have the dimensions product, region, customer, and store. Facts, for example, the sales totals, are viewed through the user-defined dimensions. When you retrieve the sales total of a particular product for a particular region, you are viewing the sales total through the product and region dimensions. The most common dimension is time because the purpose of multidimensional analytical queries is to find trends.


Product
Type
Category
Sale - Product
Region
Country
State
Town
Sale - Region
Sale
Total
Subtotal
Daily
Sale
Sale - Store
Store
Size
Discount
Sale - Customer
Customer
Category

Multidimensional diagram objects

You can create the following objects in a multidimensional diagram:

Object	Tool	Symbol	Description
Cube	▦	▦ Cube	Collection of measures related to aspects of the business and used to carry out a decision support investigation. See "Cubes (PDM)" on page 198.
Dimension	◺	◺ Dim	Axis of investigation of a cube (time, product, geography). See "Dimensions (PDM)" on page 209.
Attribute	[none]	[none]	Used to qualify a dimension. For example, attribute Year qualifies the Date dimension. See "Attributes (PDM)" on page 211.
Fact	[none]	[none]	Group of measures used among cubes. See "Facts (PDM)" on page 214.
Measure	[none]	[none]	Variable linked to a fact, used as the focus of a decision support investigation. See "Measures (PDM)" on page 216.
Hierarchy	[none]	[none]	Organizational structure that describes a traversal pattern though a dimension. See "Hierarchies (PDM)" on page 218.
Association	▦◺	→	Association that relates a cube to a dimension. See "Associations (PDM)" on page 220.

Creating a multidimensional diagram

You can create a multidimensional diagram in an existing PDM in any of the following ways:

- Right-click the model in the Browser and select New ➤ Multidimensional Diagram from the contextual menu
- Right-click the background of any diagram and select Diagram ➤ New Diagram ➤ Multidimensional Diagram from the contextual menu.

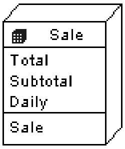
To create a new PDM with a multidimensional diagram, select File ➤ New, choose Physical Data Model from the Model type list, choose Multidimensional Diagram as the first diagram, and click OK.

Cubes (PDM)

A cube is a collection of measures (see "Measures (PDM)" on page 216) corresponding to values stored into each of its data cells. The measures are organized into dimensions (see "Dimensions (PDM)" on page 209) to provide for faster retrieval and drill-down.

Usually a cube is associated with a fact that allows to define and share measures among cubes.

In a multidimensional diagram, the cube represents an OLAP cube. Cubes need to be created and populated via a text file in the OLAP engine. This text file contains a query used to extract data from a data warehouse or operational database to fill the cubes in the OLAP engine, and is defined in the Query tab of the cube's property sheet.



Creating a cube

You can create a cube in any of the following ways:

- Use the Cube tool in the diagram Palette.
- Select Model ➤ Cubes to access the List of Cubes, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤ Cube.
- Rebuild a cube from a fact table or view defined in a physical diagram (see "Retrieving Multidimensional Objects" on page 199).

☞ For general information about creating objects, see the Objects chapter in the *Core Features Guide* .

Cube properties

You can modify an object's properties from its property sheet. To open a cube property sheet, double-click its diagram symbol or its Browser entry in the Cubes folder. The following sections detail the property sheet tabs that contain the properties most commonly entered for cubes.

The General tab contains the following properties:

Property	Description
Name	Specifies the name of the item, which should be clear and meaningful, and should convey the item's purpose to non-technical users
Code	Specifies the technical name of the object, which is used for generating code or scripts, which may be abbreviated, and should not generally include spaces
Comment	Specifies a descriptive label for the cube
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined
Fact	Specifies the fact used by the cube. You can use the tools to the right of the list to create an object, browse the complete tree of available objects or view the properties of the currently selected object

The following tabs are also available:

- Fact Measures - lists the Measures linked to the fact used by the cube (see "Measures (PDM)" on page 216).
- Queries - displays the SQL statement required to generate the cube data text file that is used to populate the OLAP cube.

Retrieving Multidimensional Objects

During the design of a data warehouse, you will need to identify which of your tables and views will represent facts, and which dimensions. In this section, we will refer to both tables and views as being either:

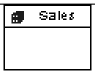
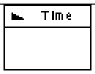
- A **fact table** – which stores variable numerical values related to aspects of a business (for example, sales, revenue, budget). These are usually the values you want to obtain when you carry out a decision support investigation.
- A **dimension table** – which stores data related to the axis of investigation of a fact (for example, geography, time, product). A dimension table should be connected to a central fact table.

You can specify your tables and views individually as facts or dimensions, or you can use Multidimensional Objects Retrieval Wizard to perform this task automatically, based on the references connecting them:

- Child tables or views become Fact tables or views

- Parent tables or views become Dimension tables or views

The new type is indicated in the Dimensional Type field in the object’s property sheet, and a type icon is displayed in the upper left corner of its symbol:

Fact table	Dimension table
	

❖ To retrieve multidimensional objects

1. Select Tools ➤ Multidimension ➤ Retrieve Multidimensional Objects to open the Multidimensional Objects Retrieval Wizard.
2. Specify the objects to be retrieved. By default both Facts and Dimensions will be retrieved.

> **Sybase AS IQ v12.0 and higher**
> If you are working with Sybase AS IQ v12.0 or higher, you can also select to automatically rebuild join indexes after retrieving multidimensional objects. For more information, see “IQ join indexes” in the DBMS-Specific Features chapter.

3. [optional] Click the Selection tab to specify which tables to retrieve multidimensional objects from..
4. Click OK to retrieve the multidimensional objects..

   The selected tables are assigned a multidimensional type.

Rebuilding Cubes

Once the fact and dimension tables of the data warehouse schema are designed, you can use this information to build the multidimensional cubes. The Rebuild Cubes Wizard transforms fact tables or views into cubes, and dimension tables or views into dimensions. You can then design the cubes taking into account the different analysis axes of the dimensions. These cubes will serve to generate the text files used to create and populate the OLAP engine.

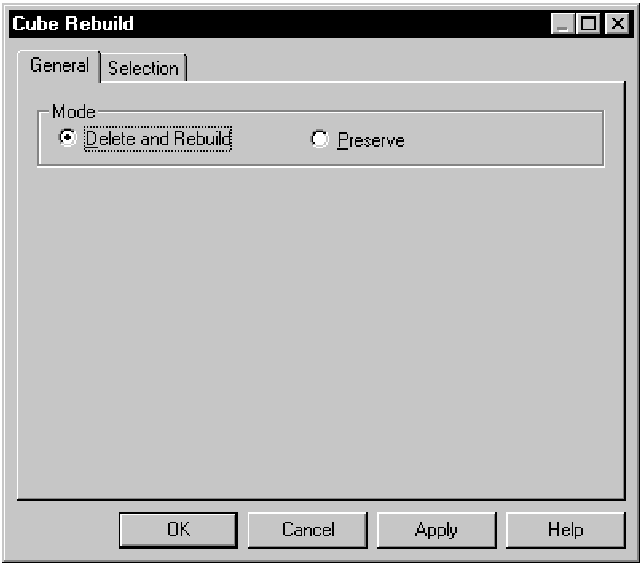
The Rebuild Cubes feature works only if there are tables in the physical diagram with a multidimensional type (Fact or Dimension). You can assign types either manually (see “Table property sheet General tab” in the Building Physical Diagrams chapter) or via the Multidimensional Objects Retrieval Wizard (see “Retrieving Multidimensional Objects” on page 199).

The Rebuild Cubes Wizard creates multidimensional objects in a new or existing multidimensional diagram as follows:

Physical object	After rebuild cubes, creates
Fact table	A fact with the name of the fact table. A cube with the name of the fact table.
Column in a Fact table (except foreign keys)	A measure with the name of the column.
Dimensions tables attached to the fact table	A dimension, named through the concatenation of the dimension tables along the path to the child table, from the furthest to the closest. A hierarchy that becomes the default hierarchy, and which contains attributes corresponding to the primary key columns of the tables converted into a dimension
Column in a Dimension table (except foreign keys)	An attribute, named through the concatenation of the dimension table name and column name if column names are ambiguous. Otherwise the name is identical to the name of the column.
Reference between a fact and a dimension table	A cube dimension association

❖ To rebuild cubes

1. Select Tools ➤ Multidimension ➤ Rebuild Cubes to open the Cube Rebuild Wizard:



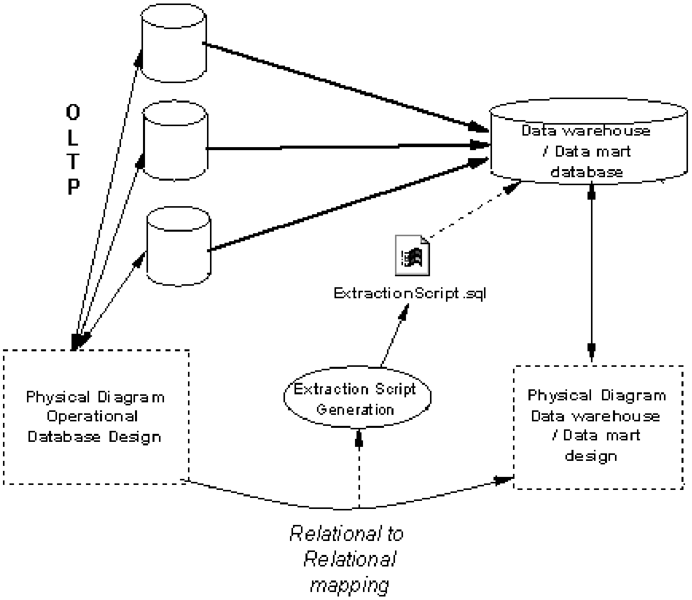
Mode
● Delete and Rebuild ○ Preserve

2. Specify a rebuild mode. You can choose between the following options:
   - ♦ Delete and Rebuild – all cubes are deleted and rebuilt, including those to which you have made modifications.
   - ♦ Preserve – only those cubes that have not been modified are deleted and rebuilt. Any cubes that you have modified are preserved.
3. [optional] Click the Selection tab to specify which tables or views will be used to rebuild cubes. Only those tables and views that have a multidimensional type are available for rebuilding.
4. Click OK to rebuild cubes.

   A message in the Output window informs you that the rebuild is successful. The cube and dimension are created and displayed in a multidimensional diagram.

Generating Extraction Scripts

The Generate Extraction Script feature allows you to generate script files that will be used to fill and update tables in a data warehouse or data mart database.


O
L
T
P
Data warehouse / Data mart database
ExtractionScript.sql
Physical Diagram Operational Database Design
Extraction Script Generation
Physical Diagram Data warehouse / Data mart design
Relational to Relational mapping

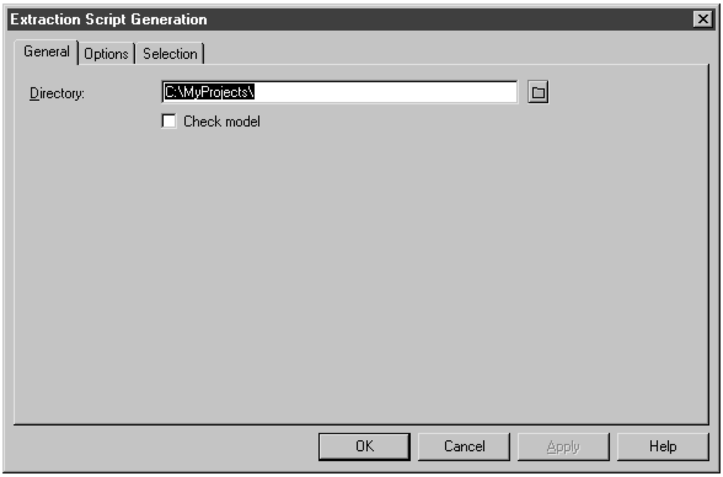
The link between the operational database and the data warehouse or data mart database is a relational to relational mapping.

☞ For more information about relational to relational mapping, see “Relational to relational mapping” in the “Creating Mappings” chapter of the *Core Features Guide* .

You can generate a script file for each data source, you can also select the tables in the data source which `select` orders will be generated in the script file. The extraction scripts list all the `select` orders defined in the table mappings.

❖ To generate an extraction script

1. In the Physical Diagram, select Database ➤ Generate Extraction Script to open the Extraction Script Generation dialog box.



Directory: C:\MyProjects\

□ Check model

2. Specify a destination directory for the generated file, and select the Check Model check box if you want to verify the PDM syntax before generation. The name of the script is identical to the name of the data source.

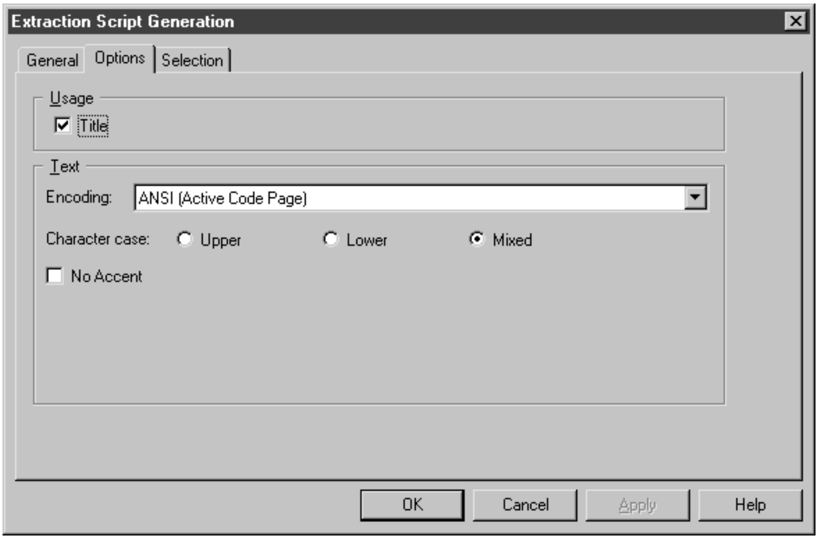
3. [optional] Click the Options tab and specify any appropriate options. For more information, see "Extraction Script Generation Options tab" on page 204.

4. [optional] Click the Selection tab, and select the tables that you want to use in the script generation.

5. Click OK to generate the script files in the specified directory.

Extraction Script Generation Options tab

The Options tab allows you to specify the format for the script.



Usage

☑ Title

Text

Encoding: ANSI (Active Code Page)

Character case: ○ Upper ○ Lower ● Mixed

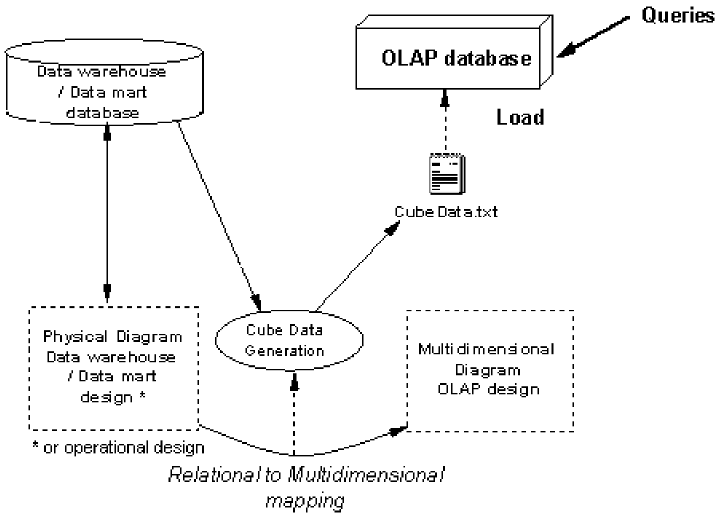
□ No Accent

The following options are available:

Option	Description
Title	Inserts the database header and the name of the tables before each select query.
Encoding	Encoding format to use for generation. You should select the encoding format that supports the language used in your model and the database encoding format.
Character Case	Defines the character case in the generated text file.
No Accent	When selected, disallows the use of accents.

Generating Cube Data

Cubes in an OLAP database have to be filled with data from a data warehouse, data mart or operational database. PowerDesigner allows you to generate a text file that will be used by an OLAP tool to create and populate cubes using data from operational sources.


Queries
OLAP database
Load
Data warehouse / Data mart database
Cube Data.txt
Physical Diagram Data warehouse / Data mart design *
* or operational design
Cube Data Generation
Multidimensional Diagram OLAP design
Relational to Multidimensional mapping

In a PDM multidimensional diagram, each cube is associated with a query. There is one cube per mapping and per data source. The query defined on a cube is used to extract data from a data warehouse or operational database to populate the cubes in the OLAP database. The link between the data warehouse database and the OLAP database is a relational to multidimensional mapping.

☞ For more information about relational to multidimensional mapping, see “Relational to multidimensional mapping” section in the Creating Mappings chapter of the *Core Features Guide* .

When you generate cube data, PowerDesigner produces one text file for each selected cube and each selected data source. The name of the generated file is a concatenation of the name of the cube and the name of the data source.

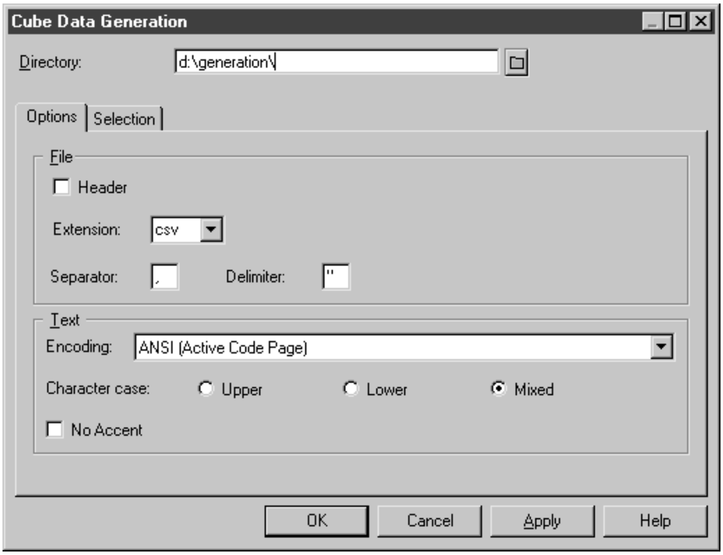
Each file contains the following fields:

Field	Details
Dimension	Lists the attributes of the cube
Member	Lists the attribute values
Data fields	Contains the values stored in the fact measures

❖ To generate cube data

1. In the multidimensional diagram, select Tools ➤ Generate Cube Data.

   The Generate Cube Data dialog box is displayed.



Directory: d:\generation\

□ Header

Extension: csv

Separator: ,

Delimiter: "

Encoding: ANSI (Active Code Page)

Character case: ○ Upper ○ Lower ● Mixed

□ No Accent

2. Define a destination directory for the generated file in the Directory box.

3. Select the generation options in the Options tab.

4. Select the cubes and data sources for which you want to generate a file from the sub- tabs in the Selection tab.

5. Click OK.

   The generated files are stored in the destination directory you have defined.

Cube data generation options

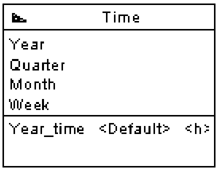
You can customize the format of the generated text files from the Generate Cube Data dialog box.

Option	Description
Header	When selected, includes the name of the attribute at the beginning of the generated text file
Extension	Extension of the generated text file, you can choose between .txt and .csv
Separator	Separator used between columns
Delimiter	String delimiter
Encoding	Encoding format to use for generation. You should select the encoding format that supports the language used in your model and the database encoding format
Character Case	Defines the character case in the generated text file
No Accent	When selected, disallows the use of accents

Dimensions (PDM)

A dimension is an axis of analysis in a multidimensional structure.

The dimension is made of an ordered list of attributes that share a common semantic meaning in the domain being modeled. Each attribute designates a unique position along the axis.



The dimension can be mapped to tables or views: this mapping allows to transfer operational data to the dimension.

☞ For more information on object mapping, see “Mapping objects in a PDM” in the Working with Data Models chapter.

A dimension may have one or more hierarchies representing attribute sets.

Creating a dimension

You can create a dimension in any of the following ways:

- Use the Dimension tool in the diagram Palette.
- Select Model ➤ Dimensions to access the List of Dimensions, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤ Dimension.

☞ For general information about creating objects, see the Objects chapter in the *Core Features Guide* .

Dimension properties

You can modify an object’s properties from its property sheet. To open a dimension property sheet, double-click its diagram symbol or its Browser entry in the Dimensions folder. The following sections detail the property sheet tabs that contain the properties most commonly entered for dimensions.

The General tab contains the following properties:

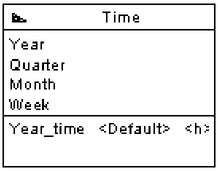
Property	Description
Name	Specifies the name of the item, which should be clear and meaningful, and should convey the item's purpose to non-technical users
Code	Specifies the technical name of the object, which is used for generating code or scripts, which may be abbreviated, and should not generally include spaces
Comment	Specifies a descriptive label for the dimension
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined
Default Hierarchy	Specifies the dimension hierarchy used by default for a cube to perform its consolidation calculations. The hierarchy used by the cube is defined on the cube dimension association

The following tabs are also available:

- Attributes - lists the attributes that qualify the dimension (see "Attributes (PDM)" on page 211).
- Hierarchies - lists the hierarchies used to organize the dimension attributes (see "Hierarchies (PDM)" on page 218).
- Mapping - defines the mapping between the current dimension and a table or a view in a data source.

Attributes (PDM)

An **attribute** is used to qualify dimensions (see "Dimensions (PDM)" on page 209) used in queries. For example, the Time dimension can contain attributes Year, Quarter, Month, and Week.



Attributes can be organized in hierarchies (see "Hierarchies (PDM)" on page 218).

Creating an attribute

You can create an attribute in any of the following ways:

- Open the Attributes tab in the property sheet of a dimension, and click the Add a Row tool.
- Right-click a dimension in the Browser, and select New ➤ Attribute.

☞ For general information about creating objects, see the Objects chapter in the *Core Features Guide* .

Attribute properties

You can modify an object's properties from its property sheet. To open an attribute property sheet, double-click its Browser entry. The following sections detail the property sheet tabs that contain the properties most commonly entered for attributes.

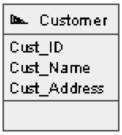
The General tab contains the following properties:

Property	Description
Name	Specifies the name of the item, which should be clear and meaningful, and should convey the item's purpose to non-technical users
Code	Specifies the technical name of the object, which is used for generating code or scripts, which may be abbreviated, and should not generally include spaces
Comment	Specifies a descriptive label for the attribute
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined
Dimension	Parent dimension of the attribute

Attribute property sheet Detail Attributes tab

A detail attribute participates in the definition of an attribute.

Detail attributes appear in the list of dimension attributes, but you can use them to further define a given attribute. For example, attributes Cust_Name and Cust_Address appear in the list of dimension attributes, however they are used as detail attributes of attribute Cust_ID.

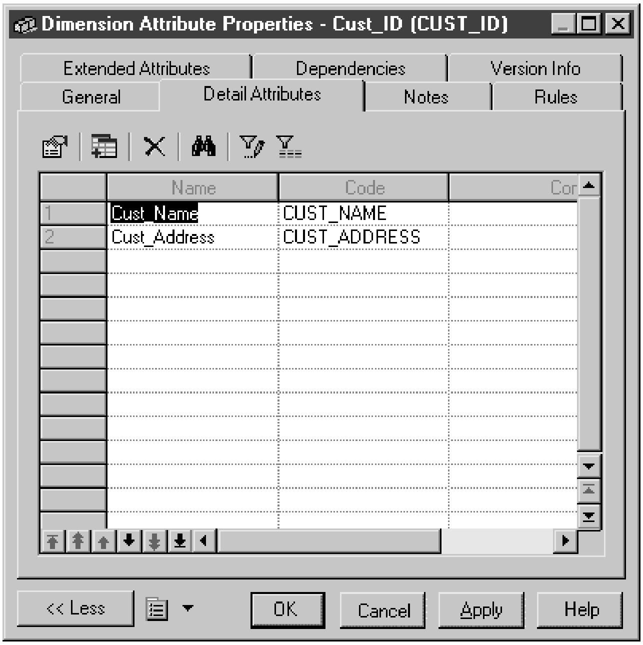


❖ To define a detail attribute

1. Open the property sheet of a dimension, and click the Attributes tab.

2. Select an attribute in the list, click the Properties tool to open its property sheet, and then click the Detail Attributes tab.

3. Click the Add Detail Attributes tool to open a Selection dialog listing the attributes available in the model, select one or more, and then click OK.

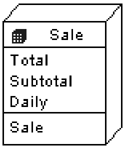


	Name	Code	Cor
1	Cust_Name	CUST_NAME	
2	Cust_Address	CUST_ADDRESS	

4. Click Apply.

Facts (PDM)

A fact corresponds to the focus of a decision support investigation. It is a set of measures (see "Measures (PDM)" on page 216) manipulated by a cube (see "Cubes (PDM)" on page 198). For example, Sale, Revenue, Budget could be facts.



Facts can be reused among different cubes.

Creating a fact

You can create a fact in any of the following ways:

- Select Model ➤ Facts to access the List of Facts, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤ Fact.

☞ For general information about creating objects, see the Objects chapter in the *Core Features Guide* .

Fact properties

You can modify an object's properties from its property sheet. To open a fact property sheet, double-click its Browser entry in the Facts folder. The following sections detail the property sheet tabs that contain the properties most commonly entered for facts.

The General tab contains the following properties:

Property	Description
Name	Specifies the name of the item, which should be clear and meaningful, and should convey the item’s purpose to non-technical users
Code	Specifies the technical name of the object, which is used for generating code or scripts, which may be abbreviated, and should not generally include spaces
Comment	Specifies a descriptive label for the fact
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined

The following tabs are also available:

- Measures - lists the measures manipulated by the cube with which the fact is associated (see “Measures (PDM)” on page 216).
- Mapping - contains the mapping between the fact and a table or a view in a data source.

Measures (PDM)

A measure is a variable that corresponds to the focus of an investigation. Measures describe the meaning of the analytical values stored in each data cell of a cube (see “Cubes (PDM)” on page 198).

Measures are most of the time numeric values like for example Price or Total.

Measures can also be the result of an operation or calculation as indicated in the formula box of the measure property sheet.

Creating a measure

You can create a measure in any of the following ways:

- Open the Measures tab in the property sheet of a fact, and click the Add a Row tool.
- Right-click a fact in the Browser, and select New ➤ Measure.

☞ For general information about creating objects, see the Objects chapter in the *Core Features Guide* .

Measure properties

You can modify an object’s properties from its property sheet. To open a measure property sheet, double-click its diagram symbol or its Browser entry. The following sections detail the property sheet tabs that contain the properties most commonly entered for measures.

The General tab contains the following properties:

Property	Description
Name	Specifies the name of the item, which should be clear and meaningful, and should convey the item's purpose to non-technical users
Code	Specifies the technical name of the object, which is used for generating code or scripts, which may be abbreviated, and should not generally include spaces
Comment	Specifies a descriptive label for the measure
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined
Fact	Specifies the parent fact of the measure
Formula	Indicates if the measure is a computed expression and allows to define this expression

Hierarchies (PDM)

A **hierarchy** defines one of two paths through a dimension (see "Dimensions (PDM)" on page 209):

♦ an **organizational** path - describes a traversal pattern through a dimension, from the most general to the most specific attribute of the dimension. It is an ordered subset of the attributes.

♦ a **consolidation** path - represents a consolidation of attributes. For example, a Time dimension with a base periodicity of days might have a hierarchy specifying the consolidation of days into weeks, weeks into months, months into quarters, and quarters into years.

Creating a hierarchy

You can create a hierarchy in any of the following ways:

♦ Open the Hierarchies tab in the property sheet of a dimension, and click the Add a Row tool.

♦ Open the Attributes tab in the property sheet of a dimension, and click the Create Hierarchy tool.

♦ Right-click a dimension in the Browser, and select New ➤ Hierarchy.

☞ For general information about creating objects, see the Objects chapter in the *Core Features Guide* .

Hierarchy properties

You can modify an object's properties from its property sheet. To open a hierarchy property sheet, double-click its Browser entry. The following sections detail the property sheet tabs that contain the properties most commonly entered for hierarchies.

The General tab contains the following properties:

Property	Description
Name	Specifies the name of the item, which should be clear and meaningful, and should convey the item's purpose to non-technical users
Code	Specifies the technical name of the object, which is used for generating code or scripts, which may be abbreviated, and should not generally include spaces
Comment	Specifies a descriptive label for the hierarchy
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined
Dimension	Specifies the parent dimension of the hierarchy

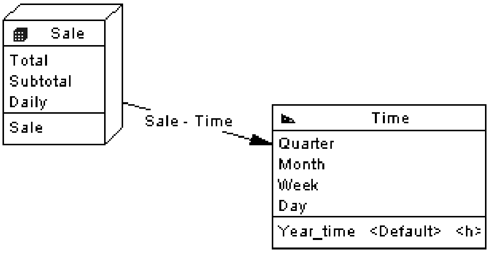
The following tabs are also available:

♦ Attributes - lists the attributes associated with the hierarchy (see "Attributes (PDM)" on page 211).

Associations (PDM)

An association relates a cube (see “Cubes (PDM)” on page 198) to the dimension (see “Dimensions (PDM)” on page 209) that defines it. It shows the axis of investigation of the dimension in the cube.

For example, the Sale cube is linked to the Time dimension by the Sale - Time association to analyze sales through the time dimension.



There can be only one association between a cube and a dimension.

Creating an association

You can create an association in any of the following ways:

- ♦ Use the Association tool in the diagram Palette.
- ♦ Select Model ➤ Associations to access the List of Associations, and click the Add a Row tool.
- ♦ Right-click the model or package in the Browser, and select New ➤ Association.

☞ For general information about creating objects, see the Objects chapter in the *Core Features Guide* .

Association properties

You can modify an object’s properties from its property sheet. To open an association property sheet, double-click its diagram symbol or its Browser entry in the Associations folder.

The General tab contains the following properties:

Property	Description
Cube	Specifies the cube origin of the association. You can click the Properties tool to view the properties of the currently selected object
Dimension	Specifies the destination dimension of the association. You can use the tools to the right of the list to create an object, browse the complete tree of available objects or view the properties of the currently selected object
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined
Hierarchy	Specifies the hierarchy used by the cube for the consolidation calculation. You can click the Properties tool to view the properties of the currently selected object

CHAPTER 5

Building Triggers and Procedures

About this chapter

This chapter presents triggers and procedures used to generate database constraints.

Contents

Triggers (PDM)

A trigger is a segment of SQL code associated with a table or a view, and stored in a database. It is invoked automatically whenever there is an attempt to modify data in the associated table with an insert, delete, or update command.

Triggers enhance the security, efficiency, and standardization of databases.

You can use triggers to enforce referential integrity, where declarative constraints are not sufficient. You can also use triggers to implement sequences for columns.

Trigger templates and template items

A trigger template is a pre-defined form for creating triggers. PowerDesigner ships templates for each supported DBMS. Depending on the current DBMS, there are pre-defined templates for insert, update, and delete trigger types.

A template item is a reusable block of SQL script that can implement referential integrity, or do any other work on database tables. PowerDesigner ships template items for each supported DBMS. A template item is inserted into a trigger template script, or a trigger script. The template item calls a corresponding SQL macro which can implement an insert, update, delete, or error message constraint on one or more tables in the database.

You can use the PowerDesigner templates and template items, copy and edit them, or create your own from scratch. For more information, see "Trigger Templates (PDM)" on page 240.

Creating Triggers for Referential Integrity

You can create triggers automatically for one or more tables in a model, when you specify that triggers are to be used to implement referential integrity and for columns where the values depend on a sequence.

❖ To implement referential integrity between tables by triggers

1. Create a reference between two tables, and then double click the reference symbol to open its property sheet.
2. Click the Integrity tab, and then select Trigger from the Implementation list.
3. Specify the form of Update and Delete constraints using the radio buttons, and then click OK to return to the diagram.
4. If you have set the Automatically rebuild triggers model option, then triggers will have been created automatically in the parent and child

tables. To verify this, double-click the table symbol to open its property sheet, and then click the Triggers tab. If the triggers are not present,, you will need to rebuild your triggers manually.

☞ For more information about the Automatically rebuild triggers model option and rebuilding triggers manually, see "Rebuilding Triggers" on page 229.

☞ For more information about selecting referential integrity options from the Reference property sheet, see section Defining referential integrity in chapter Building Physical Diagrams.

You can instruct PowerDesigner to implement referential integrity between tables using triggers by default.

❖ To implement referential integrity between tables by triggers by default

1. Select Tools Model Options and click Reference under Model Settings in the Category list.

2. Select Trigger from the Default implementation list, and then click OK.

   From now, when you create a reference between tables, referential integrity will be implemented using triggers by default.

Creating Other Triggers

You can create your own triggers from the property sheet of a table (or view, if supported by your DBMS).

You can write a trigger from scratch directly in its property sheet, but we recommend that you use a trigger template and/or trigger template items to define the trigger code. These allow you to create triggers in a modular fashion, to make re-use of your trigger code easier and give your triggers more portability.

☞ For more information, see "Trigger Templates (PDM)" on page 240.

Creating a trigger from a template

You can create a trigger based on one of the PowerDesigner templates or on a template of your own.

❖ To create a trigger from a template

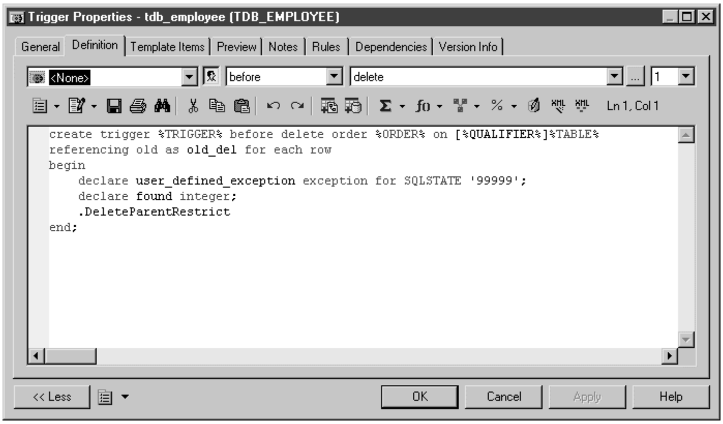
1. Double-click a table symbol to open its property sheet, and then click the Triggers tab.

2. Click the Add a Row tool to create a new trigger, and type a name and code.

3. Click Apply to commit the creation of the new trigger, and then click the Properties tool to open its property sheet.

4. Click the Definition tab, and select a trigger template from the Template list.

   The time and event fields will be set and the template code copied into the definition box.



```
create trigger %TRIGGER% before delete order %ORDER% on [%QUALIFIER%]%TABLE%
referencing old as old_del for each row
begin
    declare user_defined_exception exception for SQLSTATE '99999';
    declare found integer;
    .DeleteParentRestrict
end;
```

5. [optional] Modify the trigger definition code. You can add trigger template items, use PDM variables and macros and various other tools available from the toolbar (see “SQL Code Definition Toolbars” on page 270).

   If you edit the code, then the trigger will be marked as user-defined and will be excluded from most forms of rebuilding (see “Rebuilding Triggers” on page 229).

6. You can also modify the trigger’s other properties. For a full list of the properties available, see “Trigger properties” on page 228.

7. Click OK in each of the dialog boxes.

Creating a trigger from scratch

You can create a trigger without basing it on a template. However, we recommend that you use a template as this will simplify reuse of your code and make your triggers more portable.

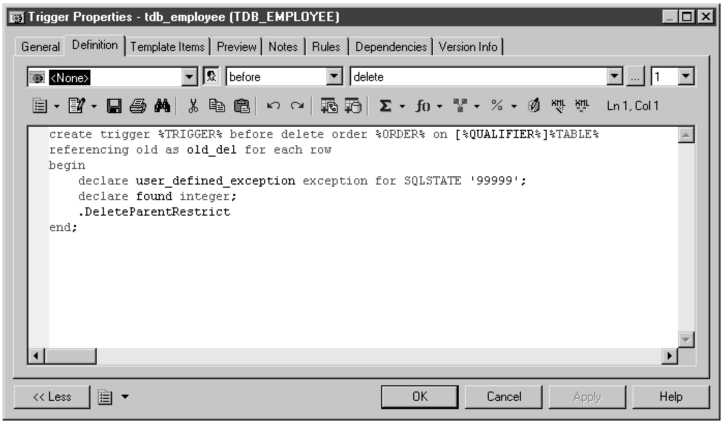
❖ To create a trigger from scratch

1. Double-click a table symbol to open its property sheet, and then click the Triggers tab.

2. Click the Add a Row tool to create a new trigger, and type a name and code.

3. Click Apply to commit the creation of the new trigger, and then click the Properties tool to open its property sheet.

4. Click the Definition tab.



```
create trigger %TRIGGER% before delete order %ORDER% on [%QUALIFIER%]%TABLE%
referencing old as old_del for each row
begin
    declare user_defined_exception exception for SQLSTATE '99999';
    declare found integer;
    .DeleteParentRestrict
end;
```

5. Enter the trigger definition code. You can add trigger template items, use PDM variables and macros and various other tools available from the toolbar (see "SQL Code Definition Toolbars" on page 270).

   The trigger will be marked as user-defined and will be excluded from most forms of rebuilding (see "Rebuilding Triggers" on page 229).

6. You can also modify the trigger's other properties. For a full list of the properties available, see "Trigger properties" on page 228.

7. Click OK in each of the dialog boxes.

Editing triggers in Eclipse
When using the PowerDesigner Eclipse plug-in, you can right-click a trigger in the Browser and select Edit in SQL Editor from the contextual menu to open it in the Eclipse SQL Editor. You can optionally connect to your database in order to obtain auto-completion for table names. The trigger definition is added as a .SQL file to the Generated SQL Files list in the Workspace Navigator.

Trigger properties

You can modify an object's properties from its property sheet. To open a trigger property sheet, double-click its Browser entry in the Triggers folder. The following sections detail the property sheet tabs that contain the properties most commonly entered for triggers.

The General tab contains the following properties:

Property	Description
Name	The name of the item which should be clear and meaningful, and should convey the item's purpose to non-technical users
Code	The technical name of the item used for generating code or scripts, which may be abbreviated, and should not generally include spaces
Comment	Descriptive label for the trigger
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined
Owner	Name of trigger owner. You choose an owner from a list of users. A trigger can only have one owner at a time. This is normally the trigger creator
Table	Table attached to trigger
Generate	Indicates to generate trigger
User-defined	[Read-only] Indicates that the trigger definition is modified by user. You modify a trigger definition when you change the trigger template script in the Definition tab of the trigger

Trigger property sheet Definition tab

The Definition tab allows you to enter code for the trigger. It contains the following properties:

Property	Description
Template	Template on which the trigger is based
User-defined	This button is automatically depressed when you modify the definition of a trigger, that is to say the trigger template code. You can click the User-Defined button to restore the trigger definition as it is provided in the template
Time	Time attribute of the template. The content of the list depends on the DBMS
Event	Event attribute of the template. The content of the list depends on the DBMS
Order	Firing order of trigger

Time and event

You can customize the trigger time and event, and you can also handle multiple events on a single trigger.

The time and event lists are filled with the values defined in the trigger template and the values defined in the Time and Event entries in the Trigger category of the DBMS editor. For example, if you define Select as an event in the DBMS definition file, it will appear in the list of events.

For more information about triggers with multiple event, see "Defining triggers with multiple events" on page 239.

For information about the Definition tab toolbar, see "SQL Code Definition Toolbars" on page 270.

Trigger property sheet Preview tab

The Preview tab displays the SQL code associated with the trigger. For more information, see "Previewing SQL statements" in the Working with Data Models chapter.

Trigger property sheet Template Items tab

The Template Items tab lists the trigger template items available for use in the trigger definition. For more information, see "Trigger Template Items (PDM)" on page 248.

Rebuilding Triggers

You rebuild triggers in order to ensure that:

- They are attached to all tables joined by a reference where referential integrity is implemented by a trigger

- Any changes to a trigger template or trigger template item are cascaded down to all associated triggers

PowerDesigner can rebuild triggers either:

- Automatically, whenever a relevant change is made, if you have enabled the Automatically rebuild triggers model option
- Manually, when you select Tools ➤ Rebuild Objects ➤ Rebuild Triggers

The Rebuild Triggers function creates new triggers based on template items that correspond to trigger referential integrity defined for references and sequence implementation for columns.

The Rebuild Triggers function does not use pre-defined template items which are not applicable to trigger referential integrity or sequence implementation.

☞ For information about rebuilding dependencies between triggers and other objects, see "Tracing Trigger and Procedure Dependencies" on page 257.

Rebuilding triggers automatically

PowerDesigner can automatically rebuild your triggers whenever a relevant change is made.

❖ To enable the automatically rebuild triggers option

1. Select Tools ➤ Model Options to open the Model Options window.
2. In the Category pane, click on Trigger under the Model Settings node to go to the Trigger model options
3. Select the Automatically rebuild triggers checkbox, and click OK.

PowerDesigner will rebuild all triggers and will, from now on, rebuild triggers whenever you make a relevant change in the model.

Rebuilding triggers manually

You can rebuild triggers manually at any time.

❖ To rebuild triggers manually

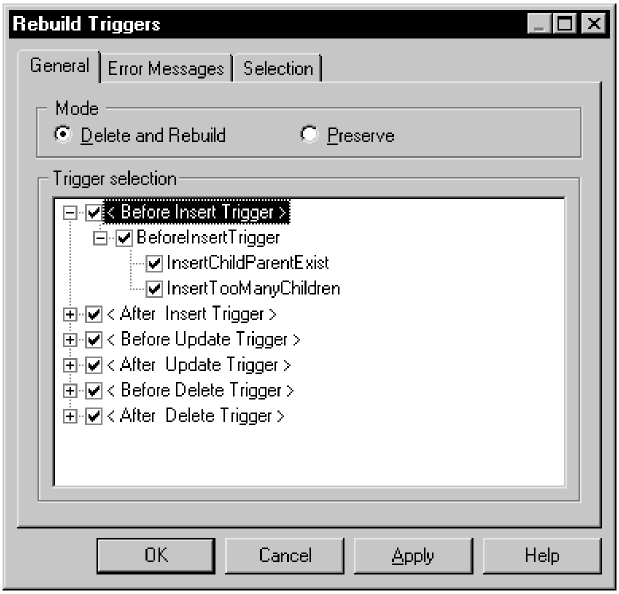
1. Select Tools ➤ Rebuild Objects ➤ Rebuild Triggers to open the Trigger Rebuild window.
2. Specify a rebuild mode. You can choose between the following options:

- Delete and Rebuild – all triggers attached to templates are deleted and rebuilt, including those to which you have made modifications
- Preserve – only those triggers attached to templates that have not been modified are deleted and rebuilt. Any triggers that you have modified are preserved.

3. The Trigger selection box shows an expandable tree view of trigger types. There are three levels in this tree:

   - All trigger types supported by the current DBMS
   - All trigger templates corresponding to the trigger types
   - All template items defined for each trigger template

   For example, in the list below, the two template items `InsertChildParentExist` and `InsertTooManyChildren` are used in the `BeforeInsertTrigger` template that is, in turn, used in all triggers with a time of `Before` and an event type of `Insert`:



Mode: ● Delete and Rebuild ○ Preserve

Trigger selection:

```
⊟ ☑ < Before Insert Trigger >
    ⊟ ☑ BeforeInsertTrigger
          ☑ InsertChildParentExist
          ☑ InsertTooManyChildren
⊞ ☑ < After  Insert Trigger >
⊞ ☑ < Before Update Trigger >
⊞ ☑ < After  Update Trigger >
⊞ ☑ < Before Delete Trigger >
⊞ ☑ < After  Delete Trigger >
```

   You can select which trigger types, trigger templates, and template items will be rebuilt by expanding the appropriate nodes and selecting or clearing the check boxes as required.

4. [optional] Click the Error Messages tab to define the types of error messages to generate. For more information about this tab, see the "Creating and generating user-defined error messages" on page 278 section.

5. [optional] Click the Selection tab. to specify which tables to rebuild the triggers for.

6. Click OK to begin the rebuild process.

   Progress is shown in the Output window. You can view the triggers that have been created from the Triggers tab of the table property sheet, or from the List of Triggers.

Which Triggers are Rebuilt?

Whether you use the automatic model option or rebuild triggers manually, all triggers that are based on a trigger template are rebuilt. These include:

- The default PowerDesigner triggers that are used to maintain referential integrity between tables

- Any other triggers that you have created via a user-defined trigger template

Triggers are rebuilt when a change in the DBMS family is made
If you change the target DBMS family, for example from Sybase to Oracle or IBM DB2, triggers are automatically rebuilt.

User-defined triggers

When a trigger based on a trigger template is modified, it becomes user-defined. The Rebuild Triggers function does not create a trigger corresponding to a trigger template attached to a user-defined trigger if you select the “Preserve” mode.

If you wish to recreate a trigger based on the same trigger template you can:

- Click the User-defined button in the Definition tab of the trigger property sheet to reset your edits. This removes the changes performed on the trigger and reapplies the trigger template

- Select <None> in the trigger template list in the Definition tab of the trigger property sheet. This detaches the trigger template from the user-defined trigger. The next time you will rebuild triggers, a new trigger based on the previously mentioned trigger template will be created

For example, Trigger1 is based on template TrigTpl1. You modify Trigger1 so PowerDesigner turns it into a user-defined trigger. If you want to recreate a trigger based on TrigTpl1 you can click the User-defined button in the Definition tab of the Trigger1 property sheet: you restore the trigger template code, but you lose the modifications on the trigger. You can also keep the changes performed on the trigger by selecting <None> in the trigger template list in the Definition tab of the trigger property sheet. The trigger template is detached from the trigger and a new trigger based on this trigger template will be created the next time you rebuild triggers.

Modifying Triggers

PowerDesigner provides various methods for editing a trigger. You can:

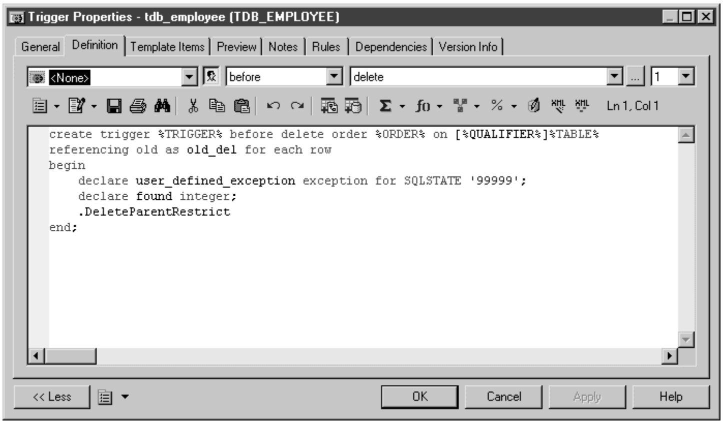
- Edit the code directly in the Definition tab of its property sheet.
- Attach a predefined trigger template or create and attach your own reusable trigger templates
- Insert predefined trigger template item code or create your own reusable trigger template items

> **Always work on a copy of your DBMS definition file**
> If you modify the definition of a DBMS trigger template or template item, you are modifying the DBMS definition file. We recommend that you only ever work on a copy of the original DBMS definition file.

❖ To modify a trigger

1. Open the trigger property sheet in one of the following ways:
   - Open the relevant table property sheet click the Triggers tab, select the trigger from the list, and then click the Properties tool
   - Select Model ➤ Triggers ➤ Triggers to open the List of Triggers, select the trigger from the list, and then click the Properties tool
   - In the Browser, find the entry for the trigger, right-click it, and select Properties from the contextual menu

2. Click the Definition tab to display the trigger code.



```
create trigger %TRIGGER% before delete order %ORDER% on [%QUALIFIER%]%TABLE%
referencing old as old_del for each row
begin
    declare user_defined_exception exception for SQLSTATE '99999';
    declare found integer;
    .DeleteParentRestrict
end;
```

3. Enter the trigger definition code. You can attach a trigger template, add template items, use PDM variables and macros and various other tools available from the toolbar (see "SQL Code Definition Toolbars" on page 270).

   The trigger will be marked as user-defined and will be excluded from most forms of rebuilding (see "Rebuilding Triggers" on page 229), if you select the "Preserve" mode.

4. You can also modify the trigger's other properties. For a full list of the properties available, see "Trigger properties" on page 228.

5. Click OK in each of the dialog boxes.

Inserting a template item into a trigger or trigger template

Template items are inserted in a trigger or trigger template definition using a dot followed by the template item name. For example, the following script contains two template items `InsertChildParentExist` and `InsertTooManyChildren`:

```
/*  Before insert trigger "%TRIGGER%" for table
         "[%QUALIFIER%]%TABLE%"  */
create trigger %TRIGGER% before insert order %ORDER% on
         [%QUALIFIER%]%TABLE%
referencing new as new_ins for each row
begin
    declare user_defined_exception exception for SQLSTATE
         '99999';
    declare found integer;
    .InsertChildParentExist
    .InsertTooManyChildren
end
/
```

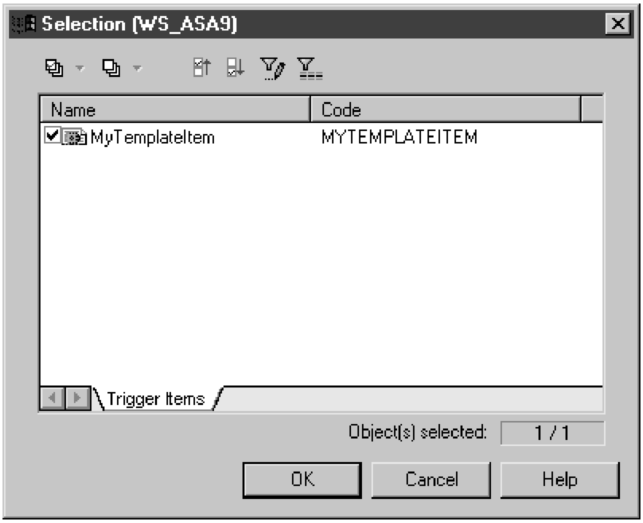
❖ To add a template item to a trigger or trigger template

1. Open the property sheet of the trigger or trigger template that you want to modify, and then click the Definition tab.

2. Click at the point in the code where you want to insert the trigger template item, and then click one of the following tools:

   - Add Trigger Item From DBMS – to open a selection box containing a list of trigger template items defined in the DBMS definition file

   - Add Trigger Item From Model – to open a selection box containing a list of trigger template items defined in the model

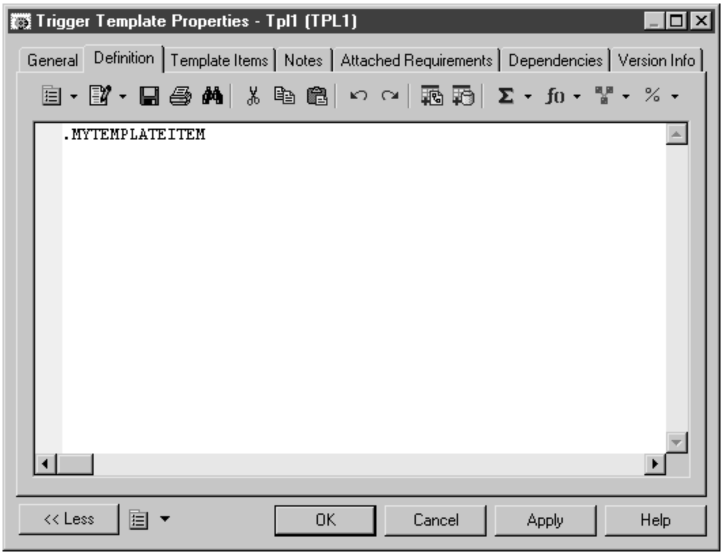


Name	Code
☑ MyTemplateItem	MYTEMPLATEITEM

Object(s) selected: 1 / 1

3. Select the item to insert and then click OK to return to the definition tab.

   The trigger template item will be inserted in your code. It will also appear in the list on the Template Items tab.



```
.MYTEMPLATEITEM
```

Declaring a template item in a trigger definition

Certain DBMS require that a cursor and variables are declared for each template item before the template item name is used in the script. This can be a statement that calls a corresponding procedure. You can use the following format to declare a template item: `Decl` *template item name.*

For example, the trigger definition for Oracle 8 contains the `.DeclInsertChildParentExist` statement which declares the following `.InsertChildParentExist` template item:

```
--  Before insert trigger "[%QUALIFIER%]%TRIGGER%" for table
         "[%QUALIFIER%]%TABLE%"
create trigger [%QUALIFIER%]%TRIGGER% before insert
on [%QUALIFIER%]%TABLE% for each row
declare
    integrity_error  exception;
    errno            integer;
    errmsg           char(200);
    dummy            integer;
    found            boolean;
    .DeclInsertChildParentExist
begin
    .InsertChildParentExist
--  Errors handling
exception
    when integrity_error then
       raise_application_error(errno, errmsg);
end;
/
```

In a generated trigger script, `.DeclInsertChildExist` corresponds to the following definition:

```
.FOREACH_PARENT()
--  Declaration of InsertChildParentExist constraint for the
         parent "[%PQUALIFIER%]%PARENT%"
.DEFINE "CURSOR" "cpk%REFNO%_%.25L:TABLE%"
cursor %CURSOR%(.JOIN("var_%.L26:FK% %.L:COLTYPE%", "", ",", ")
         is")
   select 1
   from   [%PQUALIFIER%]%PARENT%
   where  .JOIN("%PK% = var_%.L26:FK%", "and   ")
    and   .JOIN("var_%.L26:FK% is not null", "and   ", "", ";")
.ENDFOR
```

Trigger naming conventions

The pre-defined trigger templates that ship with PowerDesigner indicate naming conventions for the trigger scripts that it generates. The naming convention consists of a prefix indicating the trigger type followed by the table code.

The default naming conventions include a variable (%L:TABLE). The name of the resulting trigger script replaces this variable with a lower-case table code. For example, a resulting trigger script may have the name ti_employee.

You can change the trigger naming convention in PowerDesigner pre-defined DBMS trigger templates from the Trigger Templates tab of the DBMS property sheet.

❖ To change trigger naming conventions in a DBMS trigger template

1. Select Database ➤ Edit Current DBMS to open the DBMS definition file in the Resource Editor, and then click the Trigger Template tab.
2. Click a trigger template in the list, and then click the Properties tool to open its property sheet .
3. Type a new trigger name in the Trigger Name text box at the bottom of the tab.

   For example, mytempl_%TABLE%
4. Click OK in each of the dialog boxes.

Calling a related procedure in a trigger template

Some target databases do not accept code within a trigger statement. For these databases, a trigger template can call a related procedure as a parameter, which is defined in a procedure template. In these cases, procedure templates are listed in the list of trigger templates.

Example

Informix does not accept code in trigger templates. The template `InsertTrigger` calls the procedure in the form of the variable `%PROC%`, as follows:

```
-- Insert trigger "[%QUALIFIER%]%TRIGGER%" for table
         "[%QUALIFIER%]%TABLE%"
create trigger [%QUALIFIER%]%TRIGGER% insert on
         [%QUALIFIER%]%TABLE%
referencing new as new_ins
   for each row (execute procedure %PROC%(.FKCOLN("new_
         ins.%COLUMN%", "", ",", "));")
/
```

The template InsertProc defines the procedure, as follows:

```
--  Insert procedure "%PROC%" for table "[%QUALIFIER%]%TABLE%"
create procedure %PROC%(.FKCOLN("new_%.14L:COLUMN% %COLTYPE%",
         "", ",", ")")
    .DeclInsertChildParentExist
    .DeclInsertTooManyChildren
    define  errno       integer;
    define  errmsg      char(255);
    define  numrows     integer;

    .InsertChildParentExist
    .InsertTooManyChildren

end procedure;
/
```

Multiple triggers

Some DBMSs allow you to have multiple triggers of the same type (time and event) defined for any given table. Triggers of the same type are triggers that are invoked for the same insert, update, or delete event.

Example

A company is considering large numbers of candidates for new positions in various posts. You want to ensure that all new employees will have a salary that is within the range of others working in the same field, and less than his or her prospective manager.

On an EMPLOYEE table, you create two *BeforeInsert triggers* , tibTestSalry1_EMPLOYEE to verify that a proposed salary falls within the correct range, and tibTestSalry2_EMPLOYEE to verify that the proposed salary is less than that of the prospective manager.

```
create trigger tibTestSalry1 before insert order 1 on EMPLOYEE
referencing new as new_ins for each row
begin

   [Trigger code]

end

create trigger tibTestSalry2 before insert order 2 on EMPLOYEE
begin

   [Trigger code]

end
```

For a specified table, you can indicate the order that a trigger executes, or fires, within a group of triggers of the same type.

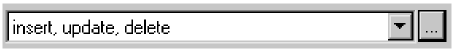
❖ To indicate trigger order

1. Click the Definition tab from the trigger property sheet.
2. Select a number from the Order dropdown list box to indicates the position in the firing order that the trigger fires.
3. Click OK in each of the dialog boxes.

Defining triggers with multiple events

Some DBMSs support multiple events on triggers. If such is the case, the Ellipsis button to the right of the Event box on the trigger definition tab is available.

You can click the Ellipsis button to open the Multiple Events Selection box. If you select several events and click OK, the different events will be displayed in the Event box, separated by the appropriate delimiter.



insert, update, delete

Trigger Templates (PDM)

PowerDesigner trigger templates and template items allow you to create triggers in a modular reusable fashion.

PowerDesigner uses pre-defined trigger templates to create triggers to implement referential integrity between tables. Trigger templates exist for each type of trigger supported by the DBMS. Each template specifies:

- a **Time** – relevant to an event. One of:
  - After
  - Before
- an **Event** – that can occur to a table row. One of:
  - Delete
  - Insert
  - Update
- **Code** – that performs the trigger action. The code may contain references to trigger template items, which are re-usable blocks of script.

You can create trigger templates and trigger template items in your DBMS definition file or in your model. These elements will be available as follows:

Trigger template or template item created in:	Available to
DBMS (shared definition file)	All models sharing the DBMS
DBMS (copied definition file	Current model only
Model	Current model only

☞ For more information on the DBMS share and copied definition files, see the “Generating a Database from a PDM” chapter.

Modifying DBMS template items

By modifying a DBMS template item, you modify the template item definition in the current DBMS. You should only modify a DBMS that is a copy of the DBMS shipped with PowerDesigner.

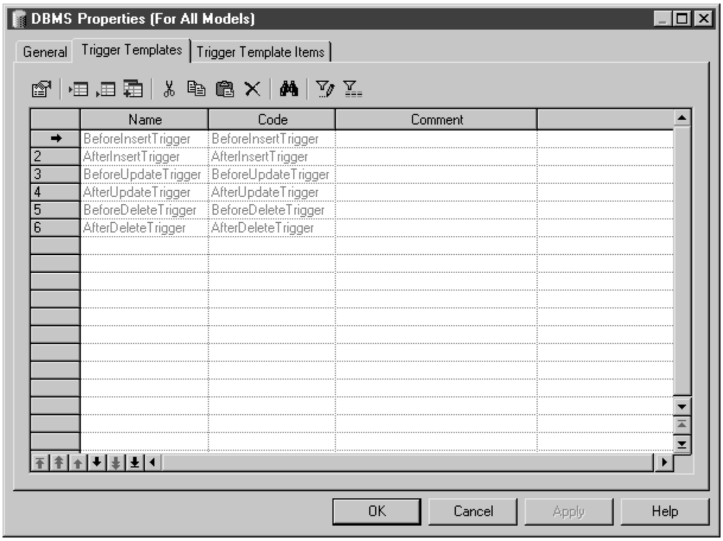
☞ For more information on working with DBMS resource files, see the DBMS Resource File Reference chapter of the Customizing and Extending PowerDesigner manual.

Creating a trigger template

You can create a new trigger template in your DBMS definition file or as part of your model. You can begin by copying an existing template or write one from scratch.two types of trigger templates:

❖ To create a trigger template

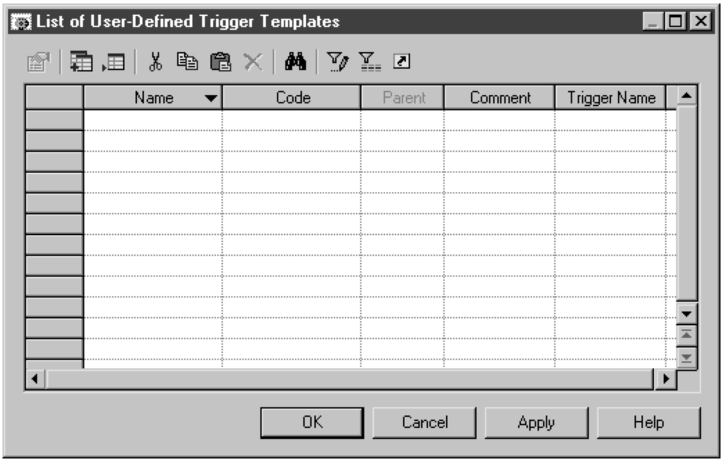
1. To create a DBMS trigger template: select Database ➤ Edit Current DBMS to open the DBMS definition file in the resource editor, and then click the Trigger Templates tab:



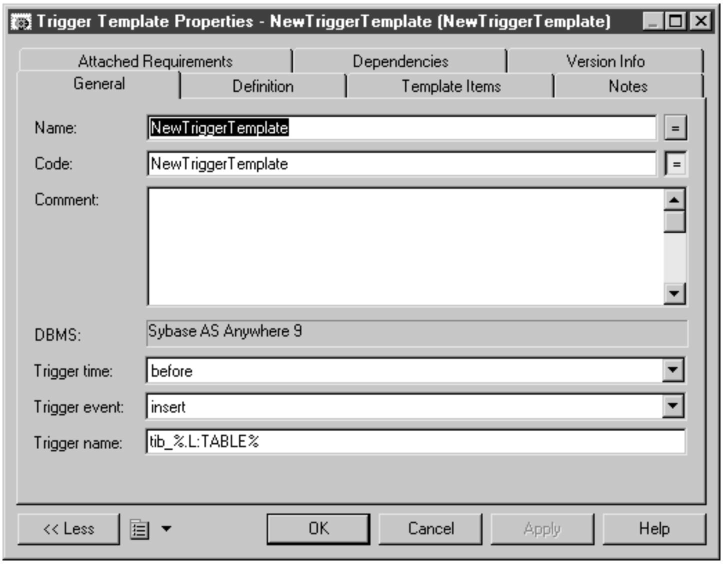
	Name	Code	Comment
➜	BeforeInsertTrigger	BeforeInsertTrigger	
2	AfterInsertTrigger	AfterInsertTrigger	
3	BeforeUpdateTrigger	BeforeUpdateTrigger	
4	AfterUpdateTrigger	AfterUpdateTrigger	
5	BeforeDeleteTrigger	BeforeDeleteTrigger	
6	AfterDeleteTrigger	AfterDeleteTrigger	

or

To create a model trigger template: select Model ➤ Triggers ➤ Trigger Templates to open the List of User-Defined Trigger Templates:



2. Click on one of the following tools:
   - **Create from DBMS trigger template** – opens a selection box listing all the trigger templates available in the current DBMS. Select a check box for the type of trigger template that you want to use as the basis for your new template and click OK to return to the trigger template list. The duplicate DBMS template has been added to the list.
   - **Add a Row** – adds a new blank template to the list.
3. Type a new name and code for the new template and click Apply to commit its creation.
4. Click the Properties tool to open the property sheet of the new trigger template:



Name: NewTriggerTemplate

Code: NewTriggerTemplate

Comment:

DBMS: Sybase AS Anywhere 9

Trigger time: before

Trigger event: insert

Trigger name: tib_%.L:TABLE%

5. Click the Definition tab and enter or modify the definition code. You can add trigger template items, use PDM variables and macros and various other tools available from the toolbar. For more information, see "SQL Code Definition Toolbars" on page 270.

6. You can also modify other of the trigger template's properties. For a full list of the properties available, see "Trigger template properties" on page 245.

7. Click OK in each of the dialog boxes.

   If you have created DBMS trigger template, a confirmation box will appear, asking if you want to save the changes to the DBMS.

   Click Yes to confirm the template creation.

PowerDesigner pre-defined trigger templates

The pre-defined DBMS templates provided with PowerDesigner control referential integrity constraints for insert, update, and delete events. Depending on the current DBMS, there is a before and after event template for each trigger type.

You can modify the code of these pre-defined trigger templates, but they cannot be deleted or renamed.

The following templates types exist, but may vary by DBMS:

Insert templates

Template type	Generates trigger/procedure executing. . .
InsertTrigger	With insert
BeforeInsertTrigger	Before insert
AfterInsertTrigger	After insert
InsertProc	When called by InsertTrigger
BeforeInsertProc	When called by BeforeInsertTrigger
AfterInsertProc	When called by AfterInsertTrigger

Update templates

Template type	Generates trigger/procedure executing. . .
UpdateTrigger	With update
BeforeUpdateTrigger	Before update
AfterUpdateTrigger	After update
UpdateProc	When called by UpdateTrigger
BeforeUpdateProc	When called by BeforeUpdateTrigger
AfterUpdateProc	When called by AfterUpdateTrigger

Delete templates

Template type	Generates trigger/procedure executing. . .
DeleteTrigger	With delete
BeforeDeleteTrigger	Before delete
AfterDeleteTrigger	After delete
DeleteProc	When called by DeleteTrigger
BeforeDeleteProc	When called by BeforeDeleteTrigger
AfterDeleteProc	When called by AfterDeleteTrigger

Modifying a trigger template

You can modify both your templates and those that are provided with PowerDesigner.

❖ To modify a trigger template

1. Open the trigger template property sheet in one of the following ways:
2. To modify a DBMS trigger template: select Database ➤ Edit Current DBMS to open the DBMS definition file in the resource editor, and then click the Trigger Templates tab.
3. To modify a model trigger template: select Model ➤ Triggers ➤ Trigger Templates to open the List of User-Defined Trigger Templates.
4. Click a trigger template in the list, and then click the Properties tool to open its property sheet.
5. Click the Definition tab and modify the trigger definition code. You can add trigger template items, use PDM variables and macros and various other tools available from the toolbar. For more information, see "SQL Code Definition Toolbars" on page 270.
6. You can also modify other of the trigger template's properties. For a full list of the properties available, see "Trigger template properties" on page 245.
7. Click OK in each of the dialog boxes.

   If you have created DBMS trigger template, a confirmation box will appear, asking if you want to save the changes to the DBMS.

   Click Yes to confirm the template creation.

Trigger template properties

You can modify a trigger templates properties from its property sheet. The following sections detail the property sheet tabs that contain the properties most commonly entered for trigger templates.

The General tab contains the following properties:

Property	Description
Name	The name of the item which should be clear and meaningful, and should convey the item's purpose to non-technical users
Code	The technical name of the item used for generating code or scripts, which may be abbreviated, and should not generally include spaces
Comment	Descriptive label for the trigger template

Property	Description
DBMS	Current DBMS
Trigger time	Time attribute of the trigger template. The list displays the values defined in the trigger templates and template items of the current DBMS
Trigger event	Event attribute of the trigger template. The list displays the values defined in the trigger templates and template items of the current DBMS
Trigger name	Name of trigger associated with template
Applies to table triggers or view triggers	For those DBMS that support view triggers, it allows you to define if the trigger template applies to table or view triggers

Trigger template property sheet Definition tab

The Definition tab contains a field for entering its definition code. For information about editing this code, see "Modifying Triggers" on page 233.

Trigger template property sheet Template Items tab

The Template Items tab list the template items that are defined in the trigger template and that will be generated when a trigger is generated from the template.

A template item that is deleted from the Template Items tab is not deleted from the trigger template definition. You can therefore limit the template items available for generation by removing template items from the Template Item tab, without having to remove them from the trigger template definition.

Rebuild triggers

When you use Rebuild Triggers to automatically create triggers for selected tables, the template items that are listed on this tab are those available for generation. Whether they are generated or not depends on the following:

- ♦ Template items are generated in a trigger if they match the trigger implemented referential integrity defined for a reference attached to the table
- ♦ Template items are generated in a trigger if they are user-defined, regardless of trigger referential integrity constraints

Adding template items to trigger template definition

You can add any template item from the model or DBMS to the Trigger template definition by clicking an Add Trigger Item tool from the Definition tab of the trigger template property sheet, and selecting a trigger item. It is automatically added to the Template Items tab.

Trigger Template Items (PDM)

Trigger template items are named reusable blocks of script that can be inserted into trigger templates or triggers.

In a generated trigger script, a template item calls a macro that implements a trigger referential integrity constraint or does any other updating work on tables in the database.

Example

A trigger template for Sybase Adaptive Server Anywhere 6 contains the `.InsertChildParentExist` template item, which corresponds to the following definition:

```
.FOREACH_PARENT()
/*  Parent "[%PQUALIFIER%]%PARENT%" must exist when inserting a
         child in "[%CQUALIFIER%]%CHILD%"  */
if (.JOIN("new_ins.%FK% is not null", "", " and", ") then")
begin
   set found = 0;
   select 1
    into  found
    from  dummy
   where  exists (select 1
                   from  [%PQUALIFIER%]%PARENT%
                  where  .JOIN("%PK% = new_ins.%FK%", "and   ",
         "", ");")
   if found <> 1 then
      message 'Error: Trigger(%TRIGGER%) of table
         [%QUALIFIER%]%TABLE%';
      message '        Parent code must exist when inserting a
         child!';
      signal user_defined_exception;
   end if;
end
end if;
.ENDFOR
```

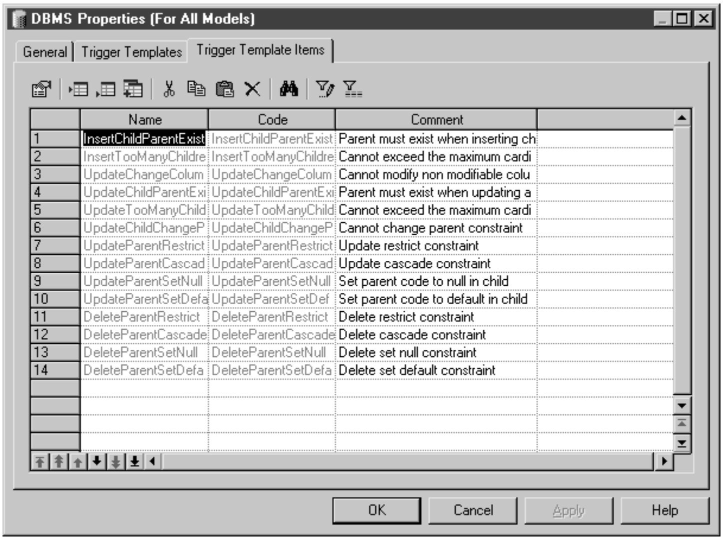
Creating a trigger template item

You usually create a template item when an existing template item is not suitable, or to create a repeatable block of code to do updating work on tables in the database.

You can create a new trigger template in your DBMS definition file or as part of your model. You can begin by copying an existing template or write one from scratch.

❖ To create a trigger template item

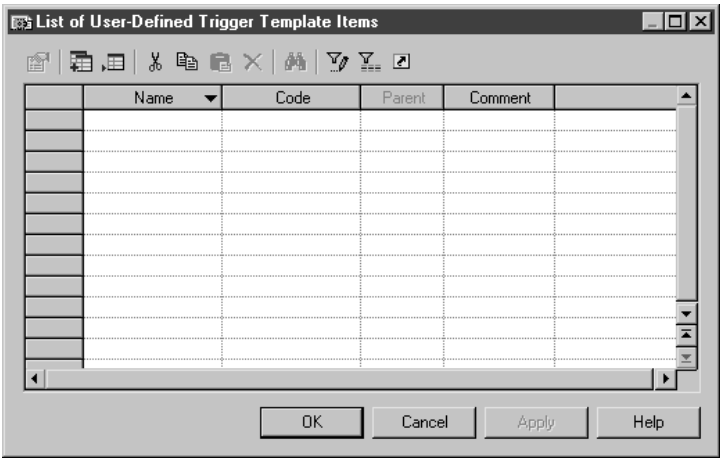
1. To create a DBMS trigger template item: select Database ➤ Edit Current DBMS to open the DBMS definition file in the resource editor, and then click the Trigger Template Items tab:



	Name	Code	Comment
1	InsertChildParentExist	InsertChildParentExist	Parent must exist when inserting ch
2	InsertTooManyChildre	InsertTooManyChildre	Cannot exceed the maximum cardi
3	UpdateChangeColum	UpdateChangeColum	Cannot modify non modifiable colu
4	UpdateChildParentExi	UpdateChildParentExi	Parent must exist when updating a
5	UpdateTooManyChild	UpdateTooManyChild	Cannot exceed the maximum cardi
6	UpdateChildChangeP	UpdateChildChangeP	Cannot change parent constraint
7	UpdateParentRestrict	UpdateParentRestrict	Update restrict constraint
8	UpdateParentCascad	UpdateParentCascad	Update cascade constraint
9	UpdateParentSetNull	UpdateParentSetNull	Set parent code to null in child
10	UpdateParentSetDefa	UpdateParentSetDef	Set parent code to default in child
11	DeleteParentRestrict	DeleteParentRestrict	Delete restrict constraint
12	DeleteParentCascade	DeleteParentCascade	Delete cascade constraint
13	DeleteParentSetNull	DeleteParentSetNull	Delete set null constraint
14	DeleteParentSetDefa	DeleteParentSetDefa	Delete set default constraint

or

To create a model trigger template item: select Model ➤ Triggers ➤ Trigger Template Items to open the List of User-Defined Trigger Template Items:



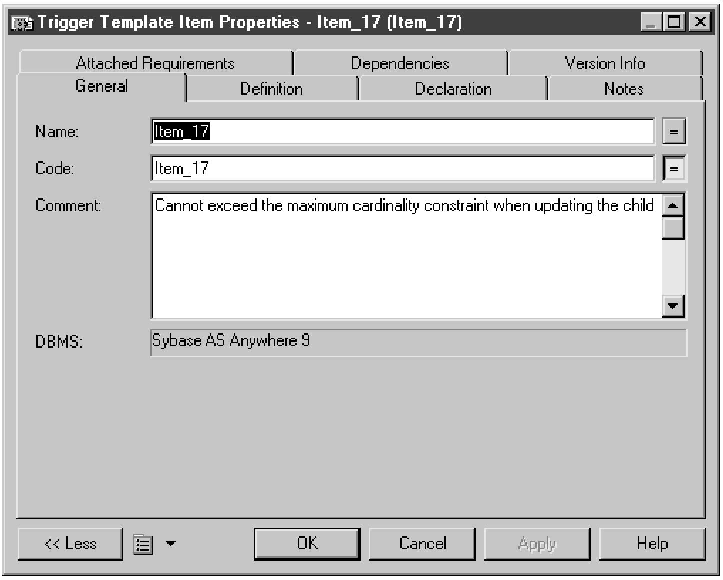
2. Click on one of the following tools:

   ♦ **Create from DBMS trigger item** – opens a selection box listing all the trigger template items available in the current DBMS. Select a check box for the type of item that you want to use as the basis for your new item and click OK to return to the trigger template item list. The duplicate DBMS template item has been added to the list.

- **Add a Row** – adds a new blank template item to the list.

3. Type a new name and code for the new template item and click Apply to commit its creation.

4. Click the Properties tool to open the property sheet of the new template item:



Trigger Template Item Properties - Item_17 (Item_17)

Name: Item_17

Code: Item_17

Comment: Cannot exceed the maximum cardinality constraint when updating the child

DBMS: Sybase AS Anywhere 9

5. Click the Definition tab and enter or modify the definition code. You can use PDM variables and macros and various other tools available from the toolbar. For more information, see "SQL Code Definition Toolbars" on page 270.

6. You can also modify other of the trigger template item's properties. For a full list of the properties available, see "Trigger template properties" on page 245.

7. Click OK in each of the dialog boxes.

   If you have created DBMS trigger template item, a confirmation box will appear, asking if you want to save the changes to the DBMS.

   Click Yes to confirm the template creation.

PowerDesigner pre-defined trigger template items

PowerDesigner ships pre-defined template items for each pre-defined trigger template defined in the supported DBMS. The Rebuild Triggers function uses both pre-defined and user-defined trigger templates to automatically create triggers for selected tables.

In the pre-defined trigger templates, each pre-defined template item corresponds to a referential integrity constraint. Although a pre-defined template item is defined in a trigger template, it is only generated in a trigger script if it implements the trigger referential integrity defined for a reference.

Template items have the following generation conditions:

Template item is listed in. . .	Template item is. . .
Template Items tab of trigger property sheet	Available for generation
Template Items tab of trigger template property sheet	Generated

You can modify the code for these pre-defined template items, but they cannot be deleted or renamed.

The PowerDesigner pre-defined template items that are available depend on the current DBMS.

Insert constraints

The template items below implement referential integrity in insert trigger templates.

Template item	Integrity constraint	Description
`DeclInsertChildParentExist` `InsertChildParentExist`	Mandatory parent	Parent must exist when inserting a child
`DeclInsertTooManyChildren` `InsertTooManyChildren`	Cannot exceed maximum cardinality constraint	Cannot insert a child if maximum cardinality has been reached
`DeclInsertSequenceColumn` `InsertSequenceColumn`	Select value in sequence list for column	Select a value for the column from a list of sequences

Update constraints

The template items below implement referential integrity in update trigger templates.

Template item	Integrity constraint	Description
DeclUpdateChildParentExist UpdateChildParentExist	Mandatory parent	Parent must exist when updating a child
DeclUpdateChildChangeParent UpdateChildChangeParent	Change parent not allowed	Cannot modify parent code in child
DeclUpdateParentRestrict UpdateParentRestrict	Restrict on update	Cannot modify parent if child exists
DeclUpdateParentCascade UpdateParentCascade	Cascade on update	Modify parent code in all children
DeclUpdateChangeColumn UpdateChangeColumn	Non-modifiable column	Cannot modify column
DeclUpdateParentSetNull UpdateParentSetNull	Set null on update	Set parent code to null in all children
DeclUpdateParentSetDefault UpdateParentSetDefault	Set default on update	Set parent code to default in all children
DeclUpdateTooManyChildren UpdateTooManyChildren	Cannot exceed maximum cardinality constraint	Cannot update a child if maximum cardinality has been reached

Delete constraints

The template items below implement referential integrity in delete trigger templates.

Template item	Integrity constraint	Description
DeclDeleteParentRestrict DeleteParentRestrict	Restrict on delete	Cannot delete parent if child exists

Template item	Integrity constraint	Description
`DeclDeleteParentCascade` `DeleteParentCascade`	Cascade on delete	Delete parent code in all children
`DeclDeleteParentSetNull` `DeleteParentSetNull`	Set null on delete	Delete in parent sets child to null
`DeclDeleteParentSetDefault` `DeleteParentSetDefault`	Set default on delete	Delete in parent sets child to default

Constraint messages

You can insert the following template items in any trigger template. They generate error messages that indicate the violation of an integrity constraint.

Template item	Description
UseErrorMsgText	Error handling without a message table
UseErrorMsgTable	Error handling with a message table

Modifying a trigger template item

You can modify both your template items and those that are provided with PowerDesigner.

❖ To modify a trigger template item

1. Open the trigger template item property sheet in one of the following ways:

2. To modify a DBMS trigger template item: select Database ➤ Edit Current DBMS to open the DBMS definition file in the resource editor, and then click the Trigger Template Items tab.

3. To modify a model trigger template item: select Model ➤ Triggers ➤ Trigger Templates Items to open the List of User-Defined Trigger Template Items.

4. Click a trigger template item in the list, and then click the Properties tool to open its property sheet.

5. Click the Definition tab and modify the trigger definition code. You can use PDM variables and macros and various other tools available from the

toolbar. For more information, see "SQL Code Definition Toolbars" on page 270.

6. You can also modify other of the trigger template item's properties. For a full list of the properties available, see "Trigger template properties" on page 245.

7. Click OK in each of the dialog boxes.

   If you have created DBMS trigger template item, a confirmation box will appear, asking if you want to save the changes to the DBMS.

   Click Yes to confirm the template creation.

Trigger template item properties

You can modify an object's properties from its property sheet. To open a trigger template item property sheet, access it via the List of User-Defined Trigger Template Items List or the DBMS Properties window. The following sections detail the property sheet tabs that contain the properties most commonly entered for trigger template items.

The General tab contains the following properties:

Property	Description
Name	The name of the item which should be clear and meaningful, and should convey the item's purpose to non-technical users
Code	The technical name of the item used for generating code or scripts, which may be abbreviated, and should not generally include spaces
Comment	Descriptive label for the template item
DBMS	Current DBMS

The following tabs are also available:

- Definition - allows you to enter the SQL code for the template item. For information about the tools available, see "SQL Code Definition Toolbars" on page 270.

- Declaration - contains the declaration for the template item in trigger scripts. For information about the tools available, see "SQL Code Definition Toolbars" on page 270.

Stored Procedures and Functions (PDM)

You can define stored procedures and functions for any DBMS that supports them.

A stored procedure is a precompiled collection of SQL statements stored under a name and processed as a unit. Stored procedures are stored within a database; can be executed with one call from an application; and allow user-declared variables, conditional execution, and other programming features.

The use of stored procedures can be helpful in controlling access to data (end-users may enter or change data but do not write procedures), preserving data integrity (information is entered in a consistent manner), and improving productivity (statements in a stored procedure only need to be written one time).

A user-defined function is a form of procedure that returns a value to the calling environment for use in queries and other SQL statements.

Creating a stored procedure or function

You can create a stored procedure or function in any of the following ways:

- Use the Procedure tool in the diagram Palette
- Open the Procedures tab in the property sheet of a table, and click the Add a Row tool
- Select Model ➤ Procedures to access the List of Procedures, and click the Add a Row tool
- Right-click the model or package in the Browser, and select New ➤ Procedure

☞ For general information about creating objects, see the Objects chapter in the *Core Features Guide* .

You can create a procedure based on one of the PowerDesigner templates or on a template of your own.

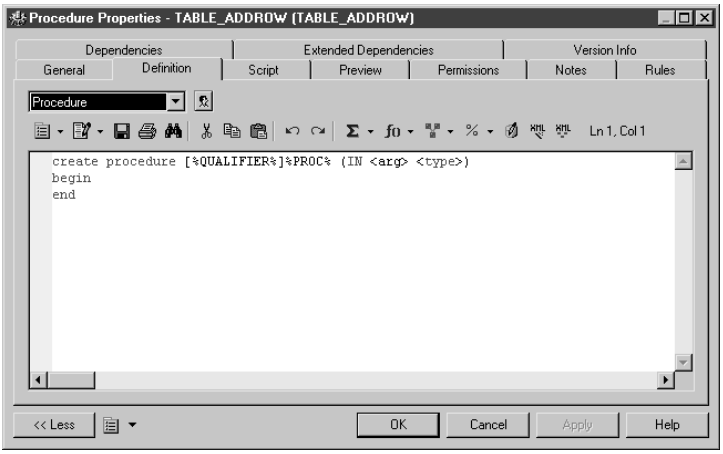
❖ To create a procedure

1. Double-click a table symbol to open its property sheet, and then click the Procedures tab.

2. Click the Add a Row tool to create a new procedure, and type a name and code.

3. Click Apply to commit the creation of the new procedure, and then click the Properties tool to open its property sheet.

4. Click the Definition tab:



```
create procedure [%QUALIFIER%]%PROC% (IN <arg> <type>)
begin
end
```

5. [optional] Select a procedure template from the Template list (see “Procedure Templates (PDM)” on page 266).

6. Modify the procedure definition code. You can use PDM variables and macros and various other tools available from the toolbar (see “SQL Code Definition Toolbars” on page 270).

7. You can also modify the procedure’s other properties. For a full list of the properties available, see “Procedure properties” on page 256.

8. Click OK in each of the dialog boxes.

Editing procedures in Eclipse

When using the PowerDesigner Eclipse plug-in, you can right-click a procedure in the Browser or diagram and select Edit in SQL Editor from the contextual menu to open it in the Eclipse SQL Editor. You can optionally connect to your database in order to obtain auto-completion for table names. The procedure definition is added to the Generated SQL Files list in the Workspace Navigator.

Procedure properties

You can modify an object’s properties from its property sheet. To open a procedure property sheet, double-click its diagram symbol or its Browser entry in the Procedures folder. The following sections detail the property sheet tabs that contain the properties most commonly entered for procedures.

The General tab contains the following properties:

Property	Description
Name	Specifies the name of the item which should be clear and meaningful, and should convey the item's purpose to non-technical users.
Code	Specifies the technical name of the item used for generating code or scripts, which may be abbreviated, and should not generally include spaces.
Comment	Specifies a descriptive label for the procedure.
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined.
Owner	Specifies the name of the procedure owner.
Table	Specifies the table to which the procedure is attached. You can use the tools to the right of the list to create an object, browse the complete tree of available objects or view the properties of the currently selected object.

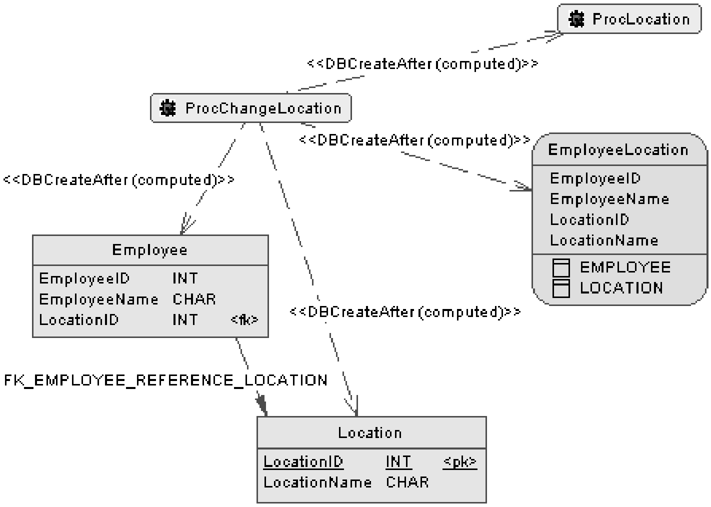
The following tabs are also available:

- Definition - allows you to enter the SQL code for the procedure. For information about the tools available, see "SQL Code Definition Toolbars" on page 270.

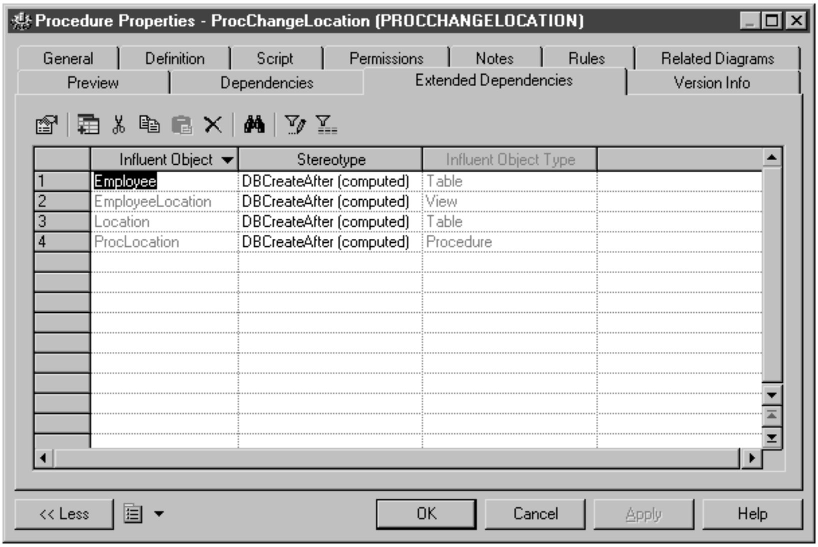
Tracing Trigger and Procedure Dependencies

When you write a trigger or procedure, PowerDesigner automatically creates dependencies to any table, view, procedure, or database package referenced in the code. These dependencies are taken into account when performing an impact analysis prior to deleting the trigger or procedure or objects on which they depend. For procedures, if the procedure has a symbol in your diagram, then any dependencies will be shown graphically by way of arrows linking the procedure to these objects.

The diagram below shows a procedure, ProcChangeLocation, which is dependent on a number of other objects:

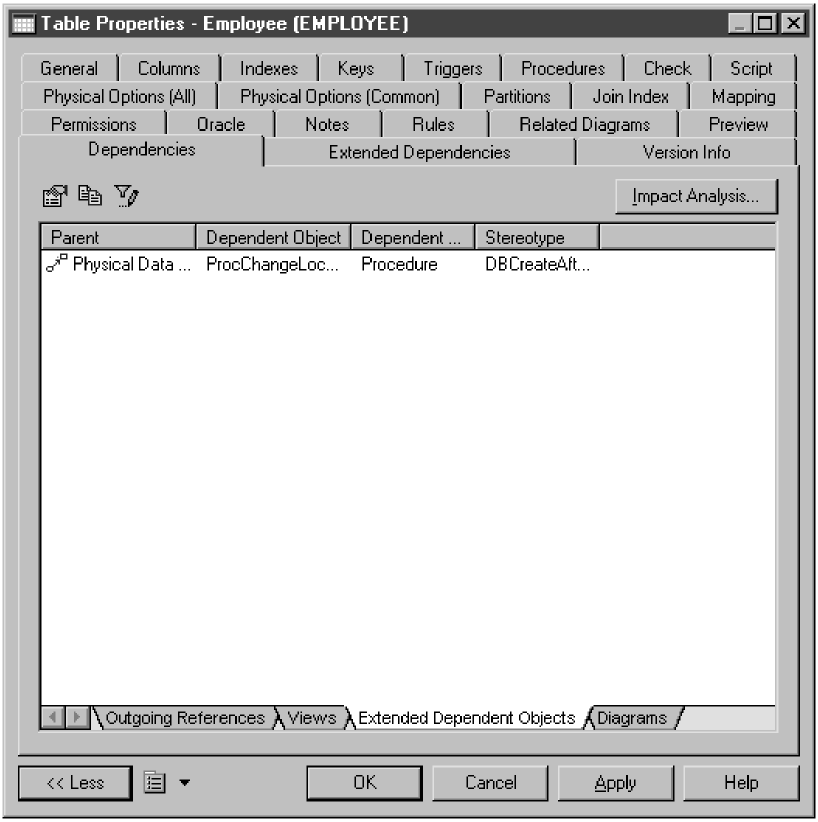


The Extended Dependencies tab of the trigger or procedure property sheet lists the objects upon which it depends. The stereotype <<DBCreateAfter (computed)>> specifies that PowerDesigner has determined that the trigger or procedure can only be created after these objects.



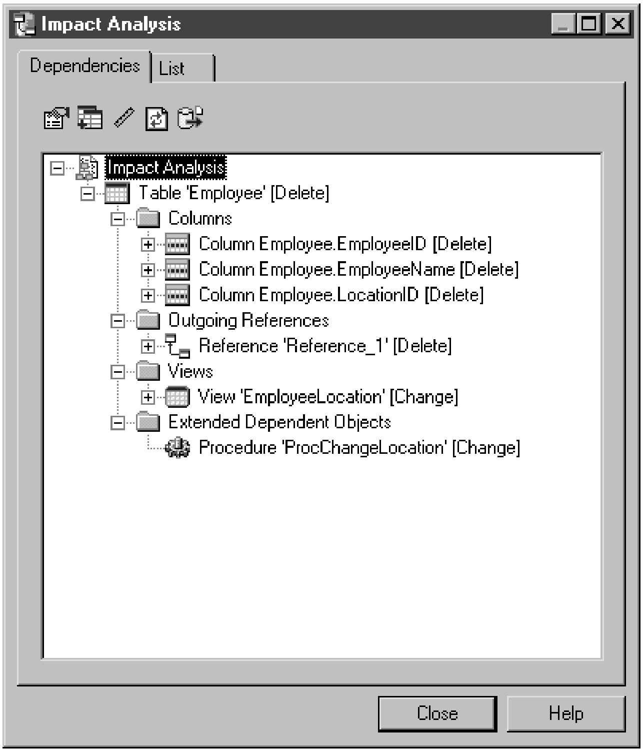
	Influent Object	Stereotype	Influent Object Type
1	Employee	DBCreateAfter (computed)	Table
2	EmployeeLocation	DBCreateAfter (computed)	View
3	Location	DBCreateAfter (computed)	Table
4	ProcLocation	DBCreateAfter (computed)	Procedure

The Dependencies tab of the Employee table property sheet shows that ProcChangeLocation is dependent upon it.



Parent	Dependent Object	Dependent ...	Stereotype
Physical Data ...	ProcChangeLoc...	Procedure	DBCreateAft...

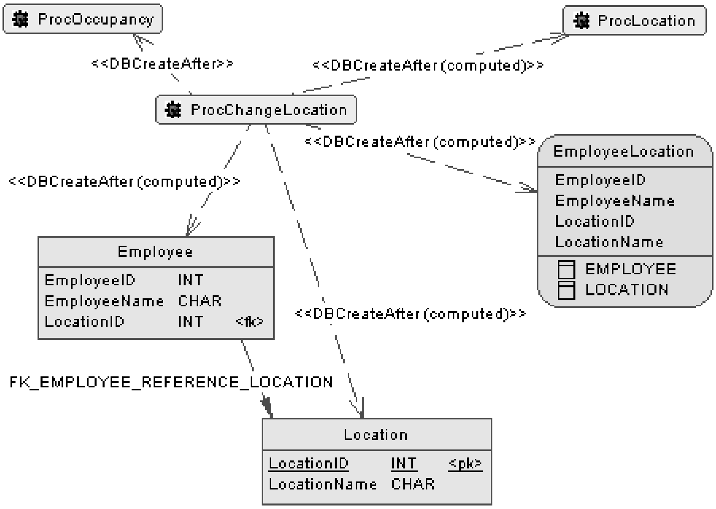
If you were to perform an impact analysis prior to deleting the Employee table, you would be warned of the dependency of the procedure upon it.



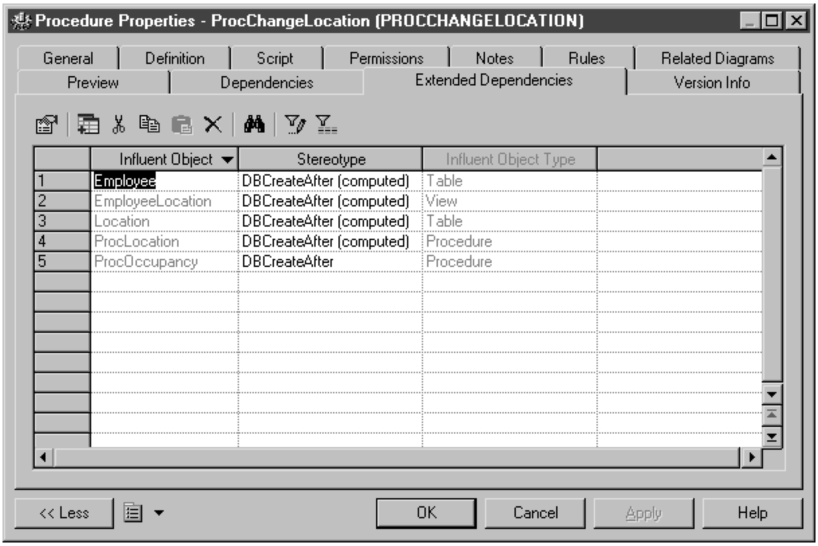
- Impact Analysis
  - Table 'Employee' [Delete]
    - Columns
      - Column Employee.EmployeeID [Delete]
      - Column Employee.EmployeeName [Delete]
      - Column Employee.LocationID [Delete]
    - Outgoing References
      - Reference 'Reference_1' [Delete]
    - Views
      - View 'EmployeeLocation' [Change]
    - Extended Dependent Objects
      - Procedure 'ProcChangeLocation' [Change]

Creating procedure dependencies manually

Since procedures have diagram symbols, you can manually add dependencies for them. To do this, use the Link/Extended Dependency tool in the palette. In the diagram below, I have added a dependency from ProcChangeLocation to a new procedure, ProcOccupancy:



Since ProcOccupancy is not directly referenced in ProcChangeLocation, I needed to manually add the stereotype <<DBCreateAfter>> to the dependency in the ProcChangeLocation property sheet Extended Dependencies tab:



	Influent Object	Stereotype	Influent Object Type
1	Employee	DBCreateAfter (computed)	Table
2	EmployeeLocation	DBCreateAfter (computed)	View
3	Location	DBCreateAfter (computed)	Table
4	ProcLocation	DBCreateAfter (computed)	Procedure
5	ProcOccupancy	DBCreateAfter	Procedure

Rebuilding trigger and procedure dependencies

Trigger and procedure dependencies are rebuilt automatically after the following actions:

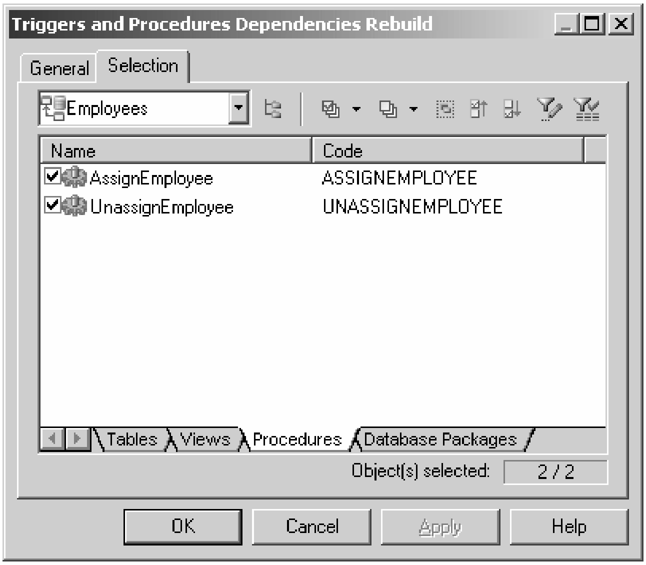
- Importing a PDM created with a former version of PowerDesigner
- Reverse engineering a database into a PDM
- Merging PDMs

You can also manually rebuild trigger and procedure dependencies at any time.

❖ To rebuild trigger and procedure dependencies

1. Select Tools ➤ Rebuild Objects ➤ Rebuild Triggers and Procedures Dependencies to open the Procedures Dependencies window.

2. Specify a rebuild mode for each of Procedures and Triggers. You can choose between the following options:
   - Delete and Rebuild – all triggers and/or procedures attached to templates are deleted and rebuilt, including those to which you have made modifications
   - Preserve – only those triggers and/or procedures attached to templates that have not been modified are deleted and rebuilt. Any triggers and/or procedures that you have modified are preserved.



Name	Code
☑ AssignEmployee	ASSIGNEMPLOYEE
☑ UnassignEmployee	UNASSIGNEMPLOYEE

Object(s) selected: 2 / 2

3. [optional] Click the Selection tab and specify the tables, views, procedures, and (for Oracle only) database packages for which you want to rebuild dependencies. By default all are selected.

4. Click OK to begin the rebuild process.

Attaching a stored procedure to a table

You can attach a stored procedure to a table when your current DBMS supports stored procedures. This feature lets you update the table or retrieve information from this table.

For example, the stored procedure TABLE_ADDROW can be attached to a table in which you need to insert rows.

Intermodel generation

When you generate an OOM from a PDM, the procedures attached to a table become operations with the <<procedure>> stereotype in the generated class. By attaching procedures to tables, you are able to define class operations in the generated OOM.

When you generate a PDM from an OOM, class operations with the <<procedure>> stereotype become stored procedures attached to the generated table. The operation body is generated as a comment in the procedure definition.

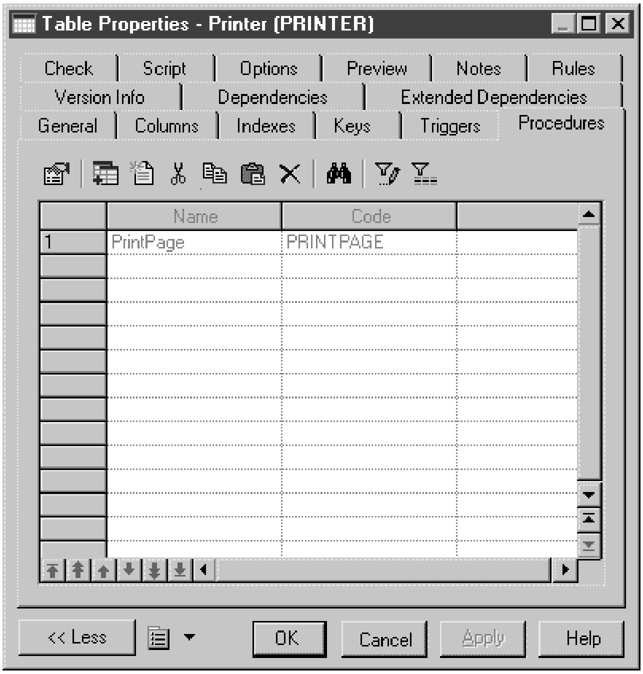
You can attach a table to a procedure from the property sheet of a procedure or the property sheet of a table.

❖ To attach a stored procedure to a table

1. Open the table property sheet and click the Procedures.

2. Click the Add Objects tool to open a selection box, choose the the stored procedure you want to attach to the table and click OK.

   The stored procedure is displayed in the list of stored procedures.



	Name	Code	
1	PrintPage	PRINTPAGE	

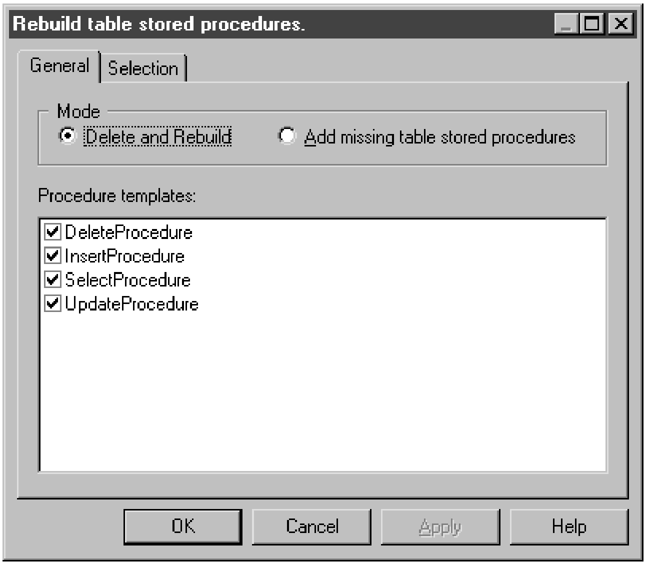
3. Click OK.

Rebuilding procedures attached to tables

You can rebuild procedures attached to tables at any time.

❖ To rebuild procedures attached to tables manually

1. Select Tools ➤ Rebuild Objects ➤ Rebuild Table Stored Procedures to open the Rebuild Table Stored Procedures window.



Mode

● Delete and Rebuild ○ Add missing table stored procedures

Procedure templates:

☑ DeleteProcedure
☑ InsertProcedure
☑ SelectProcedure
☑ UpdateProcedure

2. Specify a rebuild mode. You can choose between the following options:
   - ♦ Delete and Rebuild – all procedures attached to tables are deleted and rebuilt
   - ♦ Add missing table stored procedures – adds procedures to any selected tables that do not presently have them.

3. [optional] Click the Selection tab to specify for which tables you want to rebuild stored procedures.

4. Click OK to begin the rebuild process.

Procedure Templates (PDM)

PowerDesigner procedure templates allow you to create procedures in a modular reusable fashion.

PowerDesigner provides certain basic procedure templates and also allows you to write your own. You can create procedure templates in your DBMS definition file or in your model. These elements will be available as follows:

Procedure template created in:	Available to
DBMS (shared definition file)	All models sharing the DBMS
DBMS (copied definition file)	Current model only

☞ For more information on the DBMS share and copied definition files, see the "Generating a Database from a PDM" chapter.

Modifying DBMS template items
By modifying a DBMS template, you modify its definition in the current DBMS. You should only modify a DBMS if you have already made a copy of the DBMS definition file provided with PowerDesigner.

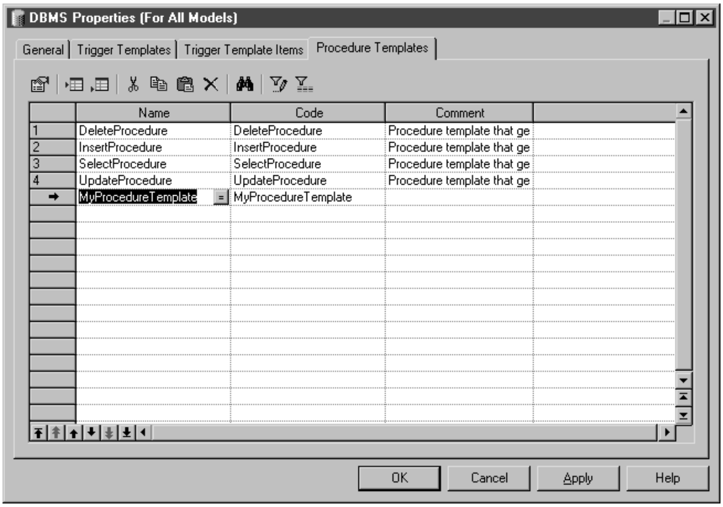
☞ For more information on working with DBMS resource files, see the DBMS Resource File Reference chapter of the Customizing and Extending PowerDesigner manual.

Creating a procedure template

You can create a new procedure template in your DBMS definition file or as part of your model. You can begin by copying an existing template or write one from scratch:

❖ To create a procedure template

1. Select Database ➤ Edit Current DBMS to open the DBMS definition file in the resource editor, and then click the Procedure Templates tab:

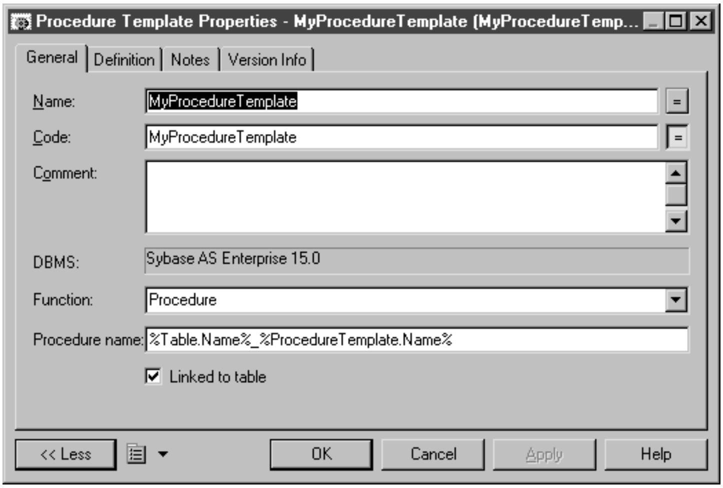


	Name	Code	Comment
1	DeleteProcedure	DeleteProcedure	Procedure template that ge
2	InsertProcedure	InsertProcedure	Procedure template that ge
3	SelectProcedure	SelectProcedure	Procedure template that ge
4	UpdateProcedure	UpdateProcedure	Procedure template that ge
➜	MyProcedureTemplate	MyProcedureTemplate	

2. Click the Add a Row tool to add a new blank template to the list.

3. Type a new name and code for the new template and click Apply to commit its creation.

4. Click the Properties tool to open the property sheet of the new procedure template:


Procedure Template Properties - MyProcedureTemplate (MyProcedureTemp...
General
Definition
Notes
Version Info
<< Less
OK
Cancel
Apply
Help

Name: MyProcedureTemplate

Code: MyProcedureTemplate

Comment:

DBMS: Sybase AS Enterprise 15.0

Function: Procedure

Procedure name: %Table.Name%_%ProcedureTemplate.Name%

☑ Linked to table

5. Click the Definition tab and enter or modify the definition code. You can

use PDM variables and macros and various other tools available from the toolbar (see “SQL Code Definition Toolbars” on page 270).

6. You can also modify other of the procedure template’s properties. For a full list of the properties available, see “Procedure template properties” on page 269.

7. Click OK in each of the dialog boxes.

   A confirmation box will appear, asking if you want to save the changes to the DBMS.

   Click Yes to confirm the template creation.

PowerDesigner pre-defined procedure templates

The pre-defined DBMS templates provided with PowerDesigner generate SQL insert, delete, update, and select statement procedures.

You can modify the code of these pre-defined procedure templates, but they cannot be deleted or renamed.

The following templates, which create procedures linked to tables, are provided for those databases that support them:

Template type	Generates procedure executing. . .
`DeleteProcedure`	a SQL Delete statement
`InsertProcedure`	a SQL Insert statement
`SelectProcedure`	a SQL Select statement
`UpdateProcedure`	a SQL Update statement

Modifying a procedure template

You can modify both your templates and those that are provided with PowerDesigner.

❖ To modify a procedure template

1. Select Database ➤ Edit Current DBMS to open the DBMS definition file in the resource editor, and then click the Procedure Templates tab.

2. Click a procedure template in the list, and then click the Properties tool to open its property sheet.

3. Click the Definition tab and modify the procedure definition code. You can use PDM variables and macros and various other tools available from

the toolbar. For more information, see “SQL Code Definition Toolbars” on page 270.

4. You can also modify other of the procedure template’s properties. For a full list of the properties available, see “Procedure template properties” on page 269.

5. Click OK in each of the dialog boxes.

   A confirmation box will appear, asking if you want to save the changes to the DBMS.

   Click Yes to confirm the template creation.

Procedure template properties

You can modify a procedure template’s properties from its property sheet. The following sections detail the property sheet tabs that contain the properties most commonly entered for procedure templates.

The General tab contains the following properties:

Property	Description
Name	The name of the item which should be clear and meaningful, and should convey the item’s purpose to non-technical users.
Code	The technical name of the item used for generating code or scripts, which may be abbreviated, and should not generally include spaces.
Comment	Descriptive label for the procedure template.
DBMS	Current DBMS.
Function	Specifies whether the template defines procedures or functions.
Procedure Name	Specifies the format of the resulting procedure names.
Linked to table	Specifies whether the resulting procedure will be linked to a table.

The following tabs are also available:

- Definition - contains a field for entering its definition code. For information about editing this code, see “Modifying a procedure template” on page 268.

SQL Code Definition Toolbars

The following tools are available on the Definition tabs of trigger, procedure and template property sheets:

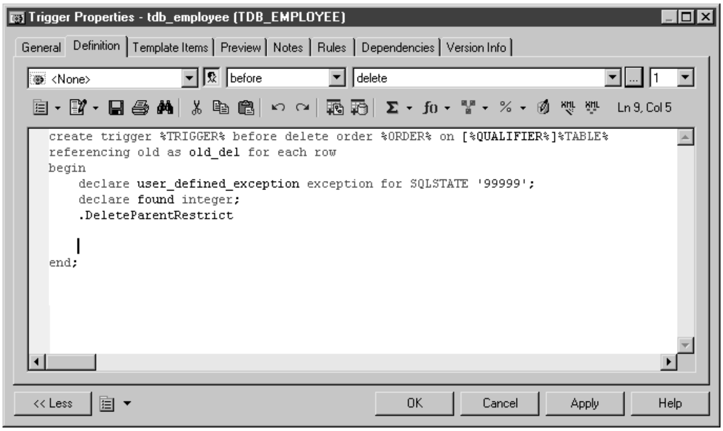
Tool	Description
	Add Trigger Item From Model - [Triggers and trigger templates only] Opens a dialog box to select a trigger template item defined in the model for insertion in the trigger definition.
	Add Trigger Item From DBMS - [Triggers and trigger templates only] Opens a dialog box to select a trigger template item defined in the DBMS definition file for insertion in the trigger definition.
Σ ▾	Operators - Lists logical operators for insertion in the trigger definition.
f() ▾	Functions - Lists group, number, string, date, conversion and other functions for insertion in the trigger definition.
▾	Macros - Lists macros for insertion in the trigger definition. For more information on macros, see the "Writing SQL Statements in PowerDesigner" appendix.
% ▾	Variables - Lists variables for use with operators and functions for insertion in the trigger definition. You can also use formatting variables that have a syntax that can force a format on their values, as follows: ♦ Force values to lower-case or upper-case characters ♦ Truncate the length of values For more information on variables, see "PDM variables" in the Writing SQL Statements in PowerDesigner chapter.
	Edit with SQL Editor - Opens the SQL Editor dialog box. Provides object types and available objects for insertion in the trigger definition.
XML	SQL/XML Wizard - Opens the SQL/XML Wizard to build a SQL/XML query from a table or a view and insert it in the trigger definition (see "Creating SQL/XML Queries with the Wizard" on page 271).
XML	Insert SQL/XML Macro - Opens a dialog box to select a global element in an XML model open in the workspace with the SQL/XML extended model definition. Inserts a SQL/XML macro referencing the selected element in the trigger definition.

Creating SQL/XML Queries with the Wizard

You can use the SQL/XML Wizard to insert a SQL/XML query in the definition of a trigger, stored procedure, or function to store or retrieve data, in an XML format, from relational databases supporting SQL/XML. The wizard, allows you to select tables and views from a PDM to build a mapped XML model. This XML model (which does not appear in the workspace) is used to generate SQL/XML queries from global elements.

❖ To insert a SQL/XML query in the definition of a trigger

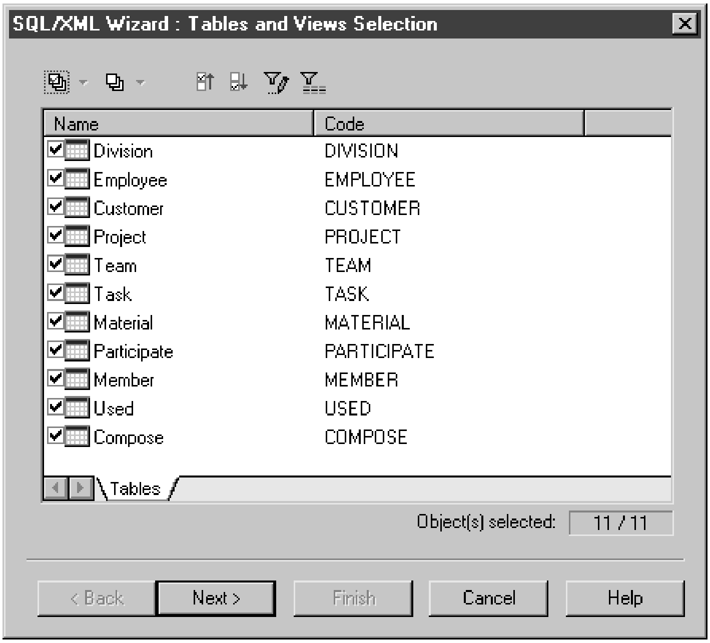
1. Open the trigger property sheet, click the **Definition** tab and position the cursor in the trigger definition where you want to insert the SQL/XML query:



```
create trigger %TRIGGER% before delete order %ORDER% on [%QUALIFIER%]%TABLE%
referencing old as old_del for each row
begin
    declare user_defined_exception exception for SQLSTATE '99999';
    declare found integer;
    .DeleteParentRestrict

    
end;
```

2. Click the **SQL/XML Wizard** tool to launch the wizard at the Tables and Views Selection page:

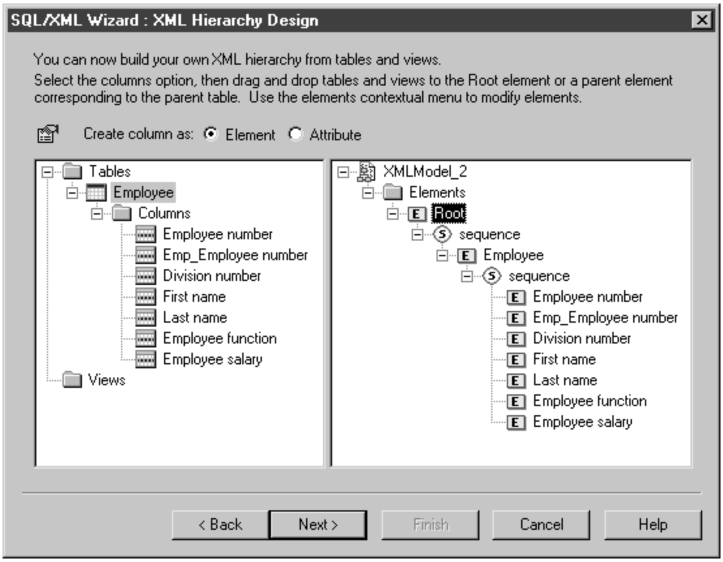


SQL/XML Wizard : Tables and Views Selection

Name	Code
☑ Division	DIVISION
☑ Employee	EMPLOYEE
☑ Customer	CUSTOMER
☑ Project	PROJECT
☑ Team	TEAM
☑ Task	TASK
☑ Material	MATERIAL
☑ Participate	PARTICIPATE
☑ Member	MEMBER
☑ Used	USED
☑ Compose	COMPOSE

Object(s) selected: 11 / 11

3. Select the tables and views that you want to include in your query and click Next to go to the XML Hierarchy Design page:



SQL/XML Wizard : XML Hierarchy Design

You can now build your own XML hierarchy from tables and views.
Select the columns option, then drag and drop tables and views to the Root element or a parent element corresponding to the parent table. Use the elements contextual menu to modify elements.

Create column as: ● Element ○ Attribute

```
Tables
  Employee
    Columns
      Employee number
      Emp_Employee number
      Division number
      First name
      Last name
      Employee function
      Employee salary
Views
```

```
XMLModel_2
  Elements
    Root
      sequence
        Employee
          sequence
            Employee number
            Emp_Employee number
            Division number
            First name
            Last name
            Employee function
            Employee salary
```

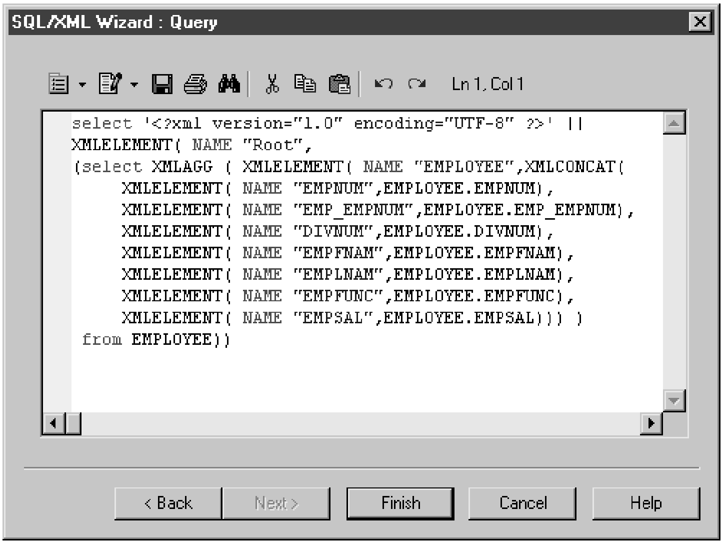
On this tab, you construct the XML hierarchy that you want to generate:

- The left-hand pane lists the tables and views that you have selected

- ♦ The right-hand pane displays the XML hierarchy to be generated, containing a default root element.

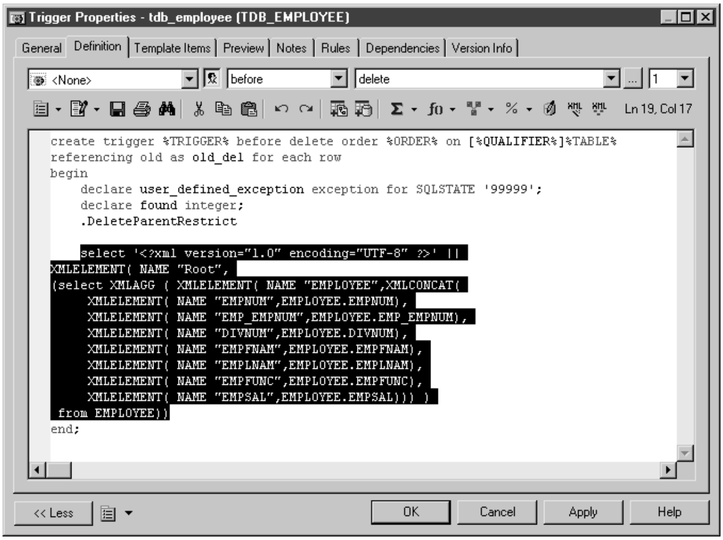
4. You can build your XML hierarchy using the following techniques:
   - ♦ Specify whether columns will be generated as elements or attributes by using the radio buttons above the panes.
   - ♦ Drag and drop a table, view, or column onto a node in the XML hierarchy. You must respect the PDM hierarchy: You cannot create an XML hierarchy between two elements if there is no **reference** between their corresponding tables, and a **parent** table cannot be placed beneath one of its children.
   - ♦ Right-click a table, view, or column and select Add from the contextual menu to add it to the last selected node in the XML hierarchy.
   - ♦ Rename an element or attribute by clicking its node and typing a new name.
   - ♦ Create new elements and attributes not in the PDM, and Sequence, Choice and All group particles, by right-clicking an XML node and selecting New ➤ *object* from the contextual menu.
   - ♦ Delete an XML node by right-clicking it and selecting Delete from the contextual menu.

5. When you have finished building your hierarchy, click Next to go to the Query tab:



```
select '<?xml version="1.0" encoding="UTF-8" ?>' ||
XMLELEMENT( NAME "Root",
(select XMLAGG ( XMLELEMENT( NAME "EMPLOYEE",XMLCONCAT(
     XMLELEMENT( NAME "EMPNUM",EMPLOYEE.EMPNUM),
     XMLELEMENT( NAME "EMP_EMPNUM",EMPLOYEE.EMP_EMPNUM),
     XMLELEMENT( NAME "DIVNUM",EMPLOYEE.DIVNUM),
     XMLELEMENT( NAME "EMPFNAM",EMPLOYEE.EMPFNAM),
     XMLELEMENT( NAME "EMPLNAM",EMPLOYEE.EMPLNAM),
     XMLELEMENT( NAME "EMPFUNC",EMPLOYEE.EMPFUNC),
     XMLELEMENT( NAME "EMPSAL",EMPLOYEE.EMPSAL))) )
 from EMPLOYEE))
```

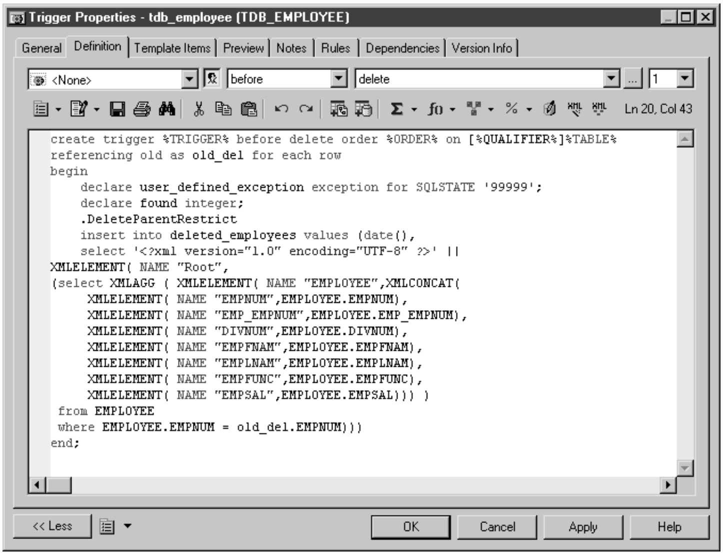
6. Review your query and click Back, if necessary, to make revisions in your hierarchy. When you are satisfied, click Finish to close the wizard and insert the SQL/XML query in the trigger definition



```
create trigger %TRIGGER% before delete order %ORDER% on [%QUALIFIER%]%TABLE%
referencing old as old_del for each row
begin
    declare user_defined_exception exception for SQLSTATE '99999';
    declare found integer;
    .DeleteParentRestrict

    select '<?xml version="1.0" encoding="UTF-8" ?>' ||
XMLELEMENT( NAME "Root",
(select XMLAGG ( XMLELEMENT( NAME "EMPLOYEE",XMLCONCAT(
     XMLELEMENT( NAME "EMPNUM",EMPLOYEE.EMPNUM),
     XMLELEMENT( NAME "EMP_EMPNUM",EMPLOYEE.EMP_EMPNUM),
     XMLELEMENT( NAME "DIVNUM",EMPLOYEE.DIVNUM),
     XMLELEMENT( NAME "EMPFNAM",EMPLOYEE.EMPFNAM),
     XMLELEMENT( NAME "EMPLNAM",EMPLOYEE.EMPLNAM),
     XMLELEMENT( NAME "EMPFUNC",EMPLOYEE.EMPFUNC),
     XMLELEMENT( NAME "EMPSAL",EMPLOYEE.EMPSAL))) )
 from EMPLOYEE))
end;
```

7. [optional] Add code to complete the SQL/XML query:



```
create trigger %TRIGGER% before delete order %ORDER% on [%QUALIFIER%]%TABLE%
referencing old as old_del for each row
begin
    declare user_defined_exception exception for SQLSTATE '99999';
    declare found integer;
    .DeleteParentRestrict
    insert into deleted_employees values (date(),
    select '<?xml version="1.0" encoding="UTF-8" ?>' ||
XMLELEMENT( NAME "Root",
(select XMLAGG ( XMLELEMENT( NAME "EMPLOYEE",XMLCONCAT(
     XMLELEMENT( NAME "EMPNUM",EMPLOYEE.EMPNUM),
     XMLELEMENT( NAME "EMP_EMPNUM",EMPLOYEE.EMP_EMPNUM),
     XMLELEMENT( NAME "DIVNUM",EMPLOYEE.DIVNUM),
     XMLELEMENT( NAME "EMPFNAM",EMPLOYEE.EMPFNAM),
     XMLELEMENT( NAME "EMPLNAM",EMPLOYEE.EMPLNAM),
     XMLELEMENT( NAME "EMPFUNC",EMPLOYEE.EMPFUNC),
     XMLELEMENT( NAME "EMPSAL",EMPLOYEE.EMPSAL))) )
 from EMPLOYEE
 where EMPLOYEE.EMPNUM = old_del.EMPNUM)))
end;
```

8. Click OK to close the trigger property sheet:

Generating Triggers and Procedures

You can create or modify database triggers to a script or to a live database connection.

❖ To generate triggers and procedures

1. Select Database ➤ Generate Database to open the Database Generation window, and specify the standard options, including whether you want to generate to a script or to a live database connection.

   ☞ For detailed information about using this window, see the "Generating a Database" section in the "Generating a Database from a PDM" chapter.

2. Select "Triggers & Procedures (with Permissions)" from the Settings set list in the Quick Launch groupbox at the bottom of the window. This settings set specifies standard options for generating triggers and procedures.

   or:

   Click the Options tab and click on Trigger in the left-hand pane to display the trigger generation options. Change the default options as appropriate.

   ☞ For detailed information about settings sets, see "Quick launch selection and settings sets" section in the "Generating a Database from a PDM" chapter.

3. [optional] Click the Selection tab and select the Table or Procedure subtab at the bottom of the tab. Select the tables or procedures that you want to generate for. Note that if you want to generate a trigger script for tables owned by a particular owner, you can select an owner from the Owner list.

4. Click OK to begin the generation.

Defining a generation order for stored procedures

You can use extended dependencies to define an order in the generation of stored procedures.

Extended dependencies are free links between PDM objects. These links help to make object relationships clearer between model objects. Usually, these links are not interpreted and checked by PowerDesigner as they are meant to be used for documentation purposes only. However, if you assign the <<**DBCreateAfter**>> stereotype to an extended dependency between stored procedures, it will be analyzed during generation.

The procedure from which you start the extended dependency is the dependent procedure and the procedure at the other end of the link is the influent procedure. The influent procedure will be generated before the dependent procedure.

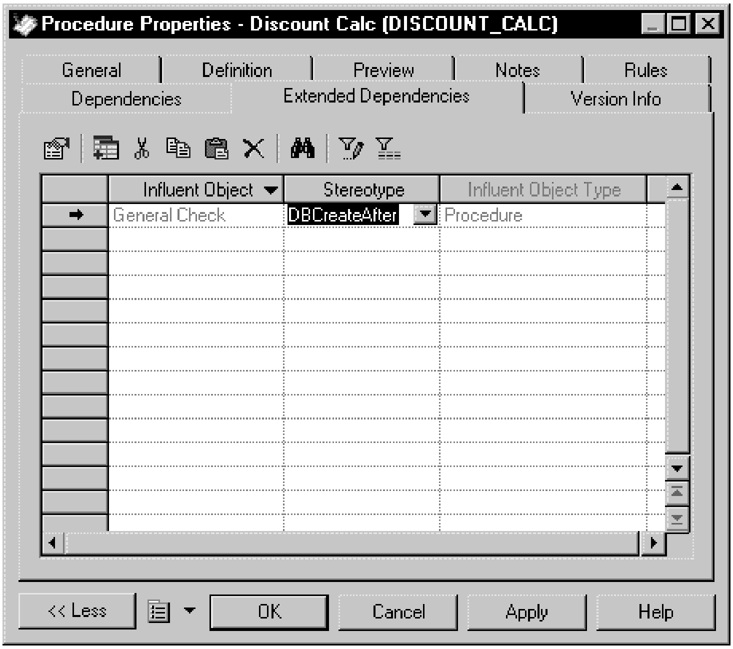
Circular extended dependencies

If you create a reflexive and/or circular set of extended dependencies with the <<**DBCreateAfter**>> stereotype, an error message is displayed during the check model. If you choose to ignore this error, the stored procedures will be generated in alphabetical order, without taking into account the generation order, which could cause errors in the creation of stored procedures in the database.

Example

A publisher may decide to sell certain books at a reduced rate (15%) when a customer's order is above 10 000$.

In this model, one stored procedure GENERAL CHECK globally verifies orders: check books availability, check the order amount, check if discount rate is calculated, and applies discount rate to order bill. During the execution of this stored procedure, the procedure DISCOUNT CALC is called to calculate the 15% discount rate. It is important to generate GENERAL CHECK before DISCOUNT CALC; you can define an extended dependency to set the generation order between these two objects. To do so, you have to open the property sheet of the dependent stored procedure, click the Extended Dependencies tab and create an extended dependency with the <<**DBCreateAfter**>> stereotype with the influent stored procedure.



Influent Object	Stereotype	Influent Object Type
General Check	DBCreateAfter	Procedure

❖ To define a generation order for stored procedures

1. Select Model ➤ Procedures to display the List of Stored Procedures.
2. Select a dependent stored procedure in the list and click the Properties tool.

   The property sheet of the stored procedure is displayed.
3. Click the Extended Dependencies tab to display the Extended Dependencies tab.
4. Click the Add Objects tool.
5. Select the Procedure tab in the Add Object selection dialog box.
6. Select the influent stored procedure check box and click OK.
7. Select the <<**DBCreateAfter**>> stereotype in the Stereotype list.
8. Click OK.

   The influent stored procedure is displayed in the Influent Object column of the list of extended dependencies of the dependent stored procedure.

☞ For more information on extended dependencies, see “Using extended dependencies”, in the Objects chapter of the *Core Features Guide.*

Creating and generating user-defined error messages

You can create user-defined error messages. The error messages are stored in a message table which you need to create in your database. When you select trigger generation parameters, you can choose to generate an error message from this table.

In the generated trigger script, the message table is called in a SELECT command. If an error number in the script corresponds to an error number value in the table column, then the standard error message is replaced by the message defined in the table.

Creating a message table

You create a message table which stores error message information.

❖ To create a message table

1. Create a table with columns to store the following information:

Column to store...	Description
Error number	Number of the error message that is referenced in the trigger script
Message text	Text of message

2. Generate the table in your database.

3. Select Database ➤ Execute SQL.

   A dialog box asks you to identify a data source and connection parameters.

4. Select a data source and fill in connection parameters.

5. Click Connect.

   An SQL query editor box is displayed.

6. Type an SQL statement to insert a message number and text in the appropriate columns. You can use the following format for example:

   insert into `table` values (`error number,'error message'`)

   insert into `ERR_MSG` values (`1004,'The value that you are trying to insert does not exist in the referenced table'`)

7. Click Execute.

   A message box tells you that the command has been successfully executed.

8. Click OK.

   You return to the SQL query box.

9. Click Close.

Generating a user-defined error message

You can choose to generate a user-defined error message from the trigger generation parameters box.

❖ To generate a user-defined error message

1. Select Tools ➤ Rebuild Objects ➤ Rebuild Triggers.

   The Rebuild Triggers dialog box is displayed.

2. Click the Error Messages tab.

   The Error Messages tab is displayed.

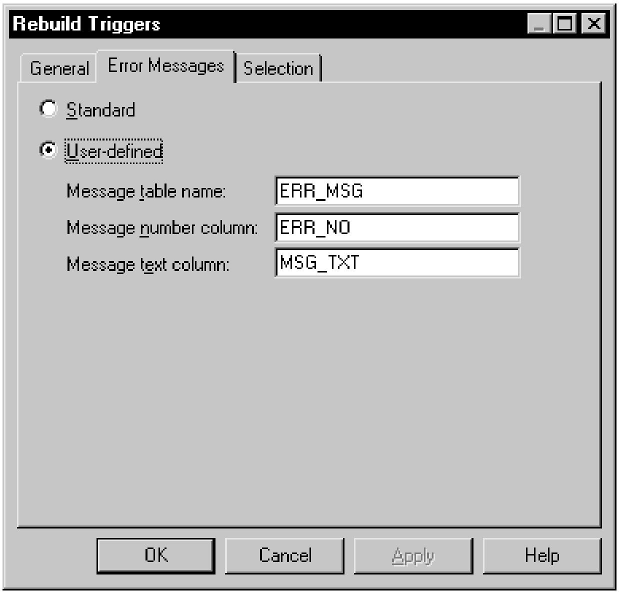
3. Select the User-defined radio button.

4. Type the name of the table that contains the error message in the Message Table Name box.

5. Type the name of the column that contains the error number in the Message Number box.

6. Type the name of the column that contains the error message text in the Message Text column.

   Below is an example of the details for a table called ERR_MSG.



○ Standard

● User-defined

Message table name: ERR_MSG

Message number column: ERR_NO

Message text column: MSG_TXT

7. Click the General tab and select the mode and triggers to create.
8. Click the Selection tab and select the tables for which you want to create triggers.

   ☞ For more information on rebuilding triggers, see section "Rebuilding Triggers" on page 229.
9. Click OK.

   The trigger rebuilding process is shown in the Output window.

   Select Database ➤ Generate Database to open the Database Generation window.
10. Select generation parameters as required.
11. Click OK.

   ☞ For information on selecting trigger generation parameters, see section "Generating Triggers and Procedures" on page 275.

CHAPTER 6

Building a Database Access Structure

About this chapter

This chapter explains how to manage access to the database using privileges and permissions, and defining users, groups, roles, and object synonyms.

Contents

Introducing database access

In this manual, we group within the concept of database access, the objects that are used to let database users carry out their jobs while preserving the privacy and integrity of the information within the database.

Depending on the target DBMS, the setting up of a secure database environment implies the following tasks:

- Creation of **users** (see “Users (PDM)” on page 283)
- Creation of **groups** (see “Groups (PDM)” on page 300)
- Creation of **roles** (see “Roles (PDM)” on page 298)
- Definition of **system privileges** (see “Granting system privileges” on page 286)
- Definition of **object permissions** (see “Granting object permissions” on page 290)

Reverse engineering

During reverse engineering by script, the permission and privilege grant orders are reverse engineered only if the user, group, or role already exist in the model or, in the case of a new model, if the user, group, or role creation orders are in the same script.

These restrictions do not apply to reverse engineering from a live database connection.

Users (PDM)

A user is a database object that identifies a person who can login or connect to the database. Users may have:

- ♦ a role that specifies their responsibilities (see “Roles (PDM)” on page 298)
- ♦ membership in a group that specifies their rights (see “Groups (PDM)” on page 300)
- ♦ privileges that control their level of access (see “Granting system privileges” on page 286)
- ♦ permissions that allow them to perform actions on objects (see “Granting object permissions” on page 290)

Note
Not all DBMSs support the concepts of user, role and group.

Creating a user

You can create a user in any of the following ways:

- ♦ Select Model ➤ Users and Roles ➤ Users to access the List of Users, and click the Add a Row tool.
- ♦ Right-click the model or package in the Browser, and select New ➤ User.

☞ For general information about creating objects, see the Objects *Core Features Guide* .

User properties

You can modify an object’s properties from its property sheet. To open a user property sheet, double-click its Browser entry in the Users folder. The following sections detail the property sheet tabs that contain the properties most commonly entered for users.

The General tab contains the following properties:

Property	Description
Name	Name of the user
Code	Code of the user used as identifier in the database
Comment	Descriptive label for the user
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined
Password	Password that may be used for database connection

The following tabs are also available:

- Privileges - lists the system privileges granted to the user (see "Granting system privileges" on page 286).
- Permissions - lists the operations that the user is permitted to perform on various database objects (see "Granting object permissions" on page 290).

Assigning an owner to an object

In a database, the user who creates an object (tables, views, stored procedures, etc) is the owner of the object and is automatically granted all permissions on it.

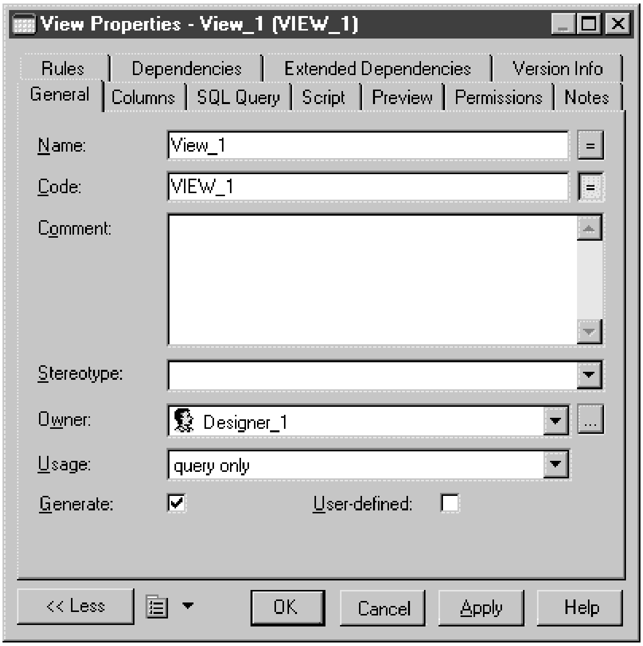
When building a PDM, you must attach the user to the object in order to make it the owner. Each object can have only one owner. In a model where many users have access to the same objects, you can restrict object modifications to the owner and define permissions for the other users.

Owners can also be used during generation: when you select to generate for a selected owner, only the tables belonging to this owner are generated, whereas when you generate as ADMIN, you generate all the tables on behalf of all their owners.

❖ To attach an object to a user

1. Open the property sheet of the object.

2. Select a user in the Owner list. You can create a new user by clicking the ellipsis button to the right of the Owner list.



View Properties - View_1 (VIEW_1)

Name: View_1

Code: VIEW_1

Comment:

Stereotype:

Owner: Designer_1

Usage: query only

Generate: ☑

User-defined: □

3. Click OK.

Specifying default owners for object types

You can specify a default owner for each type of object that supports the concept of ownership. The default owner will automatically be linked to all the objects of this type that you create after making this change.

❖ To specify a default owner for tables

1. Select Tools ➤ Model Options and then select Table and View in the left-hand pane.

2. Select a user in the Default owner list in the Table groupbox. You can create a new user by clicking the ellipsis button to the right of the Default owner list.

3. Click OK.

For more information, see “Setting PDM Model Options” in the Customizing the PDM Environment chapter.

Granting system privileges

A system privilege is a set of rights assigned to a database user, group, or role. You use system privileges to create user profiles with different levels of influence over the database content.

System privileges are used in association with object permissions (see “Granting object permissions” on page 290) to evaluate the rights of a user, group, or role. For example, if a user has the modify privilege, he cannot modify an object on which he has no update permission.

Terminology
In some DBMS, system privileges are called permissions. In this manual, the term privilege is used for any right granted to a user, a group, or a role. Permissions are defined for objects.

System privileges vary according to the DBMS you are using. The list of privileges also includes predefined roles (like connect, or resource) for an easier use.

System privileges are **granted** to a user. A user with administrative profile is also allowed to **revoke** a privilege in order to prevent a user from performing certain actions over the database content.

The list of predefined system privileges allowed in the current DBMS is available in the System entry under Script\Object\Privileges in the DBMS resource file. The Privileges category also contains entries used to provide the syntax for the different order corresponding to the DBMS privileges.

Granting system privileges to a user, role, or group

The procedure for defining privileges is identical for users, groups, and roles.

Adding a system privilege
You can add a system privilege to the list of available privileges from the DBMS editor. To do so, select Database ➤ Edit Current DBMS, expand categories Script\Objects\Privileges and type each new system privilege on a new line at the end of the list of privileges in the Value box

Inherited privileges

By default, a user belonging to a group or having a role inherits the group or role privileges. Inherited privileges appear in the Privileges tab of the user property sheet.

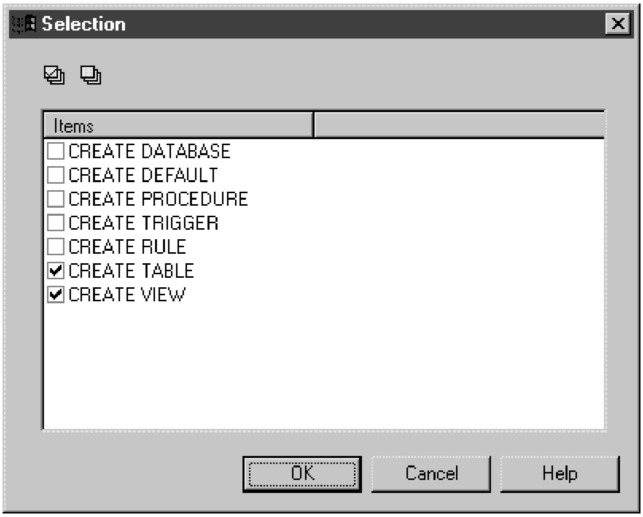
When you select specific privileges for the user, the list of privileges displays the user privilege above the group privilege. The following table summarizes the different privilege combinations:

Privilege combination	Description
✓	Privilege granted to user
✓	Privilege inherited from group
✗ ✓	Privilege inherited from group and revoked to user
✓ ✓	Privilege inherited from group overloaded by “with admin option”

❖ To grant system privileges to a user, role, or group

1. Open the property sheet of a user, role, or group, and click the Privileges tab.

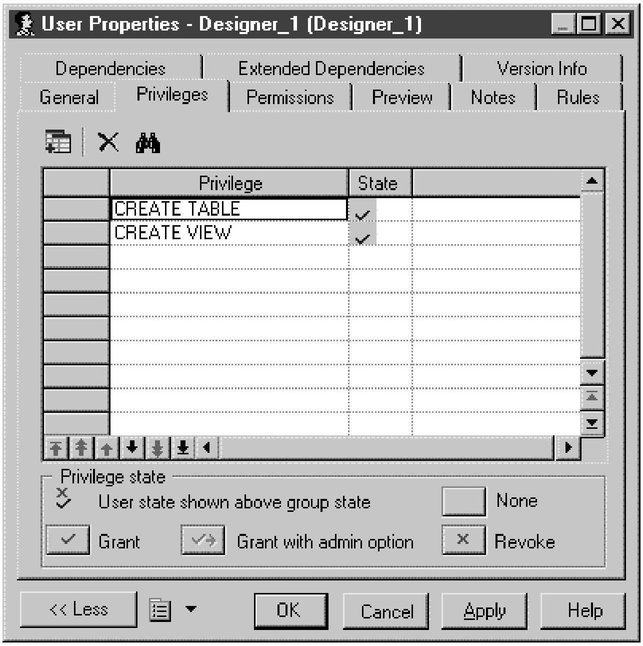
2. Click the Add Objects tool to open a selection box listing all the privileges available in the DBMS.



☐ CREATE DATABASE
☐ CREATE DEFAULT
☐ CREATE PROCEDURE
☐ CREATE TRIGGER
☐ CREATE RULE
☑ CREATE TABLE
☑ CREATE VIEW

3. Select one or more privileges and click OK to add them to the list of privileges of the user, role, or group. By default, privileges are granted.

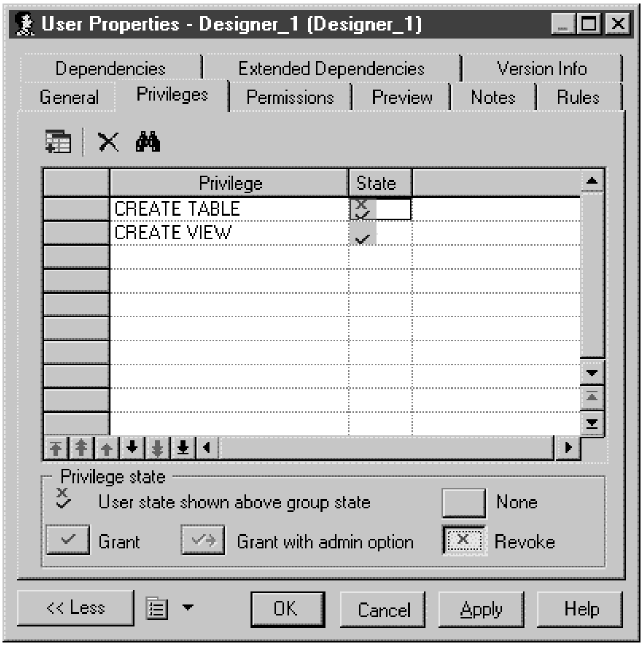
In the example below, the privileges in red are granted to the group to which the user belongs.



Privilege	State
CREATE TABLE	✓
CREATE VIEW	✓

4. [optional] To change the state of a privilege, click in the State column until the desired state is displayed, or select one of the Privilege state tools at the bottom of the tab:

   - Grant – [default] Assigns the privilege to the user.
   - Grant with admin option - Assigns the privilege to the user, and allows the recipient to pass on the privilege to other users; groups, or roles. For example, you assign the CREATE TABLE privilege for user Designer_1 and then click the Grant With Admin Option button to permit Designer_1 to grant this privilege to other users.
   - Revoke – Revokes the privilege inherited from a group or role for the current user or group. This option is only available when the user has inherited a privilege from a group or a role.
   - None - Cancels any state and cleans up the current cell.



Privilege	State
CREATE TABLE	☒
CREATE VIEW	✓

5. Click OK.

Generating privileges

You can generate privileges to a script or to a live database connection.

❖ To generate privileges

1. Select Database ➤ Generate Database to open the Database Generation window, and specify the standard options, including whether you want to generate to a script or to a live database connection.

   ☞ For detailed information about using this window, see "Generating a Database" section in the "Generating a Database from a PDM" chapter.

2. Select "Users & Groups (with privileges)" from the Settings set list in the Quick Launch groupbox at the bottom of the window. This settings set specifies standard options for generating privileges.

   or:

   Click the Options tab and click on User in the left-hand pane to display the user generation options. Change the default options as appropriate.

   ☞ For detailed information about settings sets, see "Quick launch selection and settings sets" section in the "Generating a Database from a PDM" chapter.

3. [optional] Click the Selection tab and select the Users sub-tab at the bottom of the tab. Select the users that you want to generate for.

4. Click OK to begin the generation.

Granting object permissions

Object permissions are operations attached to particular database objects.

PowerDesigner allows you to define permissions on tables, views, columns, procedures, packages, and other objects depending on your DBMS. Some or all of the following may be available

Permission	Description
Select	To observe information contained in object
Insert	To insert rows into object
Alter	To alter table with ALTER TABLE command
Delete	To delete rows from object
References	To create indexes in tables and foreign key referencing tables
Update	To update row in object
Index	To create an index with the CREATE INDEX command
Execute	To execute procedure or function

☞ For more information on the permissions allowed in your DBMS, see your DBMS documentation.

Object owner

The owner of an object (see "Assigning an owner to an object" on page 284) automatically has permission to carry out any operation on that object. These permissions do not appear in the Permissions tab of the object property sheet but they are implemented during generation and reverse engineering.

Granting permissions to a user, role, or group

The procedure for defining privileges is identical for users, groups, and roles.

Inherited permissions

A user belonging to a group or having a role with permissions inherits these permissions. You can display or hide users with inherited permissions using the following tools in the upper part of the Permissions tab:

Tool	Action
‾✓	Adds users that inherit permissions
+✓	Hides users that do not inherit permissions

When you select specific permissions for the user, the list of permissions displays the user permission above the group permission. The following table summarizes the different permission combinations:

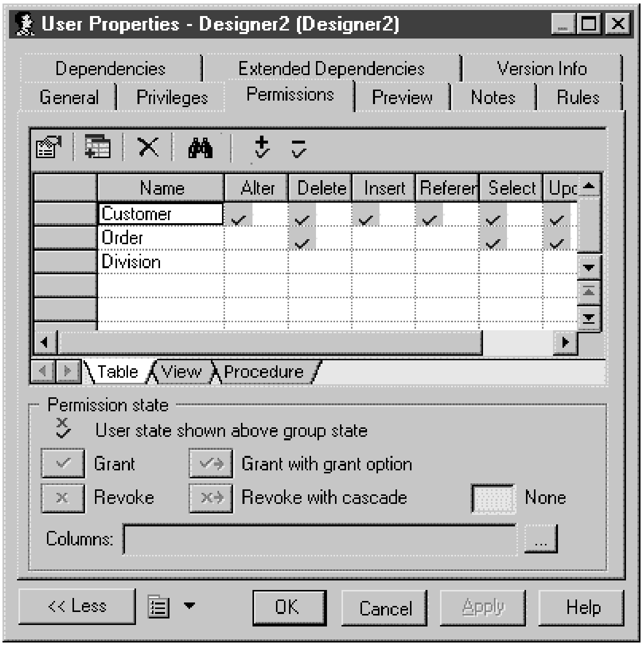
Permission combination	Description
✓	Permission granted to user
✓	Permission inherited from group
× ✓	Permission granted to group and revoked to user
✓→ ✓	Permission granted to group and overloaded by "with admin option"
×→ ✓	Permission granted to group and revoked with cascade to user

❖ To grant permissions to a user, role, or group

1. Open the property sheet of a user, role, or group, and click the Permissions tab. The columns in the list show the permissions available for a given type of object in the current DBMS. A sub-tab is displayed for each type of object supporting permissions in the current DBMS.

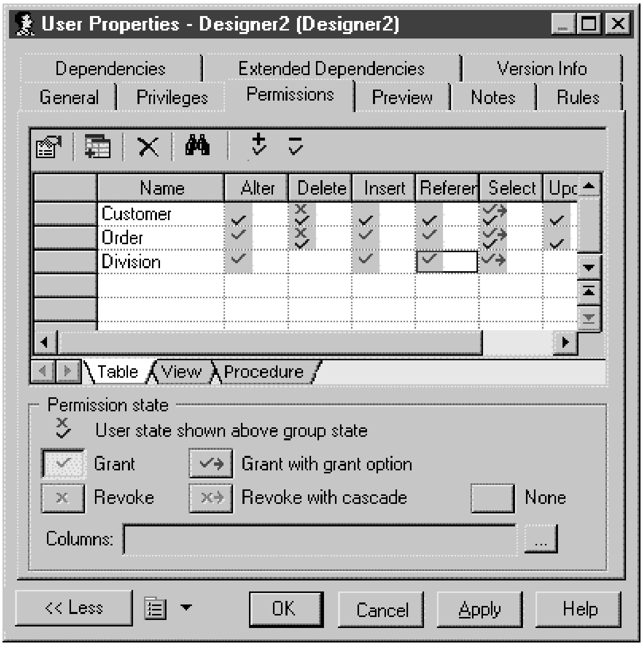
2. Click the Add Objects tool to open a selection box listing all the objects of the present type in the model.

3. Select one or more objects and click OK to add them to the list of permissions of the user, role, or group. If the current user belongs to a group with permissions on the selected objects, these permissions appear in red in the list.



Name	Alter	Delete	Insert	Referer	Select	Upc
Customer	✓	✓	✓	✓	✓	✓
Order		✓			✓	✓
Division						

4. [optional] To change the state of a permission, click in the appropriate column until the desired state is displayed, or select one of the Permission state tools at the bottom of the tab:
   - Grant – Assigns the permission to the user.
   - Grant with admin option - Assigns the permission to the user, and allows the recipient to pass on the permission to other users; groups, or roles.
   - Revoke – Revokes the permission inherited from a group or role for the current user or group. This option is only available when the user has inherited a permission from a group or a role.
   - Revoke with cascade – Revokes the permission inherited from a group or role for the current user or group and revokes any permission granted by the user.
   - None - Cancels any state and cleans up the current cell.



Name	Alter	Delete	Insert	Referer	Select	Upc
Customer	✓	✗ ✓	✓	✓	✓→ ✓	✓
Order	✓	✗ ✓	✓	✓	✓→ ✓	✓
Division	✓		✓	✓	✓→	

5. [optional] For tables, you can specify permissions on individual columns (see "Defining column permissions" on page 295).

6. Click OK.

Granting permissions for an object from its property sheet

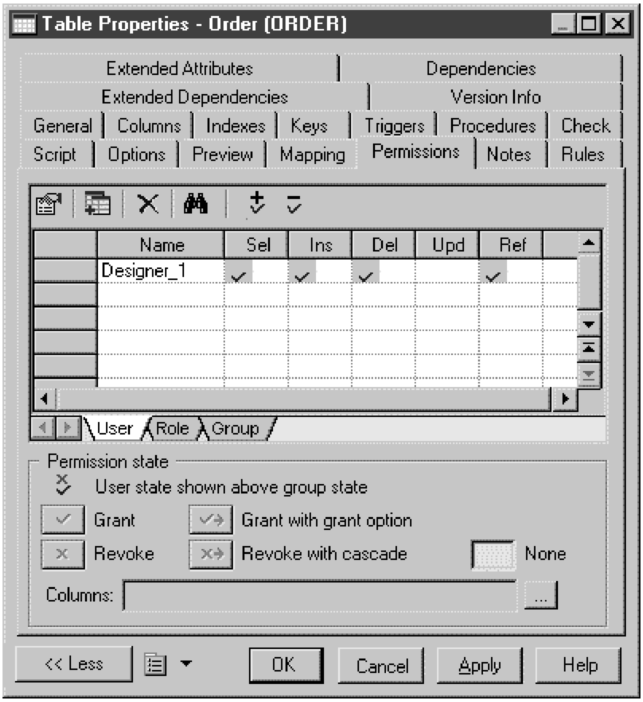
You can also grant permissions for an object directly in its property sheet.

❖ To define permissions from the object property sheet

1. Open the property sheet of the object and click the Permissions tab. The columns in the list show the permissions available for the object in the current DBMS. Sub-tabs allow you to grant permissions to Users, Roles, and Groups.

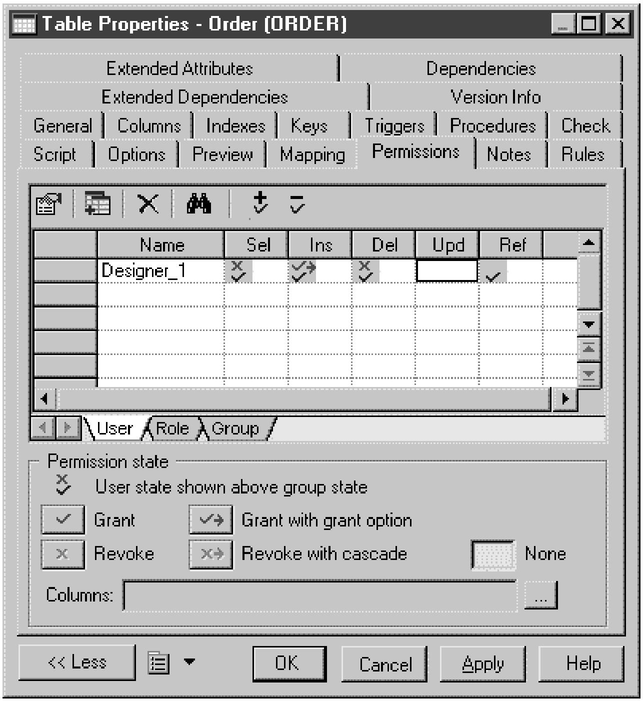
2. Click the Add Objects tool to open a selection box listing all the users (or roles or groups) in the model.

3. Select one or more users and click OK to add them to the list of users with permissions on the object. If the user belongs to a group with permissions on the object, these permissions appear in red in the list.



	Name	Sel	Ins	Del	Upd	Ref
	Designer_1	✓	✓	✓		✓

4. [optional] To change the state of a permission, click in the appropriate column until the desired state is displayed, or select one of the Permission state tools at the bottom of the tab:



Name	Sel	Ins	Del	Upd	Ref
Designer_1	✗✓	✓→	✗✓		✓

5. [optional] For tables, you can specify permissions on individual columns (see "Defining column permissions" on page 295).

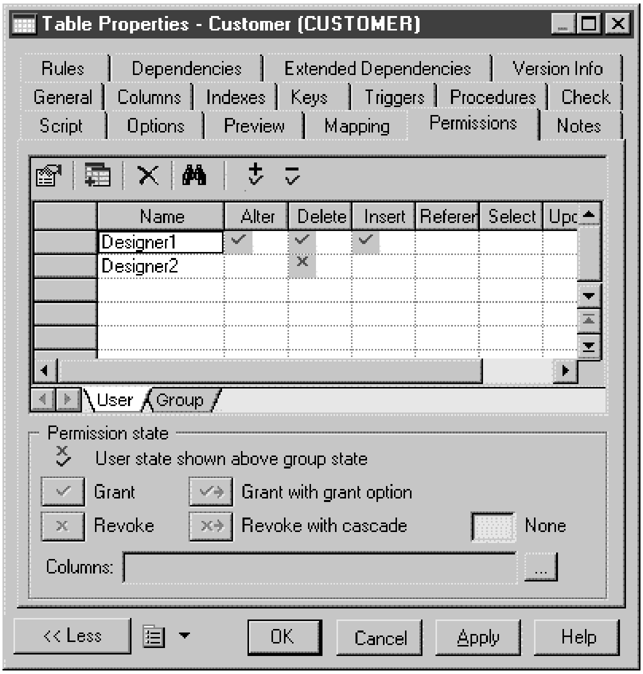
6. Click OK.

Defining column permissions

You can fine tune the permissions on a table by specifying permissions on a column-by-column basis. The available column permissions are specified in the DBMS resource file. Note that any new or modified permission may not be supported during generation or reverse-engineering.

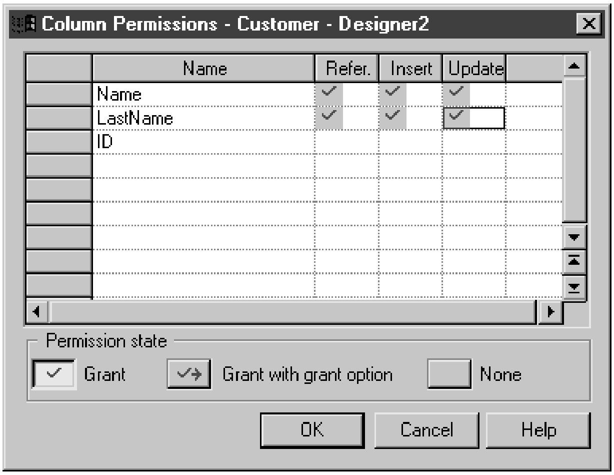
❖ To define column permissions

1. Open the property sheet of a table, click the Permissions tab, and select a user, group or role:



Name	Alter	Delete	Insert	Referer	Select	Upc
Designer1	✓	✓	✓			
Designer2		✗				

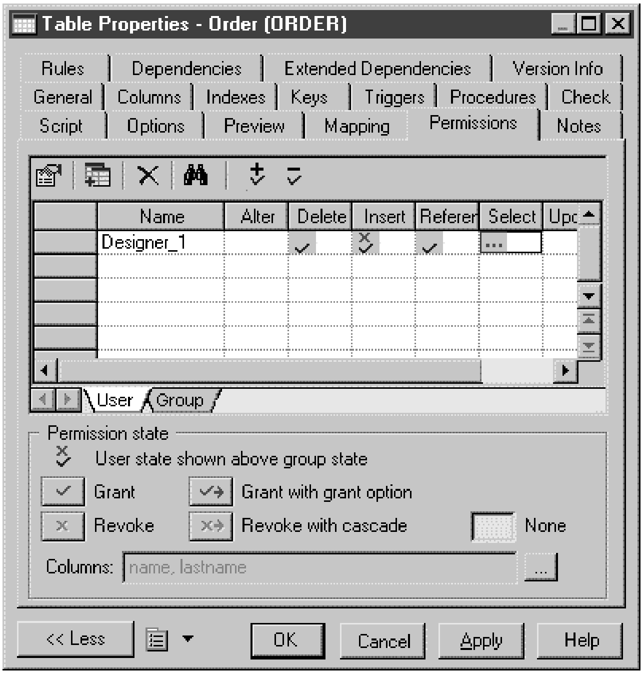
2. Click the Ellipsis button beside the Columns box to open the Column Permissions dialog box. The columns in the list show the permissions available for each of the table's columns.



Name	Refer.	Insert	Update
Name	✓	✓	✓
LastName	✓	✓	✓
ID			

3. Click inside a column cell until the desired state is displayed, or select one of the Permission state tools at the bottom of the tab.

4. Click OK to close the Column Permissions dialog box and return to the property sheet. The cells corresponding to selected column permissions contain ellipsis symbols. Click on one of these symbols to display the associated column permissions information in the Columns box.



	Name	Alter	Delete	Insert	Referer	Select	Upc
	Designer_1		✓	✗✓	✓	...	

Columns: name, lastname

5. Click OK.

Roles (PDM)

A role is a predefined profile that can be assigned to users, or roles in those DBMS that support this concept. Roles are reverse engineered in your model and you can also create user-defined roles.

Roles may have:

- privileges that control their level of access (see "Granting system privileges" on page 286)
- permissions that allow them to perform actions on objects (see "Granting object permissions" on page 290)

Creating a role

You can create a role in any of the following ways:

- Select Model ➤ Users and Roles ➤ Roles to access the List of Roles, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤ Role.

☞ For general information about creating objects, see the Objects *Core Features Guide* .

Role properties

You can modify an object's properties from its property sheet. To open a role property sheet, double-click its Browser entry in the Roles folder. The following sections detail the property sheet tabs that contain the properties most commonly entered for roles.

The General tab contains the following properties:

Property	Description
Name	Name of the role
Code	Code of the role used as identifier in the database
Comment	Descriptive label for the role
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined

The following tabs are also available:

- Privileges - lists the system privileges granted to the role (see "Granting system privileges" on page 286).

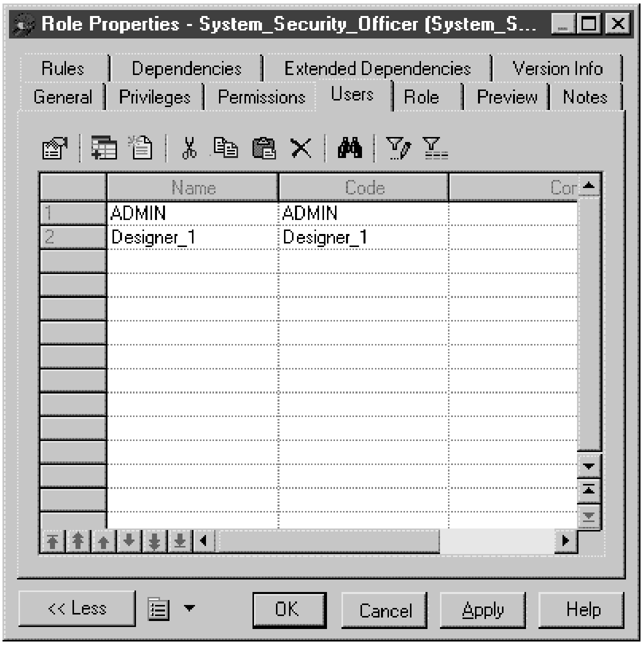
- Permissions - lists the operations that the role is permitted to perform on various database objects (see “Granting object permissions” on page 290).

Assigning a user to a role

Once you have created a role, you can assign users to it.

❖ To assign a role to a user

1. Select Model ➤ Users and Roles ➤ Roles to open the List of Roles.
2. Select a role in the list, click the Properties tool to open its property sheet and then click the Users tab.
3. Click the Add Objects tool to open a selection box listing the users available in the model.
4. Select one or more users and click OK to assign these users to the role.



	Name	Code	Cor
1	ADMIN	ADMIN	
2	Designer_1	Designer_1	

5. Click OK.

Groups (PDM)

Groups are used to facilitate the granting of privileges and permissions to users. They prevent the time-consuming and error-prone process of assigning privileges and permissions individually to each user.

Groups may have:

- privileges that control their level of access (see "Granting system privileges" on page 286)
- permissions that allow them to perform actions on objects (see "Granting object permissions" on page 290)

Creating a group

You can create a group in any of the following ways:

- Select Model ➤ Users and Roles ➤ Groups to access the List of Groups, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤ Group.

☞ For general information about creating objects, see the Objects *Core Features Guide* .

Group properties

You can modify an object's properties from its property sheet. To open a group property sheet, double-click its Browser entry in the Groups folder. The following sections detail the property sheet tabs that contain the properties most commonly entered for groups.

The General tab contains the following properties:

Property	Description
Name	Name of the group
Code	Code of the group used as identifier in the database
Comment	Descriptive label for the group
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined
Password	Password that may be used for database connection

The following tabs are also available:

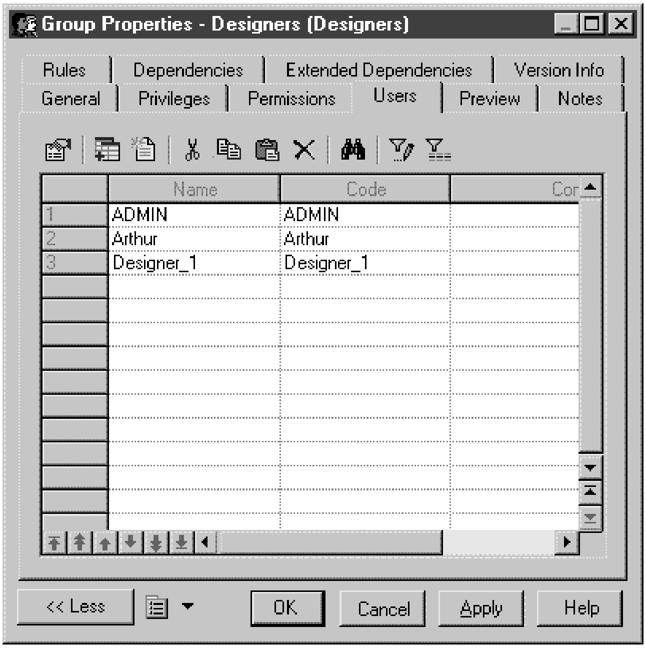
- ♦ Privileges - lists the system privileges granted to the group (see "Granting system privileges" on page 286).
- ♦ Permissions - lists the operations that the group is permitted to perform on various database objects (see "Granting object permissions" on page 290).

Inserting a user into a group

Once you have created a group, you can insert users into it.

❖ To insert a user into a group

1. Select Model ➤ Users and Roles ➤ Groups to open the List of Groups.
2. Select a group in the list, click the Properties tool to open its property sheet and then click the Users tab.
3. Click the Add Objects tool to open a selection box listing the users available in the model.
4. Select one or more users and click OK to insert these users into the group.



	Name	Code	Cor
1	ADMIN	ADMIN	
2	Arthur	Arthur	
3	Designer_1	Designer_1	

5. Click OK.

Synonyms (PDM)

A synonym is an alternative name for various types of objects (table, view, sequence, procedure, function, synonym or database package). Synonyms are created to:

- Mask the name and owner of an object
- Provide location transparency for remote objects of a distributed database
- Simplify SQL statements for database users

Example

For example, table SALES_DATA is owned by user JWARD. A standard select statement on this table would be:

```
SELECT * FROM jward.sales_data
```

The database administrator can create a synonym for this table and owner and call it SALES. In this case, the SQL statement is simplified in the following way:

```
SELECT * FROM sales
```

In PowerDesigner synonyms are created for **base objects**. The base object is the object used to create a synonym. Base objects support multiple synonyms while each synonym must have only one base object. You can view the synonyms depending on a base object in the Dependencies tab of the base object property sheet.

If you delete the base object of a synonym, the synonym is deleted as well.

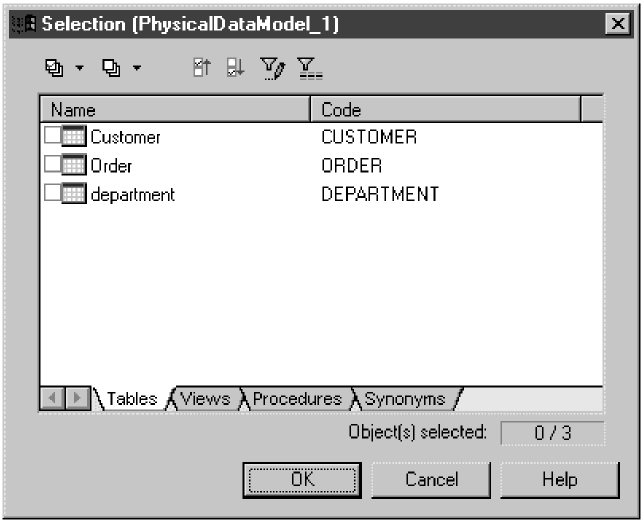
Creating a synonym

You can create a synonym as follows:

❖ To create a synonym

1. Select Model ➤ Synonyms to open the List of Synonyms.

2. Click the Create Synonyms tool to open a selection box listing all the available objects in the model on various sub-tabs.

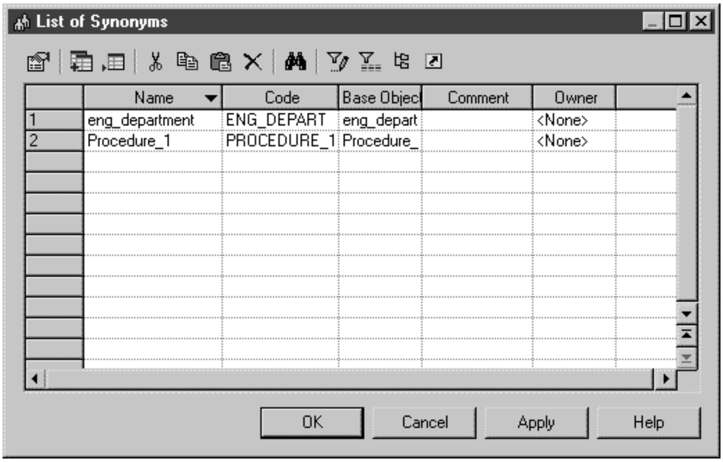


Name	Code
□ Customer	CUSTOMER
□ Order	ORDER
□ department	DEPARTMENT

Object(s) selected: 0 / 3

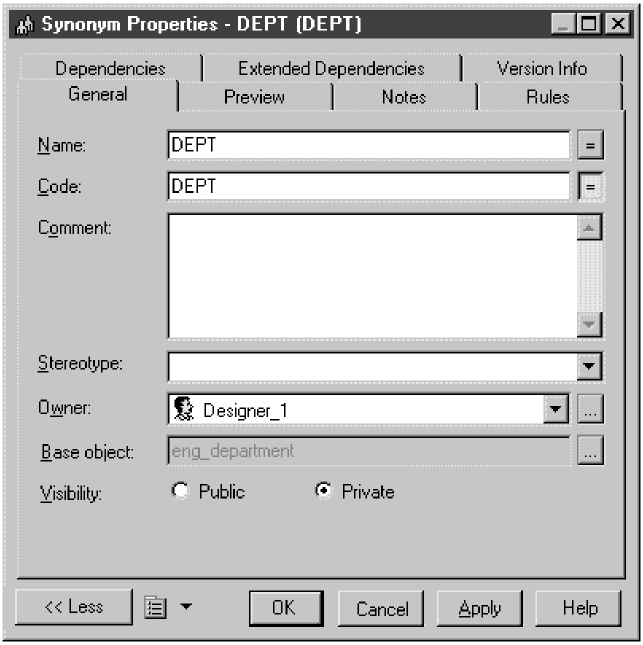
3. Select one or more objects and click OK.

   Synonyms for each of the selected objects are created in the List of Synonyms. By default, a synonym has the same name as its base object. If the Base Object column is not shown in the list, click the Customize Columns and Filter tool, select Base Object in the list of available columns, and click OK.



	Name	Code	Base Objec	Comment	Owner
1	eng_department	ENG_DEPART	eng_depart		<None>
2	Procedure_1	PROCEDURE_1	Procedure_		<None>

4. Click in the Name column of one of the rows and enter a new name for the synonym. Alternatively, you can click the Properties tool to open the property sheet of the synonym and edit its name and other properties there.



Synonym Properties - DEPT (DEPT)

Name: DEPT

Code: DEPT

Comment:

Stereotype:

Owner: Designer_1

Base object: eng_department

Visibility: ○ Public ● Private

5. Click OK in each of the dialog boxes.

☞ For general information about creating objects, see the Objects *Core Features Guide* .

Synonym properties

You can modify an object's properties from its property sheet. To open a synonym property sheet, double-click its Browser entry in the Synonyms folder.

The General tab contains the following properties:

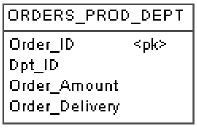
Property	Description
Name	Name of the synonym (usually corresponding to a simplified name of the base object)
Code	Code of the synonym
Comment	Additional information about the synonym

Property	Description
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined
Owner	Name of the synonym owner. You choose an owner from a list of users. A synonym can only have one owner at a time
Base Object	Name of the object origin of the synonym. The Ellipsis button displays a selection dialog box that lets you select objects among all the models opened in the current workspace and belonging to the same DBMS family as the current DBMS
Visibility	Allows to define a synonym as public (accessible to all database users) or private (available to a specific user)
Type	For those DBMS that support it (for example DB2) you can create an alias instead of a synonym. In PowerDesigner synonyms and aliases are managed in the same way whereas their behavior in the database may be different

☞ For more information on aliases, see the DB2 documentation.

Creating a view for a synonym

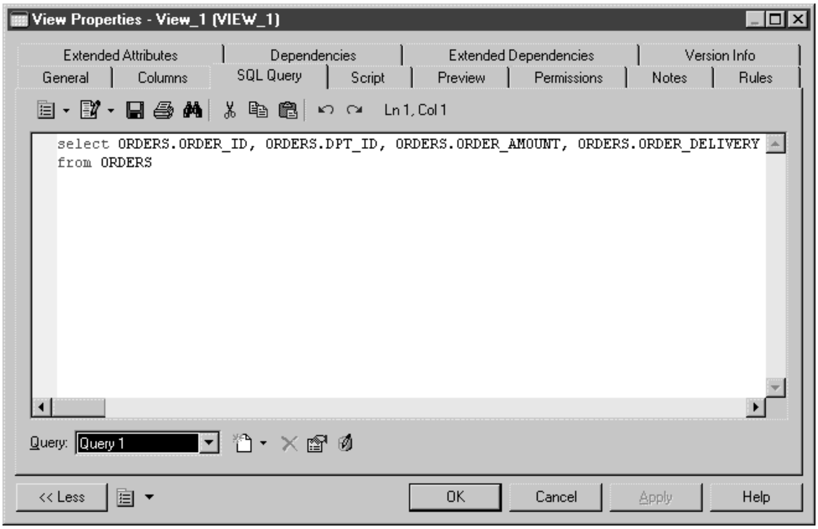
You can create views for synonyms in the same way as you create views for tables. The view query displays the content of the object used for the synonym. For example, the ORDERS_PROD_DEPT table has a synonym ORDERS:



ORDERS_PROD_DEPT

Order_ID <pk>
Dpt_ID
Order_Amount
Order_Delivery

If you create a view for the ORDERS synonym, the view query displays the select order of the table content:



```
select ORDERS.ORDER_ID, ORDERS.DPT_ID, ORDERS.ORDER_AMOUNT, ORDERS.ORDER_DELIVERY
from ORDERS
```

Query: Query 1

❖ To create a view from a synonym

1. Ensure that no objects are selected in the diagram and select Tools ➤ Create View to open a selection box listing all the available objects in the model.

2. Click the Synonyms tab and select one or more synonyms to add to the view.

3. Click OK. The view is created in the diagram.

☞ For more information about creating views, see “Views (PDM)” in the Building Physical Diagrams chapter.

Generating synonyms

You can generate synonyms from the Database Generation dialog box. For more information, see “Generating a Database” section in the “Generating a Database from a PDM” chapter.

Reverse engineering synonyms

PowerDesigner supports the reverse engineering of synonyms.

When you reverse engineer synonyms, the link with the base object is preserved if both objects are reverse engineered and if the base object is displayed before the synonym in the script.

You can reverse a synonym without its base object; in this case you should not forget to define a base object for the synonym.

For more information about reverse engineering, see the Reverse Engineering a Database into a PDM chapter.

CHAPTER 7

Building Web Services

About this chapter

This chapter describes how to create, generate and reverse engineer web services in the following databases:

- Sybase Adaptive Server Anywhere 9 and over
- Sybase Adaptive Server Enterprise 15 and over
- Sybase IQ12.6 and over
- IBM DB2 v8.1 and over

Contents

Introducing Web Services

Web services are applications stored on web servers that you can access remotely through standard web protocols (HTTP, SOAP) and data formats (HTML, XML. . .), whatever the systems and programming languages.

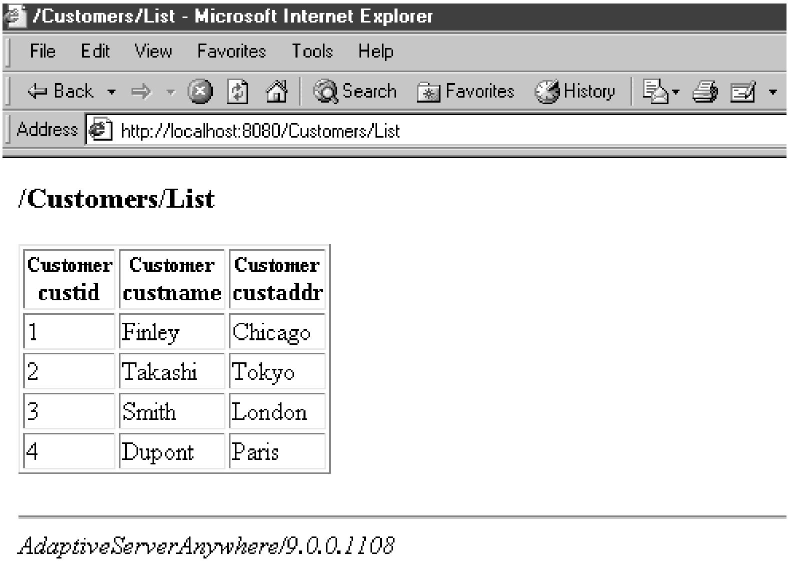
In SOAP requests, queries are encapsulated into services, whereas in HTTP requests, operations are invoked directly. In PowerDesigner, you can design web services for both protocols.

PowerDesigner supports web services for the following databases:

- Sybase Adaptive Server Anywhere 9 and over
- Sybase Adaptive Server Enterprise 15 and over
- Sybase IQ12.6 and over
- IBM DB2 v8.1 and over

Web services can be used to simplify access to databases. If you use web services to query databases, you no longer need drivers to communicate with those databases.

In the following example, you can see the result of an HTTP request for a database web service:



http://localhost:8080/Customers/List

/Customers/List

Customer custid	Customer custname	Customer custaddr
1	Finley	Chicago
2	Takashi	Tokyo
3	Smith	London
4	Dupont	Paris

AdaptiveServerAnywhere/9.0.0.1108

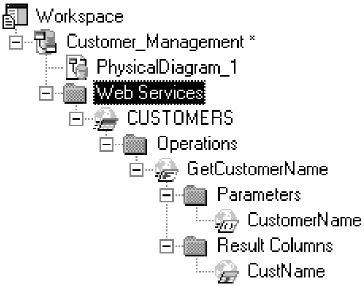
Web services are made of a set of operations. Each operation contains a SQL query for retrieving data from a database.

Web Services (PDM)

In PowerDesigner, web services are made of web operations which themselves contain web parameters and result columns:

- **Web operations** specify the SQL statements used to retrieve data from databases
- **Web parameters** are the parameters which appear in the SQL statements
- **Result columns** are the columns in which the results are displayed

These objects have no symbols, but they appear in the Browser tree view.



```
Workspace
  Customer_Management *
    PhysicalDiagram_1
    Web Services
      CUSTOMERS
        Operations
          GetCustomerName
            Parameters
              CustomerName
            Result Columns
              CustName
```

This structure is compatible with the definition of web services in the supported databases.

Import Web service as service provider

You can import a web service as a service provider into a Business Process Model (BPM) to define the links between a concrete implementation of service interfaces and operations and their abstract definition.

☞ For more information, see “Importing a service provider from an OOM or a PDM” in the Building Business Process Diagrams chapter of the Business Processs Modeling guide.

Web services in Sybase ASA, ASE, and IQ

PowerDesigner supports web services for ASA 9 and over, ASE 15 and over and IQ 12.6 and over. You must specify the type of the web service in the Service type list on the General tab of its property sheet (see “Web service properties” on page 313).

Web services can be invoked by either of two protocols:

- A web service invoked via an HTTP request can have a RAW, HTML or XML type.

  When several web services concern the same table in a database, their name usually starts with the name of the table, followed by a slash and a specific name identifying the query (e.g. Customer/List, Customer/Name). In that case, the name of the table is called the **local path** (which is defined on the General tab of the web service property sheet).

PowerDesigner treats HTTP web operations which share a local path as belonging to the web service with that local path name.

- [ASA and IQ only] A web service invoked in a SOAP request can have a SOAP or a DISH type.

  PowerDesigner treats SOAP web services for these databases as web operations belonging to a DISH web service.

Implementation (SQL statement)

When you create a web service, you must type a **SQL statement** to select which data you want to retrieve from the database in the **Implementation** tab of the property sheet of its web operation(s). For DISH web services, SQL statements are defined in the SOAP web services bearing their prefix name.

Web services in IBM DB2

PowerDesigner supports web services for IBM DB2 v8.1 and over.

In IBM DB2, web services are defined by Document Access Definition Extension (DADX) files.

☞ For more information about generating DADX files, see "Generating web services for IBM DB2 v8.1" on page 324.

A DADX file specifies a web service through a set of **operations** defined by **SQL statements** or Document Access Definition (DAD) files, which specify the mapping between XML elements and DB2 tables.

☞ For more information on DAD files, see "Generating a DAD file for IBM DB2", in the Exchanging Data with Databases Supporting XML" chapter of the *XML* Modeling guide.

Creating a web service

You can create a web service in any of the following ways:

- Select Model ➤ Web Services to access the List of Web Services, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤ Web Service.

☞ For general information about creating objects, see the Objects *Core Features Guide* .

Web service properties

You can modify an object's properties from its property sheet. To open a web service property sheet, double-click its Browser entry in the Web Services folder. The following sections detail the property sheet tabs that contain the properties most commonly entered for web services.

The General tab contains the following properties:

Property	Description
Name	Name of the web service. Used in URIs to access the web service. It can neither start with a slash nor contain two consecutive slashes
Code	Code of the web service
Comment	Descriptive label for the web service
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined
Local path	Name prefixing the web service. If you type a path, the **User-Defined** tool (beside the Local path box) is pushed-in. Click the User-Defined tool to recover the original path. The default value is the name of the web service

Property	Description
Service type	[ASA, ASE, and IQ only] Specifies the type of web service. You can choose from: ♦ **DISH** - [ASA and IQ only] acts as a proxy for a group of SOAP services and generates a **WSDL** (Web Services Description Language) file for each of its SOAP services. When you create a DISH service, you must specify a **prefix** name (on the Extended Attributes tab) for all the SOAP services to which the DISH service applies. PowerDesigner treats SOAP web services as **Web operations** (see "Web Operations (PDM)" on page 316) of DISH web services. ♦ **HTML** – [ASA and IQ only] the result of the SQL statement or procedure is formatted as an HTML document (with a table containing rows and columns). ♦ **RAW** - the result of the SQL statement or procedure is sent without any additional formatting. ♦ **SOAP** - [ASE only] generates a **WSDL** (Web Services Description Language) file. ♦ **XML** - the result of the SQL statement or procedure is sent in XML. By default, the result is converted into XML RAW format.

Web service property sheet Operations tab

This tab lists the web operations associated with the web service (see "Web Operations (PDM)" on page 316).

Web service property sheet Sybase tab

Only available with ASA, ASE, and IQ.

This tab displays the following properties:

Property	Description
Port number	Specifies the port number for the test URL.
Server name	Specifies the server name for the test URL.
Name prefix (dish services only)	[ASA and IQ only] Specifies the name prefix for DISH services.
Database name	[ASE only] Specifies the database name for the test URL.

Web service property sheet Security tab

Only available with ASA and IQ.

This tab displays the following properties:

Property	Description
Secured connection	If selected, only HTTPS connections are accepted. If cleared, both HTTP and HTTPS connections are accepted
Required authorization	If selected, all users must provide a name and a password. When cleared, a single user must be identified
Connection User	When authorization is required, you can select <None> or a list of user names. When authorization is not required, you must select a user name. Default value is <None>, which means all users are granted access

Web service property sheet Namespaces tab

Only available with IBM DB2.

This tab displays a list of namespaces with their prefix, URI and comment.

An XML Schema can be specified where elements and data types used in web parameters and result columns are defined.

Web Operations (PDM)

A web operation is a child object of a web service. It allows you to define the SQL statement of a web service and to display its parameters and result columns.

Creating a web operation

You can create a web operation as follows:

- ♦ Open the Operations tab in the property sheet of a web service, and click the Add a Row tool.

☞ For general information about creating objects, see the Objects *Core Features Guide* .

Web operation properties

You can modify an object's properties from its property sheet. To open a web operation property sheet, double-click its Browser entry in the Operations folder beneath the parent web service. The following sections detail the property sheet tabs that contain the properties most commonly entered for web operations.

The General tab contains the following properties:

Property	Description
Name	Name of the web operation. In URIs, it comes after the name of the web service followed by a slash. It can neither start with a slash nor contain two consecutive slashes
Code	Code of the web operation
Comment	Descriptive label for the web operation
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined
Web Service	Code of the web service containing the web operation. You can click the Properties tool (beside the web Service box) to display the web service property sheet
Owner	[ASE 15 only] Specifies the owner of the operation.

Property	Description
Operation Type	[IBM DB2 only] Specifies the type of operation. You can choose from the following: ♦ **call** - invokes a stored procedure with parameters and result columns for the web operation ♦ **query** - retrieves relational data using the SQL select statement in the Implementation tab ♦ **retrieveXML** - retrieves an XML document from relational data. The mapping of relational data to XML data is defined by a DAD file with SQL or RDB as MappingType ♦ **storeXML** - stores an XML document as relational data. The mapping of XML data to relational data is defined by a DAD file, with RDB as MappingType ♦ **update** - executes the SQL update statement with optional parameters. Parameters can be created from the Parameters tab in the web operation property sheet

Web operation property sheet Parameters tab

This tab lists the web parameters associated with the web operation. These parameters are part of the SQL statement defined on the Implementation tab. They can be created on this tab, before generating a web service, or reverse engineered from a web service (ASA, ASE, and IQ only).

You can use the following tools to add parameters on this tab:

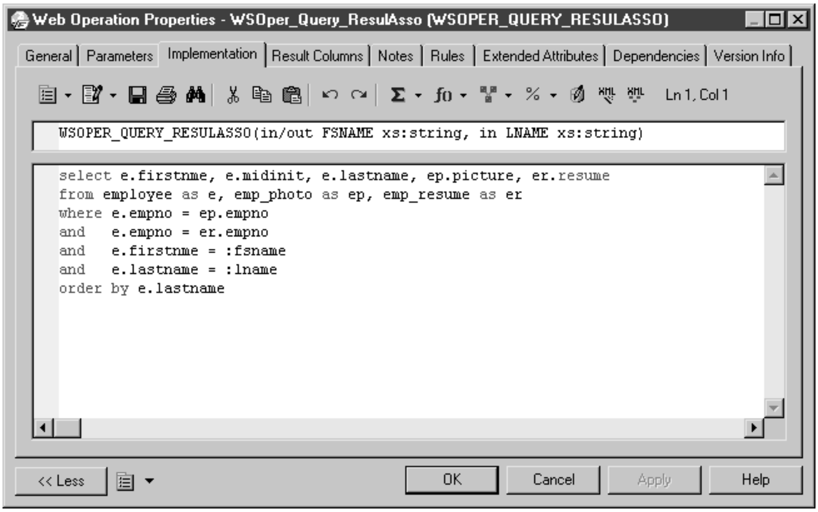
Tool	Name	Description
▦	Add a Row	Adds a parameter to the end of the list.
▦	Insert a Row	Inserts a parameter above the current line in the list.
▦	Add Parameters from SQL Implementation	[ASA, ASE, and IQ only] Display the parameters resulting from the reverse engineering of the web service .

Web operation property sheet Implementation tab

This tab displays the **SQL statement** for the web service. You can use the following tools to type the SQL statement:

Tool	Name	Description
Σ ▾	Operators	Provides logical operators
fo ▾	Functions	Provides group, number, string, date, conversion and other functions
▾	Macros	Provides macros to accelerate the creation of a SQL statement
% ▾	Variables	Provides variables for use with operators and functions
	Edit with SQL Editor	Opens the SQL Editor dialog box. Provides object types and available objects to insert in the SQL statement
XML	SQL/XML Wizard	Opens the SQL/XML Wizard to build a SQL/XML query from a table or a view, and insert it in the SQL statement
XML	Insert SQL/XML Macro	Opens a dialog box to select a global element in an XML model open in the workspace with the SQL/XML extended model definition. Inserts a SQLXML macro referencing the selected element in the SQL statement

For example:



```
WSOPER_QUERY_RESULASSO(in/out FSNAME xs:string, in LNAME xs:string)
```

```
select e.firstnme, e.midinit, e.lastname, ep.picture, er.resume
from employee as e, emp_photo as ep, emp_resume as er
where e.empno = ep.empno
and   e.empno = er.empno
and   e.firstnme = :fsname
and   e.lastname = :lname
order by e.lastname
```

Web operation property sheet Result Columns tab

This tab lists the result columns associated with the web operation (see "Web Operation Result Columns" on page 320).

You can use the following tools to add result columns on this tab:

Tool	Name	Description
▦	Add a Row	Adds a result column to the end of the list.
▦	Insert a Row	Inserts a result column above the current line in the list.
▦	Add Result Columns from Executing SQL Statement	Display the result columns resulting from the execution of the SQL statement in the database.

Web operation property sheet Security tab

Only available with ASA, and IQ.

This tab displays the following properties:

Property	Description
Secured connection	If selected, only HTTPS connections are accepted. If cleared, both HTTP and HTTPS connections are accepted
Required authorization	If selected, all users must provide a name and a password. When cleared, a single user must be identified
Connection User	When authorization is required, you can select <None> or a list of user names. When authorization is not required, you must select a user name. Default value is <None>, which means all users are granted access

Web operation property sheet Sybase tab

Only available with ASE.

This tab displays the following properties:

Property	Description
Alias	Specifies the user-defined database alias.
Secure	Specifies the form of security. You can choose between: ♦ Clear – Use standard HTTP ♦ SSL – Use HTTPS

Web Operation Result Columns

Result columns are sub-objects of web operations. They are part of the SQL statement defined in the Implementation tab of a web operation property sheet, and belong to a table in the target database. They are listed in the Result Columns tab of a web operation property sheet.

The General tab of a result column property sheet displays the following properties:

Property	Description
Name	Name of the result column
Code	Code of the result column
Comment	Descriptive label for the result column
Data Type	[IBM DB2 only] Select an XML schema data type from the list, or click the **Select Object** tool to open a selection dialog box where you select a global element in an XML model open in the workspace
Is element	[IBM DB2 only] Checked and greyed when a global element is attached to a result column

Web Parameters (PDM)

Web parameters are child objects of web operations. They are part of the SQL statement defined in the Implementation tab of a web operation property sheet. They are listed in the Parameters tab of a web operation property sheet.

Creating a web parameter

You can create a web parameter in any of the following ways:

- Open the Parameters tab in the property sheet of a web operation, and click the Add a Row tool.
- Right-click a web operation in the Browser, and select New ➤ Web Parameter.

☞ For general information about creating objects, see the Objects *Core Features Guide* .

Web parameter properties

You can modify an object's properties from its property sheet. To open a web parameter property sheet, double-click its Browser entry in the Parameters folder beneath its parent web operation.

The General tab contains the following properties:

Property	Description
Name	Name of the web parameter.
Code	Code of the web parameter.
Comment	Descriptive label for the web parameter.
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined.
Web Operation	Name of the web operation containing the web parameter.
Parameter Type	Select **in** if you want the web parameter to be an input parameter. Select **in/out** if you want the web parameter to be both an input and output parameter. Select **out** if you want the web parameter to be an output parameter.

Property	Description
Default Value	[ASE only] Specifies a default value for the parameter.
Data Type	[For IBM DB2] Select an XML schema data type from the list, or click the **Select Object** tool to open a selection dialog box where you select a global element in an XML model open in the workspace. [For ASE] Select a datatype from the list.
Is element	[IBM DB2 only] Checked and greyed when a global element is attached to a web parameter.

Testing Web Services

PowerDesigner provides a method for testing web services within the model environment. You must be connected to the appropriate database.

❖ To test a web service

1. Right-click the browser entry for a web service of type DISH or SOAP and select Show WSDL

   or

   Right-click the browser entry for a web service operation belonging to a web service of another type and select Test Web Service Operation from the contextual menu.

2. Review the generated URL and then click OK.

   For a web service of type SOAP, the WSDL file will be displayed in your browser

   or

   For a web service of type RAW, the result will be displayed in your browser

Generating Web Services

You generate web services in order to implement them on target databases.

Generating web services for Sybase ASA, ASE, and IQ

You can generate database web services to a script or to a live database connection.

❖ To generate web services for Sybase ASA, ASE, and IQ

1. Select Database ➤ Generate Database to open the Database Generation window, and specify the standard options, including whether you want to generate to a script or to a live database connection.

   ☞ For detailed information about using this window, see “Generating a Database” section in the “Generating a Database from a PDM” chapter.

2. [optional] Click the Options tab and click on Web Service in the left-hand pane to display the web service generation options. Change the default options as appropriate.

3. [optional] Click the Selection tab and select the Web Services subtab at the bottom of the tab. Select the web services that you want to generate.

4. Click OK to begin the generation.

Note that for web services generated to a live database connection, you may have to refresh the Web Services folder before they appear.

Generating web services for IBM DB2 v8.1

In IBM DB2, web services are defined by Document Access Definition Extension (DADX) files.

A PDM with the appropriate **extended model definition** allows you to generate a DADX file for each web service defined for IBM DB2.

You can attach the **DADX** extended model definition at model creation. In the New dialog box, once you have selected Physical Data Model in the Model type list, and IBM DB2 UDB 8.x Common Server in the DBMS list, click the Extended Model Definitions tab and select DADX.

❖ To generate DADX files for IBM DB2

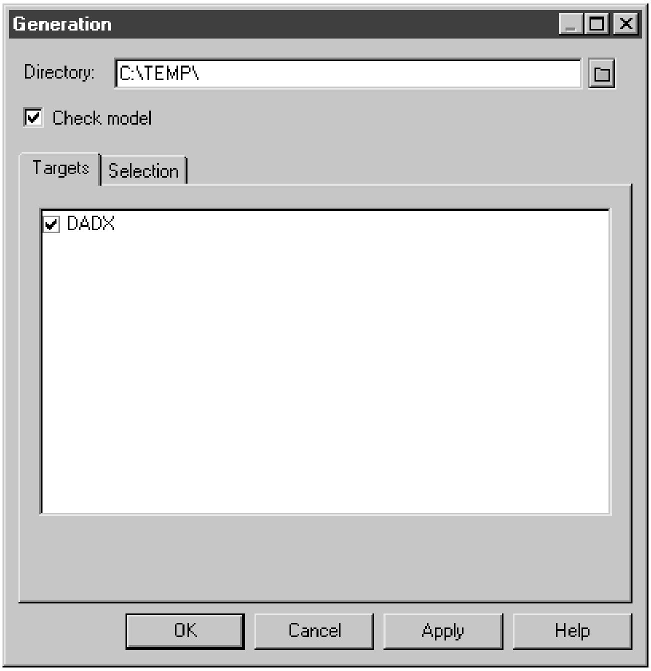
1. Select Model ➤ Extended Model Definitions to open the List of Extended Model Definitions.

2. Click the **Import an Extended Model Definition** tool to open the Extended Model Definition Selection dialog box, select **DADX** in the list of extended model definitions and click OK.

   DADX is displayed in the List of Extended Model Definitions.

3. Click OK to return to the main window.

4. Select **Tools ➤ Extended Generation** to open the Generation dialog box with DADX selected in the Targets tab.



Generation

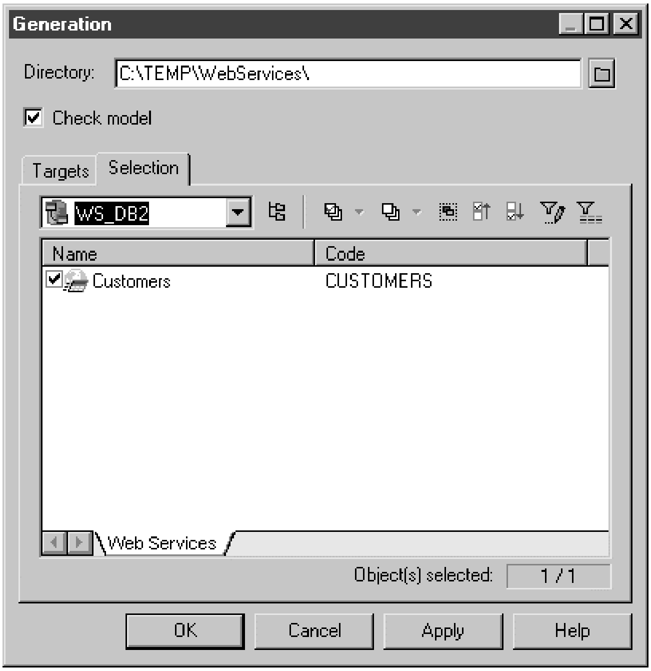
Directory: C:\TEMP\

☑ Check model

☑ DADX

5. Click the **Select a Path** button, to the right of the Directory box, and specify a path for the DADX files.

6. Click the Selection tab, and select the web services for which you want to generate a DADX file.



Directory: C:\TEMP\WebServices\

☑ Check model

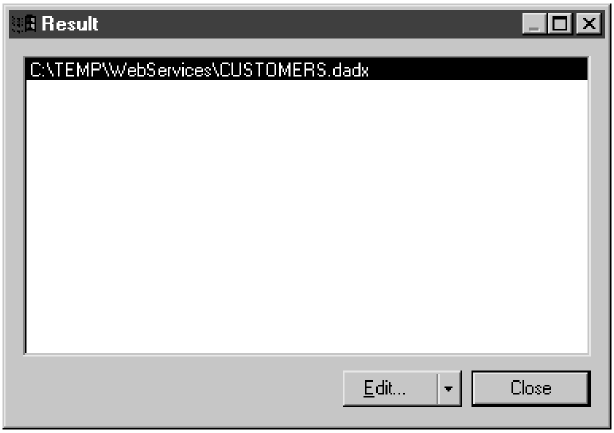
WS_DB2

Name	Code
☑ Customers	CUSTOMERS

Object(s) selected: 1 / 1

7. Click OK to begin generation.

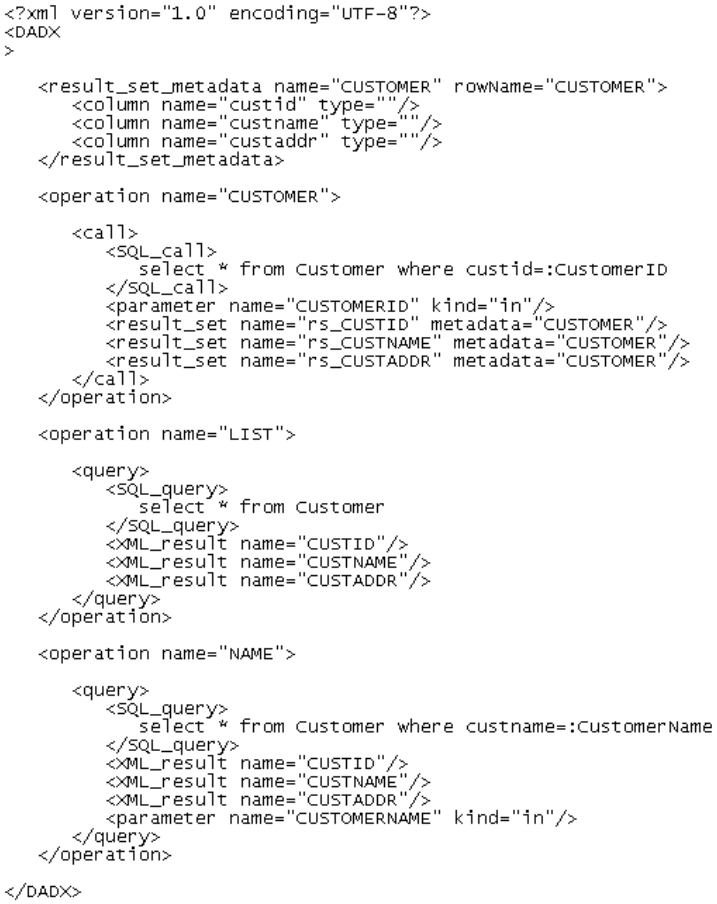
   When generation is complete, the Result dialog box will be displayed with the paths of the DADX files.



C:\TEMP\WebServices\CUSTOMERS.dadx

8. Select the path of a DADX file and click Edit.

   The DADX file is displayed in the editor window.



```
<?xml version="1.0" encoding="UTF-8"?>
<DADX
>

   <result_set_metadata name="CUSTOMER" rowName="CUSTOMER">
      <column name="custid" type=""/>
      <column name="custname" type=""/>
      <column name="custaddr" type=""/>
   </result_set_metadata>

   <operation name="CUSTOMER">

      <call>
         <SQL_call>
            select * from Customer where custid=:CustomerID
         </SQL_call>
         <parameter name="CUSTOMERID" kind="in"/>
         <result_set name="rs_CUSTID" metadata="CUSTOMER"/>
         <result_set name="rs_CUSTNAME" metadata="CUSTOMER"/>
         <result_set name="rs_CUSTADDR" metadata="CUSTOMER"/>
      </call>
   </operation>

   <operation name="LIST">

      <query>
         <SQL_query>
            select * from Customer
         </SQL_query>
         <XML_result name="CUSTID"/>
         <XML_result name="CUSTNAME"/>
         <XML_result name="CUSTADDR"/>
      </query>
   </operation>

   <operation name="NAME">

      <query>
         <SQL_query>
            select * from Customer where custname=:CustomerName
         </SQL_query>
         <XML_result name="CUSTID"/>
         <XML_result name="CUSTNAME"/>
         <XML_result name="CUSTADDR"/>
         <parameter name="CUSTOMERNAME" kind="in"/>
      </query>
   </operation>

</DADX>
```

9. Click Close in the Result dialog box.

You can now use the DADX files for SOAP requests in IBM DB2 UDB web services Object Runtime Framework (WORF).

Reverse Engineering Web Services

You reverse engineer web services from a database to a PDM, when you want to reuse these web services in the PDM. Once reverse engineered, you can modify and generate them in the database.

You can only reverse engineer web services from Sybase ASA, ASE, and IQ.

You can reverse engineer web services into a new or existing PDM from a script or live database connection via the Database Reverse Engineering dialog box.

☞ For information about using the Database Reverse Engineering dialog box, see the "Reverse Engineering a Database into a PDM" chapter.

The following list shows how web service objects in these databases are treated in PowerDesigner:

- Database HTTP web services with a common **local path** are grouped as PowerDesigner web operations of an HTTP web service with the specified local path:

Software	Web service name	Type	Web operation name
Database	Customers/Name	HTML	—
PowerDesigner	Customers	HTML	Name

- Database HTTP web services without a common local path are grouped as PowerDesigner web operations of an HTTP web service named raw, xml or html:

Software	Web service name	Type	Web operation name
Database	Customers	HTML	—
PowerDesigner	html	HTML	Customers

- Database SOAP web services with a **prefix name** are considered as PowerDesigner web operations of a DISH web service with the prefix name:

Software	Web service name	Type	Web operation name
Database	DishPrefix/Name	SOAP	—
PowerDesigner	Customers (with DishPrefix as prefix)	DISH	Name

♦ Database SOAP web services without a prefix name are considered as PowerDesigner web operations of a DISH web service without a prefix name:

Software	Web service name	Type	Web operation name
Database	Customers	SOAP	—
PowerDesigner	WEBSERVICE_1	DISH	Customers

♦ Database DISH web services with or without a prefix name are considered identically in PowerDesigner:

Software	Web service name	Type	Web operation name
Database	Customers	DISH	—
PowerDesigner	Customers (with or without DishPrefix as prefix)	DISH	—

CHAPTER 8

Working with Data Models

About this chapter

This chapter describes how to work with data models.

Contents

Customizing the Data Modeling Environment

The PowerDesigner data modeling environment includes a set of parameters and configuration options that define various aspects of the model content and behavior. You can set these parameters:

- At model creation
- After creating a model with default options and parameters
- When creating a model template

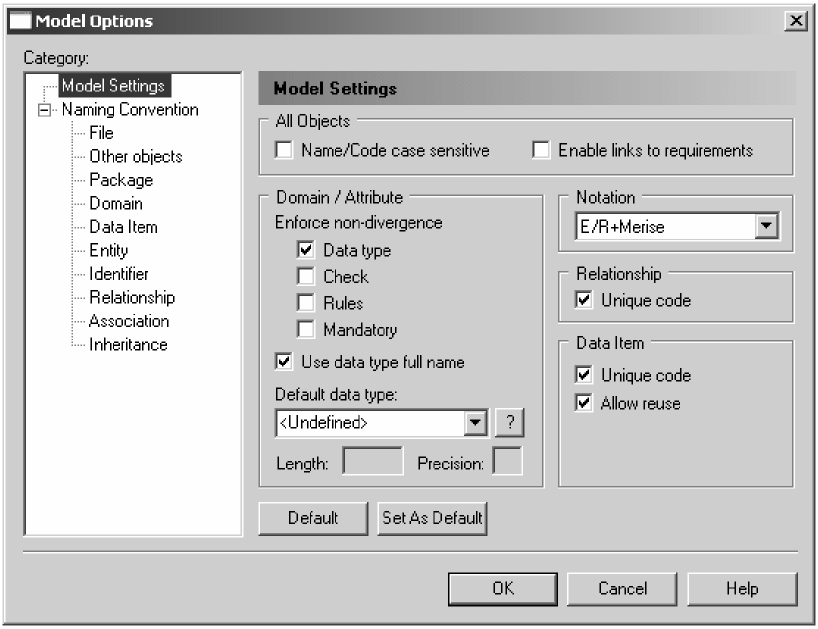
Setting CDM/LDM model options

This section explains how to set global options for the objects in your CDM/LDM. These options apply only to the current CDM/LDM.

For information about controlling the naming conventions of your models, see “Naming Conventions” section in the Models chapter of the *Core Features Guide* .

Setting Model Settings

To set Model Settings, select Tools ➤ Model Options or right-click the diagram background and select Model Options from the contextual menu.



Model Settings

All Objects
□ Name/Code case sensitive
□ Enable links to requirements

Domain / Attribute
Enforce non-divergence
☑ Data type
□ Check
□ Rules
□ Mandatory
☑ Use data type full name
Default data type: <Undefined>
Length:
Precision:

Notation: E/R+Merise

Relationship
☑ Unique code

Data Item
☑ Unique code
☑ Allow reuse

The options on this tab affect all the objects in the model, including those already created, while changes to the object-specific options on the sub-category tabs only affect objects created subsequently.

You can set the following options on this tab:

Option	Description
Name/Code case sensitive	Specifies that the names and codes for all objects are case sensitive, allowing you to have two objects with identical names or codes but different cases in the same model. If you change case sensitivity during the design process, we recommend that you check your model (Tools ➤ Check Model) to verify that your model does not contain any duplicate objects.
Enable links to requirements	Displays a Requirements tab in the property sheet of every object in the model, which allows you to attach requirements to objects in your model. These attached requirements are kept synchronized with your requirements model. For more information, see the *Requirements Modeling* guide.
Enforce non-divergence	Enforces non-divergence between a domain definition and the attributes using the domain. You can select any or all of the following attribute properties: ♦ Data type - Data type, length, and precision ♦ Check - Check parameters, such as minimum and maximum ♦ Rules - Business rules ♦ Mandatory - Attribute mandatory property When you apply these options, you are asked if you want to apply domain properties to attributes currently attached to the domain. If you click **OK**, the attribute properties are modified for consistency with the domain. When you modify the properties of a domain, the properties of the attributes attached to it are updated provided they are selected here. When you select an attribute property under Enforce non-divergence, that property cannot be modified in the lists of attributes and the property sheets of attributes. If you want to modify an attribute property that is defined as non-divergent, you must detach the attribute from its domain, or clear the appropriate Enforce non-divergence model option.

Option	Description
Use data type full name	Specifies that the complete data type is displayed in entity symbols.
Default data type	Specifies a default data type to apply to domains and attributes if none is selected for them.
Notation	You can choose between the following notations: ♦ Entity / Relationship [Default – used throughout this manual] Entity/relationship notation connects entities with links representing one of four relationships between them. These relationships have properties that apply to both entities involved in the relationship ♦ Merise - uses associations instead of relationships ♦ E/R + Merise - both entity/relationship and Merise are used in the same model ♦ IDEF1X - data modeling notation for relationships and entities. In this notation, each set of relationship symbols describes a combination of the optionality and cardinality of the entity next to it ♦ Barker – inheritances are represented by placing the child entities inside the parent entity symbol, and relationships are drawn in two parts, each reflecting the multiplicity of the associated entity role. For more information about these notations, see "Supported CDM/LDM notations" on page 361
Unique code	Requires that data items or relationships have unique codes
Allow n-n relationships	[LDM only] Allows n-n relationships to be displayed.

Option	Description
Allow reuse	Allows the reuse of one data item as an attribute for more than one entity provided the attributes have same name and data type and do not belong to a primary key. When deselected or when the attribute belongs to a primary key, the data item cannot be reused. In this case, if the Unique code check box is selected, a new data item with identical name but different code is created, otherwise a new data item with identical name and code is created. When you delete an entity or entity attributes, these options determine whether or not the corresponding data items are also deleted, as follows: ♦ Both – deletes the entity attribute. ♦ Unique Code only – deletes the entity attribute. ♦ Allow Reuse only – deletes the entity attribute and the corresponding data item (if it is not used by another entity). ♦ None – deletes the entity attribute and the corresponding data item.

Setting Migration Settings (LDM only)

To set migration settings, select Tools ➤ Model Options, and select the Migration settings sub-category in the left-hand Category pane.

Option	Description
Migrate attribute properties	Enables the domain, the checks or the rules to be kept when an attribute is migrated.

Option	Description
Foreign attribute name	Specifies the naming convention for migrated foreign identifiers. You can select one of the default templates from the list or enter your own using the following variables: ♦ %PARENT% - Name/Code of the parent entity ♦ %ATTRIBUTE% - Name/Code of the parent attribute ♦ %IDENTIFIER% - Name/Code of the identifier constraint attached to the relationship ♦ %RELATIONSHIP% - Name/Code of the relationship ♦ %PARENTROLE% - Role of the entity that generated the parent entity, this variable proceeds from the conceptual environment. If no role is defined on the relationship, %PARENTROLE% takes the content of %PARENT% to avoid generating an attribute with no name The following example checks the %PARENTROLE% value; if it is equal to the parent name (which is the replacement value) then the template “%.3:PARENT%_%ATTRIBUTE%” will be used, otherwise template “%PARENTROLE% will be used because the user has entered a parent role for the relationship: Note that customized naming templates reappear in the generation dialog box the next time you open it, but are not saved to the list of predefined templates.
Use template	Controls when the primary identifier attribute name template will be used. You can choose between the following radio buttons: ♦ Always use template. ♦ Only use template in case of conflict.

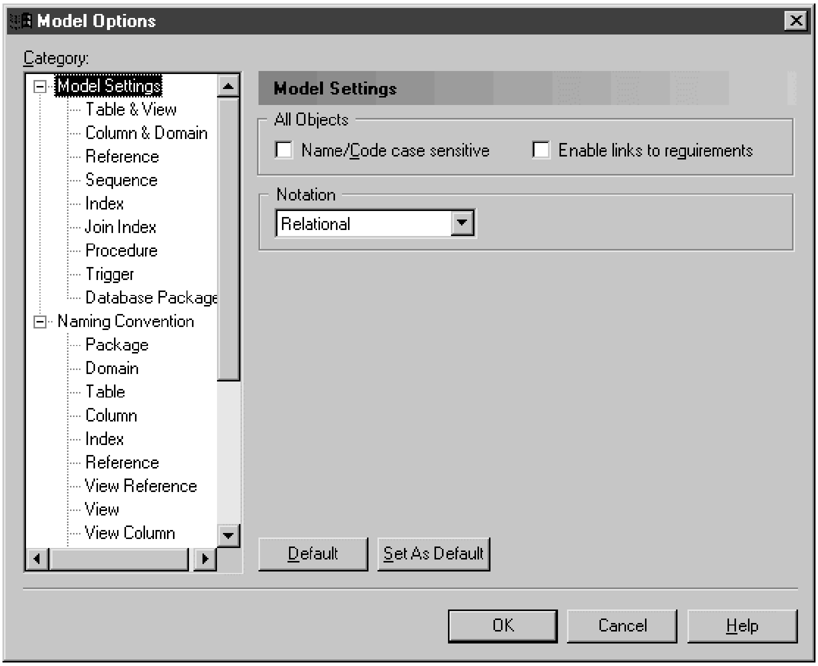
These options are available for relationships and automatic for inheritances, except the Only use template in case of conflict option, which is always triggered.

Setting PDM Model Options

This section explains how to set global options for the objects in your PDM. For information about controlling the naming conventions of your models, see “Naming Conventions” in the Models chapter of the *Core Features Guide* .

Setting Model Settings

To set Model Settings, select Tools ➤ Model Options or right-click the diagram background and select Model Options from the contextual menu.



Model Options

Category:

- Model Settings
  - Table & View
  - Column & Domain
  - Reference
  - Sequence
  - Index
  - Join Index
  - Procedure
  - Trigger
  - Database Package
- Naming Convention
  - Package
  - Domain
  - Table
  - Column
  - Index
  - Reference
  - View Reference
  - View
  - View Column

Model Settings

All Objects

□ Name/Code case sensitive □ Enable links to requirements

Notation

Relational

The options on this tab affect all the objects in the model, including those already created, while changes to the object-specific options on the sub-category tabs only affect objects created subsequently.

You can set the following options on this tab:

Option	Function
Name/Code case sensitive	Specifies that the names and codes for all objects are case sensitive, allowing you to have two objects with identical names or codes but different cases in the same model. If you change case sensitivity during the design process, we recommend that you check your model (Tools ➤ Check Model) to verify that your model does not contain any duplicate objects.

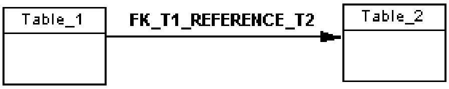
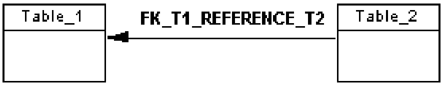
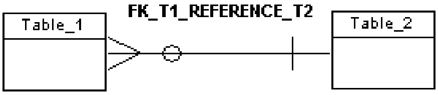
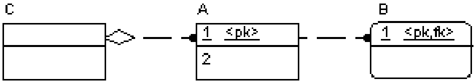
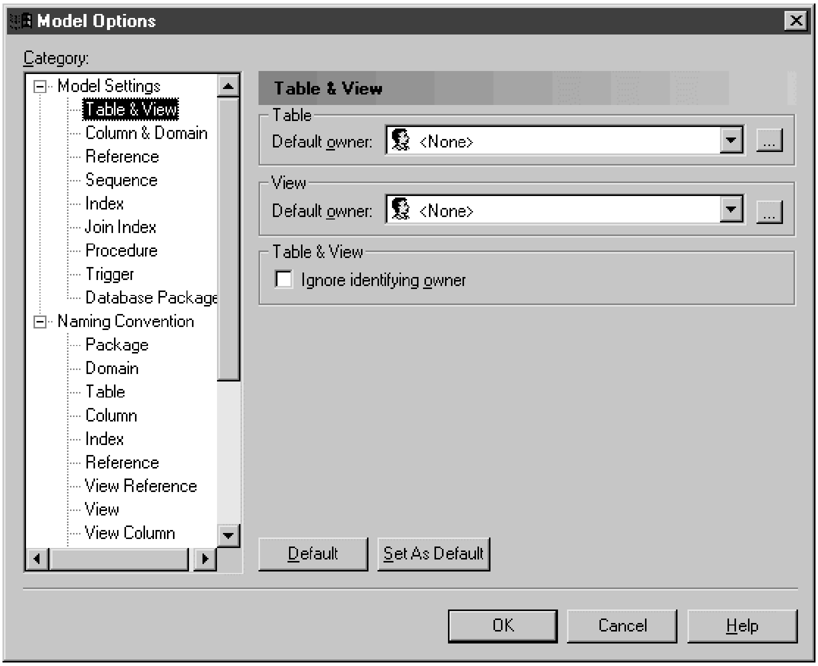
Option	Function
Enable links to requirements	Displays a Requirements tab in the property sheet of every object in the model, which allows you to attach requirements to objects in your model. These attached requirements are kept synchronized with your requirements model. For more information about requirements, see the *Requirements Modeling* guide.
Notation	Specifies the use of one of the following notation types for the model. You can choose between: ♦ Relational - Arrow pointing to primary key. This option is the default, and is used in this manual.  ♦ CODASYL - Arrow pointing to foreign key.  ♦ Conceptual - Cardinality displayed in IE format (crow's feet).  ♦ IDEF1X - Cardinality and mandatory status displayed on reference, primary columns in separate containers and dependent tables with rounded rectangles.  When you change notation, all symbols in all diagrams are updated accordingly. If you switch from Merise to IDEF1X, all associations are converted to relationships.

Table and view model options

To set model options for tables and views, select Tools ➤ Model Options, and select the Table & View sub-category in the left-hand Category pane.



Model Options

Table & View

Table

Default owner: <None>

View

Default owner: <None>

Table & View

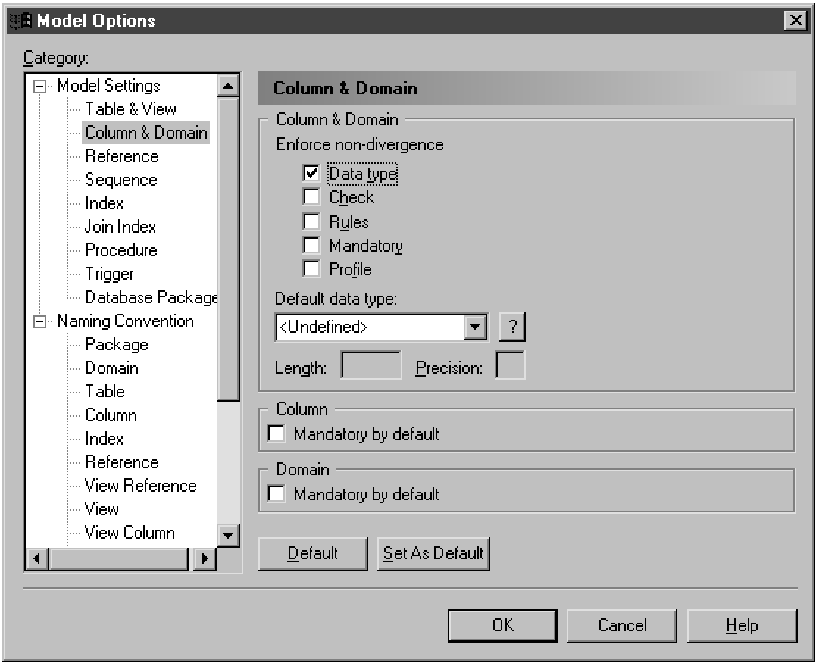
□ Ignore identifying owner

You can set the following options on this tab:

Option	Function
Table: Default owner	Specifies a default owner for the tables in your model from the list of users (see the section "Creating a user" in the Managing Database Users chapter). To create a User, click on the ellipsis button to open the List of Users, and click the Add a Row tool. If the user specified is subsequently deleted, this option (and the ownership of all associated objects) will be reset to none.
View: Default owner	Specifies a default owner for the views in your model from the list of users (see the section "Creating a user" in the Managing Database Users chapter). To create a User, click on the ellipsis button to open the List of Users, and click the Add a Row tool. If the user specified is subsequently deleted, this option (and the ownership of all associated objects) will be reset to none.
Ignore identifying owner	Specifies that the owner of a table or view is ignored for identification purposes. Since, by default, both the name/code and the owner are considered during a uniqueness check, this option enables you to enforce distinct names for these objects. For example, if a model contains a table called "Table_1", which belongs to User_1, and another table, also called "Table_1", which belongs to User_2, it will, by default, pass a uniqueness check because of the different owners.

Column and domain model options

To set model options for columns and domains, select Tools ➤ Model Options, and select the Column & Domain sub-category in the left-hand Category pane.



Column & Domain

Column & Domain

Enforce non-divergence

☑ Data type

□ Check

□ Rules

□ Mandatory

□ Profile

Default data type: <Undefined>

Length: Precision:

Column

□ Mandatory by default

Domain

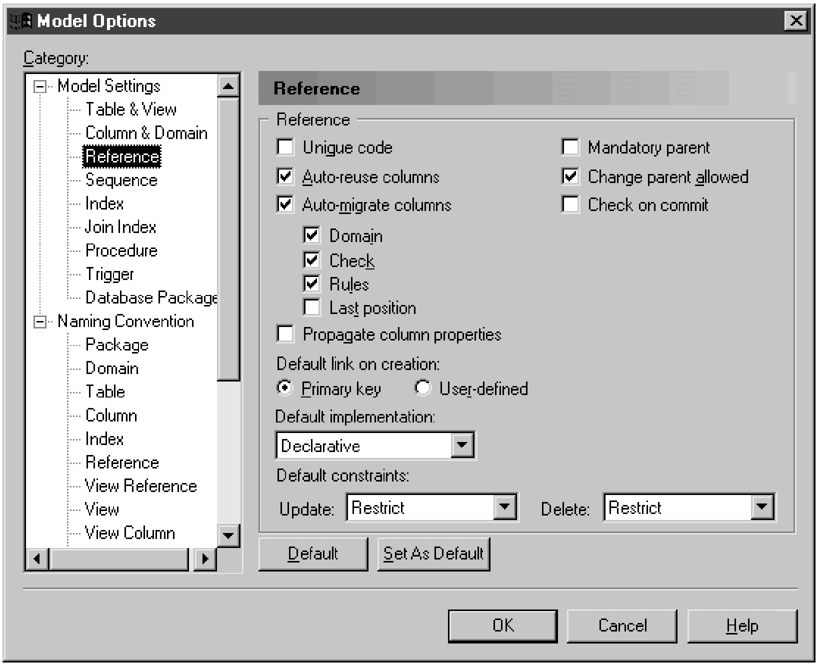
□ Mandatory by default

You can set the following options on this tab:

Option	Function
Enforce non-divergence	Specifies that columns attached to a domain must remain synchronized with the selected properties, any or all of: Data type, Check, Rules, Mandatory, and Profile.
Default data type	Specifies a default data type to be applied to columns and domains.
Column: Mandatory by default	Specifies that columns are created, by default, as mandatory.
Domain: Mandatory by default	Specifies that domains are created, by default, as mandatory.

Reference model options

To set model options for references, select Tools ➤ Model Options, and select the Reference sub-category in the left-hand Category pane.



Reference

- □ Unique code
- ☑ Auto-reuse columns
- ☑ Auto-migrate columns
  - ☑ Domain
  - ☑ Check
  - ☑ Rules
  - □ Last position
- □ Propagate column properties
- □ Mandatory parent
- ☑ Change parent allowed
- □ Check on commit

Default link on creation: ● Primary key ○ User-defined

Default implementation: Declarative

Default constraints:

Update: Restrict Delete: Restrict

You can set the following options on this tab:

Option	Function
Unique code	Requires that references have unique codes. If this option is not selected then different references can have the same code (except when two references share the same child table).
Auto-reuse columns	Enables the reuse of columns in a child table as foreign key columns if the following conditions are satisfied: ♦ Child column has same code as migrating primary key column ♦ Child column is not already a foreign key column ♦ Data types are compatible For more information, see “Automatic reuse and migration of columns” in the Building Physical Diagrams chapter.

Option	Function
Auto-migrate columns	Enables the automatic migration of primary key columns from the parent table as foreign key columns to the child table. If you select both Auto-migrate columns and any of the following sub-options, then the relevant column property of the PK will also be migrated to the FK at reference creation: ♦ Domain ♦ Check ♦ Rules ♦ Last position For more information, see “Automatic reuse and migration of columns” in the Building Physical Diagrams chapter.
Mandatory parent	Specifies that the relationship between child and parent tables is, by default, mandatory, i.e., each foreign key value in the child table must have a corresponding key value, in the parent table.
Change parent allowed	Specifies that a foreign key value can change to select another value in the referenced key in the parent table.
Check on commit	Specifies that referential integrity is checked only on commit, rather than immediately after row insertion. This feature can be useful when working with circular dependencies. Not available with all DBMSs.
Propagate column properties	Propagates changes made to the name, code, stereotype, or data type of a parent table column to the corresponding child column.
Default link on creation	Specifies how reference links are created. You can select either: ♦ Primary key – automatically create links from primary key to foreign key columns at creation ♦ User-defined – manually create your own links

Option	Function
Default implementation	Specifies how referential integrity is implemented in the reference. You can select either: ♦ Declarative – referential integrity is defined by constraint in foreign declarations ♦ Trigger – referential integrity is implemented by triggers For more information on referential integrity, see “Reference property sheet Integrity tab” in the Building Physical Diagrams chapter.
Default Constraints: Update	Controls how updating a key value in the parent table will, by default, affect the foreign key value in the child table. Depending on your DBMS, you can choose from some or all of the following settings: ♦ None – no effect ♦ Restrict – cannot update parent value if one or more matching child values exist (no effect) ♦ Cascade - update matching child values ♦ Set null - set matching child values to NULL ♦ Set default – set matching child values to default value
Default Constraints: Delete	Controls how deleting a key value in the parent table will, by default, affect the foreign key value in the child table. Depending on your DBMS, you can choose from some or all of the following settings: ♦ None – no effect ♦ Restrict – cannot delete parent value if one or more matching child values exist (no effect) ♦ Cascade - delete matching child values ♦ Set null - set matching child values to NULL ♦ Set default – set matching child values to default value

Sequence model options

To set model options for sequences, select Tools ➤ Model Options, and select the Sequence sub-category in the left-hand Category pane.

You can set the following options on this tab:

Option	Function
Default Owner	Specifies a default owner for the sequences in your model from the list of users (see the section “Creating a user” in the Managing Database Users chapter). To create a User, click on the ellipsis button to open the List of Users, and click the Add a Row tool. If the user specified is subsequently deleted, this option (and the ownership of all associated objects) will be reset to none.

Index model options

To set model options for indexes, select Tools ➤ Model Options, and select the Index sub-category in the left-hand Category pane.

You can set the following options on this tab:

Option	Function
Table: Default Owner	Specifies a default owner for the table indexes in your model from the list of users (see the section “Creating a user” in the Managing Database Users chapter). To create a User, click on the ellipsis button to open the List of Users, and click the Add a Row tool. If no default user is specified here, then the owner of the parent table is used by default. If the user specified is subsequently deleted, this option (and the ownership of all associated objects) will be reset to none.
View: Default Owner	Specifies a default owner for the view indexes in your model from the list of users (see the section “Creating a user” in the Managing Database Users chapter). To create a User, click on the ellipsis button to open the List of Users, and click the Add a Row tool. If no default user is specified here, then the owner of the parent view is used by default. If the user specified is subsequently deleted, this option (and the ownership of all associated objects) will be reset to none.

Join index model options

To set model options for join indexes, select Tools ➤ Model Options, and select the Join Index sub-category in the left-hand Category pane.

You can set the following options on this tab:

Option	Function
Default owner	Specifies a default owner for the join indexes in your model from the list of users (see the section “Creating a user” in the Managing Database Users chapter). To create a User, click on the ellipsis button to open the List of Users, and click the Add a Row tool. If the user specified is subsequently deleted, this option (and the ownership of all associated objects) will be reset to none.

Procedure model options

To set model options for procedures, select Tools ➤ Model Options, and select the Procedure sub-category in the left-hand Category pane.

You can set the following options on this tab:

Option	Function
Default owner	Specifies a default owner for the procedures in your model from the list of users (see the section “Creating a user” in the Managing Database Users chapter). To create a User, click on the ellipsis button to open the List of Users, and click the Add a Row tool. If the user specified is subsequently deleted, this option (and the ownership of all associated objects) will be reset to none.

Trigger model options

To set model options for triggers, select Tools ➤ Model Options, and select the Trigger sub-category in the left-hand Category pane.

You can set the following options on this tab:

Option	Function

Option	Function
Table: Default Owner	Specifies a default owner for the table triggers in your model from the list of users (see the section “Creating a user” in the Managing Database Users chapter). To create a User, click on the ellipsis button to open the List of Users, and click the Add a Row tool. If no default user is specified here, then the owner of the parent table is used by default. If the user specified is subsequently deleted, this option (and the ownership of all associated objects) will be reset to none.
View: Default Owner	Specifies a default owner for the view triggers in your model from the list of users (see the section “Creating a user” in the Managing Database Users chapter). To create a User, click on the ellipsis button to open the List of Users, and click the Add a Row tool. If no default user is specified here, then the owner of the parent view is used by default. If the user specified is subsequently deleted, this option (and the ownership of all associated objects) will be reset to none.
Rebuild automatically triggers	Automatically rebuilds the triggers on the child and parent tables of a reference when you: ♦ change the implementation of a reference ♦ change the referential integrity rules of a reference implemented by a trigger ♦ change the child or parent table of a reference implemented by a trigger (new and old) ♦ create or delete a reference implemented by a trigger ♦ change the maximum cardinality of the references Note: If this option is not selected, you can manually instruct PowerDesigner to rebuild triggers at any time by selecting Tools ➤ Rebuild Objects ➤ Rebuild Triggers.

Database package model options

To set model options for database packages, select Tools ➤ Model Options, and select the Database Package sub-category in the left-hand Category pane.

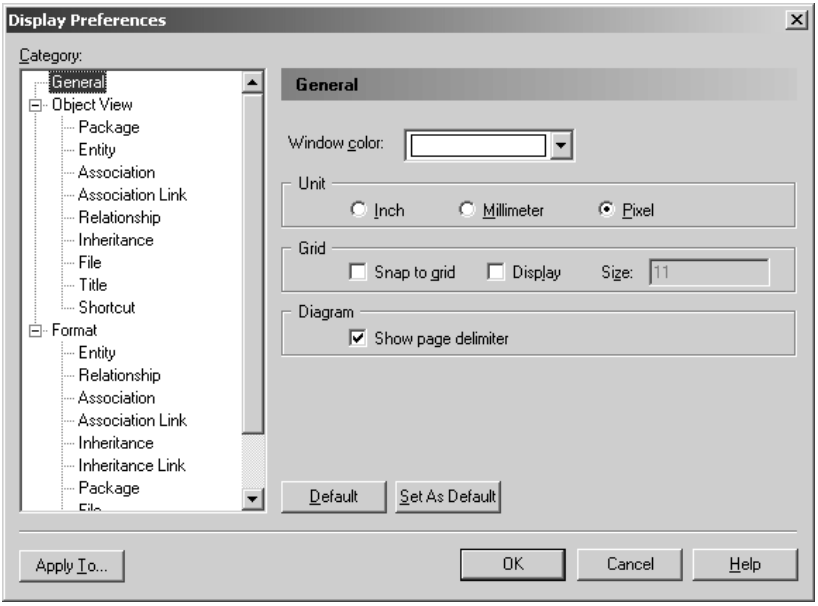
You can set the following options on this tab:

Option	Function
Default owner	Specifies a default owner for the database packages in your model from the list of users (see the section "Creating a user" in the Managing Database Users chapter). To create a User, click on the ellipsis button to open the List of Users, and click the Add a Row tool. If the user specified is subsequently deleted, this option (and the ownership of all associated objects) will be reset to none.

Setting Data Model Display Preferences

PowerDesigner display preferences allow you to customize the format of object symbols, and the information that is displayed on them.

To set a data model display preferences, select Tools ➤ Display Preferences or right-click the diagram background and select Display Preferences from the contextual menu.



Window color:

Unit: ○ Inch ○ Millimeter ● Pixel

Grid: □ Snap to grid □ Display Size: 11

Diagram: ☑ Show page delimiter

For information about changing the format of symbols, see "Format display preferences" in the Customizing your Modeling Environment chapter of the *Core Features Guide* . The following sections list the options available to customize the information displayed on a data model object symbols. Note that the objects available to be customized in the Display Preferences window are dependant upon the current diagram type.

Default options
Click the Default button to display default display preferences. Click the Set As Default button to set current display preferences as default selections.

Association display preferences (CDM)

To set display preferences for associations in a CDM, select Tools ➤ Display Preferences, and select the Association sub-category in the left-hand Category pane.

Association

Preference	Display description
Attributes	Displays the name of the association.
Display Limit	Controls the maximum number of attributes displayed in symbol.
Stereotype	Displays the stereotype of the association.
Comment	Displays the comment of the association. When selected, all other check boxes are deselected, except for Stereotype.

Association attributes

Preference	Display description
Data types	Displays the data types of association attributes.
Replace by domains	Displays the domains of association attributes. You can only display domains when the Data types check box is selected.
Domains	Displays the domain of association attributes. For details of how this and the Data types preferences interact, see "Entity display preferences (CDM/LDM)" on page 351.
Mandatory	Displays the letter *M* beside mandatory association attributes.
Stereotype	Displays the stereotype of association attributes.

Association display preferences (PDM)

To set display preferences for associations in a PDM, select Tools ➤ Display Preferences, and select the Association sub-category in the left-hand

Category pane.

Preference	Description
Name	Displays the name of the association.
Hierarchy	Displays the name of the hierarchy used to compute the cube
Stereotype	Displays the stereotype of the association.

Association Link display preferences (CDM)

To set display preferences for association links, select Tools ➤ Display Preferences, and select the Association Link sub-category in the left-hand Category pane.

Preference	Description
Role	Displays the role of the association link
Cardinality	Displays the cardinality of the association link
Stereotype	Displays the stereotype of the association link

Cube display preferences (PDM)

To set display preferences for cubes, select Tools ➤ Display Preferences, and select the Cube sub-category in the left-hand Category pane.

Preference	Description
Fact Measures	Displays fact measures. When selected, you can additionally specify whether to display all fact measures, or to limit them to a given number.
Fact	Displays cube facts.
Stereotype	Displays the stereotype of the cub. You can also display the stereotype of the cube measures and facts.

Dimension display preferences (PDM)

To set display preferences for dimensions, select Tools ➤ Display Preferences, and select the Dimension sub-category in the left-hand Category pane.

Preference	Description
Attributes	Displays the dimension attributes. When selected, you can additionally specify whether to display all attributes, or to limit them to a given number.
Hierarchies	Displays dimension hierarchies. When selected, you can also specify whether to display <Default> beside the default hierarchy in dimension symbols.
Stereotype	Displays the stereotypes of dimensions. You can also display the stereotype of the dimension attributes and hierarchies.

Dimension Attributes

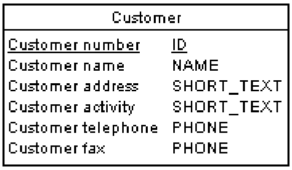
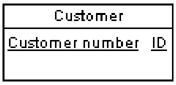
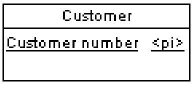
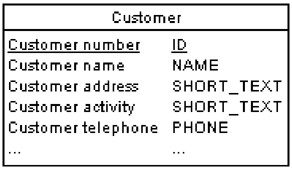
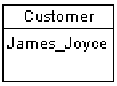
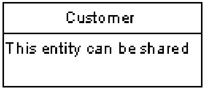
The following display preferences are available for dimension attributes:

Preference	Description
Indicators of Hierarchy	Displays the hierarchies indicators of dimension attributes. When selected, you can also specify whether to display the hierarchy level of dimension attributes.

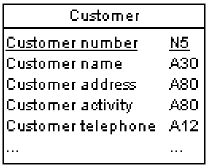
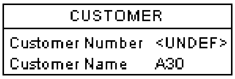
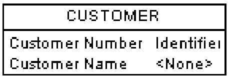
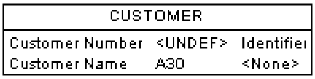
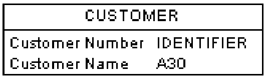
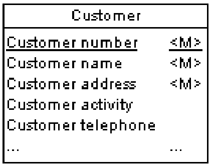
Entity display preferences (CDM/LDM)

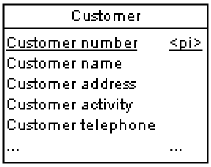
To set display preferences for entities, select Tools ➤ Display Preferences, and select the Entity sub-category in the left-hand Category pane.

Entity

Preference	**Display description**
Attributes	Specifies whether Attributes are displayed on entity symbols. If selected, you can choose between displaying: ♦ All attributes - All attributes: Customer Customer number ID Customer name NAME Customer address SHORT_TEXT Customer activity SHORT_TEXT Customer telephone PHONE Customer fax PHONE ♦ Primary attributes - Only primary identifier attributes: Customer Customer number ID ♦ Identifying attributes - All identifier attributes: Customer Customer number <pi> ♦ Display limit - Number of attributes shown depends on defined value. For example, if set to 5: Customer Customer number ID Customer name NAME Customer address SHORT_TEXT Customer activity SHORT_TEXT Customer telephone PHONE ... ...
Identifiers	All identifier attributes for the entity are listed at the bottom of the entity symbol: Customer James_Joyce
Stereotype	Stereotype of the entity.
Comment	Comment of the entity. When selected, all other check boxes are deselected, except for Stereotype: Customer This entity can be shared

Entity attributes

Preference	Display description
Data types	Data type for each entity attribute:  Customer Customer number N5 Customer name A30 Customer address A80 Customer activity A80 Customer telephone A12 ... ...
Replace by Domains	Domain for each entity attribute. You can only display domains when the Data type check box is selected.
Domains	Domain of an attribute in an entity. This display option interacts with the selection for Data types. As a result, there are four display options: ♦ Data types - Displays only the data type, if any:  CUSTOMER Customer Number <UNDEF> Customer Name A30 ♦ Domains - Displays only the domain, if any:  CUSTOMER Customer Number Identifie Customer Name <None> ♦ Data types and Domain - Displays both data type and domain, if any:  CUSTOMER Customer Number <UNDEF> Identifie Customer Name A30 <None> ♦ Data types and Replace by domains - Displays either data type or domain, if any, and domain if both are present:  CUSTOMER Customer Number IDENTIFIER Customer Name A30
Mandatory	<*M*> indicators are displayed next to each mandatory attribute:  Customer Customer number <M> Customer name <M> Customer address <M> Customer activity Customer telephone ... ...

Preference	Display description
Identifier indicators	<*pi*> indicators are displayed next to primary identifiers and <*ai*> indicators next to non-primary identifiers:  Customer Customer number <pi> Customer name Customer address Customer activity Customer telephone ... ...
Stereotype	Displays the stereotype of the entity attributes

Entity identifiers

Preference	Display description
Stereotype	Displays the stereotype of the entity identifiers

Inheritance display preferences (CDM/LDM)

To set display preferences for inheritances, select Tools ➤ Display Preferences, and select the Inheritance sub-category in the left-hand Category pane.

Preference	Description
Name	Displays the name of the inheritance
Stereotype	Displays the stereotype of the inheritance

Package display preferences (CDM/LDM/PDM)

To set display preferences for packages, select Tools ➤ Display Preferences, and select the Package sub-category in the left-hand Category pane.

Preference	Description
Stereotype	Displays the stereotype of the package
Comment	Displays the comment of the package

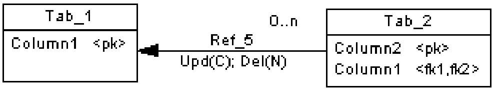
Procedure display preferences (PDM)

To set display preferences for procedures, select Tools ➤ Display Preferences, and select the Procedure sub-category in the left-hand Category pane.

Preference	Description
Owner	Displays the names of procedure owners
Stereotype	Displays the stereotypes of procedures

Reference display preferences (PDM)

To set display preferences for dimensions, select Tools ➤ Display Preferences, and select the Reference sub-category in the left-hand Category pane.

Preference	Description
Name	Displays the name or code of the reference depending on whether the Name or Code radio button is selected
Constraint name	Displays the referential integrity constraint name
Join	Displays the statement of linked columns between the two tables
Referential integrity	Displays the update and delete referential integrity constraints as follows: ♦ upd() - Update ♦ del() - Delete ♦ cpa - Change Parent Allowed A letter between the parentheses indicates the type of constraint, as follows: ♦ ( ) - None ♦ (R) - Restrict ♦ (C) - Cascade ♦ (N) - Set null ♦ (D) - Set default The referential integrity label shown below indicates the following: ♦ Cascade on update ♦ Set null on delete ♦ Cardinality is `0..n` (any number of children is acceptable).  Tab_1 Column1 <pk> 0..n Ref_5 Upd(C); Del(N) Tab_2 Column2 <pk> Column1 <fk1,fk2>

Preference	Description
Stereotype	Displays the stereotypes of references
Role names	Displays the role names for parent and child tables of references
Cardinality	Displays the minimum and maximum number of instances in a child table that can appear for each corresponding instance in the parent table. The default cardinality labels for PowerDesigner indicate the minimum and maximum number of children as follows: **[***minimum***..***maximum***]** For example, the cardinality label `[0..n]` indicates that any number of children is acceptable.
Implementation	Displays the implementation mode of referential integrity, declarative or by triggers

Reference display mode

☞ For information about changing the notation of references, see "Setting Model Settings" on page 337.

Moving text on a reference symbol
When a reference symbol displays text, the text position is based on the position of handles. You can add a handle on the reference symbol by pressing CTRL while you click the symbol.

Relationship display preferences (CDM/LDM)

To set display preferences for relationships, select Tools ➤ Display Preferences, and select the Relationship sub-category in the left-hand Category pane.

Preference	Display description
Name	Displays the name of the relationship*
Role	Displays the role of each direction in the relationship
Join	[LDM only] Displays the statement of linked attributes between the two entities
Cardinality	Displays the minimum and maximum number of instances that the first entity can have relative to the second entity
Dominance	Displays the letter *D* on the dominant entity side of the relationship
Stereotype	Displays the stereotype of the relationship

Table display preferences (PDM)

To set display preferences for tables, select Tools ➤ Display Preferences, and select the Table sub-category in the left-hand Category pane.

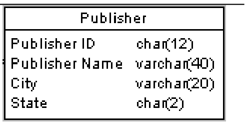
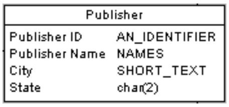
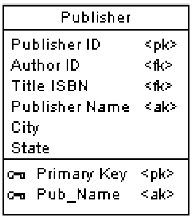
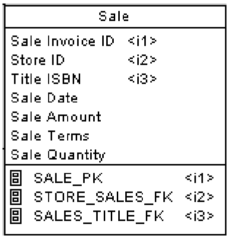
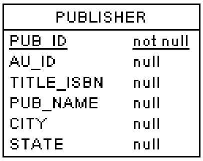
Preference	Description
Columns	Displays table columns. When selected, you can choose between displaying all columns, only primary key columns, only key columns, or limit the number of columns displayed to a given number.
Indexes	Displays all indexes defined for table columns
Keys	Displays all keys defined on table
Triggers	Displays all triggers defined for table
Owner	Displays name of table owner
Stereotype	Displays stereotype of the table. You can also display the stereotype of columns, keys, indexes, and triggers
Comment	Displays comment of the table. When selected, all other check boxes are deselected, except for Stereotype

Table Columns display

You can display information about columns in each table.

Keys and indexes are represented by indicators in the table symbol. Each key and index indicator is assigned a number. You can use these numbers to keep track of the different groups of alternate keys, foreign keys, and indexes in your model.

You can display the following column information in a table symbol:

Preference	Displays	Example
Data types	Data type for each column	 Publisher Publisher ID char(12) Publisher Name varchar(40) City varchar(20) State char(2)
Replace by domains	Domain codes for each column attached to a domain	 Publisher Publisher ID AN_IDENTIFIER Publisher Name NAMES City SHORT_TEXT State char(2)
Domains	Domain of an attribute in the table. This display option interacts with the selection for Data types. As a result, there are four display options	See Display domain and data type for options and examples
Key Indicators	<pk>, <fk>, and <ak> indicators next to primary key, foreign key, and alternate key columns respectively. When the Keys preference is also selected, the key names are listed at the bottom of the table symbol	 Publisher Publisher ID <pk> Author ID <fk> Title ISBN <fk> Publisher Name <ak> City State Primary Key <pk> Pub_Name <ak>
Index indicators	<i(*n*)> indicator next to indexed columns. When the Indexes preference is also selected, the index names and corresponding numbers are listed at the bottom of the table symbol	 Sale Sale Invoice ID <i1> Store ID <i2> Title ISBN <i3> Sale Date Sale Amount Sale Terms Sale Quantity SALE_PK <i1> STORE_SALES_FK <i2> SALES_TITLE_FK <i3>
NULL/NOT NULL	Column indicator: null, not null, identity, or with default (DBMS-dependent)	 PUBLISHER PUB_ID not null AU_ID null TITLE_ISBN null PUB_NAME null CITY null STATE null

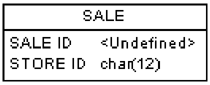
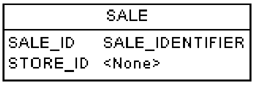
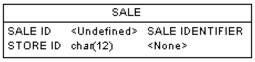
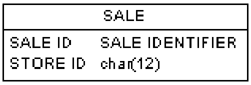
Displaying foreign key indicator numbers

You can display foreign key numbers next to their corresponding foreign key names on the references links between the appropriate parent and child tables.

☞ For information on displaying text with a reference symbol, see section “Reference display preferences (PDM)” on page 355.

Display domain and data type

You can display the domain of an attribute in the symbol of a table. There are four display options available:

Preference	Displays	Example
Data types	Only the data type, if it exists	 SALE SALE ID <Undefined> STORE ID char(12)
Domains	Only the domain, if it exists	 SALE SALE_ID SALE_IDENTIFIER STORE_ID <None>
Data types and Domains	Both data type and domain, if they exist	 SALE SALE ID <Undefined> SALE IDENTIFIER STORE ID char(12) <None>
Data types and Replace by domains	If domain exists and data type does not exist, then displays domain. If domain does not exist and data type exists, then displays data type.	 SALE SALE ID SALE IDENTIFIER STORE ID char(12)

Default options

Click the Default button to display default table display preferences. Click the Set As Default button to set current display preferences as default selections.

Table notation

☞ For information about changing the notation of tables, see “Setting Model Settings” on page 337.

View display preferences (PDM)

To set display preferences for views, select Tools ➤ Display Preferences, and select the View sub-category in the left-hand Category pane.

Preference	Description
Columns	Displays the columns in the view. When selected, you can choose between displaying all the columns, or to limit them to a given number.
Tables	Displays the tables in the view.
Indexes	For those DBMS that support query tables, displays the indexes in the views of type materialized view, summary table or snapshot.
Owner	Displays the names of view owners.
Stereotype	Displays the stereotypes of views.
Comment	Displays the comments associated with views. When selected, all other check boxes except for Stereotype are deselected.

View column display preferences

The following display preferences are available for view columns:

Preference	Description
Name	Displays the view column names.
Expression	Displays the SQL expressions for view columns.
Data types	Displays the data type for view columns.
Index indicator	For those DBMS that support query tables, displays the indicator next to indexed columns.

View reference display preferences (PDM)

To set display preferences for views, select Tools ➤ Display Preferences, and select the View reference sub-category in the left-hand Category pane.

Preference	Description
Name	Displays the name or code of view references.
Join	Displays the statement of linked columns between the two tables or views.
Stereotype	Displays the stereotype of view references
Role names	Displays the role names for parent and child tables or views of view references.

View reference display mode

☞ For information about changing the notation of view references, see

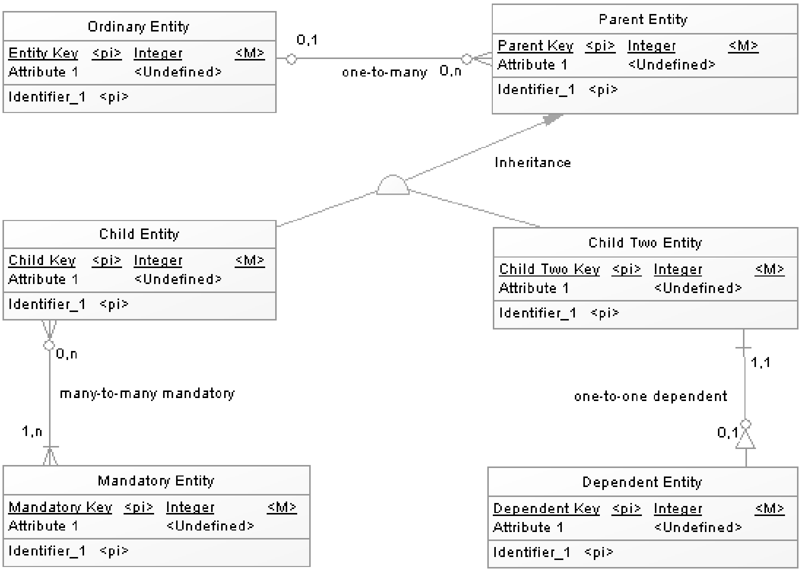
"Setting Model Settings" on page 337.

Supported CDM/LDM notations

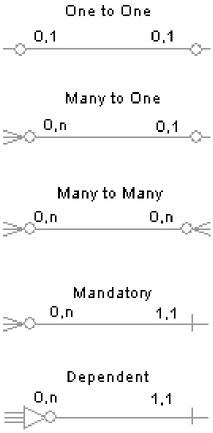
PowerDesigner supports the most popular data modeling notations in the CDM and LDM via the notation model option (see "Setting Model Settings" on page 332).

Entity/relationship notation

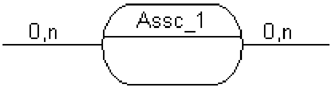
In the Entity/relationship notation, entities are represented as rectangles and divided in three compartments: name, attributes, and identifiers.



The termination points of relationships indicate the cardinality as follows:



(Note that the Merise notation uses associations instead of relationships):

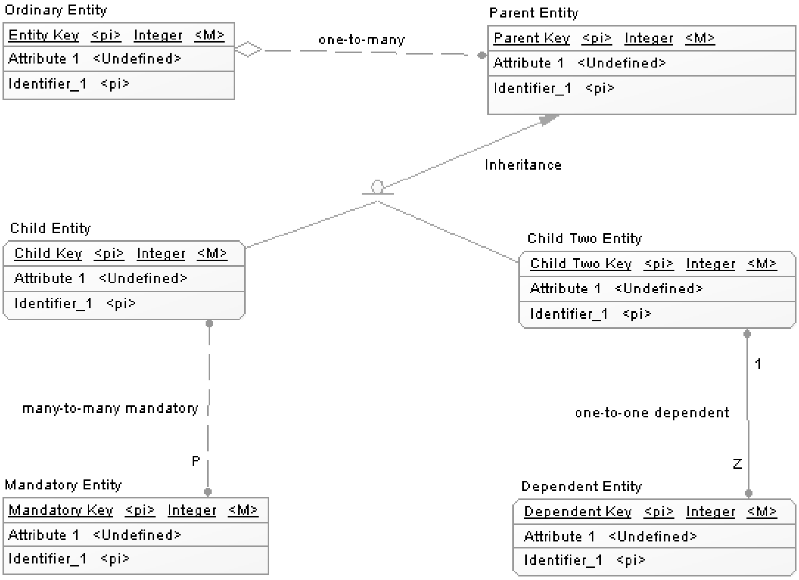


Inheritance symbols indicate if they are complete and if they have mutually exclusive children:

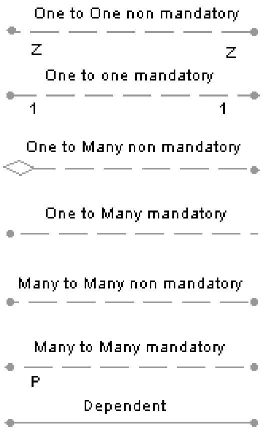
Complete	Mutually exclusive	Symbol
No	No	
Yes	No	
No	Yes	
Yes	Yes	

IDEF1X notation

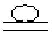
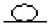
In the Idef1x notation, entity names are displayed outside the symbol, and dependent entities are drawn with round corners.



Relationship symbols indicate the cardinality as follows:

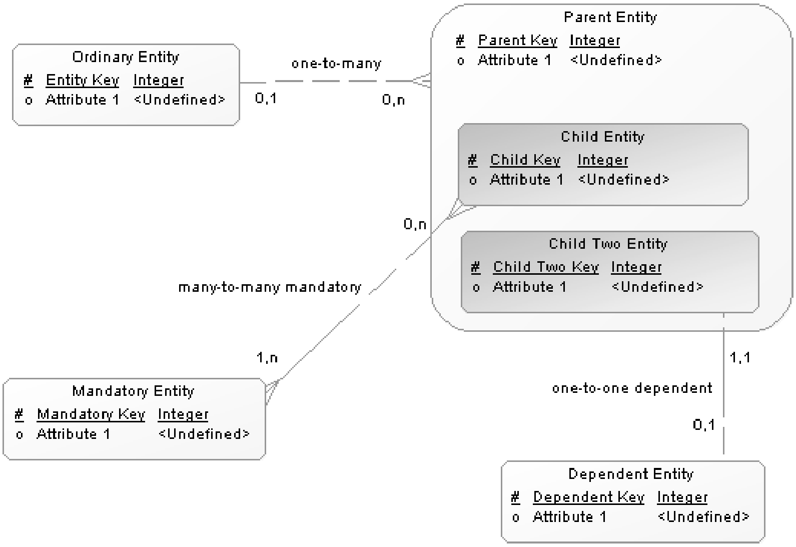


Inheritance symbols indicate if the inheritance is complete:

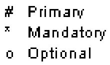
Complete	Symbol
Yes	
No	

Barker notation

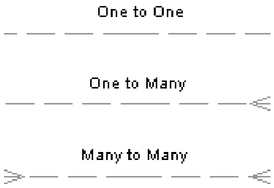
In the Barker notation, entities are drawn with round corners, and inheritances are displayed by placing children inside the parent entity.



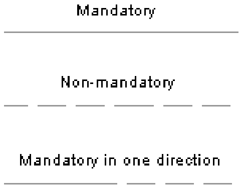
Only attributes are listed and a symbol specifies whether each attribute is a key, a mandatory or an optional attribute as follows:



Relationship symbols indicate the cardinality as follows:



The line style specifies if a relationship is mandatory:



Specifying the PDM DBMS

PowerDesigner can be used with many different DBMSs. Each supported DBMS has its own DBMS definition file, which provides PowerDesigner with the syntax for generating databases, triggers, and stored procedures for a that DBMS.

☞ For more information on the DBMS definition file, see the DBMS Resource File Reference chapter in the *Customizing and Extending PowerDesigner* manual.

Not certified resource file
Some resource files are delivered with “Not Certified” in their names. Sybase will perform all possible validation checks, however Sybase does not maintain specific environments to fully certify these resource files. Sybase will support the definition by accepting bug reports and will provide fixes as per standard policy, with the exception that there will be no final environmental validation of the fix. Users are invited to assist Sybase by testing fixes of the definition provided by Sybase and report any continuing inconsistencies.

Changing the target DBMS

When you create a PDM, you select a target DBMS (see “Creating a PDM” in the Building Physical Diagrams chapter). You can change the DBMS of a model at any time.

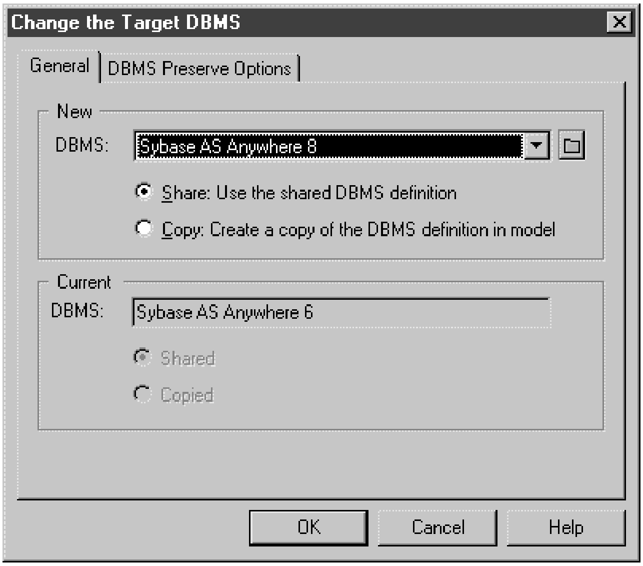
You may be required to change the DBMS if you open a PDM and the associated DBMS file has been deprecated, or is otherwise unavailable. In this case the Choose DBMS dialog box will open, inviting you to select a DBMS from the list.

If you change the target DBMS, the model may be altered to conform with the new DBMS as follows:

- All data types specified in your model will be converted to their equivalents in the new DBMS. For more information about data types, see Script/Data Type Category in the DBMS Resource File Reference chapter of the *Customizing and Extending PowerDesigner* manual.
- Any objects not supported by the new DBMS will be deleted
- Certain objects, whose behavior is heavily DBMS-dependent may lose their values. You can choose to preserve the values associated with the following database objects, if they are supported by the new DBMS:
    - Triggers and stored procedures – note that triggers are always rebuilt when you change DBMS.
    - Physical options - if the syntax of an option is incompatible with the new DBMS, the values will be lost, even if you have selected to preserve the physical option. For example, the physical option *in* used by ASA is not supported by Oracle and any values associated with that option will be lost.
    - Database objects: databases, storages, tablespaces, abstract data types, sequences
    - Extended attributes

❖ To change the target database for a PDM

1. Select Database ➤ Change Current DBMS to open the Change the Target DBMS dialog box:



New

DBMS: Sybase AS Anywhere 8

● Share: Use the shared DBMS definition

○ Copy: Create a copy of the DBMS definition in model

Current

DBMS: Sybase AS Anywhere 6

● Shared

○ Copied

2. Select a target DBMS from the list.

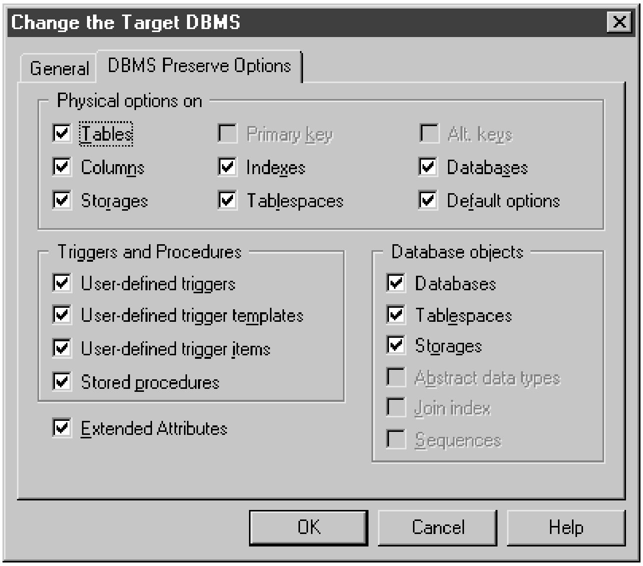
3. Select one of the following radio buttons:

   ♦ Share the DBMS definition – use the original DBMS file in the “Resource Files\DBMS” directory. Any changes made to the DBMS are shared by all linked PDMs.

   ♦ Copy the DBMS definition in model – make a copy of the original DBMS file in the “Resource Files\DBMS” directory. The current DBMS is independent of the original DBMS, so modifications made to the DBMS in the DBMS directory are not available to the PDM. The copied object language is saved with the PDM and cannot be used without it.

   ☞ For more information on DBMS properties and customizing a DBMS, see the DBMS Resource File Reference chapter in the *Customizing and Extending PowerDesigner* manual.

4. [optional] Click the DBMS Preserve Options tab, which displays options for database objects that can be preserved or lost when the target DBMS is changed:



Physical options on

☑ Tables
☑ Columns
☑ Storages
□ Primary key
☑ Indexes
☑ Tablespaces
□ Alt. keys
☑ Databases
☑ Default options

Triggers and Procedures

☑ User-defined triggers
☑ User-defined trigger templates
☑ User-defined trigger items
☑ Stored procedures

☑ Extended Attributes

Database objects

☑ Databases
☑ Tablespaces
☑ Storages
□ Abstract data types
□ Join index
□ Sequences

5. Select check boxes for the objects and options that you want to preserve.

> **Preserve options selected by default**
> If you are changing target database within a database family, for example between Sybase ASE 12.5 and 15, all preserve options available are selected by default. The database objects not supported by the old and new target databases are disabled.

6. Click OK.

   A message box opens to tell you that the target database has been changed.

7. Click OK to return to the model with the new DBMS.

Working with data model extended model definitions

Extended model definitions (.XEM files) provide means for customizing and extending PowerDesigner metaclasses, parameters and generation. Extended model definitions are typed like models in PowerDesigner. You create an extended model definition for a specific type of model and you cannot share these files between heterogeneous models.

Extended model definitions may contain:

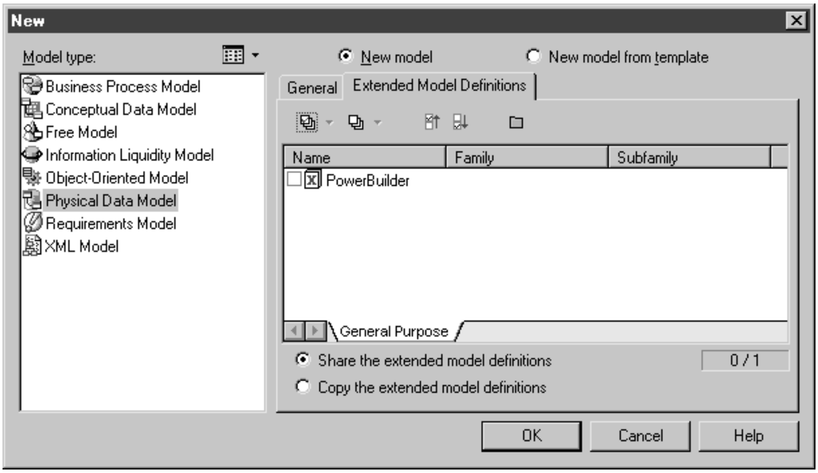
- Extended attributes for applicable objects in order to further define their properties

- ♦ Stereotypes to define extended dependencies established between model objects

  ☞ For more information on extended dependencies, see “Working with data model extended dependencies” on page 369.

- ♦ Generation targets and commands to complement the generation of an object model, or to perform an extended generation

When you create a new data model, or when you reverse engineer into a new data model, you can select one or several extended model definitions and attach them to the model from the Extended Model Definitions tab of the New dialog box.



● New model ○ New model from template

□ PowerBuilder

● Share the extended model definitions 0 / 1

○ Copy the extended model definitions

You can choose one of the following options:

Option	Definition
Share	Current extended model definition constantly refers to the extended model definition stored in the Resource Files\Extended Model Definitions directory. Any changes made to the extended model definition are shared by all linked XEM
Copy	Current extended model definition is a unique copy of the extended model definition stored in the Resource Files\Extended Model Definitions directory. The current extended model definition is independent of the original one, so modifications made to the extended model definition in the Resource Files\Extended Model Definitions directory are not available to the copied XEM. This one is saved with the PDM and cannot be used without it

☞ For more information on extended model definitions, see “Extended

Model Definitions” in the Resource Files and the Public Metamodel chapter of the *Customizing and Extending PowerDesigner* manual.

Working with data model extended dependencies

Extended dependencies are links between PDM objects. These links help to make object relationships clearer but are not interpreted and checked by PowerDesigner as they are meant to be used for documentation purposes only.

You can complement these links by applying stereotypes. Stereotypes are used to define extended dependencies between objects in the PDM.

You can type stereotypes directly in the Stereotype column of the object property sheet or select a value from the list if you have previously defined stereotypes in an embedded or imported extended model definition (.XEM).

☞ For more information on extended model definitions, see “Extended Model Definitions” in the Resource Files and the Public Metamodel chapter of the *Customizing and Extending PowerDesigner* manual.

Generating Other Models from a Data Model

You can generate the following types of models from CDMs, LDMs, and PDMs:

Data Model	CDM	LDM	PDM	OOM	XSM
CDM	X	X	X	X	
LDM	X	X	X		
PDM	X	X	X	X	X

❖ To generate a model from a CDM, LDM, or PDM

1. Select Tools, and then one of the following to open the appropriate Model Generation Options Window:
   - ♦ Generate Conceptual Data Model… Ctrl+Shift+C
   - ♦ Generate Logical Data Model… Ctrl+Shift+L
   - ♦ Generate Physical Data Model… Ctrl+Shift+P
   - ♦ Generate Object-Oriented Model… Ctrl+Shift+O
   - ♦ Generate XML Model… Ctrl+Shift+M
2. On the General tab, select a radio button to generate a new or update an existing model, and complete the appropriate options.
3. [optional – PDM-PDM generation only] Click the DBMS Preserve Options tab and set any appropriate options.
4. [optional] Click the Detail tab and set any appropriate options. We recommend that you select the Check model checkbox to check the model for errors and warnings before generation.
5. [optional] Click the Target Models tab and specify the target models for any generated shortcuts.
6. [optional] Click the Selection tab and select or deselect objects to generate.
7. Click OK to begin generation.

Generation options
For detailed information about the options available on the various tabs of the Generation window, see the Linking and Synchronizing Models chapter of the *Core Features Guide* .

Generating other models from a CDM

The following table details how CDM objects are generated to other models:

CDM	OOM	PDM
Entity	Class	Table
Entity attribute	Attribute	Table column
Primary identifier	-	Primary or foreign key depending on independent or dependent relationship
Identifier	-	Alternate key
Association	Relationship or association	-
Binary association with attributes	Association class	-
Inheritance	Generalization	-
Relationship	-	Reference

Persistent entities (OOM)

All entities are generated as persistent classes with the "Generate table" persistence mode.

When the Generate check box of an entity is not selected, the generated class has the "Migrate columns" persistence mode.

Changing the name of a column automatically

Two columns in the same table cannot have the same name. If column names conflict due to foreign key migration, PowerDesigner automatically renames the migrated columns. The new name is composed of the first three letters of the original entity name followed by the code of the attribute.

Generating PDM table keys from CDM entity identifiers

The type of key that is generated in the PDM depends on the cardinality and type of dependency defined for a relationship in the CDM. Primary identifiers generate primary and foreign keys. Other identifiers that are not primary identifiers generate alternate keys:

- A **primary key** is a column or columns whose values uniquely identify a row in a table.

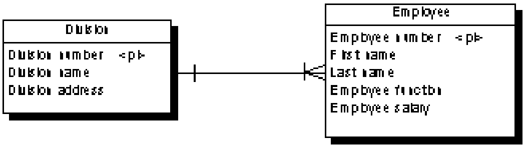
- A **foreign key** is a column or columns that depend on and migrate from a primary key column in another table.
- An **alternate key** is a column or columns whose values uniquely identify a row in a table, and is not a primary key.

Independent one-to-many relationships

In independent one-to-many relationships, the primary identifier of the entity on the one side of the relationship is generated as a:

- Primary key in the table generated by the entity on the one side of the relationship
- Foreign key in the table generated by the entity on the many side of the relationship

The following CDM shows an independent relationship. Each division contains one or more employees:



The following PDM will be generated:

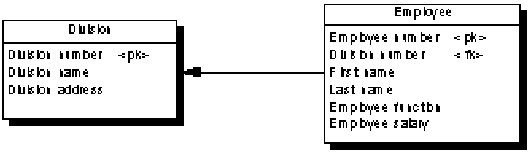
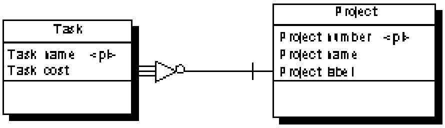


Table	Primary key	Foreign key
Division	Division number	—
Employee	Employee number	Division number

Dependent one-to-many relationships

In dependent relationships, the primary identifier of the nondependent entity is generated as a primary/foreign key in the table generated by the dependent entity. The migrated column is integrated into the primary key if it already exists.

The following CDM shows a dependent relationship. Each task must have a project number.



The following PDM will be generated:

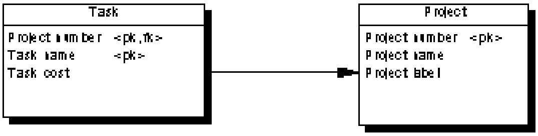
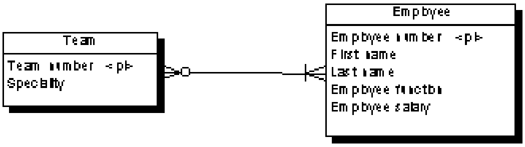


Table	Primary key	Foreign key
Project	Project number	—
Task	Project number/Task number	Project number

Independent many-to-many relationships

In independent many-to-many relationships, the primary identifiers of both entities migrate to a join table as primary/foreign keys. The CDM below shows an independent relationship. Each employee can be a member of one or more teams, and each team can have one or more employees as members.



The following PDM will be generated:

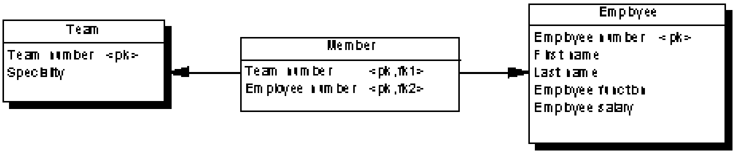


Table	Primary key	Foreign key
Team	Team number	—
Employee	Employee number	—
Member	Team number/Employee number	Team number/Employee number

Independent one-to-one relationships

In independent one-to-one relationships, the primary identifier of one entity migrates to the other generated table as a foreign key.

Generating tables from entities with inheritance links

Two properties influence the generation of tables from entities with inheritance links.

Object	Property	When selected generates
Entity	Generate table	Table for the entity (parent or child)
Inheritance	Generation mode	Parent and/or children as indicated

Generating Other Models from an LDM

The following table details how LDM objects are generated to other models:

LDM	CDM	PDM
Business rule	Business rule	Business rule
Domain	Domain	Domain
Entity	Entity	Table
Identifier	Identifier	Key
Entity attribute	Entity attribute	Column table
Inheritance	Inheritance	References
Relationship	Relationship	Reference

Generating Other Models from a PDM

The following table details how PDM objects are generated to other models:

PDM	CDM	LDM	OOM	XSM
Domain	Domain		Domain	Simple Type
Table	Entity		Class	Element
Table column	Entity attribute		Attribute	Attribute or element
Primary key	Primary identifier		Primary identifier	-
Alternate key	Identifier		Identifier	-
Foreign key	-		-	Keyref constraint
Stored-Procedures	-		Operation	-
View	-		-	Element
View column	-		-	Attribute

PDM	CDM	LDM	OOM	XSM
Index	-		-	Unique
Abstract data type	-		-	Complex type
Reference	Relationship		Association	-

XML model naming conventions
If the code of the generated XML model objects does not correspond to the target language naming conventions, you can define a code naming convention script to convert object names into codes. For more information on conversion scripts, see ".convert_name & .convert_code macros" in the Models chapter.

XML Specifics

Generation of column as attribute or element is controlled by generation option

Foreign keys - When a foreign key is not a composition, it is generated as a KeyRef constraint

Oracle 8 and Interbase sequence translation

When a CDM is generated from a PDM, the data type of a table column attached to a sequence is translated to a serial data type in the new model.

The resulting CDM serial data type for an entity property has the format `NO%n`, where `%n` is a number indicating the length of the data type.

☞ For more information on sequences, see sections on Oracle and Interbase in chapter DBMS-specific Features.

OOM Specifics

All tables are generated as persistent classes with the "Generate table" persistence mode.

All abstract data types are generated as persistent classes with the "Generate ADT" persistence mode.

Table - Class. The cardinality of a class is translated from the number of estimated records in a table

Table with migrated keys from only two other tables - Class linked with an association class between the two classes generated by the two parent tables

Stored-Procedures and stored functions attached to selected table - If the parent table is generated as a class, the stored procedure or the stored function is generated as an operation attached to the class

OOM naming conventions
If the code of the generated OOM objects does not correspond to the target language naming conventions, you can define a code naming convention script to convert object names into codes. For more information on conversion scripts, see ".convert_name & .convert_code macros" in the Models chapter.

Configuring the generated model options

When you configure the options of a CDM to generate, you may define options diverging from the PDM options.

To avoid conflicts, PowerDesigner applies the following rule for default values of CDM options: an option defined for the generated CDM should respect the equivalent option of the PDM.

Equivalent Enforce non-divergence model options are available in both the PDM and CDM.

PDM option	CDM option	Result in generated CDM
✔ Enforce non-divergence	—	Enforce non-divergence in model according to PDM options. Data items and attributes attached to the domain cannot have divergent definitions
—	✔ Enforce non-divergence	Enforce non-divergence in model according to CDM options defined using the Configure Model Options feature

Relationships unique code

(CDM) Unique Code for relationships is not selected by default in the CDM options. However, if you select Unique Code for relationships in the CDM options, relationships are renamed during the generation of a PDM to a CDM.

Options with no equivalent, like Enforce Profile in the PDM without any corresponding option in a CDM, are generated using default values found in the registry.

Options with no equivalent in the models

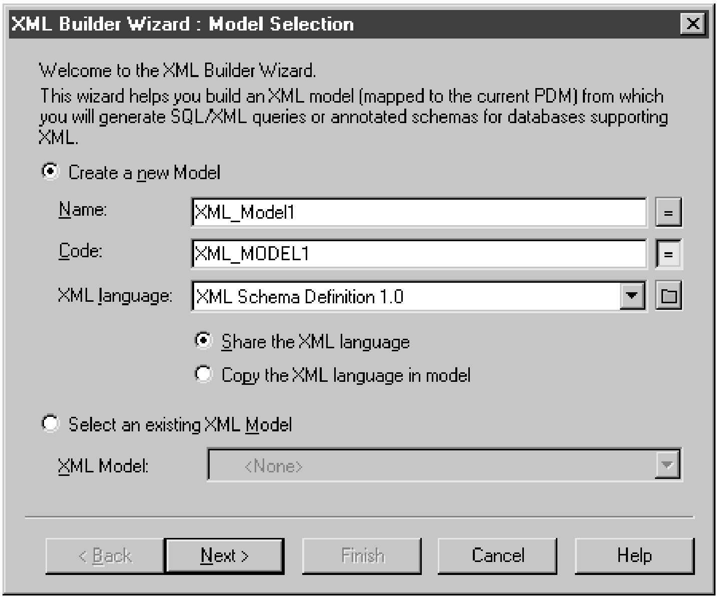
(OOM) Options with no equivalent, like Enforce Profile in the PDM without any corresponding option in an OOM, are generated using default values found in the registry.

Generating an XSM from a PDM via the XML Builder Wizard

The XML Builder Wizard helps you build an XML model (XSM) that will be used to generate SQL/XML queries for retrieving data from databases. It is more powerful than the standard PDM-XSM generation, as it helps you to customize the XML hierarchy to be built, and sets up the XSM to retrieve data from relational databases supporting SQL/XML, and for generation of an annotated schema (see the Working with XML and Databases chapter in the *XML* Modeling guide).

❖ To generate an XSM from a PDM via the XML Builder Wizard

1. In your PDM, select Tools ➤ XML Builder Wizard to open the XML Builder Wizard to the Model Selection page:



XML Builder Wizard : Model Selection

Welcome to the XML Builder Wizard.
This wizard helps you build an XML model (mapped to the current PDM) from which you will generate SQL/XML queries or annotated schemas for databases supporting XML.

● Create a new Model

Name: XML_Model1

Code: XML_MODEL1

XML language: XML Schema Definition 1.0

● Share the XML language

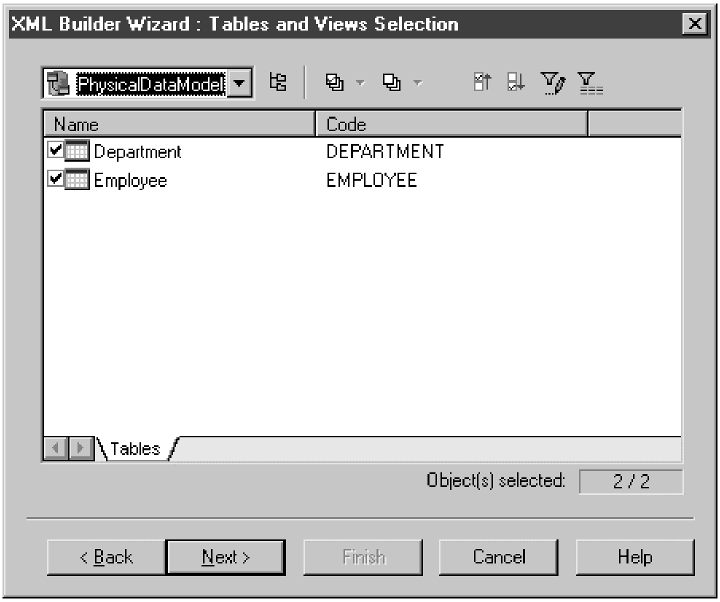
○ Copy the XML language in model

○ Select an existing XML Model

XML Model: <None>

   Select whether you want to create a new XML model or update an existing XML model currently open in your workspace.

2. Click Next to go to the Tables and Views Selection tab:



XML Builder Wizard : Tables and Views Selection

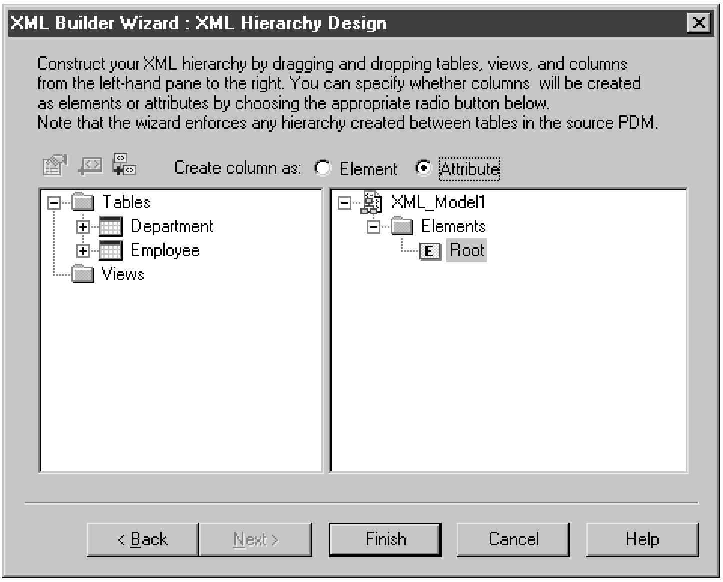
PhysicalDataModel

Name	Code
☑ Department	DEPARTMENT
☑ Employee	EMPLOYEE

Object(s) selected: 2 / 2

Select the tables and views you want to generate. By default, all tables and views are selected.

3. Click Next to go to the XML Hierarchy Design tab:



XML Builder Wizard : XML Hierarchy Design

Construct your XML hierarchy by dragging and dropping tables, views, and columns from the left-hand pane to the right. You can specify whether columns will be created as elements or attributes by choosing the appropriate radio button below. Note that the wizard enforces any hierarchy created between tables in the source PDM.

Create column as: ○ Element ● Attribute

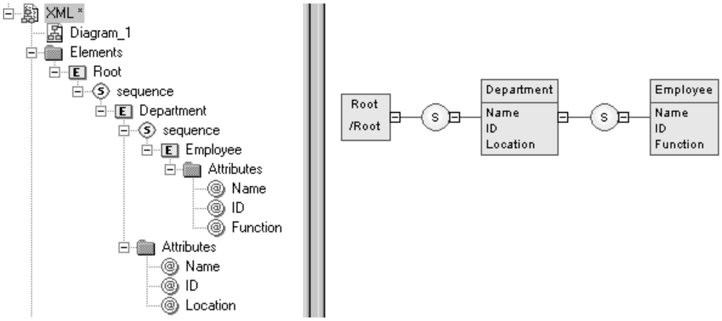
On this tab, you construct the XML hierarchy that you want to generate:

- The left-hand pane lists the tables and views that you have selected

- The right-hand pane displays the XML hierarchy to be generated, containing a default root element.

4. You can build your XML hierarchy using the following techniques:
   - Specify whether columns will be generated as elements or attributes by using the radio buttons above the panes.
   - Drag and drop a table, view, or column onto a node in the XML hierarchy. You must respect the PDM hierarchy: You cannot create an XML hierarchy between two elements if there is no **reference** between their corresponding tables, and a **parent** table cannot be placed beneath one of its children.
   - Right-click a table, view, or column and select Add from the contextual menu to add it to the last selected node in the XML hierarchy.
   - Rename an element or attribute by clicking its node and typing a new name.
   - Create new elements and attributes not in the PDM, and Sequence, Choice and All group particles, by right-clicking an XML node and selecting New ➤ *object* from the contextual menu.
   - Delete an XML node by right-clicking it and selecting Delete from the contextual menu.

5. Click Finish to generate the XSM.



In the case of an update to an existing XSM, your hierarchy will be created as a new root in the model.

The **SQL/XML** extended model definition is automatically attached to the XML model to enable you to generate SQL/XML queries from global elements. For more information, see the Exchanging Data with Databases supporting XML chapter in the *XML* Modeling guide.

XML Builder Wizard tools

The following tools are available to help you build your hierarchy:

Tool	Description
	Properties – Opens the property sheet for the selected table, view, or column.
	Add Object - Adds the selected PDM object to the XML hierarchy.
	Create Default Hierarchy for Selected Objects - Adds the selected PDM objects to the XML hierarchy.

Checking a Data Model

You can check the validity of your data model at any time. A valid data model conforms to the following kinds of rules:

- Each object name in a data model must be unique
- Each entity in a CDM must have at least one attribute
- Each relationship in a LDM must be attached to at least one entity
- Each index in a PDM must have a column

Check your data model before generation
We recommend that you check your data model before generating another model or a database from it. If the check encounters errors, generation will be stopped. The Check model option is enabled by default in the Generation dialog box.

You can check a data model in any of the following ways:

- Press F4, or
- Select Tools ➤ Check Model, or
- Right-click the diagram background and select Check Model from the contextual menu

The Check Model Parameters window opens, which allows you to specify the kinds of checks to perform, and the objects to apply them to. For detailed information about this window and correcting problems reported, see “Checking a Model” in the Models chapter of the *Core Features Guide* .

The following sections document the data model-specific checks available by default. For information about checks made on generic objects available in all model types, see “Checking a Model” in the Models chapter of the *Core Features Guide* .

Abstract data type checks (PDM)

The following PDM model checks are made on abstract data types:

Check	Description and Correction
Abstract data type name and code uniqueness	Abstract data type names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Abstract Data Type code maximum length	The code of the ADT is longer than the maximum allowed by the DBMS. Manual correction: Reduce the length of the code Automatic correction: Reduces the code to a permissible length
Instantiable object type must have attributes and no abstract procedures	If an abstract data type of type Object (or SQLJ Object) is instantiable (Abstract option not checked), then it must have attributes and no abstract procedure. Manual correction: Define at least one attribute in the ADT Attributes tab and clear the Abstract option in the procedures property sheet Automatic correction: None
Abstract object type must not have tables based on it	If an abstract data type of type Object (or SQLJ Object) is not instantiable (Abstract option checked), then it must not have tables based on it. Manual correction: Set the Based on property to <None> in the tables property sheet Automatic correction: None

Abstract data type procedure checks (PDM)

The following PDM model checks are made on abstract data type procedures:

Check	Description and Correction
Abstract data type procedure name and code uniqueness	Abstract data type procedure names and codes must be unique in the abstract data type. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code

Check	Description and Correction
Abstract Data Type procedure code maximum length	The code of the ADT procedure is longer than the maximum allowed by the DBMS. Manual correction: Reduce the length of the code Automatic correction: Reduces the code to a permissible length
Procedure cannot have the same name as an attribute	An abstract data type procedure cannot have the same name as an attribute. Manual correction: Change the name of the ADT procedure Automatic correction: None
Abstract data type procedure definition empty	An abstract data type procedure must have a definition. Manual correction: Create an ADT procedure definition in the Definition tab of the ADT procedure property sheet Automatic correction: None
Inconsistent return type	If the abstract data type procedure is a function, a map or an order, you should define a return data type for the function, map or order. Manual correction: Select a return data type in the Return data type list Automatic correction: None

Association checks (CDM)

The following CDM model checks are made on associations:

Check	Description and Correction
Association name and code uniqueness	Association names and codes must be unique in the namespace. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code.
Number of links >= 2	An association is isolated and therefore does not define a relationship between entities. Manual correction: Define at least two links between the isolated association and one or several entities. Automatic correction: None.

Check	Description and Correction
Number of links = 2 with an identifier link	An identifier link introduces a dependency between two entities. An association with this type of link must be binary. Manual correction: Delete the unnecessary links or clear the Identifier check box for a link. Automatic correction: None.
Number of identifier links <= 1	An identifier link introduces a dependency between two entities. There can only be one identifier link between two entities otherwise a circular dependency is created. Manual correction: Clear the Identifier check box for one of the links. Automatic correction: None.
Absence of properties with identifier links	An association with an identifier link cannot have any properties. Manual correction: Move the association properties into the dependent entity (the one linked to the association with an identifier link). Automatic correction: None.
Bijective association between two entities	There are bijective associations between two entities when a two-way one to one association between the entities exist. This is equivalent to a merge of two entities. Manual correction: Merge the entities or modify the cardinality links. Automatic correction: None.
Maximal cardinality links	An association with more than two links can only have links with a maximum cardinality greater than one. Manual correction: Change the maximum cardinality of such links to be greater than 1. Automatic correction: None.
Reflexive identifier links	An identifier link introduces a dependency between two entities. An association with this type of link cannot therefore be reflexive. Manual correction: Change the relationship between the entities or clear the Identifier check box for a link. Automatic correction: None.

Check	Description and Correction
Name uniqueness constraint between many-to-many associations and entities	A many-to-many association and an entity cannot have the same name or code. Manual correction: Change the name or code of the many-to-many association or the name or code of the entity. If you do not, PDM generation will rename the generated table. Automatic correction: None.

Association checks (PDM)

The following PDM model checks are made on associations:

Check	Description and Correction
Existence of hierarchy	Association names and codes must be unique in the model. Manual correction: Select a hierarchy in the Hierarchy list in the association property sheet Automatic correction: None

Attribute checks (PDM)

The following PDM model checks are made on attributes:

Check	Description and Correction
Attribute name and code uniqueness	Attribute names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code

Column checks (PDM)

The following PDM model checks are made on columns:

Check	Description and Correction
Column name and code uniqueness	Column names and codes must be unique in a table. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code

Check	Description and Correction
Column code maximum length	The column code length is limited by the maximum length specified in the DBMS definition (MaxLen entry, in the Objects ➤ Column category) or in the naming conventions of the model options. Manual correction: Modify the column code length to meet this requirement Automatic correction: Truncates the code length to the maximum length specified in the DBMS definition
Domain divergence	Divergence is verified between columns, domains, and data types. Various checks and attributes are also examined. One or more of the Enforce non divergence model options must be selected. Manual correction: Select one or more of the Enforce non divergence model options to enforce non divergence Automatic correction: Restores divergent attributes from domain to column (domain values overwrite column values)
Column mandatory	In some DBMS, the columns included in a key or a unique index should be mandatory. Manual correction: Select the Mandatory check box in the column property sheet Automatic correction: Makes the column mandatory
Detect inconsistencies between check parameters	The values entered in the check parameters tab are inconsistent for numeric and string data types: default does not respect minimum and maximum values, or default does not belong to list of values, or values in list are not included in minimum and maximum values, or minimum is greater than maximum value. Check parameters must be defined consistently. Manual correction: Modify default, minimum, maximum or list of values in the check parameters tab Automatic correction: None
Precision > Maximum length	The data type precision should not be greater than the length. Note that some DBMS accept a precision higher than the length. Manual correction: Make the data type length greater than the precision Automatic correction: None

Check	Description and Correction
Undefined data type	A model should not contain columns with undefined data type, all columns should have a defined data type. Manual correction: Select a data type in the column property sheet Automatic correction: None
Foreign key column data type and constraint parameters divergence	Primary/alternate and foreign key columns involved in a join should have consistent data types and constraint parameters. Manual correction: Modify foreign key data types and constraint parameters to make them consistent Automatic correction: Parent column overwrites existing data type and constraint parameters in the foreign key column
Column with sequence not in a key	Since a sequence is used to initialize a key, it should be attached to a column that is part of a key. This applies to those DBMS that support sequences. Manual correction: Attach the sequence to a column that is part of a key Automatic correction: None
Auto-incremented column with data type not numeric	An auto-incremented column must have a numeric data type. Manual correction: Change the column data type Automatic correction: Changes data type to numeric data type
Auto-incremented column is foreign key	A foreign key column should not be auto-incremented. Manual correction: Deselect the Indentity check box in the column property sheet Automatic correction: None
Missing computed column expression	A computed column should have a computed expression defined. Manual correction: Add a computed expression to the column in the Details tab of the column property sheet Automatic correction: None

Cube checks (PDM)

The following PDM model checks are made on cubes:

Check	Description and Correction
Cube name and code uniqueness	Cubes names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Existence of association	A cube must have at least one association with a dimension. Manual correction: Create an association between the cube and a dimension Automatic correction: None
Existence of fact	A cube must be associated to a fact. Manual correction: Click the Ellipsis button beside the Fact box in the cube property sheet, and select a fact from the List of Facts Automatic correction: None
Duplicated association with the same dimension	A cube cannot have more than one association with the same dimension. Manual correction: Delete one of the associations Automatic correction: None

Database checks (PDM)

The following PDM model checks are made on databases:

Check	Description and Correction
Database name and code uniqueness	Database names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Database code maximum length	The code of the database is longer than the maximum allowed by the DBMS. Manual correction: Reduce the length of the code Automatic correction: Reduces the code to a permissible length

Check	Description and Correction
Database not used	The database you have created is not used in the model. Manual correction: Delete the database or apply the database as a physical option to a table, an index, a key, a column, a storage, a tablespace or a view (Options tab of the object property sheet) Automatic correction: None

Database package checks (PDM)

The following PDM model checks are made on database packages:

Check	Description and Correction
Database package name and code uniqueness	Database package names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Database package name and code maximum length	The database package name and code length is limited by the maximum length specified in the DBMS definition (MaxLen entry, in the Objects ➤ DB Package category) and in the naming conventions of the model options. Manual correction: Modify the name/code length to meet this requirement Automatic correction: Truncates the name/code length to the maximum length specified in the DBMS definition
Existence of package procedure	A database package is an encapsulated collection of related procedures. It should contain at least one procedure. Manual correction: Create one or several procedures in the database package or use existing stored procedures and duplicate them in the database package Automatic correction: None
Existence of package cursor	This check is to suggest that a database package can contain cursors to define a work area and access its stored information. Manual correction: Create one or several cursors in the Cursors tab of the database package property sheet Automatic correction: None

Check	Description and Correction
Existence of package variable	This check is to suggest that a database package can contain variables to capture or provide a value when one is needed. Manual correction: Create one or several variables in the Variables tab of the database package property sheet Automatic correction: None
Existence of package type	This check is to suggest that a database package can contain user-defined data types called types. Manual correction: Create one or several types in the Types tab of the database package property sheet Automatic correction: None
Existence of package exception	This check is to suggest that a database package can contain exceptions to handle internal and user-defined error conditions. Manual correction: Create one or several exceptions in the Exceptions tab of the database package property sheet Automatic correction: None

Database package cursor checks (PDM)

The following PDM model checks are made on package cursors:

Check	Description and Correction
Package cursor name and code uniqueness	Database package cursor names and codes must be unique in the database package. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Package cursor definition empty	A package cursor must have a definition. Manual correction: Create the cursor definition in the Definition tab of the cursor property sheet Automatic correction: None
Check for undefined return types	You should define a return data type for a cursor. Manual correction: Select a return data type in the cursor property sheet Automatic correction: None

Check	Description and Correction
Existence of parameter	A cursor must contain parameters for input values. Manual correction: Create one or several parameters in the Parameters tab of the cursor property sheet Automatic correction: None

Database package exception checks (PDM)

The following PDM model checks are made on exceptions:

Check	Description and Correction
Package exception name and code uniqueness	Exception names and codes must be unique in the database package. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code

Database package procedure checks (PDM)

The following PDM model checks are made on package procedures:

Check	Description and Correction
Package procedure name and code uniqueness	Database package procedure names and codes must be unique in the database package. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Package procedure definition empty	A package procedure must have a definition. Manual correction: Create the package procedure definition in the Definition tab of the package procedure property sheet Automatic correction: None
Existence of parameter	A package procedure must contain parameters for input and output values. Manual correction: Create one or several parameters in the Parameters tab of the package procedure property sheet Automatic correction: None

Check	Description and Correction
Check for undefined return types	If the package procedure is a function, you should define a return data type for the function. Manual correction: Select a return data type in the Return Data Type list Automatic correction: None

Database package type checks (PDM)

The following PDM model checks are made on package types:

Check	Description and Correction
Package type name and code uniqueness	Package type names and codes must be unique in the database package. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Package type definition empty	A package type must have a definition. Manual correction: Create the type definition in the Definition tab of the package type property sheet Automatic correction: None

Database package variable checks (PDM)

The following PDM model checks are made on package variables:

Check	Description and Correction
Package variable name and code uniqueness	Variable names and codes must be unique in the database package. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Undefined data type	You should define a data type for a variable. Manual correction: Select a data type in the variable property sheet Automatic correction: None

Data item checks (CDM)

The following CDM model checks are made on data items:

Check	Description and Correction
Data item name and code uniqueness	Data item names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code.
Data item not used	There are unused data items. These are useless for PDM generation. Manual correction: To use a data item, add it to an entity. If you do not need an unused data item, delete it to allow PDM generation. Automatic correction: None.
Data item used multiple times	There are entities using the same data items. This can be tolerated if you defined this check as a warning. Manual correction: Take care to ensure consistency when defining data item properties. Automatic correction: None.
Detect differences between data item and associated domain	There is a divergence between data items and associated domains. This can be tolerated if you defined this check as a warning. Manual correction: Ensure consistency when defining data item properties Automatic correction: Restores divergent attributes from domain to data items (domain values overwrite data item values).
Detect inconsistencies between check parameters	The values entered in the check parameters page are inconsistent for numeric and string data types: default does not respect minimum and maximum values, or default does not belong to list of values, or values in list are not included in minimum and maximum values, or minimum is greater than maximum value. Check parameters must be defined consistently. Manual correction: Modify default, minimum, maximum or list of values in the check parameters page Automatic correction: None.

Check	Description and Correction
Precision > maximum length	The data type precision should not be greater than or equal to the length. Manual correction: Make the data type length greater than or equal to the precision. Automatic correction: None.
Undefined data type	Undefined data types for data items exist. To be complete, a model should have all its data items data types defined. Manual correction: While undefined data types are tolerated, you must select data types for currently undefined data types before you can generate a PDM. Automatic correction: None.
Invalid data type	Invalid data types for data items exist. To be complete, a model should have all its data types for data items correctly defined. Manual correction: While tolerated, you must select valid data types for currently non-valid data types to generate the PDM. Automatic correction: None.

Data source checks (PDM)

The following PDM model checks are made on data sources:

Check	Description and Correction
Data source name and code uniqueness	Data source names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Existence of physical data model	A data source must contain at least one physical data model in its definition. Manual correction: Add a physical data model from the Models tab of the property sheet of the data source Automatic correction: Deletes data source without physical data model

Check	Description and Correction
Data source containing models differing DBMS types	The models in a data source should share the same DBMS since they represent a single database. Manual correction: Delete models with different DBMS or modify the DBMS of models in the data source Automatic correction: None

Default checks (PDM)

The following PDM model checks are made on defaults:

Check	Description and Correction
Default name and code uniqueness	Default names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Default code maximum length	The default code length is limited by the maximum length specified in the DBMS definition (MaxLen entry, in the Objects ➤ Default category). Manual correction: Modify the default code length to meet this requirement Automatic correction: Truncates the default code length to the maximum length specified in the DBMS definition
Default value empty	You must type a value for the default, this value is used during generation. Manual correction: Type a value in the Value box of the default property sheet Automatic correction: None
Several defaults with same value	A model should not contain several defaults with identical value. Manual correction: Modify default value or delete defaults with identical value Automatic correction: None

Dimension checks (PDM)

The following PDM model checks are made on dimensions:

Check	Description and Correction
Dimension name and code uniqueness	Dimension names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Existence of attribute	A dimension must have at least one attribute. Manual correction: Create an attribute in the Attributes tab of the dimension property sheet Automatic correction: None
Existence of hierarchy	A dimension must use at least one hierarchy. Manual correction: Create a hierarchy in the Hierarchies tab of the dimension property sheet Automatic correction: None
Dimension have duplicated hierarchies	Dimensions should not have duplicated hierarchies, that is to say hierarchies organizing identical attributes. Manual correction: Remove one of the duplicated hierarchies Automatic correction: None
Dimension without a default hierarchy	A dimension should have a default hierarchy. Manual correction: Select a hierarchy in the Default Hierarchy list of the dimension property sheet Automatic correction: None
Dimension mapping not defined	A dimension should be mapped to tables or views in an operational model in order to be populated by data from this model. Manual correction: Map the dimension to a table or a view. You may need to create a data source before you can create the mapping Automatic correction: Destroys the mapping for the dimension. This removes the data source from the Mapping list in the dimension Mapping tab
Attribute mapping not defined	Attributes must be mapped to columns in the data source tables or views. Manual correction: Map the attributes to columns in the data source Automatic correction: None

Check	Description and Correction
Incomplete dimension mapping for multidimensional generation	All attributes, detail attributes and hierarchies of the dimension must be mapped to tables and columns. You must map the dimension objects before generation. Manual correction: Map dimension objects to tables and columns Automatic correction: None

Dimension hierarchy checks (PDM)

The following PDM model checks are made on dimension hierarchies:

Check	Description and Correction
Dimension hierarchy name and code uniqueness	Dimension hierarchy names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Existence of attribute	A dimension hierarchy must have at least one attribute. Manual correction: Add an attribute to the hierarchy from the Attributes tab of the hierarchy property sheet Automatic correction: None

Domain checks (CDM/LDM/PDM)

The following model checks are made on domains:

Check	Description and Correction
Domain name and code uniqueness	Domain names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code

Check	Description and Correction
Domain code maximum length	[PDM only] The domain code length is limited by the maximum length specified in the DBMS definition (MaxLen entry, in the Objects ➤ Domain category) or in the naming conventions of the model options. Manual correction: Modify the domain code length to meet this requirement Automatic correction: Truncates the domain code length to the maximum length specified in the DBMS definition
Detect Inconsistencies between check parameters	The values entered in the Check Parameters tab are inconsistent for numeric and string data types. Default does not respect minimum and maximum values, or default does not belong to list of values, or values in list are not included in minimum and maximum values, or minimum is greater than maximum value. Check parameters must be defined consistently. Manual correction: Modify default, minimum, maximum or list of values in the check parameters tab Automatic correction: None
Precision > maximum length	The data type precision should not be greater than the length. Manual correction: Make the data type length greater than the precision Automatic correction: None
Undefined data type	A model should not contain domains with undefined data type, all domains should have a defined data type. Manual correction: Select a data type from the domain property sheet Automatic correction: None
Invalid data type	[CDM/LDM only] Invalid data types for domains exist. To be complete, a model should have all its domain data types correctly defined. Manual correction: While tolerated, you must select valid data types for currently non-valid data types to generate the PDM. Automatic correction: None.

Check	Description and Correction
Domain missing default object	[PDM only] A domain cannot have a default value without being attached to a default object. Manual correction: Create a default object for the domain or use the rebuild default feature Automatic correction: Creates a default object for domain

Entity attribute checks (CDM)

The following CDM model checks are made on entity attributes:

Check	Description and Correction
Entity attribute name and code uniqueness	Attribute names and codes must be unique in the entity. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code.

Entity attribute checks (LDM)

The following LDM model checks are made on entity attributes:

Check	Description and Correction
Entity attribute name and code uniqueness	Attribute names and codes must be unique in the entity. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code.
Detect differences between attribute and associated domain	There is a divergence between attributes and associated domains. This can be tolerated if you defined this check as a warning. Manual correction: Ensure consistency when defining attribute properties Automatic correction: Restores divergent attributes from domain to attributes (domain values overwrite attribute values).

Check	Description and Correction
Detect inconsistencies between check parameters	The values entered in the Check Parameters page are inconsistent for numeric and string data types. Default does not respect minimum and maximum values, or default does not belong to list of values, or values in list are not included in minimum and maximum values, or minimum is greater than maximum value. Check parameters must be defined consistently. Manual correction: Modify default, minimum, maximum or list of values in the check parameters page Automatic correction: None.
Precision > maximum length	The data type precision should not be greater than or equal to the length. Manual correction: Make the data type length greater than or equal to the precision. Automatic correction: None.
Undefined data type	Undefined data types for attributes exist. To be complete, a model should have all its attributes data types defined. Manual correction: While undefined data types are tolerated, you must select data types for currently undefined data types before you can generate a PDM. Automatic correction: None.
Invalid data type	Invalid data types for attributes exist. To be complete, a model should have all its data types for attributes correctly defined. Manual correction: While tolerated, you must select valid data types for currently non-valid data types to generate the PDM. Automatic correction: None.

Entity identifier checks (CDM/LDM)

The following CDM/LDM model checks are made on entity identifiers:

Check	Description and Correction
Entity identifier name and code uniqueness	Entity identifier names and codes must be unique in the namespace. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code.
Existence of entity attribute	At least one attribute must exist for an entity identifier. Manual correction: Add an attribute to the entity identifier or delete the identifier. Automatic correction: None.
Identifier inclusion	An identifier cannot include another one. Manual correction: Delete the identifier that includes an existing identifier. Automatic correction: None.

Entity checks (CDM/LDM)

The following CDM/LDM model checks are made on entities:

Check	Description and Correction
Entity name and code uniqueness	Entity names and codes must be unique in the namespace. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code.
Entity name and code maximum length	The entity name and code length is limited to a maximum length of 254 characters specified in the naming conventions of the model options. Manual correction: Modify the entity name/code length to meet this requirement. Automatic correction: Truncates the entity name/code length to the maximum length specified in the naming conventions.
Existence of attributes	An entity must always contain at least one attribute. Manual correction: Add an attribute to the entity or delete the entity. Automatic correction: None.

Check	Description and Correction
Number of serial types > 1	An entity cannot have more than one serial type attribute. Serial types are automatically calculated values. Manual correction: Change the types of the appropriate entity attributes to have only one serial type attribute. Automatic correction: None.
Existence of identifiers	An entity must contain at least one identifier. Manual correction: Add an identifier to the entity or delete the entity. Automatic correction: None.
Existence of relationship or association link	An entity must have at least one relationship or association link. Manual correction: Add a relationship or an association link to the entity or delete the entity. Automatic correction: None.
Redundant inheritance	An entity inherits from another entity more than once. This is redundant and adds nothing to the model. Manual correction: Delete redundant inheritances Automatic correction: None.
Multiple inheritance	An entity has multiple inheritance. This is unusual but can be tolerated if you defined this check as a warning. Manual correction: Make sure that the multiple inheritance is necessary in your model. Automatic correction: None.
Parent of several inheritances	An entity is the parent of multiple inheritances. This is unusual but can be tolerated if you defined this check as a warning. Manual correction: Verify if the multiple inheritances could not be merged. Automatic correction: None.
Redefined primary identifier	Primary identifiers in child entities must be the same as those in their parents. Manual correction: Delete those primary identifiers in the child entities that are not in the parent entity. Automatic correction: None.

Fact checks (PDM)

The following PDM model checks are made on facts:

Check	Description and Correction
Fact name and code uniqueness	Fact names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Existence of measure	A fact must have at least one measure. Manual correction: Create a measure in the Measures tab of the fact property sheet Automatic correction: None
Fact mapping not defined	A fact must be mapped to tables or views in an operational model in order to be populated by data from this model. Manual correction: Map the fact to tables or views. You may need to create a data source before you can create the mapping Automatic correction: Destroys the mapping for the fact. This removes the data source from the Mapping list in the fact Mapping tab
Measure mapping not defined	Fact measures must be mapped to columns in the data source tables or views. Manual correction: Map the fact measure to columns in the data source Automatic correction: Destroys the mapping for the measure. This removes the measures that are not mapped to any object in the Measures Mapping tab of the fact Mapping tab

Fact measure checks (PDM)

The following PDM model checks are made on fact measures:

Check	Description and Correction
Fact measure name and code uniqueness	Fact measure names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code

Group checks (PDM)

The following PDM model checks are made on groups:

Check	Description and Correction
Group name and code uniqueness	Group names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Group code maximum length	The group code length is limited by the maximum length specified in the DBMS definition (MaxLen entry, in the Objects ➤ Group category) or in the naming conventions of the model options. Manual correction: Modify the group code length to meet this requirement Automatic correction: Truncates the group code length to the maximum length specified in the DBMS definition
Existence of user	A group is created to factorize privilege and permission granting to users. A group without user members is useless. Manual correction: Add users to group or delete group Automatic correction: Deletes unassigned group
Group password empty	Groups must have a password to be able to connect to the database (for those DBMS that support passwords for groups.) Manual correction: Define a password for the group Automatic correction: None

Horizontal partitioning checks (PDM)

The following PDM model checks are made on horizontal partitioning objects:

Check	Description and Correction
Horizontal partitioning name and code uniqueness	Horizontal partitioning names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Existence of partition	A horizontal partitioning object cannot be empty, it must contain at least one partition. Manual correction: Delete the horizontal partitioning object or create at least one partition in the horizontal partitioning object property sheet Automatic correction: Deletes empty horizontal partitioning object
Unavailable target table	A partition should have a corresponding table. You could delete a table that actually corresponds to a partition, this check verifies that each partition has a corresponding table. Manual correction: Delete the partition with no corresponding table Automatic correction: Deletes the partition with no corresponding table

Index checks (PDM)

The following PDM model checks are made on indexes:

Check	Description and Correction
Index name and code uniqueness	Depending on the DBMS, a model or a table cannot contain two indexes with identical name and/or code. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code

Check	Description and Correction
Index code maximum length	The index code length is limited by the maximum length specified in the DBMS definition (MaxLen entry, in the Objects ➤ Index category) or in the naming conventions of the model options. Manual correction: Modify the index code length to meet this requirement Automatic correction: Truncates the index code length to the maximum length specified in the DBMS definition
Existence of index column	An index must have at least one index column. Manual correction: Add an index column from the Column tab of the index property sheet or delete the index Automatic correction: Deletes the index without column
Undefined index type	An index type must be specified. Manual correction: Specify a type in the index property sheet or delete the index with no type Automatic correction: None
Index column count	The current DBMS does not support more than the number of index columns specified in the MaxColIndex entry of the current DBMS. Manual correction: Delete one or more columns in the index property sheet. You can create additional indexes for these columns Automatic correction: None
Uniqueness forbidden for HNG index type	An index of HNG (HighNonGroup) type cannot be unique. Manual correction: Change the index type or set the index as non unique Automatic correction: None
Index inclusion	An index should not include another index. Manual correction: Delete the index that includes an existing index Automatic correction: None

Inheritance checks (CDM/LDM)

The following CDM/LDM model checks are made on inheritances:

Check	Description and Correction
Inheritance name and code uniqueness	Inheritance names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code.
Existence of inheritance link	An inheritance must have at least one inheritance link, from the inheritance to the parent entity. Manual correction: Define the inheritance link or delete the inheritance. Automatic correction: None.
Incomplete inheritance with ungenerated ancestor	[LDM only] If an inheritance is incomplete, the parent should be generated because you can lose information. Manual correction: Generate parent entity or define the inheritance as complete. Automatic correction: None.

Join index checks (PDM)

The following PDM model checks are made on join indexes and bitmap join indexes:

Check	Description and Correction
Join index name and code uniqueness	Join index names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Existence of base table	Join index must have a base table. Manual correction: Select a base table in the join index property sheet Automatic correction: None
Join Index tables owners	The tables associated to a join index must have the same owner. Manual correction: Modify the join index owner or the table owner Automatic correction: None

Check	Description and Correction
Join index references connection	Join index references must be connected to selected table on a linear axis. Manual correction: Delete or replace references in the join index Automatic correction: None
Duplicated join indexes	Join indexes cannot have the same set of references. Manual correction: Delete one of the duplicated join indexes Automatic correction: None

Key checks (PDM)

The following PDM model checks are made on keys:

Check	Description and Correction
Key name and code uniqueness	Key names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Key code length	The key code length is limited by the maximum length specified in the DBMS definition (MaxConstLen entry, in the Object ➤ Key category). Manual correction: Modify the key code length to meet this requirement Automatic correction: Truncates the key code length to the maximum length specified in the DBMS definition
Key column exists	Each key must have at least one column. Manual correction: Add a column to the key from the Column tab of the key property sheet Automatic correction: Deletes key without column
Key inclusion	A key cannot include another key (on some columns, regardless of their order). Manual correction: Delete the key that includes an existing key Automatic correction: None

Check	Description and Correction
Multi-column key has sequence column	Since the column initialized by a sequence is already a key, it should not be included in a multi-column key. Manual correction: Detach the sequence from a column that is already part of a multi-column key Automatic correction: None

Package checks (CDM/LDM/PDM)

The following model checks are made on packages:

Check	Description and Correction
Circular references	[PDM only] A circular reference occurs when a table refers to another table, and so on until a loop is created between tables. A package cannot contain circular references. Manual correction: Resolve the circular reference by correcting the reference, deleting its source, or clearing the Mandatory parent or Check on commit option Automatic correction: None
Constraint name uniqueness	[PDM only] A constraint name is a unique identifier for the constraint definition of tables, columns, primary and foreign keys in the database. You define the constraint name in the following tabs: Check tab of the table property sheet Additional Check tab of the column property sheet General tab of the key property sheet A constraint name must be unique in a model. Manual correction: Modify the duplicated constraint name in the corresponding tab Automatic correction: Modifies the duplicated constraint name of a selected object by appending a number to its current name

Check	Description and Correction
Constraint name maximum length	[PDM only] The constraint name length cannot be longer than the length specified in the DBMS definition: either in the MaxConstLen entry, in the Object category, or in each object category. Manual correction: Modify the constraint name to meet this requirement Automatic correction: Truncates the constraint name to the maximum length specified in the DBMS definition
Circular dependencies	[PDM only] Extended dependencies with the stereotype <<DBCreateAfter>> can be used between stored procedures to define a generation order for stored procedures. An extended dependency with the stereotype <<DBCreateAfter>> should not introduce a circular dependency in the model. Manual correction: Remove the <<DBCreateAfter>> extended dependency Automatic correction: None
Circular dependency	[CDM/LDM only] A circular dependency occurs when an entity depends on another and so on until a dependency loop is created between entities. A package cannot contain circular dependencies. Manual correction: Clear the Dependent check box for the link or delete an inheritance link. Automatic correction: None.
Circularity with mandatory links	[CDM/LDM only] A circular dependency occurs when an entity depends on another and so on until a dependency loop is created between entities through mandatory links. Manual correction: Clear the Mandatory parent check box or delete a dependency on a relationship. Automatic correction: None.
Shortcut code uniqueness	Shortcuts codes must be unique in a namespace. Manual correction: Change the code of one of the shortcuts Automatic correction: None

Check	Description and Correction
Shortcut potentially generated as child table of a reference	[CDM/LDM only] The package should not contain associations or relationships with an external shortcut as child entity. Although this can be tolerated in the CDM, the association or relationship will not be generated in a PDM if the external shortcut is generated as a shortcut. Manual correction: Modify the design of your model in order to create the association or relationship in the package where the child entity is defined. Automatic correction: None.

Procedure checks (PDM)

The following PDM model checks are made on procedures:

Check	Description and Correction
Procedure name and code uniqueness	Procedure names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Procedure code maximum length	The procedure code length is limited by the maximum length specified in the DBMS definition (MaxLen entry, in the Objects ➤ Procedure category). Manual correction: Modify the procedure code length to meet this requirement Automatic correction: Truncates the procedure code length to the maximum length specified in the DBMS definition
Procedure definition body empty	A procedure definition should have a body to specify its functionality. Manual correction: Specify a procedure body from the Definition tab of the procedure property sheet Automatic correction: None
Existence of permission	Permissions are usage restrictions set on a procedure for a particular user, group or role. Manual correction: Define permissions on the procedure for users, groups and roles Automatic correction: None

Reference checks (PDM)

The following PDM model checks are made on references:

Check	Description and Correction
Reference name and code uniqueness	Reference names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Reflexive and mandatory reference	A reflexive reference exists should not have a mandatory parent which could lead to inconsistent joins. Manual correction: Correct the reference by clearing the Mandatory parent check box Automatic correction: None
Existence of reference join	A reference must have at least one reference join. Manual correction: Create a reference join for the reference or delete the reference Automatic correction: Deletes reference without join
Reference code maximum length	The reference code length is limited by the maximum length specified in the DBMS definition (MaxConstLen entry, in the Object ➤ Reference category) or in the naming conventions of the model options. Manual correction: Modify the reference code length to meet this requirement Automatic correction: Truncates the reference code length to the maximum length specified in the DBMS definition
Incomplete join	Joins must be complete. Manual correction: Select a foreign key column or activate the primary key column migration Automatic correction: None

Check	Description and Correction
Join order	The join order must be the same as the key column order for some DBMS. Manual correction: If required, change the join order to reflect the key column order Automatic correction: The join order is changed to match the key column order During a reference check, the following object controls are made.

Relationship checks (CDM/LDM)

The following CDM/LDM model checks are made on relationships:

Check	Description and Correction
Relationship name and code uniqueness	Relationship names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code.
Reflexive dependency	A dependency means that one entity is defined through a relationship with another. A dependent relationship cannot therefore be reflexive. Manual correction: Change or delete the reflexive dependency. Automatic correction: None.
Reflexive mandatory	A reflexive mandatory relationship exists. Manual correction: Deselect the Mandatory check boxes for the relationship to be non-mandatory. Automatic correction: None.
Bijective relationship between two entities	There is a bijective relationship between two entities when there is a two-way one to one relationship between the entities. This is equivalent to a merge of two entities. Manual correction: Merge the entities or modify the relationship. Automatic correction: None.

Check	Description and Correction
Name uniqueness constraint for many to many relationships and entities	A many-to-many relationship and an entity cannot have the same name or code. Manual correction: Change the name or code of the many-to-many relationship or the name or code of the entity. If you do not, PDM generation will rename the generated table. Automatic correction: None.
Consistency between dominant and dependent relationships	A dependent relationship between entities cannot also be a dominant relationship. Manual correction: Select the Dominant check box on the other (correct) side of the relationship. Automatic correction: None.
Relationship with child shortcut	External shortcut could be generated as child table. An entity that is at the “many” end of a one-to-many relationship or which is non-dominant should not be an external shortcut as the reference will not be generated during the generation of a PDM. Manual correction: Change the cardinality of the relationship cardinality or the entity, which should not be an external shortcut. Automatic correction: None.
‘Many-many’ relationships	[LDM only] ‘Many-to-many’ relationships are not permitted. Manual correction: Create an intermediary entity, which contains the primary identifiers of the previous ‘many-to-many’ entities. Automatic correction: None.

Role checks (PDM)

The following PDM model checks are made on roles:

Check	Description and Correction
Role name and code uniqueness	Role names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code

Check	Description and Correction
Role code maximum length	The role code length is limited by the maximum length specified in the DBMS definition (MaxLen entry, in the Objects ➤ Role category) or in the naming conventions of the model options. Manual correction: Modify the role code length to meet this requirement Automatic correction: Truncates the role code length to the maximum length specified in the DBMS definition
Existence of user	A role is used to create predefined profile that can be assigned to users or roles. A role that is not assigned to any user or role is useless. Manual correction: Assign role to users or delete role Automatic correction: Deletes unassigned role

Sequence checks (PDM)

The following PDM model checks are made on sequences:

Check	Description and Correction
Sequence name and code uniqueness	Default names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Sequence code maximum length	The code of the sequence is longer than the maximum allowed by the DBMS. Manual correction: Reduce the length of the code Automatic correction: Reduces the code to a permissible length

Storage checks (PDM)

The following PDM model checks are made on storages:

Check	Description and Correction
Storage name and code uniqueness	Storage names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Storage code maximum length	The code of the storage is longer than the maximum allowed by the DBMS. Manual correction: Reduce the length of the code Automatic correction: Reduces the code to a permissible length
Storage not used	The storage you have created is not used in the model. Manual correction: Delete the storage or apply the storage as a physical option to a table, an index, a key, a column, a tablespace or a view (Options tab of the object property sheet) Automatic correction: None

Synonym checks (PDM)

The following PDM model checks are made on synonyms:

Check	Description and Correction
Synonym name and code uniqueness	Synonym names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Synonym name and code maximum length	The synonym name and code length is limited by the maximum length specified in the DBMS definition (MaxLen entry, in the Objects ➤ Synonym category) and in the naming conventions of the model options. Manual correction: Modify the name/code length to meet this requirement Automatic correction: Truncates the name/code length to the maximum length specified in the DBMS definition

Check	Description and Correction
Existence of the base object	A synonym must correspond to a model object. By default, when you create synonyms from the List of Synonyms using the Add a Row tool, they are not attached to any base object. Manual correction: Select a base object from the synonym property sheet Automatic correction: Deletes the synonym

Table checks (PDM)

The following PDM model checks are made on tables:

Check	Description and Correction
Table name and code uniqueness	Tables names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Table name and code length	The table name and code length is limited by the maximum length specified in the DBMS definition (MaxLen entry, in the Objects ➤ Table category) and in the naming conventions of the model options. Manual correction: Modify the name/code length to meet this requirement Automatic correction: Truncates the name/code length to the maximum length specified in the DBMS definition
Constraint name conflict with index name	A constraint name of the table cannot be the same as an index name. Manual correction: Change the name of the table constraint Automatic correction: None
Existence of column, reference, index, key	A table should contain at least one column, one index, one key, and one reference. Manual correction: Add missing item to the definition of the table Automatic correction: None

Check	Description and Correction
Number of auto-incremented columns	Auto-incremented columns contain automatically calculated values. A table cannot contain more than one auto-incremented column. Manual correction: Delete all but one auto-incremented column Automatic correction: None
Table index definition uniqueness	Identical indexes are indexes with the same columns, order and type. A table cannot have identical indexes. Manual correction: Delete index or change its properties Automatic correction: None
Table mapping not defined	When a table belongs to a model containing one or several data sources, it must be mapped to tables or views in the data source in order to establish a relational to relational mapping. Manual correction: Map the current table to one or several tables or views in the model belonging to the data source Automatic correction: Destroys the mapping for the table. This removes the data source from the Mapping list in the table Mapping tab
Column mapping not defined	When a column belong to a table in a model containing one or several data sources, it should be mapped to columns in the data source in order to establish a relational to relational mapping. Manual correction: Map the current column to one or several columns in the models belonging to the data source Automatic correction: Destroys the mapping for the column. This removes the columns that are not mapped to any object in the Columns Mapping tab of the table Mapping tab
Existence of permission	Permissions are usage restrictions set on a table for a particular user, group or role. Manual correction: Define permissions on the table for users, groups and roles Automatic correction: None

Table collapsing checks (PDM)

The following PDM model checks are made on table collapsing:

Check	Description and Correction
Table collapsing name and code uniqueness	Table collapsing names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Existence of target table	A table collapsing must have a table as result of the collapsing. Manual correction: Delete the table collapsing object Automatic correction: None
Unavailable target table	The table resulting from the collapsing should be available. Manual correction: Delete the table collapsing object Automatic correction: Deletes the table collapsing object

Tablespace checks (PDM)

The following PDM model checks are made on tablespaces:

Check	Description and Correction
Tablespace name and code uniqueness	Tablespace names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Tablespace code maximum length	The code of the tablespace is longer than the maximum allowed by the DBMS. Manual correction: Reduce the length of the code Automatic correction: Reduces the code to a permissible length

Check	Description and Correction
Tablespace not used	The tablespace you have created is not used in the model. Manual correction: Delete the tablespace or apply the tablespace as a physical option to a table, an index, a key, a column, a storage or a view (Options tab of the object property sheet) Automatic correction: None

Trigger checks (PDM)

The following PDM model checks are made on triggers:

Check	Description and Correction
Trigger name and code uniqueness	Trigger names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Trigger code length	The trigger code length is limited by the maximum length specified in the DBMS definition (MaxLen entry, in the Objects ➤ Trigger category). Manual correction: Modify the trigger code length to meet this requirement Automatic correction: Truncates the trigger code length to the maximum length specified in the DBMS definition

User checks (PDM)

The following PDM model checks are made on users:

Check	Description and Correction
User name and code uniqueness	User names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code

Check	Description and Correction
User code maximum length	The user code length is limited by the maximum length specified in the DBMS definition (MaxLen entry, in the Objects ➤ User category). Manual correction: Modify the user code length to meet this requirement Automatic correction: Truncates the user code length to the maximum length specified in the DBMS definition
User password empty	Users must have a password to be able to connect to the database. Manual correction: Define a password for the user Automatic correction: None

Vertical partitioning checks (PDM)

The following PDM model checks are made on partitions:

Check	Description and Correction
Vertical partitioning name and code uniqueness	Vertical partitioning names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Existence of partition	A vertical partitioning object cannot be empty, it must contain at least one partition. Manual correction: Delete the vertical partitioning object or create at least one partition in the vertical partitioning object property sheet Automatic correction: Deletes empty vertical partitioning object
Unavailable target table	A partition should have a corresponding table. You could delete a table that actually corresponds to a partition, this check verifies that each partition has a corresponding table. Manual correction: Delete the partition with no corresponding table Automatic correction: Deletes the partition with no corresponding table

View checks (PDM)

The following PDM model checks are made on views:

Check	Description and Correction
View name and code uniqueness	View names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
View code maximum length	The view code length is limited by the maximum length specified for the table code length. Manual correction: Modify the view code length to meet this requirement Automatic correction: Truncates the view code length to the maximum length specified in the DBMS definition
Existence of permission	Permissions are usage restrictions set on a view for a particular user, group or role. Manual correction: Define permissions on the view for users, groups and roles Automatic correction: None

View index checks (PDM)

The following PDM model checks are made on view indexes:

Check	Description and Correction
Index name and code uniqueness	Depending on the DBMS, a model or a view cannot contain two view indexes with identical name and/or code. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code

Check	Description and Correction
Index code maximum length	The view index code length is limited by the maximum length specified in the DBMS definition (MaxLen entry, in the Objects ➤ Index category) or in the naming conventions of the model options. Manual correction: Modify the view index code length to meet this requirement Automatic correction: Truncates the view index code length to the maximum length specified in the DBMS definition
Existence of index column	A view index must have at least one index column. Manual correction: Add an index column from the Column tab of the view index property sheet or delete the index Automatic correction: Deletes the view index without column
Index column count	The current DBMS does not support more than the number of index columns specified in the MaxColIndex entry in the Index category of the current DBMS. Manual correction: Delete one or more columns in the view index property sheet. You can create additional view indexes for these columns Automatic correction: None
View index inclusion	A view index should not include another index. Manual correction: Delete the view index that includes an existing index Automatic correction: None

View reference checks (PDM)

The following PDM model checks are made on view references:

Check	Description and Correction
View reference name and code uniqueness	View reference names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Existence of view reference join	A view reference must have at least one view reference join. Manual correction: Create a view reference join for the view reference or delete the reference Automatic correction: Deletes view reference without join

Web operation checks (PDM)

The following PDM model checks are made on Web operations:

Check	Description and Correction
Web operation name and code uniqueness	Web operation names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Web operation code maximum length	The Web operation code length is limited by the maximum length specified in the DBMS definition (Maxlen entry, in the Objects ➤ Web Operation category). Manual correction: Modify the Web operation code length to meet this requirement Automatic correction: Truncates the Web operation code length to the maximum length specified in the DBMS definition

Web service checks (PDM)

The following PDM model checks are made on Web services:

Check	Description and Correction
Web service name and code uniqueness	Web service names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Web service code maximum length	The Web service code length is limited by the maximum length specified in the DBMS definition (Maxlen entry, in the Objects ➤ Web Service category). Manual correction: Modify the Web service code length to meet this requirement Automatic correction: Truncates the Web service code length to the maximum length specified in the DBMS definition

Working with SQL

Generating a PDM produces SQL code for your target DBMS. PowerDesigner provides tools to change and preview SQL code during PDM development. These tools are:

- The SQL editor
- The SQL preview

The SQL editor allows you to define queries and the SQL preview allows you to see a SQL query script before it is generated.

Defining queries with the SQL Editor

You can use the editing features of the SQL Editor to define a query. The SQL Editor dialog box is divided into specific panes containing the information shown below:

Information	Pane location
Objects types	Upper left part of the dialog box
Available objects	Upper right part of the dialog box
Query script textbox	Lower part of the dialog box

The list of available objects depends on the selected object type. You can select individual objects from the list of available objects for insertion in the query script textbox.

You can also define expressions by entering basic arithmetic operators such as add, subtract, multiply, divide (+, -, *, /) and with the syntax tools shown below:

Syntax tool	Content
Functions	Provides group, number, string, date, conversion and other functions
Operators	Provides logical operators
Variables	Provides variables for use with operators and functions
Macros	Provides macros to accelerate the creation of a template item definition

These syntax tools allow greater flexibility in defining complex expressions.

The SQL Editor can be used for the following tasks:

- Defining a query for a view, a procedure, a trigger
- Defining a computed column

☞ For more information on defining views, see Defining a query for a view in chapter Building Physical Diagrams.

☞ For more information on defining computed columns, see Creating a computed column in chapter Building Physical Diagrams.

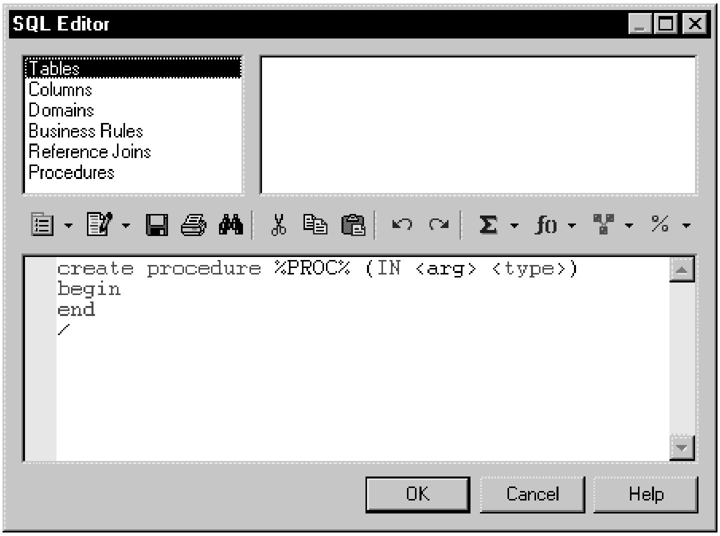
❖ To define a query with the SQL Editor

1. Click the query script textbox where you want to insert the SQL script.
2. Select an object type from the upper left part of the dialog box.

   For example, select Tables to display the list of available tables. The list of available objects of this type is displayed in the upper right part of the dialog box.
3. Double-click the available object that you want to add to the script.

   The item is added to the query script.
4. Select a function or an operator to add to the script from the Function or Operator lists.

   The item is added to the query script.



```
create procedure %PROC% (IN <arg> <type>)
begin
end
/
```

5. Click OK.

Previewing SQL statements

The Preview tab shows an SQL query script before it is generated. The displayed script reflects the options selected in the Database Generation dialog box. This script cannot be modified.

☞ For more information on database generation options, see the Generating a Database from a PDM chapter.

The text in the script preview is color coded as follows:

Text color	Represents
Blue	SQL reserved word
Black	Statement body
Red	Variable
Green	Comment

You can use the following tools and keyboard shortcuts from the Preview toolbar:

Tool	Description	Keyboard shortcut
☰▾	Editor contextual menu	Shift + F11
⟳	Refresh	F5
☷	This tool is available when at least one extended model definition flagged for generation is linked to the model and when it contains GeneratedFiles entries for the current object. When available, it displays the list of targets for the current object. If you add a generation target, the corresponding tab is added to the Preview tab. If you deselect a generation target, the corresponding tab disappears from the Preview tab	Ctrl + F6
✎	Show generation options	Ctrl + W
☑	Ignore generation options	Ctrl + D

☞ For more information on extended model definitions used for generation, see “Extended Model Definitions” in the Resource Files and the Public Metamodel chapter of the *Customizing and Extending PowerDesigner* manual.

Changing generation options

If you click the Change Generation Options tool, the Generation Options dialog box is displayed. You can change generation options and preview their effect on the code.

Ignore generation options

If you click the Ignore Generation Options tool, the preview ignores generation options selected by using the Change generation options tool but uses a predefined set of options.

Selected tool	Effect on generation options	Effect on preview
Change generation options	You can select generation options	Visible in Preview if options are applicable
Ignore generation options	Generation options currently selected are overridden by predefined set of options	Only predefined options are visible in Preview
Change generation options + Ignore generation options	You can select generation options	Changes ignored in Preview

The predefined set of generation options selects these items:

Generation Option Tab	Selected items
Tables and Views	All items except drop options
Keys and Indexes	All items except options represented differently in some DBMS. For example, if a database is auto indexed, the index options corresponding to the keys are not selected
Database	All items except drop options
Options	All user-defined options are used

SQL script bookmarks

In the SQL Editor or SQL Preview tab, you can add and remove bookmarks at specific points in the SQL script and then navigate forwards or backwards from bookmark to bookmark:

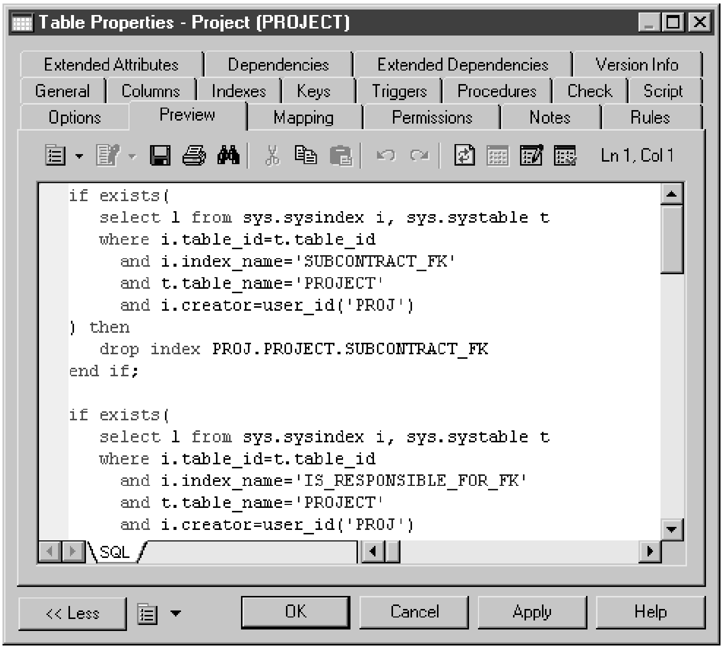
Keyboard shortcut	Description
CTRL + F2	Adds a new bookmark. A blue bookmark box is displayed. If you repeat this action from the same position, the bookmark is deleted and the blue marker disappears
F2	Jumps to next bookmark
SHIFT + F2	Jumps to the previous bookmark

Note that bookmarks are not printable and are lost if you use the Refresh, Change Generation or Ignore Generation options.

❖ To preview the code of a table

1. Double-click a table in the diagram to display the table property sheet.

2. Click the Preview tab to display the Preview tab.



```
if exists(
   select 1 from sys.sysindex i, sys.systable t
   where i.table_id=t.table_id
     and i.index_name='SUBCONTRACT_FK'
     and t.table_name='PROJECT'
     and i.creator=user_id('PROJ')
) then
   drop index PROJ.PROJECT.SUBCONTRACT_FK
end if;

if exists(
   select 1 from sys.sysindex i, sys.systable t
   where i.table_id=t.table_id
     and i.index_name='IS_RESPONSIBLE_FOR_FK'
     and t.table_name='PROJECT'
     and i.creator=user_id('PROJ')
```

3. Click OK.

PART II

WORKING WITH DATABASES

This part explains how to link your Physical Data Models to your databases, including how to generate a database from a PDM, and reverse-engineer a PDM from a database.

CHAPTER 9

Generating a Database from a PDM

About this chapter

This chapter explains how to connect to a database, and to generate and modify databases via scripts or a live database connection.

Contents

Connecting to a Database

PowerDesigner provides various methods for connecting to your database.

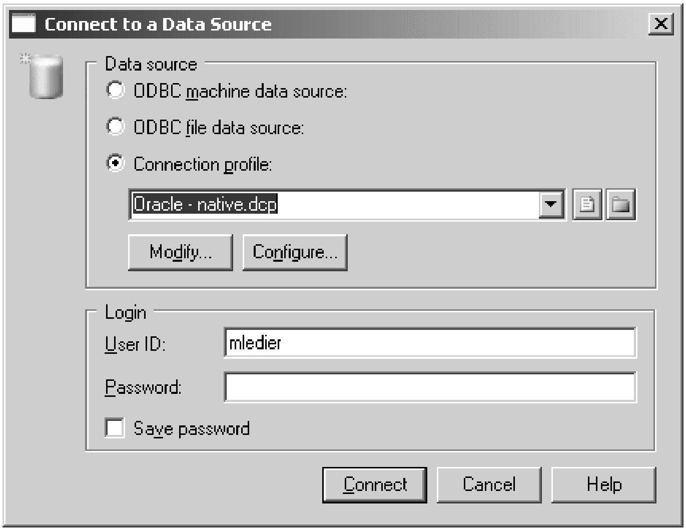
Before connecting to your database for the first time, you will have to configure a PowerDesigner connection profile. Your choice will depend on the interface that you have installed:

You have	Configure a connection of type:
ODBC driver	ODBC machine or file data source
DBMS client	Native connection profile
JDBC driver	JDBC connection profile
ADO.NET driver	ADO.NET connection profile
OLE DB driver	OLE DB connection profile
DirectConnect driver	DirectConnect connection profile

☞ For detailed information about creating, configuring, and using connection profiles, see "Connecting to a Database" section in the Models Chapter of the *Core Features Guide* .

❖ To connect to a database

1. Select Database ➤ Connect to open the Connect to a Data Source window:



Connect to a Data Source

Data source

○ ODBC machine data source:

○ ODBC file data source:

● Connection profile: Oracle - native.dcp

Login

User ID: mledier

Password:

□ Save password

2. Select one of the following radio buttons, depending on your chosen

method for connecting to your database:

- ODBC machine data source
- ODBC file data source
- Connection profile (for native, JDBC, ADO.NET, OLE DB or DirectConnect connections)

You can use the tools to the right of the data source field to browse to a new connection profile file or directory, and the Modify and Configure buttons to modify or configure your data source connection.

3. Enter your user ID and password, and then click Connect. If prompted by your database, you may need to enter additional connection parameters.

> **Connection time**
> You stay connected until you disconnect or terminate the shell session.

Displaying information about a connected database

The extent of information available about a database depends on the DBMS and the connection profile in use.

❖ To display information about a connected database

1. Connect to a data source.
2. Select Database ➤ Connection Information.

Disconnecting from a data source

❖ To disconnect from a data source

1. Select Database ➤ Disconnect.

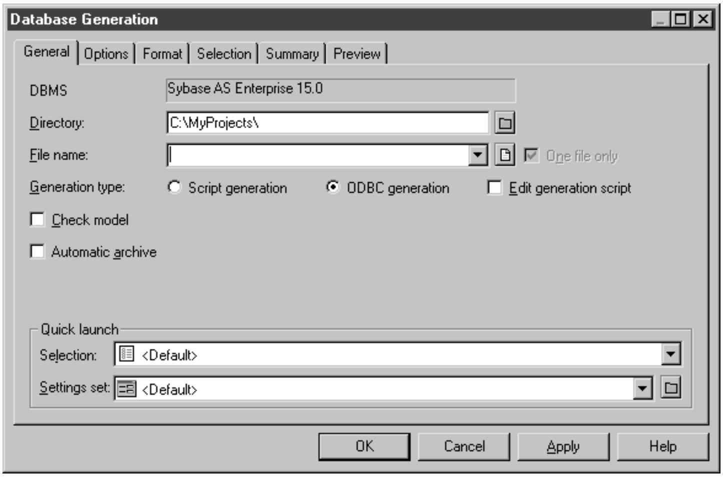
Generating a Database

This section explains how to generate a database from a PDM.

PowerDesigner can generate a database creation script that you can run in your DBMS environment or generate a database structure directly to a live database connection.

❖ To generate a database

1. Select Database ➤ Generate Database to open the Database Generation dialog box (for more information, see "Database Generation dialog General tab" on page 440).



DBMS: Sybase AS Enterprise 15.0

Directory: C:\MyProjects\

File name: ☑ One file only

Generation type: ○ Script generation ● ODBC generation □ Edit generation script

□ Check model

□ Automatic archive

Quick launch

Selection: <Default>

Settings set: <Default>

2. Type a destination directory and a filename for the script file in the Directory and File Name boxes.

3. Select the Script generation or Direct Generation radio button.

4. [optional] Click the Options tab, and specify creation options for your database objects (for more information, see "Database Generation dialog Options tab" on page 443).

5. [optional] Click the Format tab, and specify format options for your database objects (for more information, see "Database Generation dialog Format tab" on page 456).

6. [optional] Click the Selection tab, and specify the database objects to be created (for more information, see "Database Generation dialog Selection tab" on page 458).

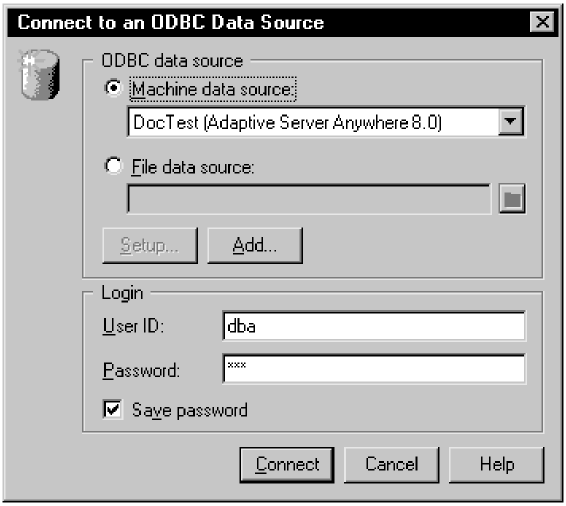
7. [optional] Click the Summary tab to view the summary of your settings and selections (for more information, see "Database Generation dialog Summary tab" on page 459).

8. [optional] Click the Preview tab to preview the database script to be used (for more information, see "Database Generation dialog Preview tab" on page 459).

9. Click OK to begin the generation.

   If you are creating a database script: The output window shows the progress of the generation process, and indicates the syntax for running the script. At the end of script generation, a Result box is displayed. It lists the file path of the generated script file. Click Edit to open the script in a text editor or Close to close the Result box.

   If you are generating a database directly: If you are not currently connected to a database, a dialog box asks you to identify a data source and connection parameters.



● Machine data source: DocTest (Adaptive Server Anywhere 8.0)

○ File data source:

User ID: dba

Password: ***

☑ Save password

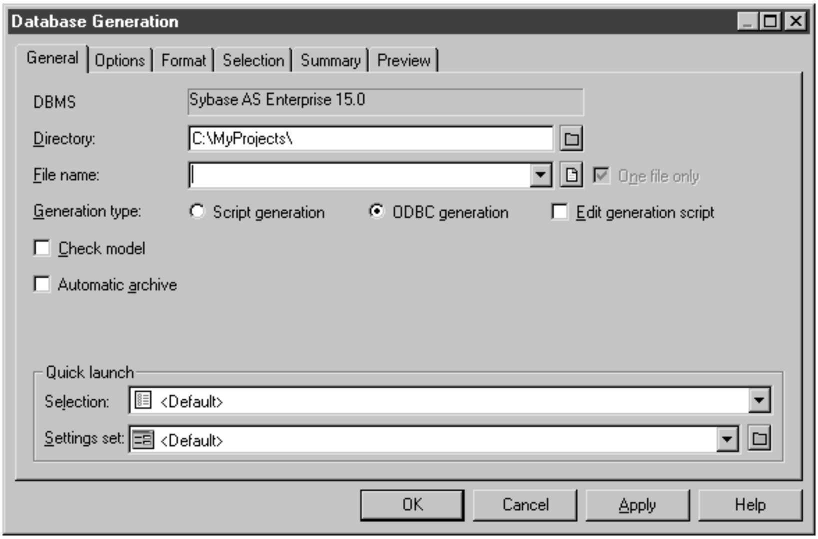
   Select a machine data source or file data source, type your user ID and password and then click Connect. You may be prompted for additional connection parameters.

Advanced generation topics

Advanced users may want to further customize database generation by, for example, customizing the order in which objects are generated, adding scripts to run before or after generation, and generating their own extended objects. For information on these and other advanced topics, the DBMS Resource File Reference chapter in the *Customizing and Extending PowerDesigner* manual.

Database Generation dialog General tab

The General tab is the main tab for controlling database generation.



DBMS: Sybase AS Enterprise 15.0

Directory: C:\MyProjects\

File name:

☑ One file only

Generation type: ○ Script generation ● ODBC generation □ Edit generation script

□ Check model

□ Automatic archive

Quick launch

Selection: <Default>

Settings set: <Default>

You can set the following options

Parameter	Description
Directory	[required] Specifies the destination directory for the script file.
File name	[required] Specifies the destination filename for the script file.
One file only	Specifies that the generation script is created as a single file. By default, a separate script file is created for each table.
Generation type	Specifies the type of generation to perform. You can choose between: ♦ Script generation - generate a script to be executed on a DBMS at a later time ♦ Direct generation – generate a script and execute it on a live database connection
Edit generation script	[available only when Direct generation is selected] Opens the generation script in a text editor for review or editing before execution on the database.

Parameter	Description
Check model	Specifies that a model check is performed before script generation.
Automatic archive	Creates an archive version of the PDM after generation.

Quick launch selection and settings sets

The Quick Launch groupbox at the bottom of the General tab allows you to load pre-configured selections and settings sets for use when generating the database.

Selection

A selection comprises:

- a set of selections of database objects made on the Selection tab (see "Database Generation dialog Selection tab" on page 458)

To save a selection, enter a name in the Selection bar at the bottom of the General or Selection tab and then click Save. The selection is saved as part of the model file.

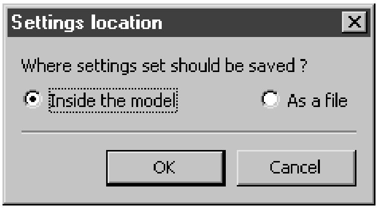
Settings set

A settings set comprises:

- a set of options selected on the Options tab (see "Database Generation dialog Options tab" on page 443) and
- the format options specified on the Format tab (see "Database Generation dialog Format tab" on page 456)

❖ To save a settings set

1. Enter a name in the Settings set bar at the bottom of the General, Options, or Format tab and then click the Save tool.
2. When the Settings location dialog box opens, specify whether you want to save the settings set as either:
   - Inside the model
   - As an external file



Settings location

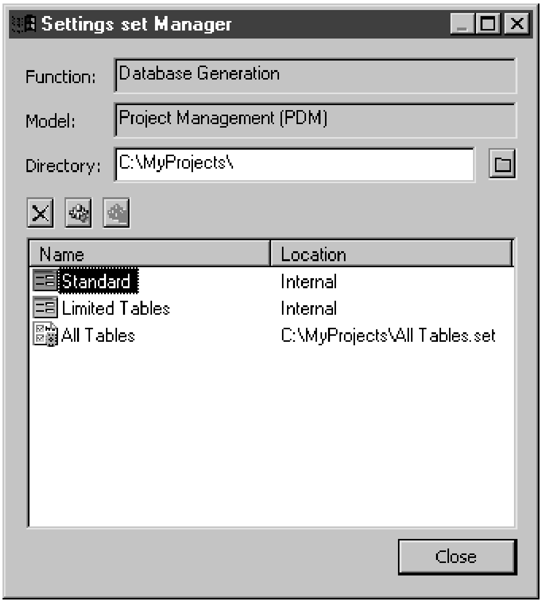
Where settings set should be saved ?

● Inside the model ○ As a file

3. Click OK.

Managing settings sets

You can review your settings sets at any time by clicking on the Settings Set Manager tool to launch the Settings Set Manager:



Settings set Manager

Function: Database Generation

Model: Project Management (PDM)

Directory: C:\MyProjects\

Name	Location
Standard	Internal
Limited Tables	Internal
All Tables	C:\MyProjects\All Tables.set

The following tools are available:

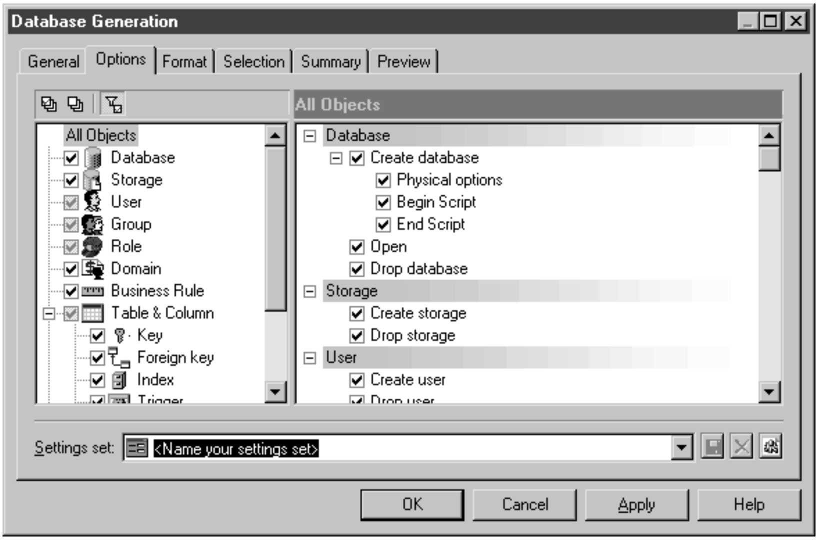
Icon	Use
📁	Browse to the settings set directory.
✕	Delete the selected settings set. Only available when an internally-saved settings set is selected. You can only delete a settings set saved to an external file through Windows Explorer.
⚙	Export the selected settings sets to an external file. Only available when an internally-saved settings set is selected.
⚙	Import the selected settings sets to inside the model. Only available when an externally-saved settings set is selected.

Note that settings sets should not be copied and renamed outside of PowerDesigner. If you want to create a variant of an existing settings set, then you should load it, make the necessary changes, and then save it under a different name.

Database Generation dialog Options tab

The Options tab allows you to specify what script elements to generate for each object type.

By default, there is an entry in the left-hand pane under the meta-category “All Objects” for each object type present in your model, and all the possible options are displayed in the right-hand pane. If you click on an object type in the left-hand pane, then the options are restricted to that object type.



All Objects
- ☑ Database
- ☑ Storage
- ☑ User
- ☑ Group
- ☑ Role
- ☑ Domain
- ☑ Business Rule
- ☑ Table & Column
  - ☑ Key
  - ☑ Foreign key
  - ☑ Index
  - ☑ Trigger

All Objects

⊟ Database
- ⊟ ☑ Create database
  - ☑ Physical options
  - ☑ Begin Script
  - ☑ End Script
- ☑ Open
- ☑ Drop database

⊟ Storage
- ☑ Create storage
- ☑ Drop storage

⊟ User
- ☑ Create user
- ☑ Drop user

Settings set: <Name your settings set>

Depending on the objects present in your model, some or all of the following options will be available.

You can save your option settings via the Settings set bar at the bottom of the tab. For more information, see “Quick launch selection and settings sets” on page 441.

Database options

These options control the generation of the database.

Parameter	Result of selection
Create database	Generates database
Physical options	Generates physical options for database
Begin script	Inserts customized script before database creation
End script	Inserts customized script after database creation
Open database	Opens database
Close database	Closes database
Drop database	Deletes an existing database, before creating new database

Tablespace options

These options control the generation of tablespaces.

Parameter	Result of selection
Create tablespace	Generates tablespaces
Begin script	Inserts customized script before tablespace creation
End script	Inserts customized script after tablespace creation
Comment	Generates tablespace comments
Drop tablespace	Deletes an existing tablespace, before creating a new tablespace

Storage options

These options control the generation of storages.

Parameter	Result of selection
Create storage	Generates storages
Comment	Generates storage comments
Drop storage	Deletes an existing storage, before creating a new storage

User options

These options control the generation of users.

Option	Result of selection
Create user	Generates users.
Physical options	Generates physical options for users
Drop user	Deletes the existing user before creating the new user.
Privilege	Generates privileges for users.

Group options

These options control the generation of groups.

Option	Result of selection
Create group	Generates groups.
Drop group	Deletes the existing group before creating the new group.
Privilege	Generates privileges for groups.

Role options

These options control the generation of roles.

Option	Result of selection
Create group	Generates roles.
Drop group	Deletes the existing role before creating the new role.
Privilege	Generates privileges for roles.

Domain options

These options control the generation of domains/user-defined data types.

Parameter	Result of selection
Create data type	Generates user-defined data types
Default value	Default value for user-defined data type
Check	Generates check parameters and validation rules for user-defined data types
Comment	Generates user-defined data type comments
Drop data type	Deletes an existing user-defined data type, before creating new user-defined data type

Business rule options

These options control the generation of business rules.

Parameter	Result of selection
Create rule	Generates rules
Comment	Generates rule comments
Drop rule	Deletes an existing rule, before creating a new rule

Default options

These options control the generation of defaults.

Parameter	Result of selection
Create default	Generates default object
Comment	Generates default comments
Drop default	Deletes an existing default before creating a new default object

If the Create and Drop check boxes are selected, the default objects will be created/dropped before domains and tables.

☞ For more information on the default generation statement, see section Default in the DBMS Resource File Reference chapter in the *Customizing and Extending PowerDesigner* manual.

Abstract data type options

These options control the generation of abstract data types.

Parameter	Result of selection
Create data type	Defines an abstract data type which is stored on a server
Begin script	Inserts customized script before data type creation
End script	Inserts customized script after data type creation
Comment	Generates data type comments
Drop data type	Deletes an existing abstract data type, before defining a abstract data type
Install JAVA class	Installs a Java class which is stored on a server
Remove JAVA class	Deletes an existing Java class, before installing a new Java class
Permission	Generates the permission statement for a given user during data type creation

Sequence options

These options control the generation of sequences.

Parameter	Result of selection
Create sequence	Generates sequence
Comment	Generates sequence comments
Drop sequence	Deletes an existing sequence, before creating a new sequence
Permission	Generates the permission statement for a given user during sequence creation

Creation parameters for tables & columns

These options control the generation of tables, columns, indexes, and keys.

Table options

These options control the generation of tables.

Parameter	Result of selection
Create table	Generates tables.
Check	Generates check parameters and validation rules for tables. If selected you can choose between: ♦ Inside Table - checks are generated during table creation ♦ Outside - checks are generated with a separate SQL command, generally using an ALTER command after the creation of the table The generation of checks outside the table is possible if the AddTableCheck entry exists in the Table category of the current DBMS.
Physical options	Generates physical options for tables.
Begin script	Inserts customized script before table creation.
End script	Inserts customized script after table creation.
Comment	Generates table comments.
Permission	Generates the permission statement for a given user during table creation.

Parameter	Result of selection
Drop table	Deletes the existing table before creating the new table.

Column options

These options control the generation of columns.

Parameter	Result of selection
User-defined type	Generates user-defined data type for column
Default value	Assigns default value to column at creation
Check	Generates check parameters and validation rules for columns. If selected you can choose between: ♦ Inside Table - checks are generated during table creation ♦ Outside - checks are generated with a separate SQL command, generally using an ALTER command after the creation of the table The generation of checks outside the table is possible if the AddTableCheck entry exists in the Table category of the current DBMS.
Comment	Generates column comments
Physical options	Generates physical options for column
Permission	Generates the permission statement for the column during creation

Primary key options

These options control the generation of primary keys.

Parameter	Result of the selection
Create primary key	Generates primary keys. If selected you can choose between: ♦ Inside Table - primary keys are generated during table creation ♦ Outside - primary keys are generated with a separate SQL command, generally using an ALTER command after the creation of the table The generation of primary keys outside the table is possible if the Create entry exists in the PKey category of the current DBMS.

Parameter	Result of the selection
Physical options	Generates physical options for primary keys
Comment	Generates key comments
Drop primary key	Deletes an existing primary key, before creating a new primary key

Alternate key options

These options control the generation of alternate keys.

Parameter	Result of the selection
Create alternate key	Generates alternate keys. If selected you can choose between: ♦ Inside Table - alternate keys are generated during table creation ♦ Outside - alternate keys are generated with a separate SQL command, generally using an ALTER command after the creation of the table The generation of alternate keys outside the table is possible if the Create entry exists in the Key category of the current DBMS.
Physical options	Generates physical options for alternate keys
Comment	Generates alternate key comments
Drop alternate key	Deletes an existing alternate key, before creating a new alternate key

Foreign keys options

These options control the generation of foreign keys.

Parameter	Result of selection
Create foreign key	Generates foreign keys. If selected you can choose between: ♦ Inside Table - foreign keys are generated during table creation ♦ Outside - foreign keys are generated with a separate SQL command, generally using an ALTER command after the creation of the table The generation of foreign keys outside the table is possible if the Create entry exists in the Reference category of the current DBMS.
Decl. Integrity	Generates declarative referential integrity for references that have Declarative selected for referential integrity implementation in the reference property sheet. You can specify any or all of the following: ♦ Update constraint restrict ♦ Update constraint cascade ♦ Update constraint set null ♦ Update constraint set default ♦ Delete constraint restrict ♦ Delete constraint cascade ♦ Delete constraint set null ♦ Delete constraint set default
Comment	Generates foreign key comments
Drop foreign key	Deletes an existing foreign key , before creating a new foreign key

Index options

These options control the generation of indexes.

Parameter	Result of selection

Create index	Generates indexes. If selected you can choose between:
	♦ Inside Table - indexes are generated during table creation
	`create table customer (`
	`   customer_id int not null,`
	`    customer_name varchar(50)`
	`)`
	`unique index CustomerIdx (customer_id);`
	♦ Outside - indexes are generated with a separate SQL command, generally using an ALTER command after the creation of the table
	`create table customer (`
	`   customer_id int not null,`
	`    customer_name varchar(50)`
	`);`
	`create unique index CustomerIdx on Customer(customer_id);`
	The generation of indexes outside the table is possible if the Create entry exists in the Index category of the current DBMS.

Physical options	Generates physical options for indexes
Comment	Generates index comments
Drop index	Deletes an existing index before creating a new index
Index Filter	You can specify from none to all of: ♦ Primary key - Generates primary key indexes ♦ Foreign key - Generates foreign key indexes ♦ Alternate key - Generates alternate key indexes ♦ Cluster - Generates cluster indexes ♦ Others - Generates indexes for all key columns with a defined index

Trigger options

These options control the generation of triggers.

Parameter	Result of selection
Create trigger	Generates triggers
Drop trigger	Deletes an existing trigger, before creating a new trigger

Parameter	Result of selection
Comment	Generates trigger comments
Trigger Filter	You can specify the creation of triggers: ♦ For insert ♦ For update ♦ For delete

View options

These options control the generation of views.

Parameter	Result of selection
Create view	Generates views

Force column list	Generates a view with a list of columns, even if this list is identical to the corresponding columns in the SQL order. Allows you to generate the list of view columns with the view creation order. By default, the list of view columns is generated only if it is different from the list of columns of the view query. For example, in the following view query:
	`select a, b from Table1`
	columns a and b are view columns by default. The default generation statement is:
	`create view V1 as select a, b from Table1`
	If you select the Force column list option, the generation statement will become:
	`create view V1(a,b) as select a, b from Table1`

Physical options	Generates physical options for view
Begin script	Inserts customized script before view creation
End script	Inserts customized script after view creation
Drop view	Deletes an existing view before creating a new view
Comment	Generates view comments

Parameter	Result of selection
Permission	Generates the permission statement for the user during view creation

Synonym options

These options control the generation of synonyms.

Option	Result of selection
Create synonym	Generates synonyms.
Drop synonym	Deletes the existing synonym before creating the new synonym.
Synonym Filter	You can specify from none to all of: ♦ Table - Generates table synonyms ♦ View - Generates view synonyms ♦ Procedure - Generates procedure synonyms ♦ Synonym - Generates synonym synonyms ♦ Database Package - Generates database package synonyms ♦ Sequence - Generates sequence synonyms

Join index options

These options control the generation of join indexes.

Parameter	Result of selection
Create join index	Generates join indexes
Physical options	Generates physical options for join indexes for those DBMS that support it
Begin script	Inserts customized script before join index creation
End script	Inserts customized script after join index creation
Drop join index	Deletes an existing join index, before creating a new join index
Comment	Generates join index comments

Procedure options

These options control the generation of procedures.

Option	Result of selection
Create procedure	Generates procedures.
Begin script	Inserts customized script before procedure creation
End script	Inserts customized script after procedure creation
Drop procedure	Deletes the existing procedure before creating the new procedure.
Comment	Generates procedure comments
Permission	Generates the permission statement for the procedure during creation

Database package options

These options control the generation of database packages.

Option	Result of selection
Create database package	Generates database packages.
Drop database package	Deletes the existing database package before creating the new database package.
Permission	Generates the permission statement for the database package during creation.

Web service options

These options control the generation of web services.

Option	Result of selection
Create web service	Generates web services.
Drop web service	Deletes the existing web service before creating the new web service.
Permission	Generates the permission statement for the web service during creation.

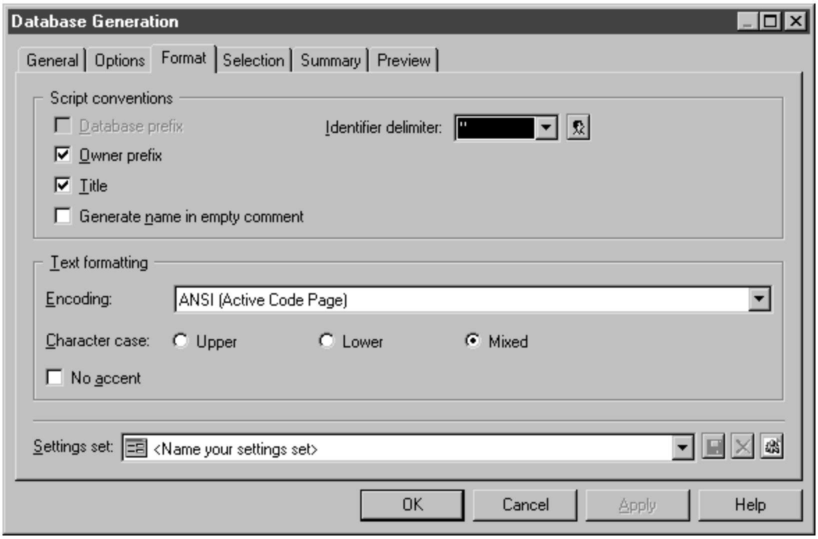
Dimension options

These options control the generation of dimensions.

Option	Result of selection
Create dimension	Generates dimensions.
Drop dimension	Deletes the existing dimension before creating the new dimension.
Permission	Generates the permission statement for the dimension during creation.

Database Generation dialog Format tab

The options on the Format tab allow you to control the format of database generation scripts.



□ Database prefix

Identifier delimiter: "

☑ Owner prefix

☑ Title

□ Generate name in empty comment

Encoding: ANSI (Active Code Page)

Character case: ○ Upper ○ Lower ● Mixed

□ No accent

Settings set: <Name your settings set>

Some of the following options may not be available, depending on your target database.

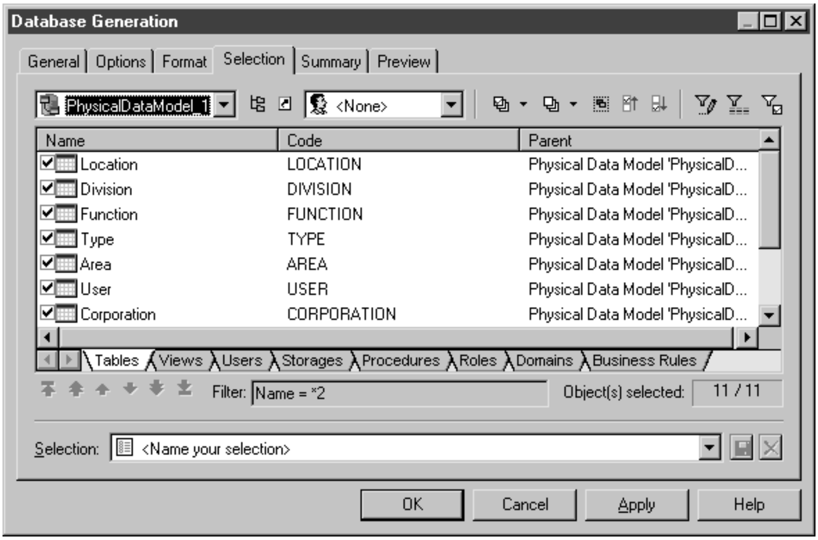
You can save your format settings via the Settings set bar at the bottom of the tab. For more information, see "Quick launch selection and settings sets" on page 441.

Option	Result of selection
Database prefix	Table and view names in the script are prefixed by the database name.
Identifier delimiter	Specifies the characters used to delimit identifiers (for example, table and view names). Most DBMSs require a double-quote character ("), but some permit other forms of delimiter.
Owner prefix	Table and view names in the script are prefixed by their owner names. For those DBMSs that support sequence owners, this option will also prefix sequence names by their owner names.
Title	Each section of the script includes commentary in the form of titles (for example, `Database Name: TUTORIAL`).

Option	Result of selection
Generate name in empty comment	For those DBMSs that support comments, this option allows to generate the name in the comment when the comment box is empty. This option applies to tables, columns, and views. The comment generated using the object name will be reversed as a comment.
Encoding	Specifies an encoding format. You should select a format that supports the language used in your model and the database encoding format.
Character case	Specifies the case to use in the script. You can choose between: ♦ Upper - all uppercase characters ♦ Lower - all lowercase characters ♦ Mixed - lowercase and uppercase characters
No accent	Non-accented characters replace accented characters in script

Database Generation dialog Selection tab

The Selection tab allows you to specify individual objects to generate.



PhysicalDataModel_1

<None>

Name	Code	Parent
☑ Location	LOCATION	Physical Data Model 'PhysicalD...
☑ Division	DIVISION	Physical Data Model 'PhysicalD...
☑ Function	FUNCTION	Physical Data Model 'PhysicalD...
☑ Type	TYPE	Physical Data Model 'PhysicalD...
☑ Area	AREA	Physical Data Model 'PhysicalD...
☑ User	USER	Physical Data Model 'PhysicalD...
☑ Corporation	CORPORATION	Physical Data Model 'PhysicalD...

Filter: Name = *2

Object(s) selected: 11 / 11

Selection: <Name your selection>

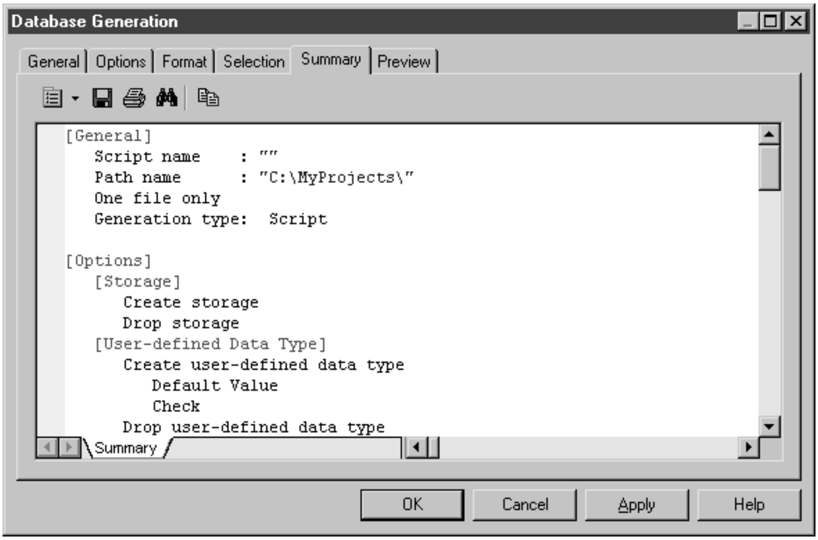
By default, all the objects in the model except for those that belong to a package, or that are shortcuts from another model are listed and selected for generation.

You can save your selection via the Selection bar at the bottom of the tab. For more information, see "Quick launch selection and settings sets" on page 441.

For more information about Selection windows, see "Adding an item from a selection list" section in the Objects chapter of the *Core Features Guide* .

Database Generation dialog Summary tab

The Summary tab allows you to view a summary of your generation options. The option summary is not editable, but you can search, save, print, and copy its contents.

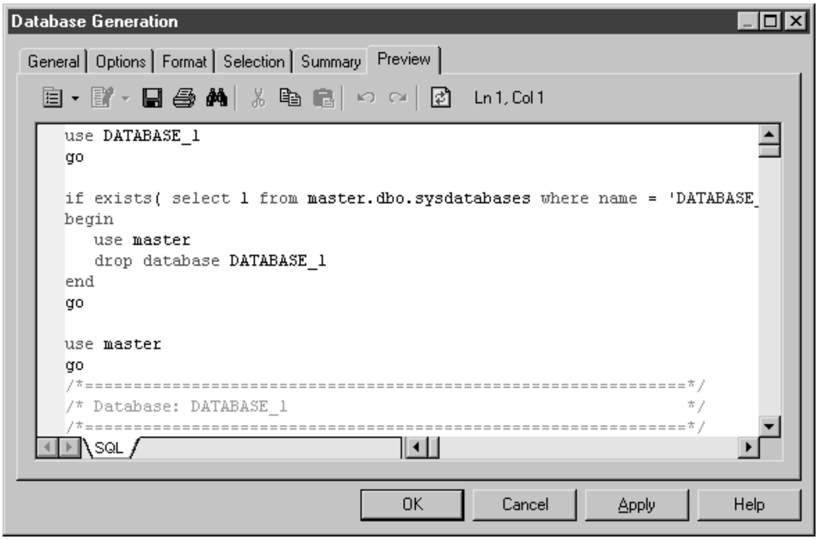


```
[General]
   Script name    : ""
   Path name      : "C:\MyProjects\"
   One file only
   Generation type:  Script

[Options]
   [Storage]
      Create storage
      Drop storage
   [User-defined Data Type]
      Create user-defined data type
         Default Value
         Check
      Drop user-defined data type
```

Database Generation dialog Preview tab

The Preview tab allows you to view the SQL script. The script is not editable, but you can search, save, print, and copy its contents.



```
use DATABASE_1
go

if exists( select 1 from master.dbo.sysdatabases where name = 'DATABASE_
begin
   use master
   drop database DATABASE_1
end
go

use master
go
/*==============================================================*/
/* Database: DATABASE_1                                         */
/*==============================================================*/
```

Generating a Microsoft Access 97 database

PowerDesigner and MS Access 97 use .DAT files to exchange information. These files are created from the PDM files via the script generation. The access.mdb database uses or creates .DAT files to generate or reverse Access databases.

You can define the database generation parameters from the access.mdb database window.

❖ To generate an MS Access database from a PowerDesigner PDM

1. Generate the database script from PowerDesigner.

2. Double-click access.mdb in the PowerDesigner \tools directory.

3. Select Generate Access Database from PowerDesigner Script File.

4. Type the destination database in the Select Database box.

5. Type the file created by PowerDesigner in the PowerDesigner File.

6. Click on the Create button.

Configuring tablespaces and storages

Tablespaces and storages are generic objects used to represent the physical location (in a named partition) of tables and indexes in a database or storage device:

- a tablespace is a partition in a database
- a storage is a partition on a storage device

For some DBMSs, a tablespace can use a specified storage in its definition.

The following table lists the DBMSs that support tablespaces and storages, explains which concept they represent and indicates the commands that implement PowerDesigner tablespace and storage options:

DBMS	Tablespace represents...	Storage represents...
ADABAS	NA	NA
IBM DB2 UDB Common Server	tablespace `create tablespace`	buffer pool `create bufferpool`
IBM DB2 UDB for OS/390	table space `create tablespace`	storage group `create stogroup`
Informix	NA	NA
Ingres	NA	NA
InterBase	NA	NA
Microsoft Access	NA	NA
Microsoft SQL Server	NA	filegroup `alter database add filegroup..`
MySQL	NA	NA
Oracle	tablespace `create tablespace`	storage structure (not physical storage)
PostgreSQL	NA	NA
Sybase ASA	database space `create dbspace`	NA
Sybase ASE	NA	segment `sp-addsegment`
Sybase AS IQ	database space `create dbspace`	NA

DBMS	Tablespace represents. . .	Storage represents. . .
Teradata	NA	NA

Tablespace or storage not applicable
When tablespace or storage options are not applicable for a DBMS, the corresponding model menu item is grayed.

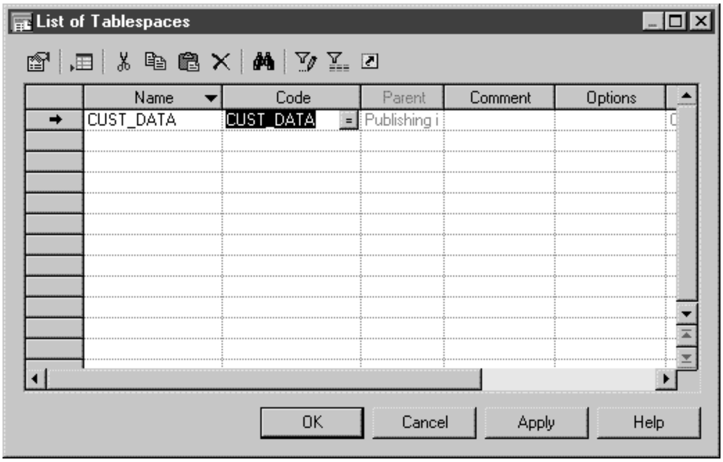
Creating a tablespace or storage

The lists of tablespace and storage options offer pre-defined parameters for each DBMS where applicable. The lists show default values and value lists for certain parameters which correspond to the recommended values for the DBMS.

☞ For more information on tablespace and storage options for a particular DBMS, see its reference manual.

❖ To create a tablespace or storage

1. Select Model ➤ Tablespaces or Model ➤ Storages to open the List of Tablespaces or List of Storages.

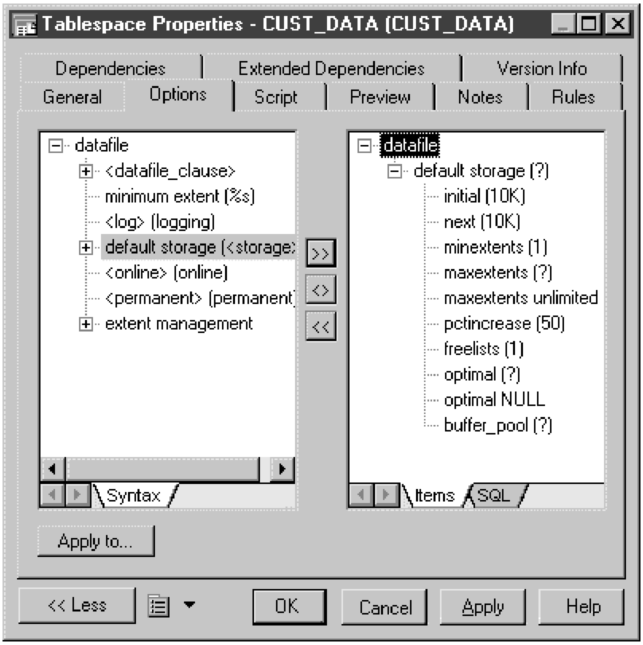


	Name	Code	Parent	Comment	Options
➔	CUST_DATA	CUST_DATA	Publishing i		

2. Click the Add a Row tool to create a tablespace or storage and then click the Properties tool to open its property sheet.

3. Click the Physical Options tab, and select and set the necessary physical options:

Simplified Physical Option tabs
PowerDesigner offers simplified physical option tabs for some DBMSs. Consequently you may have the option of choosing between the Physical Options (Common) and Physical Options (All) tabs.

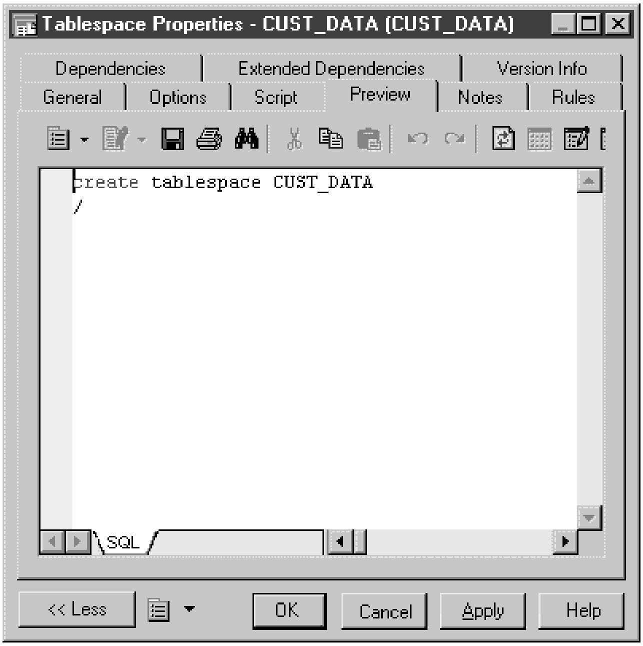


datafile
<datafile_clause>
minimum extent (%s)
<log> (logging)
default storage (<storage>
<online> (online)
<permanent> (permanent)
extent management

datafile
default storage (?)
initial (10K)
next (10K)
minextents (1)
maxextents (?)
maxextents unlimited
pctincrease (50)
freelists (1)
optimal (?)
optimal NULL
buffer_pool (?)

For more information, see “Physical Options” in the Building Physical Diagrams chapter.

4. [optional] Click the Preview tab to review the SQL code to be generated for the tablespace or storage.



```
create tablespace CUST_DATA
/
```

5. Click OK.

   The selected options appear in Options column of the List of Tablespaces or List of Storages.

Customizing scripts

You can customize scripts as follows:

- Insert scripts at the beginning and end of database creation script
- Insert scripts before and after a table creation command

Customizing a creation script allows you to add descriptive information about a generated script, or manipulate the script in such a way that is not provided by PowerDesigner.

Examples

If a development project archives all the database creation scripts that are generated, a header script can be inserted before each creation script, which may indicate the date, time, and any other information specific to the generated script.

If an organization requires that generated scripts are filed using a naming system which may be independent from a script name, a header script could direct a generated script to be filed under a different name than the name indicated in the creation script.

Access rights can be added as a footer to a table creation script.

Script toolbar properties

You can use the following tools and keyboard shortcut from the script toolbar:

Tool	Description	Keyboard shortcut
	Editor context menu	SHIFT + F11
	Edit With. Opens the default editor you previously defined	CTRL + E

☞ For more information on defining a default editor, see section “Specifying text editors” in the Models chapter of the *Core Features Guide* .

Inserting begin and end scripts for database creation

In a database creation script, you can insert the following scripts:

Script	Where it is inserted
Begin script	Before the command that creates the database
End script	After the last command in the database creation script

You can use the following variables in these scripts:

Variable	Description
%DATABASE%	Name of the current PDM
%DATE%	Date of script generation
%DBMSNAME%	Name of the DBMS for the target database
%NAMESCRIPT%	Filename of script file
%PATHSCRIPT%	Filename and path of script file
%STARTCMD%	Command that runs the script
%AUTHOR%	Author of the current model

❖ To insert begin and end scripts for database creation

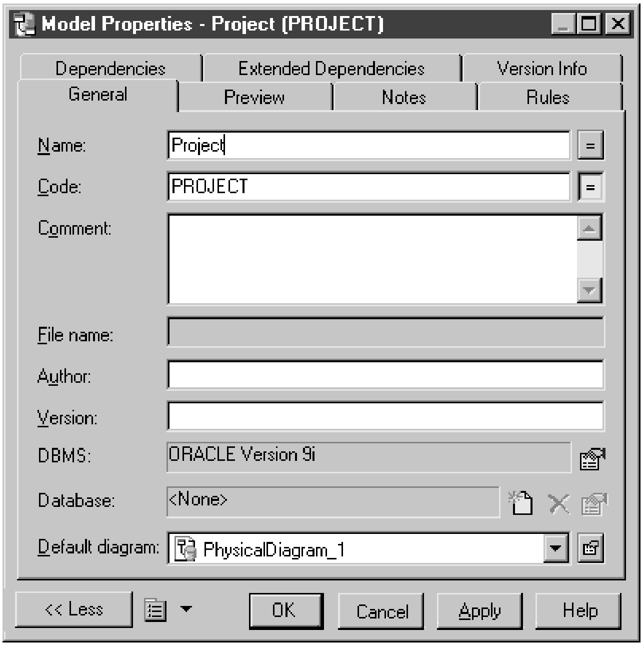
1. Select Model ➤ Model Properties.

   or

   Right click the diagram background.

   Select Properties from the contextual menu.

   The model property sheet is displayed.

   Model Properties - Project (PROJECT)

   Name: Project

   Code: PROJECT

   Comment:

   File name:

   Author:

   Version:

   DBMS: ORACLE Version 9i

   Database: <None>

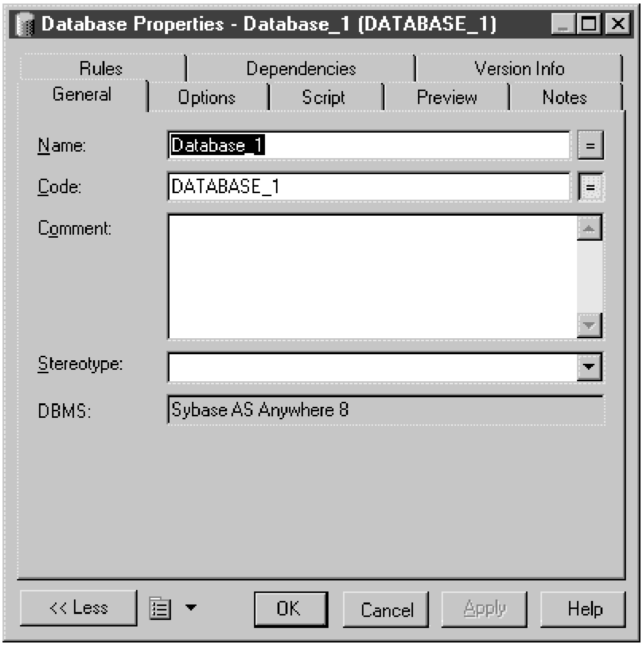
   Default diagram: PhysicalDiagram_1

2. Click the Create tool next to the Database box.

   A confirmation box is displayed asking you to commit the creation of the database script object.

3. Click Yes.

   The database property sheet is displayed.



Database Properties - Database_1 (DATABASE_1)

Name: Database_1

Code: DATABASE_1

Comment:

Stereotype:

DBMS: Sybase AS Anywhere 8

4. Type a name and code for the database.
5. Click the Script tab.

   The Script tab opens to the Begin tab.
6. Type the DBMS-specific commands to insert in the beginning of the script.
7. Click the End tab at the bottom of the tab.
8. Type the DBMS-specific commands to insert at the end of the script.
9. Click OK in each of the dialog boxes.

Inserting begin and end scripts for table creation

For each table, you have the option to insert the following scripts:

Script	Where it is inserted
Begin script	Immediately before the table creation command (after the table title)
End script	Immediately after the table creation command

These scripts can appear in database creation scripts and database modification scripts.

You can use the following variables in these scripts:

Variable	Description
%COLNLIST%	*Column list*
%DATABASE%	Code of the current PDM
%DATE%	Date of script generation
%DBMSNAME%	Code of the DBMS for the target database
%NAMESCRIPT%	Filename of script file
%OWNER%	Table owner
%OWNERPREFIX%	Owner prefix of table owner
%PATHSCRIPT%	Filename and path of script file
%STARTCMD%	Command that runs the script
%TABLE%	Name or code of current table (based on display preferences)
%TCODE%	Code of the current table
%TLABL%	Label of the current table
%TNAME%	Name of the current table
%AUTHOR%	Author of the current model

❖ To insert begin and end scripts for table creation

1. Double-click a table symbol.

   The table property sheet is displayed.

2. Click the Script tab.

   The Script tab opens to the Begin tab.

3. Type the DBMS-specific commands to insert in the beginning of the script.

4. Click the End tab at the bottom of the tab.

5. Type the DBMS-specific commands to insert at the end of the script.

6. Click OK.

Inserting begin and end script for tablespace creation

For each tablespace, you have the option to insert the following scripts:

Script	Where it is inserted
Begin script	Immediately before the tablespace creation command (after the table title)
End script	Immediately after the tablespace creation command

These scripts can appear in database creation scripts and database modification scripts.

You can use the following variables in these scripts:

Variable	Description
%DATABASE%	Code of the current PDM
%DATE%	Date of script generation
%DBMSNAME%	Code of the DBMS for the target database
%NAMESCRIPT%	Filename of script file
%PATHSCRIPT%	Filename and path of script file
%STARTCMD%	Command that runs the script
%TABLESPACE%	Code of the tablespace
%OPTIONS%	Physical options of the tablespace
%AUTHOR%	Author of the current model

❖ To insert begin and end scripts for tablespace creation

1. Double-click a tablespace in the list of tablespace.

   The tablespace property sheet is displayed.

2. Click the Script tab.

   The Script tab opens to the Begin tab.

3. Type the DBMS-specific commands to insert in the beginning of the script.

4. Click the End tab at the bottom of the tab.

5. Type the DBMS-specific commands to insert at the end of the script.

6. Click OK.

Formatting variables in customized scripts

Variables have a syntax that can force a format on their values. Typical uses are as follows:

- Force values to lowercase or uppercase characters
- Truncate the length of values
- Enquote text

You embed formatting options in variable syntax as follows:

```
%[[?][-][width][.[-
          ]precision][c][H][F][U|L][T][M][q][Q]:]<varname>%
```

The variable formatting options are the following:

Format option	Description
?	Mandatory field, if a null value is returned the translate call fails
n (where n is an integer)	Blanks or zeros added to the left to fill the width and justify the output to the right
-*n*	Blanks or zeros added to the right to fill the width and justify the output to the left
width	Copies the specified minimum number of characters to the output buffer
.[-]precision	Copies the specified maximum number of characters to the output buffer
.L	Lower-case characters
.U	Upper-case characters
.F	Combined with L and U, applies conversion to first character
.T	Leading and trailing white space trimmed from the variable
.H	Converts number to hexadecimal
.c	Upper-case first letter and lower-case next letters

Format option	Description
.*n*	Truncates to *n* first characters
.-*n*	Truncates to *n* last characters
M	Extracts a portion of the variable name, this option uses the width and precision parameters to identify the portion to extract
q	Enquotes the variable (single quotes)
Q	Enquotes the variable (double quotes)

☞ For more examples on variable formatting, see section Defining variable formatting options in the DBMS Resource File Reference chapter in the *Customizing and Extending PowerDesigner* manual.

Using Test Data

Test data is sample data that you can define and generate for one or more tables in a PDM. When you generate test data, PowerDesigner automatically generates rows of data in database tables.

You normally use test data to verify the performance of a database that is being developed. You can use test data to do the following:

- Verify the performance of the database when it is filled with large amounts of data
- Verify the performance of the database when it is accessed by different applications or users
- Verify the operational performance of the database when it is accessed by different applications or users
- Estimate the amount of memory space the database will take up
- Examine data formatting in the database

You can generate test data for selected tables in a PDM, or for all the tables in a PDM.

You can add test data to an empty database or to a database that already contains data.

Test Data profile

You generate test data for a table based on test data profiles. A test **data profile** is a named class of data types that has a defined data generation source. A test data profile can be assigned to one or more columns.

When you assign a test data profile to a column, it acts as a representative for the data type of that column. Test data for the column is then generated using the data profile and its defined data source.

You can also select a test data profile to apply to a domain. You can then choose to have the profile automatically assigned to all columns that use that domain.

A test data profile can use one of the following:

- Number
- Character
- Date/Time

Example

For a column named Employee Location, you create a test data profile named Address using the Character class. You then define a test data source for Address. You can also assign it to other columns, such as Store Location, and Client Address.

Default test data profiles

If you do not assign a test data profile for a column, a default test data profile is assigned to the column when you generate test data for the table. You define default test data profiles for each class used to generate test data.

Test data restrictions

The following objects are not taken into account when you generate test data:

- Alternate keys
- Foreign keys
- Business and validation rules
- Binary, sequential, OLE, text or image data types
- Trigger contents

❖ To set up a test data profile

1. Create a test data profile for one or more columns.
2. Define a test data generation source for each data profile.
3. Define column fill parameters for columns that need to be populated with data in a particular way.
4. Assign test data profiles to all relevant columns.
5. Use the test data profiles to generate test data.

Creating a test data profile

When you create a test data profile you need to define the following properties:

Property	Description
Name	Name of the profile
Code	Code of the data profile
Profile class	You can assign one of the following data type classes: Number Character Date/Time

When you define the profile class, make sure it reflects the DBMS limitations, for example, someDBMS do not support dates prior to a certain date.

Generation source

You define a test data generation source for each data profile. You can use the following test data generation sources:

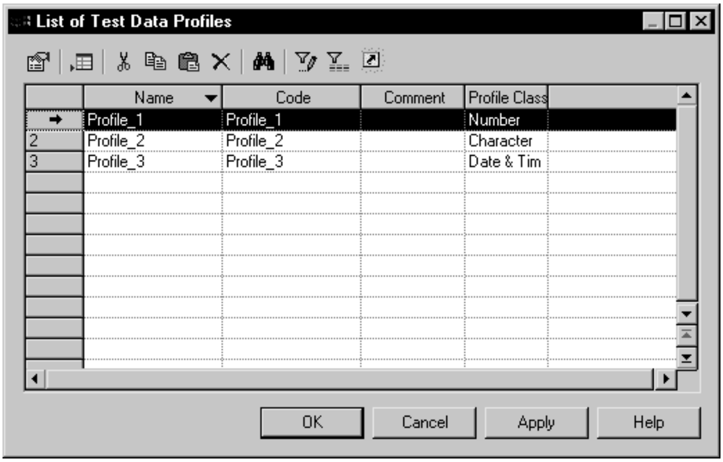
Generation source	Test data values are generated
Automatic	By PowerDesigner
List	From a list of test data values
Direct	From a live database connection
File	From a source file

Each data generation source has its own set of options for defining generation parameters.

❖ To create a test data profile

1. Select Model ➤ Test Data Profiles.

   The List of Test Data Profiles is displayed.



	Name	Code	Comment	Profile Class
➜	Profile_1	Profile_1		Number
2	Profile_2	Profile_2		Character
3	Profile_3	Profile_3		Date & Tim

2. Click a blank line in the list.

3. Type a test data Profile name and a profile code.

4. Select a data type class from the Profile Class list.

Display the column you need

If you do not see the column you need, display it with the Customize Columns and Filter tool. For details, see "Customizing object list columns and filtering lists" section in the Objets chapter of the *Core Features Guide* .

5. Click Apply.
6. Double-click the new profile to display its property sheet.
7. Define generation source parameters.
8. Click OK in each of the dialog boxes.

Automatic test data generation source parameters

There are different automatic test data generation source parameters for each profile class.

Defining an automatic test data generation source for numbers

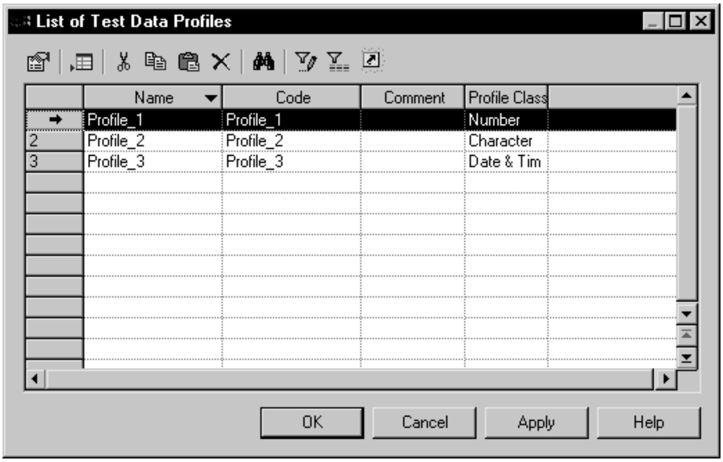
You have the following options to define generation source parameters for the profile class Number:

Option	Description
Random	Indicates to generate random data
Sequential	Indicates to generate sequential data
From/To	Indicates the range for random or sequential data
Step	Indicates interval between each sequential number step
Generate decimal numbers	Generates decimal numbers
Decimal digits number	Indicates the maximum number of decimal places to be generated

❖ To define an automatic data generation source

1. Select Model ➤ Test Data Profiles.

   The List of Test Data Profiles is displayed.

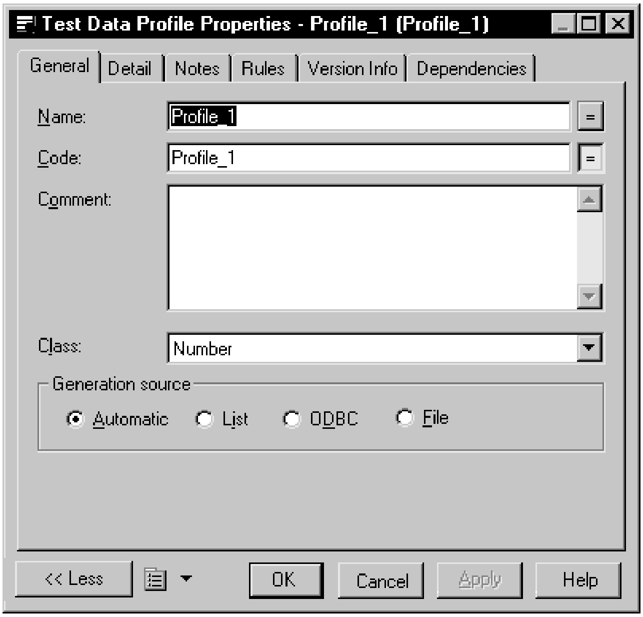


	Name	Code	Comment	Profile Class
➔	Profile_1	Profile_1		Number
2	Profile_2	Profile_2		Character
3	Profile_3	Profile_3		Date & Tim

2. Double-click a test data profile using numbers.

   The Profile properties sheet opens to the General tab.

3. Select the Automatic radio button.



Name: Profile_1

Code: Profile_1

Comment:

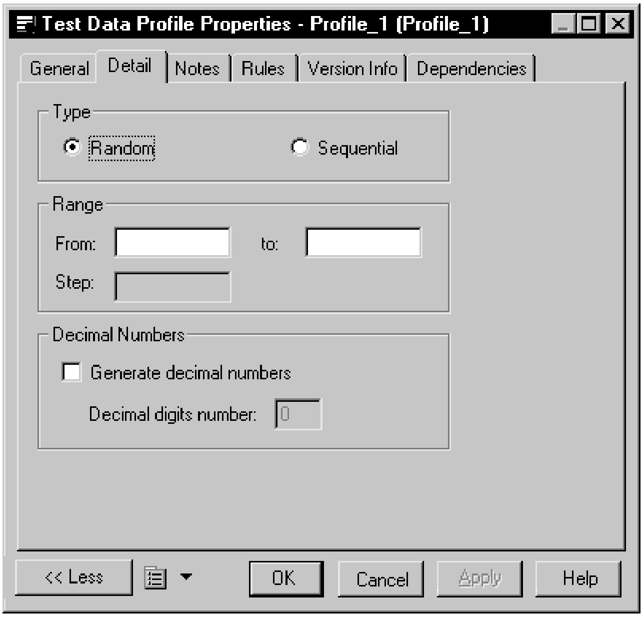
Class: Number

Generation source

● Automatic ○ List ○ ODBC ○ File

4. Click the Detail tab.

The Detail tab is displayed.



Type

● Random ○ Sequential

Range

From: to:

Step:

Decimal Numbers

□ Generate decimal numbers

Decimal digits number: 0

5. Specify test data generation parameters.

6. Click OK in each of the dialog boxes.

Defining an automatic test data generation source for characters

You have the following options to define generation source parameters for the profile class Character:

Option	Parameter	Description
Verify	Valid characters	List of authorized characters
	All	Accepts all characters
	Invalid characters	List of unauthorized characters
	No accents	Replaces accents with unaccented characters
	Mask	String of mask characters.
Case	Upper	Indicates authorized letter cases
	First uppercase	
	Lower	
	Mixed	

Option	Parameter	Description
Length	Exact	You indicate exact character length
	From/To	You indicate character length range

Character selection

You can specify valid characters to accept and invalid characters to refuse. You use a comma (**,**) to separate each single character, character interval, or string.

The following syntax applies to valid and invalid characters.

Character set	Syntax	Example
Interval of characters	Characters in single quotation marks separated by a dash	`'a'-'z'`
Single character or Character string	Characters in quotation marks	`"a" | "xyz"`

Mask characters

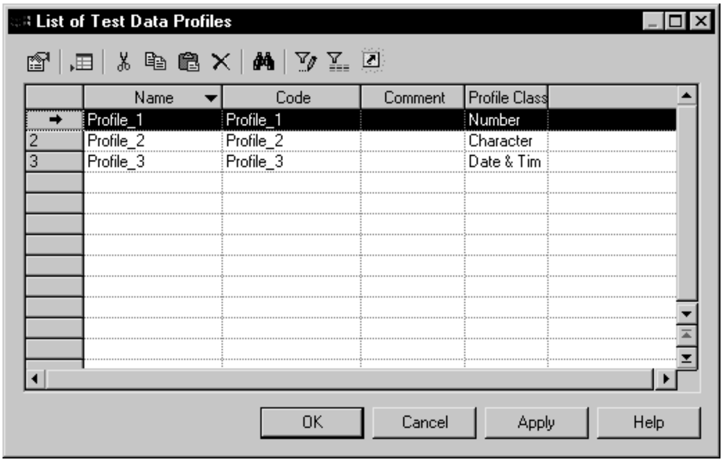
A mask character is a pre-defined character that indicates to users that they need to type a particular piece of information. The test data that is generated respects the following mask characters:

Mask character	Prompts to type
A	Letter
9	Number
?	Any character

❖ To define an automatic data generation source

1. Select Model ➤ Test Data Profiles.

   The List of Test Data Profiles is displayed.

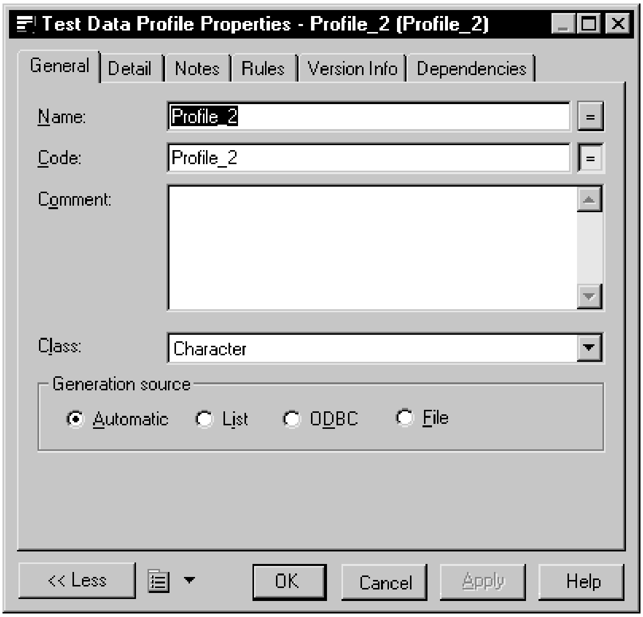


	Name	Code	Comment	Profile Class
➔	Profile_1	Profile_1		Number
2	Profile_2	Profile_2		Character
3	Profile_3	Profile_3		Date & Tim

2. Double-click a test data profile using characters.

   The Profile properties sheet opens to the General tab.

3. Select the Automatic radio button.



Name: Profile_2

Code: Profile_2

Comment:

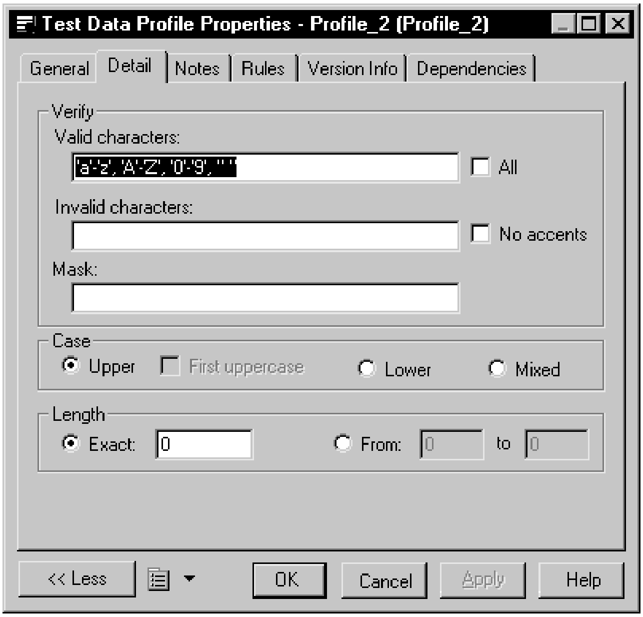
Class: Character

Generation source

● Automatic ○ List ○ ODBC ○ File

4. Click the Detail tab.

The Detail tab is displayed.



Verify

Valid characters: 'a'-'z', 'A'-'Z', '0'-'9', ' ' □ All

Invalid characters: □ No accents

Mask:

Case

● Upper □ First uppercase ○ Lower ○ Mixed

Length

● Exact: 0 ○ From: 0 to 0

5. Specify test data generation parameters.
6. Click OK in each of the dialog boxes.

Defining an automatic test data generation source for date and time

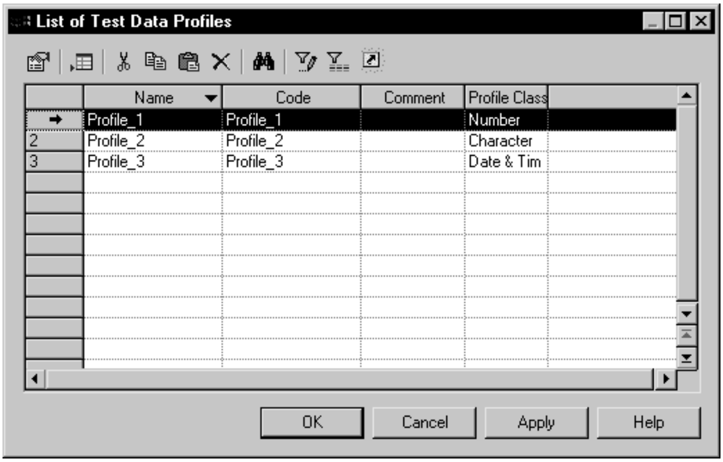
You can define the following generation source parameters for the profile class Date/Time:

Value	Parameter	Description
Date range	From/To	Indicates range of date values
Time range	From/To	Indicates range of time values
Step	Date/Time step	Indicates date and time intervals for sequential data values.
Values	Random	Indicates to generate random data
	Sequential	Indicates to generate sequential data

❖ To define an automatic data generation source

1. Select Model ➤ Test Data Profiles.

   The List of Test Data Profiles is displayed.

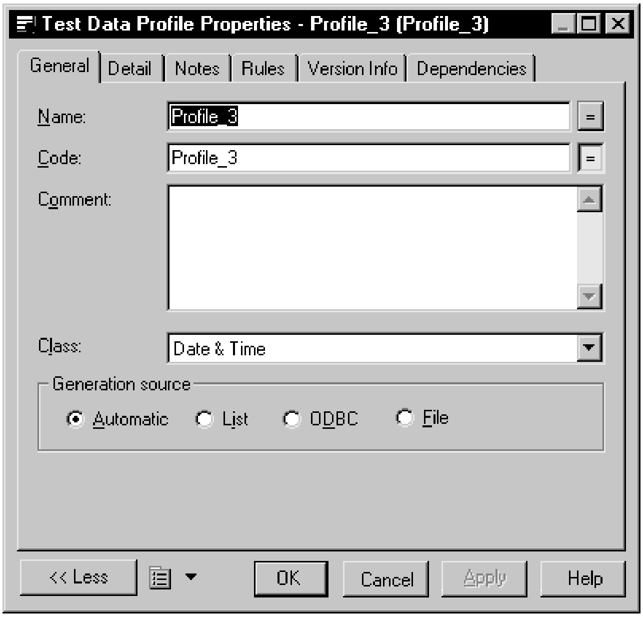


	Name	Code	Comment	Profile Class
➔	Profile_1	Profile_1		Number
2	Profile_2	Profile_2		Character
3	Profile_3	Profile_3		Date & Tim

2. Double-click a data profile using date and time.

   The Profile properties sheet opens to the General tab.

3. Select the Automatic radio button.



Name: Profile_3

Code: Profile_3

Comment:

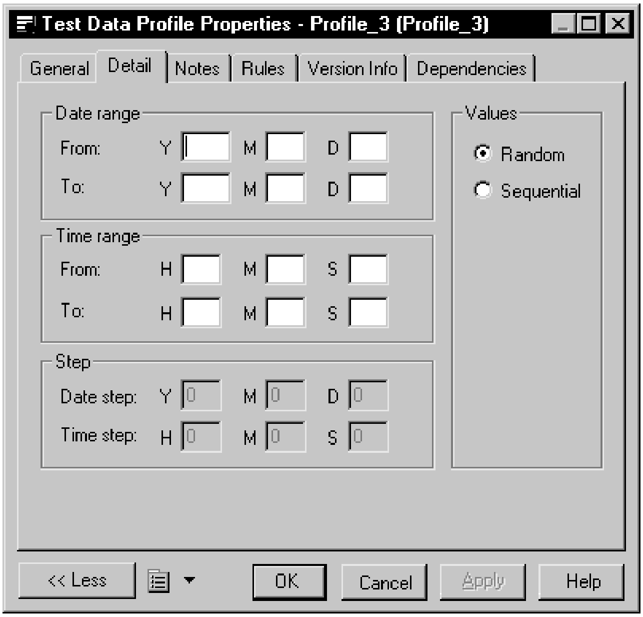
Class: Date & Time

Generation source

● Automatic ○ List ○ ODBC ○ File

4. Click the Detail tab.

The Detail tab is displayed.



Date range

From: Y M D

To: Y M D

Time range

From: H M S

To: H M S

Step

Date step: Y 0 M 0 D 0

Time step: H 0 M 0 S 0

Values

● Random

○ Sequential

5. Specify test data generation parameters.

6. Click OK in each of the dialog boxes.

Defining a list as a test data generation source

You can use information from a list to define a test data generation source for a data profile.

❖ To define a list as a test data generation source

1. Select Model ➤ Test Data Profiles.

   The List of Test Data Profiles is displayed.

2. Double-click a data profile.

   The Profile properties sheet opens to the General tab.

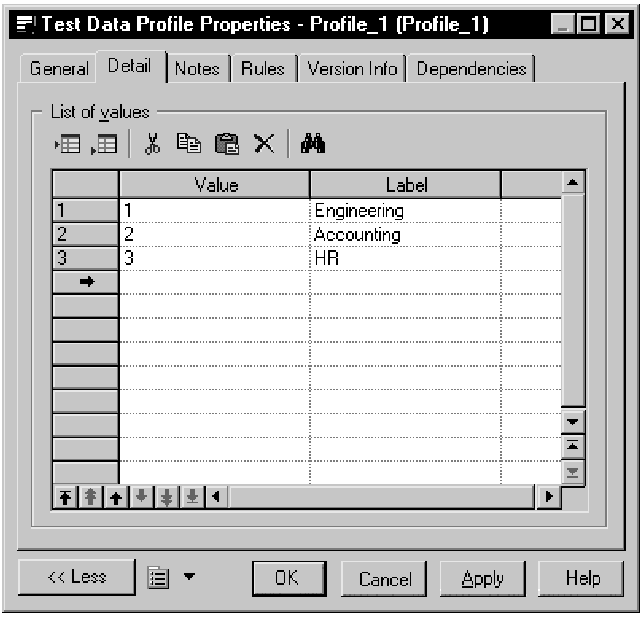
3. Select the List radio button.

4. Click the Detail tab.

   The Detail tab is displayed.

5. Select a Value profile.

6. Type a value and label.



	Value	Label
1	1	Engineering
2	2	Accounting
3	3	HR

7. Click OK in each of the dialog boxes.

Defining a live database connection as a test data generation source

You can use information from a live database connection to define a test data generation source for a data profile.

❖ To define a database as a test data generation source

1. Select Model ➤ Test Data Profiles to open the List of Test Data Profiles.

2. Double-click a data profile to open its property sheet.

3. Select the Database Generation radio button, and then click the Detail tab.

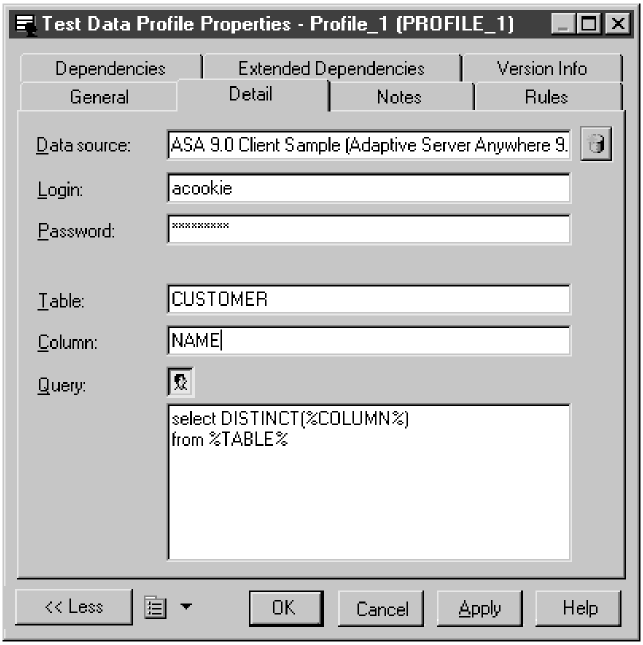
4. Click the Select a data source tool beside the Data source box and select a machine or file data source. For more information, see “Connecting to a Database” section in the Models chapter of the *Core Features Guide* .

5. Type the login and password corresponding to your data source.

6. Type the name of the table and column to query in the data source.

   The variables in the default query are replaced by the table and column values.

You can also type a user-defined query in the Query box. The user-defined button indicates when the default query has been modified. You can click this button to recover the default query.



Data source: ASA 9.0 Client Sample (Adaptive Server Anywhere 9.

Login: acookie

Password: *********

Table: CUSTOMER

Column: NAME

Query:

```
select DISTINCT(%COLUMN%)
from %TABLE%
```

7. Click OK.

   You return to the List of Test Data Profiles.

8. Click OK.

Defining a file as a test data generation source

You can import a CSV file to define a data generation source for a data profile. You can define the following generation source parameters for the CSV file:

Value	Parameter	Description
Values	Random	Indicates to import file lines in random order
	Sequential	Indicates to import file lines sequentially

CSV import restriction

Each CSV file that you import for each data profile must come from the same directory. If you have not previously used this directory as a CSV file test data source, you will need to select the correct directory to define the file path.

Example test data files
PowerDesigner provides example test data files in the \TESTDATA directory.

❖ To define a file as a test data generation source

1. Select Model ➤ Test Data Profiles.

   The List of Test Data Profiles is displayed.

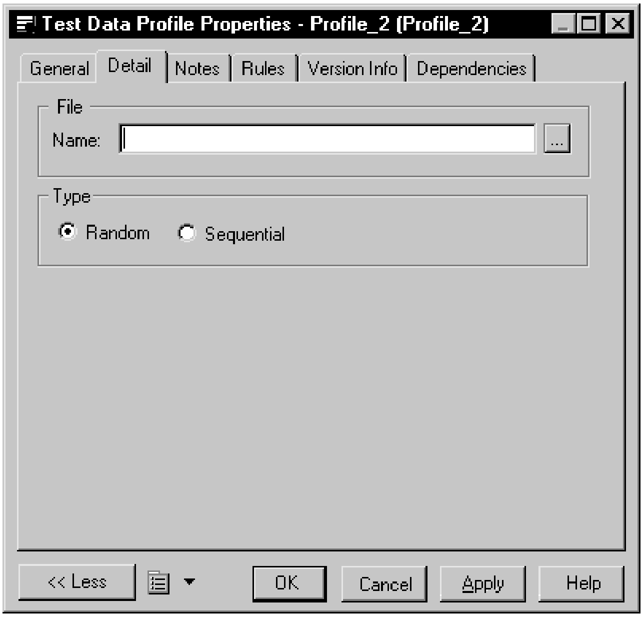
2. Double-click a data profile.

   The Profile properties sheet opens to the General tab.

3. Select the File radio button.

4. Click the Detail tab.

   The Detail tab is displayed.



Type: ● Random ○ Sequential

A dialog box asks you to select a file.

Specifying the CSV directory
If you are importing a CSV file for the first time from a specific directory, you need to select the appropriate directory.

5. Select a file and click Open.

   You return to the Detail tab.

6. Specify test generation parameters.
7. Click OK.

   You return to the List of Test Data Profiles.
8. Click OK.

Defining column fill parameters

Fill parameters determine how a column is filled or populated with test data. You can define column fill parameters when you assign a data profile to the column, or as an independent procedure.

You can define the following fill parameters:

Value	Description
Null values	Percentage of column entries containing a null value
Distinct values	Indication of the percentage of column rows that contain unique entries. This is a maximum value, and can change automatically depending on the referential integrity parameters of primary key columns. Alternately, a specific value can be entered, without a percentage sign, to indicate the exact number of column rows that contain unique entries.

Average Length
The Average Length box is only used for the Estimate data base size function. The value that is displayed by default, is the maximum length for the data type defined for a selected column.

Indicated column properties

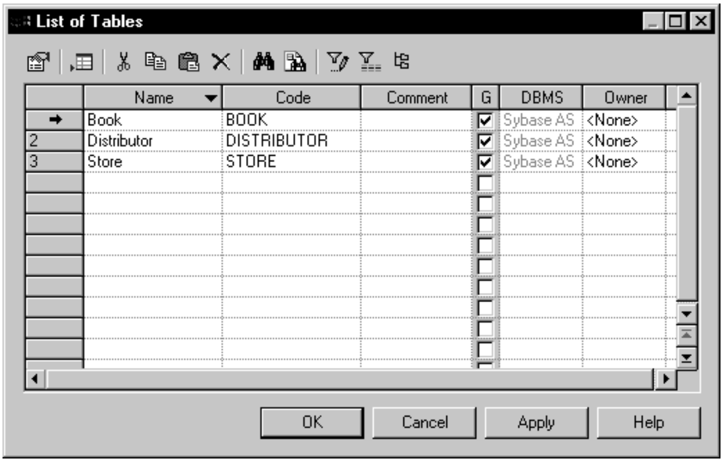
The following column properties are indicated on the Columns sheet:

Indicator	Property	When selected, indicates that...
M	Mandatory	Column must be assigned a value. The fill parameter Null value is automatically defined as 0%
U	Unique column	Column is the only column in a primary key, alternate key, or unique index. The fill parameters Null Values and Distinct Values are automatically defined as 0% and 100% respectively
F	Foreign	Column is a foreign key column. You cannot assign a data profile to this column. It automatically takes the data profile of the corresponding primary key column in the parent table

❖ To define column fill parameters

1. Select Model ➤ Tables.

   The List of tables is displayed.



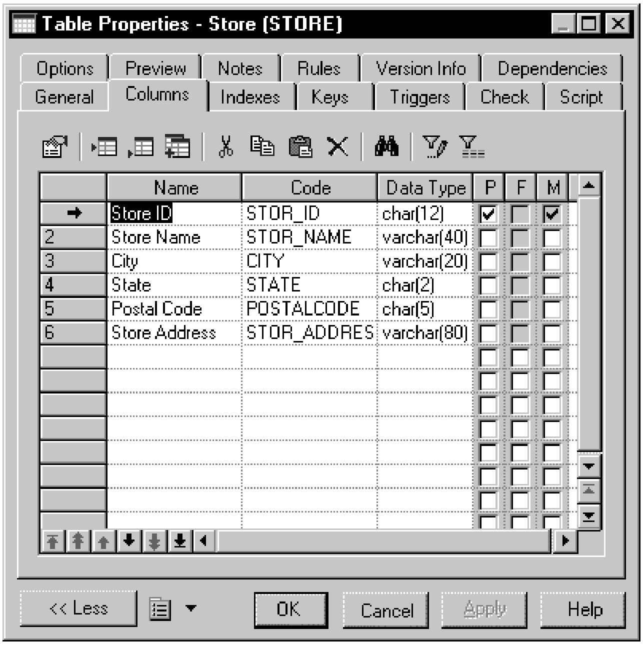
	Name	Code	Comment	G	DBMS	Owner
→	Book	BOOK		☑	Sybase AS	<None>
2	Distributor	DISTRIBUTOR		☑	Sybase AS	<None>
3	Store	STORE		☑	Sybase AS	<None>

2. Double-click a table from the list.

   The Profile properties sheet opens to the General tab.

3. Select the Columns tab.

   The Columns tab is displayed.



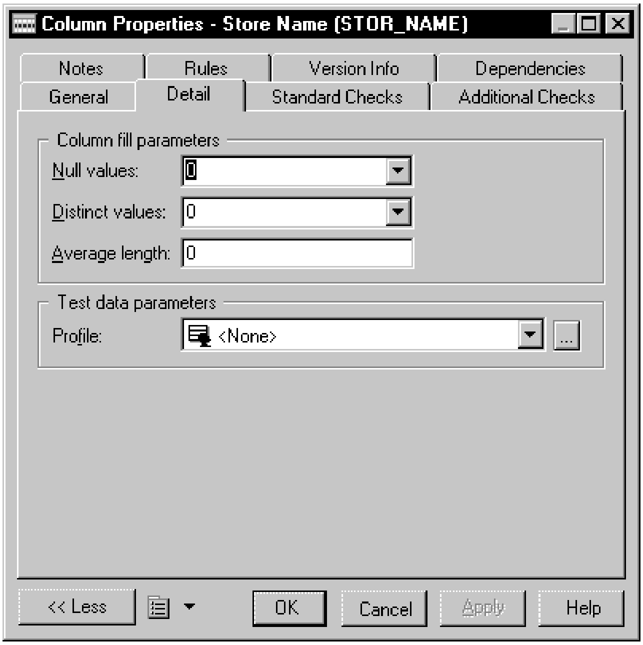
	Name	Code	Data Type	P	F	M
→	Store ID	STOR_ID	char(12)	☑	□	☑
2	Store Name	STOR_NAME	varchar(40)	□	□	□
3	City	CITY	varchar(20)	□	□	□
4	State	STATE	char(2)	□	□	□
5	Postal Code	POSTALCODE	char(5)	□	□	□
6	Store Address	STOR_ADDRES	varchar(80)	□	□	□

4. Double-click a column name.

   The Column properties dialog box is displayed.

5. Select the Detail tab.

   The Detail tab is displayed.



Column Properties - Store Name (STOR_NAME)

Column fill parameters

Null values: 0

Distinct values: 0

Average length: 0

Test data parameters

Profile: <None>

6. Select or type Column fill parameters.

7. Select a profile from the test data parameters dropdown list box.

8. Click OK in each dialog box.

   You return to the Columns tab.

9. Click OK.

Assigning a data profile

When you assign a data profile to a column, the data profile defines the type of test data that you generate for that column.

You can assign a data profile to each column in a table, except for the foreign key columns. The data profiles for foreign keys are assigned by default.

When you assign a data profile to a column that contains check parameters, the generated test data respects the column constraints.

There are three approaches to assigning data profiles:

- Assign a data profile from the list of tables
- Assign a data profile from the list of columns
- Select a data profile for a domain, so that all columns attached to that domain are automatically assigned the data profile.

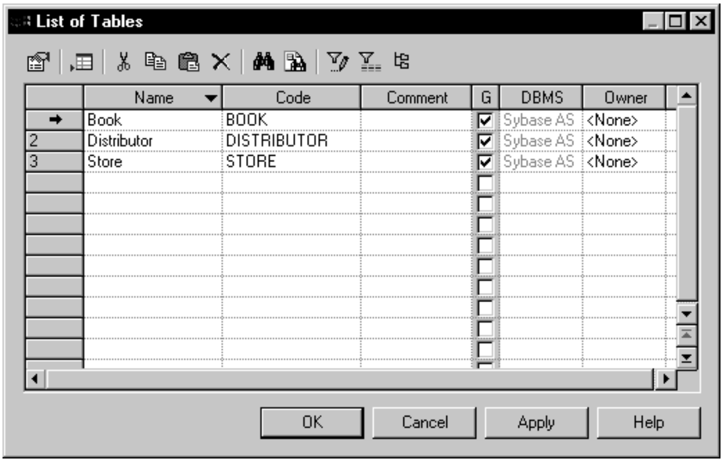
☞ For more information on how to select a data profile for a domain, see chapter Building Physical Diagrams.

Assigning a data profile from the list of tables

❖ To assign a data profile from the list of tables

1. Select Model ➤ Tables.

   The List of tables is displayed.



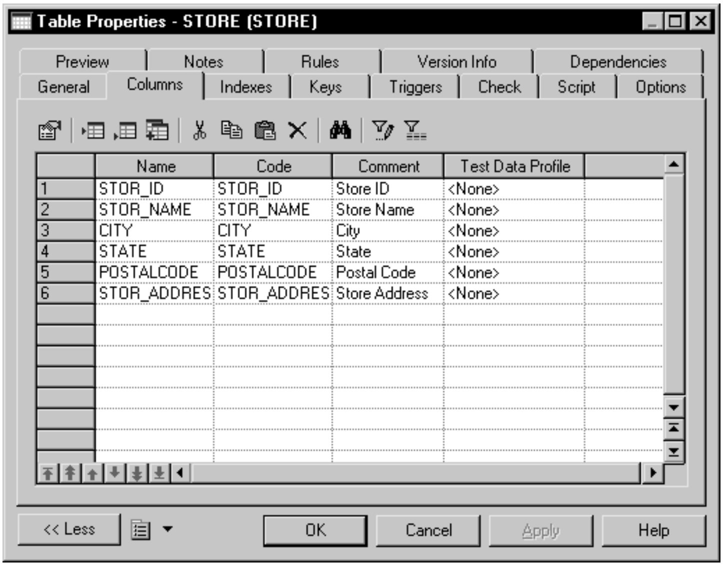
	Name	Code	Comment	G	DBMS	Owner
→	Book	BOOK		☑	Sybase AS	<None>
2	Distributor	DISTRIBUTOR		☑	Sybase AS	<None>
3	Store	STORE		☑	Sybase AS	<None>

2. Double-click a table in the list.

   The table property sheet opens to the General tab.

3. Select the Columns tab.

   The Columns tab is displayed.



	Name	Code	Comment	Test Data Profile
1	STOR_ID	STOR_ID	Store ID	<None>
2	STOR_NAME	STOR_NAME	Store Name	<None>
3	CITY	CITY	City	<None>
4	STATE	STATE	State	<None>
5	POSTALCODE	POSTALCODE	Postal Code	<None>
6	STOR_ADDRES	STOR_ADDRES	Store Address	<None>

4. Select a column in the list.

5. Select a test data profile from the dropdown list box.

Display the column you need
If you do not see the column you need, display it with the Customize Columns and Filter tool. For details, see “Customizing object list columns and filtering lists” section in the Objets chapter of the *Core Features Guide* .

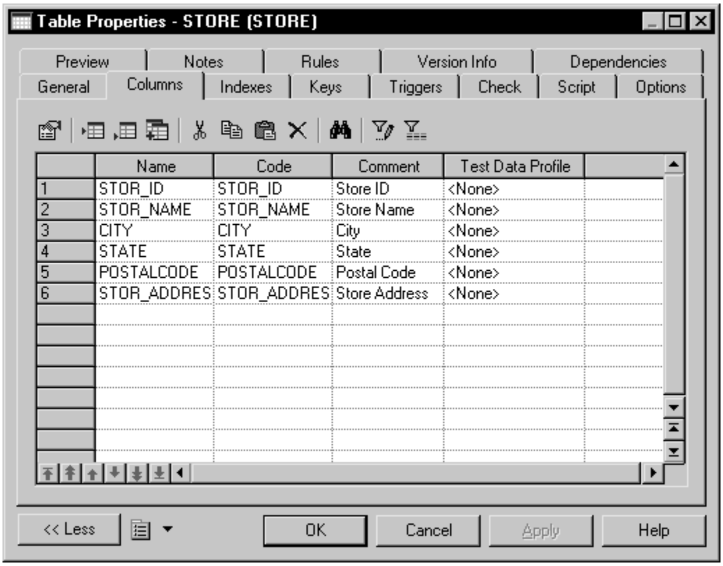
6. Click OK.

Assigning a data profile from the list of columns

❖ To assign a data profile from the list of columns

1. Select Model ➤ Columns.

   The List of Columns is displayed.



	Name	Code	Comment	Test Data Profile
1	STOR_ID	STOR_ID	Store ID	<None>
2	STOR_NAME	STOR_NAME	Store Name	<None>
3	CITY	CITY	City	<None>
4	STATE	STATE	State	<None>
5	POSTALCODE	POSTALCODE	Postal Code	<None>
6	STOR_ADDRES	STOR_ADDRES	Store Address	<None>

2. Select a column in the list.

3. Select a test data profile from the dropdown list box.

Display the column you need
If you do not see the column you need, display it with the Customize Columns and Filter tool. For details, see "Customizing object list columns and filtering lists" section in the Objets chapter of the *Core Features Guide* .

4. Click OK.

Importing a data profile

You can import data profiles in XPF format from another PDM. The imported profiles appear in the list of profiles.

❖ To import a data profile

1. Select Tools ➤ Test Data Profiles ➤ Import.

   A standard Open dialog box is displayed.

2. Browse to the EXAMPLES/TUTORIAL directory.

3. Select or type a file name with the XPF extension.

4. Click Open.

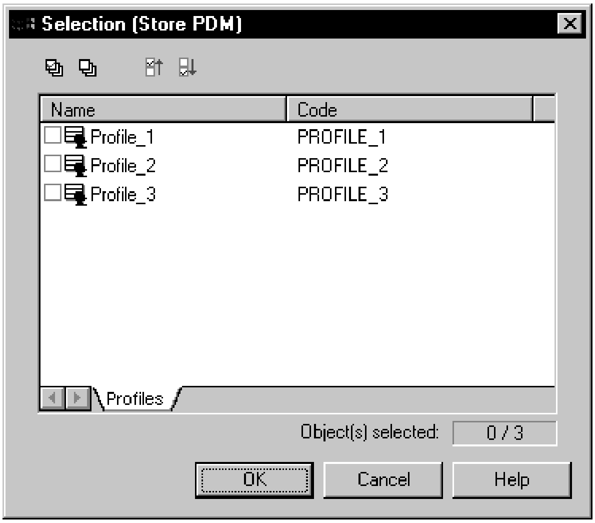
Exporting a data profile

You can export data profiles to a XPF file. In another PDM, you can then import the data profiles defined in this XPF file.

❖ To export a data profile

1. Select Tools ➤ Test Data Profiles ➤ Export.

   A Selection dialog box is displayed.



Name	Code
□ Profile_1	PROFILE_1
□ Profile_2	PROFILE_2
□ Profile_3	PROFILE_3

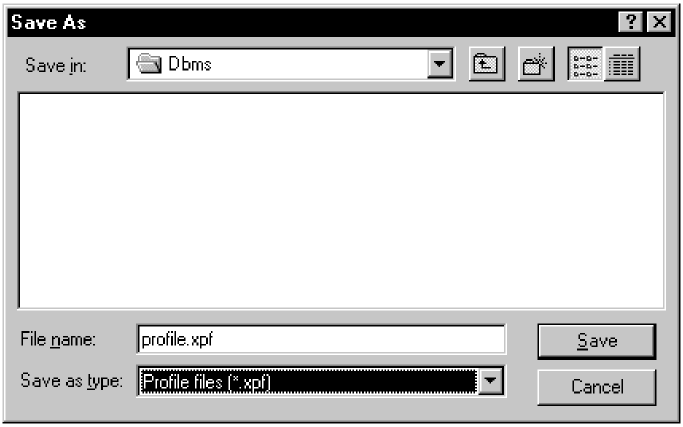
Object(s) selected: 0 / 3

2. Select one or more data profiles that you want to export.

3. Click OK.

   A dialog box is displayed.

4. Type a file name with the XPF extension.



File name: profile.xpf

Save as type: Profile files (*.xpf)

5. Click Save.

Generating test data

You can generate test data for all the tables in a PDM, or for selected tables.

You can generate data for either an empty database or for an existing database.

Avoid generating triggers
Do not implement triggers if you generate a new database. It is also recommended that you remove triggers from an existing test database. Triggers can considerably increase the time required to generate the database and can block insertions. In addition, they are not needed for the type of test we are performing.

❖ To generate test data

1. Assign data profiles to the columns in each table for which you want to generate test data (see "Assigning a data profile" on page 488).

   Default values for data profiles
   Test data generation is typically based on user-defined data profiles. If you have no data profile definitions, you are automatically provided with default values. The default number of rows is set to 20 and the value <default> is displayed in each of the following three default profile boxes.

2. Select Database ➤ Generate Test Data to open the Test Data Generation dialog box.

3. Specify a directory and filename for your test data file, and then choose between Script and Direct generation. For more information, see "Test Data Generation General tab" on page 494.

4. [optional] To change the number of rows to be generated for specific tables, click the Number of Rows tab. For more information, see "Test Data Generation Number of Rows tab" on page 496.

5. [optional] To modify script formatting options, click the Format tab. For more information, see "Test Data Generation Format tab" on page 497.

6. [optional] To control which tables will have test data generated, click the Selection tab. For more information, see "Test Data Generation Selection tab" on page 499.

7. Click OK to start the generation.

   If you are generating a test data script : then a Result dialog box asks you if you want to Edit or Close the newly generated file.

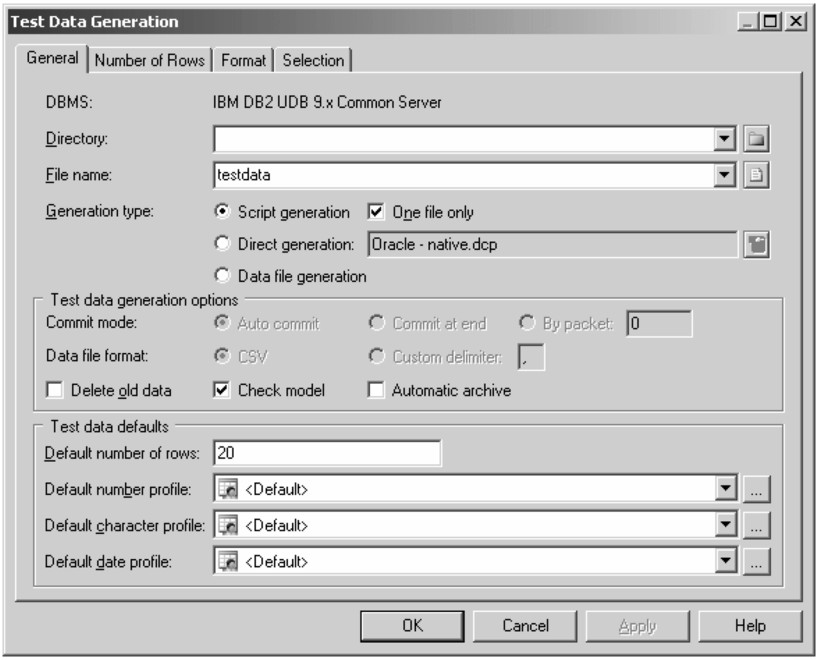
   If you are generating test data to a live database connection , then a Connect to a Data Source dialog box opens.

   Select a data source, and then click Connect.

   A message in the Output window indicates that the test data generation is completed.

Test Data Generation General tab

The General tab allows you to set general generation options.



Test Data Generation

General | Number of Rows | Format | Selection

DBMS: IBM DB2 UDB 9.x Common Server

Directory:

File name: testdata

Generation type: ● Script generation ☑ One file only

○ Direct generation: Oracle - native.dcp

○ Data file generation

Test data generation options

Commit mode: ● Auto commit ○ Commit at end ○ By packet: 0

Data file format: ● CSV ○ Custom delimiter: ,

□ Delete old data ☑ Check model □ Automatic archive

Test data defaults

Default number of rows: 20

Default number profile: <Default>

Default character profile: <Default>

Default date profile: <Default>

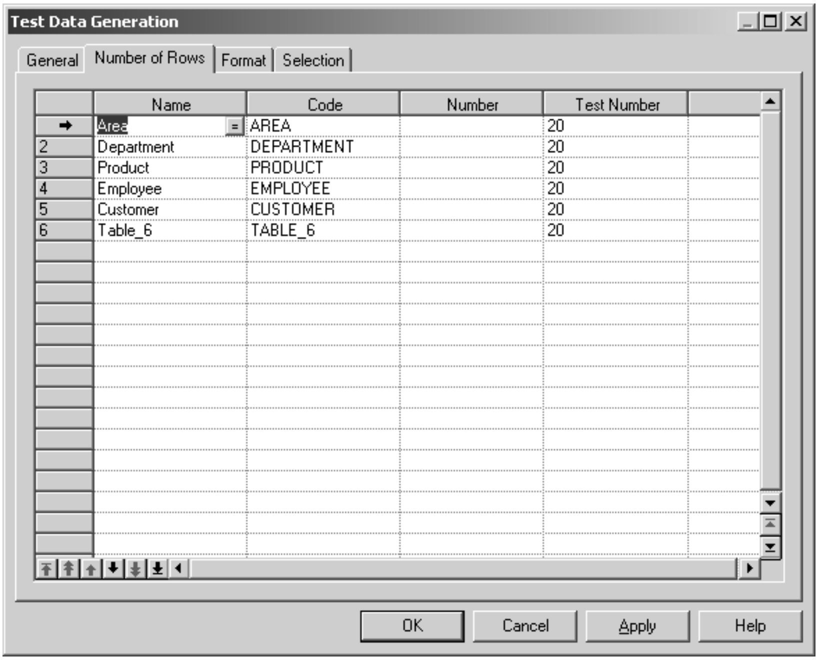
The following options are available on this tab:

Option	Result of selection
Directory	Specifies the directory in which the file will be saved.
File name	Specifies the name of the file. Select the One file only checkbox to specify that only a single file can be generated.
Generation type	Specifies how the test data will be generated. The following settings are available: ♦ Script generation - in DBMS-specific syntax ♦ Direct generation – to a live database connection ♦ Data file – as a set of values in a file

Option	Result of selection
Commit mode	Specifies when the data will be committed. The following settings are available: ♦ Auto - automatically during script generation ♦ At end - at end of script generation ♦ By packet - at defined intervals during script generation
Data file format	For use with the data file option. The following settings are available: ♦ CSV – comma-separated values ♦ Custom delimiter – specify a custom delimiter
Delete old data	Deletes existing data before generating new data..
Check model	Checks the PDM before generating the test database or script, and stops generation if an error is found.
Automatic archive	Creates an archive of any previous test data.
Default number of rows	Specifies the default number of rows for table
Default number profile	Specifies the default number profile for table
Default character profile	Specifies the default character profile for table
Default date profile	Specifies the default date profile for table

Test Data Generation Number of Rows tab

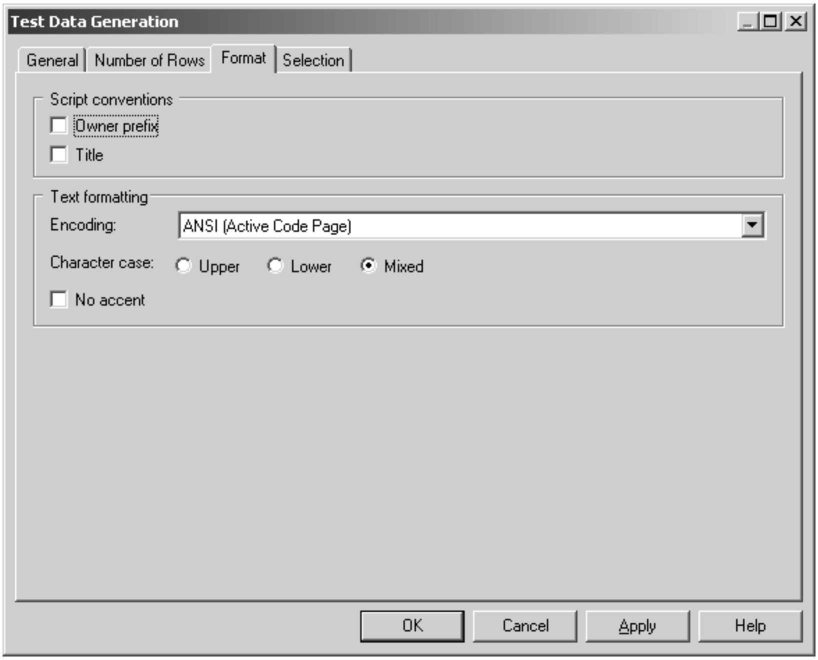
The Number of Rows tab allows you to specify the number of rows of test data to generate. Enter the required number in the Test Number column.



	Name	Code	Number	Test Number
➔	Area	AREA		20
2	Department	DEPARTMENT		20
3	Product	PRODUCT		20
4	Employee	EMPLOYEE		20
5	Customer	CUSTOMER		20
6	Table_6	TABLE_6		20

Test Data Generation Format tab

The Format tab allows you to set script format options.



Script conventions

□ Owner prefix

□ Title

Text formatting

Encoding: ANSI (Active Code Page)

Character case: ○ Upper ○ Lower ● Mixed

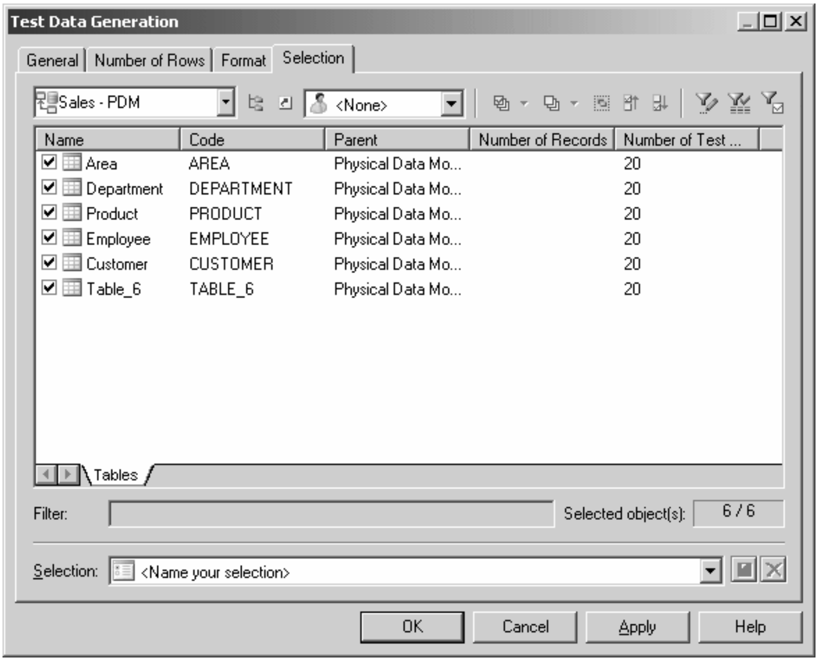
□ No accent

The following options are available on this tab:

Option	Result of selection
Owner prefix	Specifies that an owner prefix is added.
Titles	Specifies that each section of the script includes commentary in the form of titles.
Encoding	Specifies the encoding format to use for test data generation. You should select the encoding format that supports the language used in your model and the database encoding format.
Character case	Specifies the character case to use. The following settings are available: ♦ Upper - all uppercase characters ♦ Lower - all lowercase characters ♦ Mixed - both uppercase and lowercase characters
No accent	Non-accented characters replace accented characters in script.

Test Data Generation Selection tab

The Selection tab allows you to specify tables for which test data will be generated.



Sales - PDM

<None>

Name	Code	Parent	Number of Records	Number of Test ...
☑ Area	AREA	Physical Data Mo...		20
☑ Department	DEPARTMENT	Physical Data Mo...		20
☑ Product	PRODUCT	Physical Data Mo...		20
☑ Employee	EMPLOYEE	Physical Data Mo...		20
☑ Customer	CUSTOMER	Physical Data Mo...		20
☑ Table_6	TABLE_6	Physical Data Mo...		20

Filter:

Selected object(s): 6 / 6

Selection: <Name your selection>

☞ For more information on selection windows, see “Adding an item from a selection list” section in the Objects chapter of the *Core Features Guide* .

Estimating Database Size

You can estimate the size of a database for all the tables in the model or for only selected tables throughout the model.

Basis of calculation

You obtain an estimate based on the following elements:

- Estimated number of records in tables
- Columns in each table
- Indexes in the model
- Tablespaces in the model
- Database storage options

Estimation depends on parameters like DBMS type, physical options, column data types, indexes and keys. The estimation formula used in this feature proceeds from the DBMS documentation.

Records in tables
You indicate a number of records in each table in your model.

Columns

The estimated database size for a column is based on the following:

- Size of fixed length data types
- Average size of variable length data types

Indexes

The estimate of the database size includes all indexes including primary key indexes, foreign key indexes, alternate key indexes and database-specific indexes such as IQ join indexes.

Automatic index constraints
The automatic indexing of keys is DBMS specific. If the target database supports the automatic indexing of keys, the resulting database size estimation includes these indexes.

Tablespace size

The size of a tablespace associated with a table is estimated by default, and is displayed as a total of the following:

- All tables in the tablespace
- All indexes in the tablespace

☞ For more information on how to associate a tablespace with an index or table, see section Creating an Index in chapter Building Physical Diagrams.

Storage options

Data storage options are DBMS specific. You define data storage options supported by the target database. These are included in the estimate of database size.

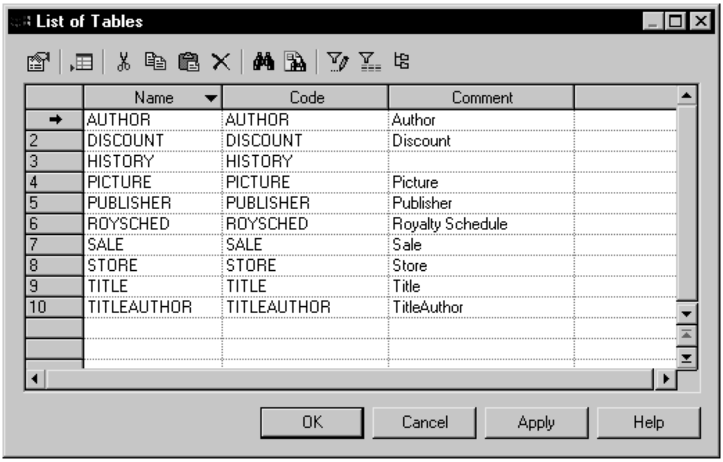
Indicating the number of records in a table

Before you estimate database size, you indicate the number of records in each table that you generate.

❖ To indicate the number of records in a table

1. Select Model ➤ Tables.

   The List of Tables is displayed.

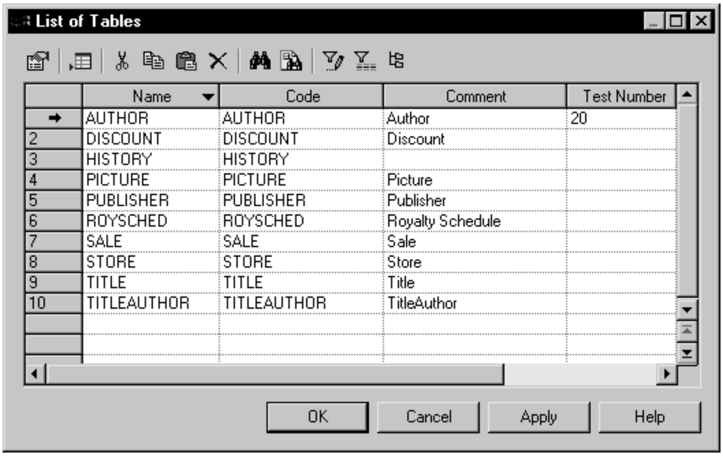


	Name	Code	Comment	
➡	AUTHOR	AUTHOR	Author	
2	DISCOUNT	DISCOUNT	Discount	
3	HISTORY	HISTORY		
4	PICTURE	PICTURE	Picture	
5	PUBLISHER	PUBLISHER	Publisher	
6	ROYSCHED	ROYSCHED	Royalty Schedule	
7	SALE	SALE	Sale	
8	STORE	STORE	Store	
9	TITLE	TITLE	Title	
10	TITLEAUTHOR	TITLEAUTHOR	TitleAuthor	

2. Display the Test Number column.

> **Display the column you need**
> If you do not see the column you need, display it with the Customize Columns and Filter tool. For details, see “Customizing object list columns and filtering lists” section in the Objets chapter of the *Core Features Guide* .

3. Type a value in the Test Number column for the table that requires an indication of the number of records.



	Name	Code	Comment	Test Number
➡	AUTHOR	AUTHOR	Author	20
2	DISCOUNT	DISCOUNT	Discount	
3	HISTORY	HISTORY		
4	PICTURE	PICTURE	Picture	
5	PUBLISHER	PUBLISHER	Publisher	
6	ROYSCHED	ROYSCHED	Royalty Schedule	
7	SALE	SALE	Sale	
8	STORE	STORE	Store	
9	TITLE	TITLE	Title	
10	TITLEAUTHOR	TITLEAUTHOR	TitleAuthor	

4. Click OK.

Indicating average data type length

You can indicate the estimated average data type length for variable length data types. This average length value is then used instead of the maximum data type length when you estimate database size.

Average size must be carefully selected
The average length value is particularly important for strings or long binary data types. A Binary Long OBject (BLOB) such as a picture can represent the largest portion of the space actually taken by a table.

❖ To indicate average data type length

1. Select Model ➤ Tables.

   The List of Tables is displayed.

2. Double-click a table name.

   The Table Properties sheet opens to the General tab.

3. Click the Columns tab.

   The Columns tab is displayed.

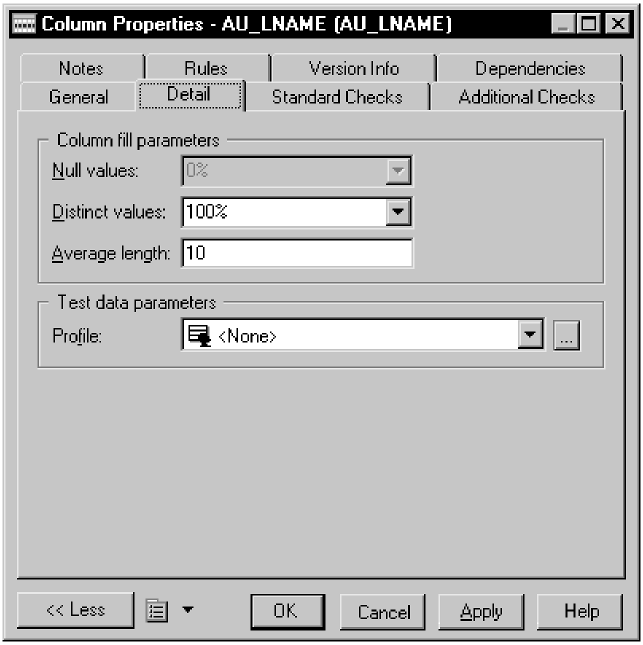
4. Double-click a column that uses a data type with variable length.

   The Column Properties sheet opens to the General tab.

5. Click the Detail tab.

   The Detail tab is displayed.

6. Type a value in the Average length box.



Column Properties - AU_LNAME (AU_LNAME)

Column fill parameters

Null values: 0%

Distinct values: 100%

Average length: 10

Test data parameters

Profile: <None>

7. Click OK in each of the dialog boxes.

Estimating the database size of a model

You can estimate the database size of all the tables in the model.

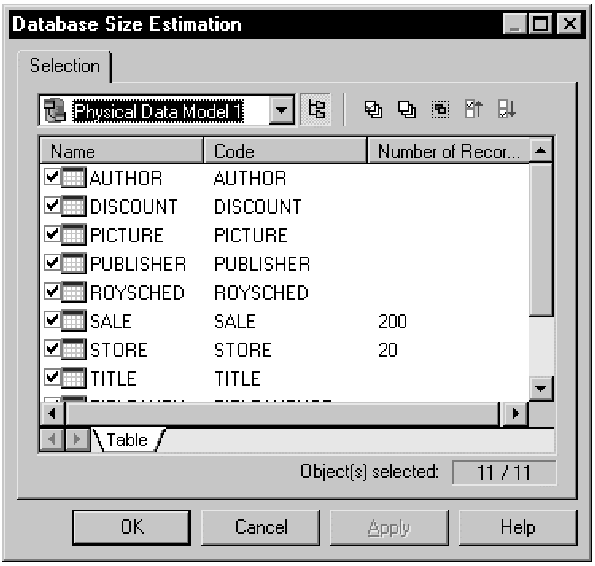
❖ To estimate the database size of all the tables in the model

1. Select Database ➤ Estimate Database Size.

   The Database Size Estimation dialog box is displayed.

2. Click the Include Sub-Packages button.

3. Select all the tables from the list.



Database Size Estimation

Selection

Physical Data Model 1

Name	Code	Number of Recor...
☑ AUTHOR	AUTHOR	
☑ DISCOUNT	DISCOUNT	
☑ PICTURE	PICTURE	
☑ PUBLISHER	PUBLISHER	
☑ ROYSCHED	ROYSCHED	
☑ SALE	SALE	200
☑ STORE	STORE	20
☑ TITLE	TITLE	

Table

Object(s) selected: 11 / 11

4. Click OK.

   The output list displays the estimated database size.

Estimating the database size of selected tables

You can estimate the database size of selected tables in the model.

❖ To estimate the database size of selected tables in the model

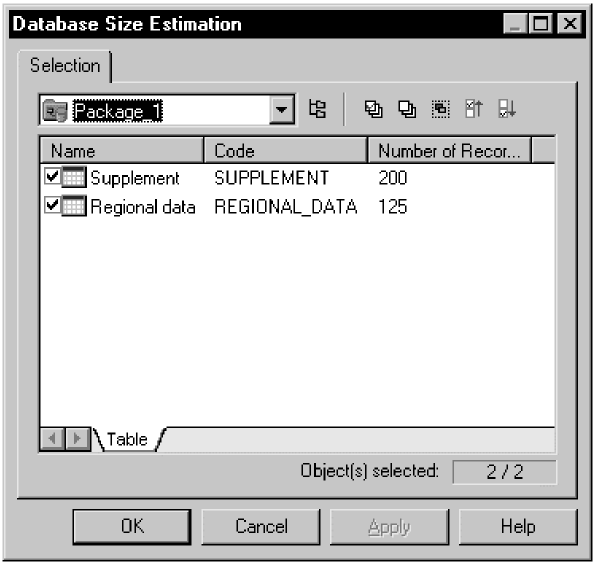
1. Select Database ➤ Estimate Database Size.

   The Database Size Estimation dialog box is displayed.

2. Select a package from the list.

   The list of tables in the currently selected package is displayed.

3. Select one or more tables from the list.



Name	Code	Number of Recor...
☑ Supplement	SUPPLEMENT	200
☑ Regional data	REGIONAL_DATA	125

Object(s) selected: 2 / 2

4. If required, repeat steps 2 and 3 until all the tables of interest have been selected throughout the model.

5. Click OK.

   A confirmation box is displayed asking if you want to keep all previously selected objects in the database size estimate.

6. Click OK.

   The output list displays the estimated database size.

Modifying a Database

You can modify an existing database schema by synchronizing it with your model. The existing schema can be in the form of:

- an archive model
- a live database connection
- a script file
- a model from the repository

The PDM (source model) and the existing database schema (target model) are merged using a database synchronization window, which allows you to choose which objects are added, deleted, or updated in the target.

❖ To modify a database

1. Select Database ➤ Apply Model Changes to Database to open the Apply Model Changes to Database dialog box.

2. Type a destination directory and filename for the script file in the Directory and File Name boxes.

3. Specify the type of generation to perform. You can choose between a script and a live database connection.

4. Specify how PowerDesigner will obtain the database schema to modify. You can choose between:
   - Using an archive model – Click the button to the right to browse to the archived model.
   - Using a data source – Click the button to the right to connect to your data source.
   - Using a script file – Select a script from the list or click the button to the right to browse to the script.
   - Using a model from repository – Click the Change Model Version tool to the right to browse to a version of the currently selected model.

5. If you want to retain your existing data, select the Backup Tables option. If this option is not selected, then all existing data will be erased. For details of this and other options on this tab, see "Apply Model Changes to Database dialog General tab" on page 508.

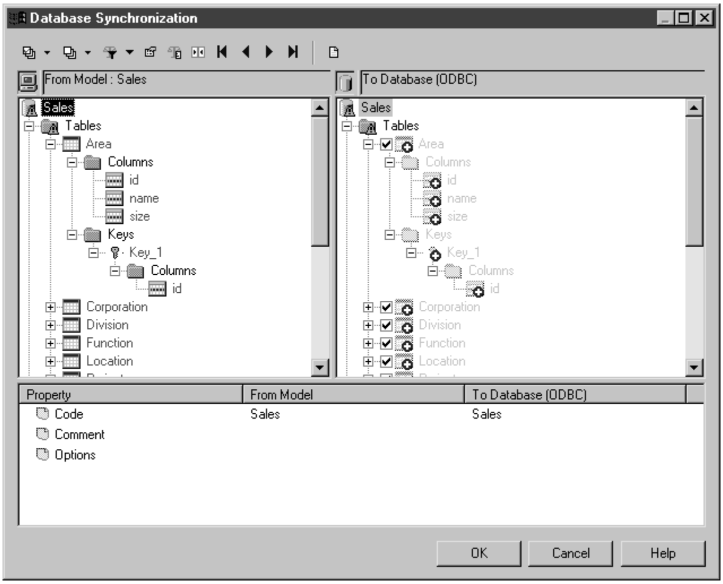
6. [optional] If you want to change the default generation options, then click the Options tab. For more information about these options, see "Apply Model Changes to Database dialog Options tab" on page 510.

7. [optional] If you want to change the format of your script, then click the Format tab. This tab has the same functionality as in the Database Generation window (see “Database Generation dialog Format tab” on page 456).

8. [optional] If you want to control which database objects will be modified, then click the Selection tab. This tab has the same functionality as in the Database Generation window (see “Database Generation dialog Selection tab” on page 458)

9. Click OK. If you are using a live database connection, then the Reverse Engineering window will open, allowing you to select or clear check boxes in the target model for objects that you want to include or remove from the source model. Make your selections and then click OK to continue.

10. The Database Synchronization window will open. Select or clear check boxes in the target model for objects that you want to include or remove from the model.



Property	From Model	To Database (ODBC)
Code	Sales	Sales
Comment		
Options		

☞ For more information on comparing and merging models, see the Comparing and Merging Models chapter in the *Core Features Guide* .

11. Click OK.

   ♦ If you are generating a script, at the end of generation a result box opens listing the file path of the generated file. To open the script in a text editor, click the file in the result box and click the Edit button. To close the Result box, click the Close button.

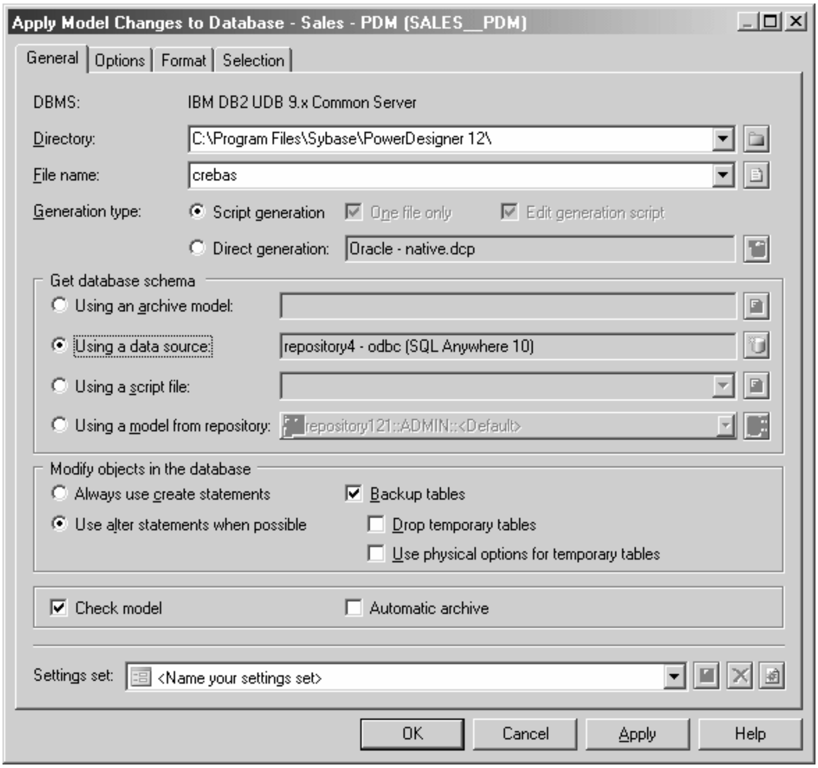
- ♦ If you are generating a database directly, a Data Source connection box is displayed. Type your connection details and click the Connect button. A message box shows the progress of the generation process. At the end of generation click OK to close the box.

Modify Database options

The following sections give detailed information about the various options that you can use to control database modification.

Apply Model Changes to Database dialog General tab

This tab controls the main options for database modification.



Apply Model Changes to Database - Sales - PDM (SALES__PDM)

DBMS: IBM DB2 UDB 9.x Common Server

Directory: C:\Program Files\Sybase\PowerDesigner 12\

File name: crebas

Generation type: ● Script generation ☑ One file only ☑ Edit generation script

○ Direct generation: Oracle - native.dcp

Get database schema

○ Using an archive model:

● Using a data source: repository4 - odbc (SQL Anywhere 10)

○ Using a script file:

○ Using a model from repository: repository121::ADMIN::<Default>

Modify objects in the database

○ Always use create statements ☑ Backup tables

● Use alter statements when possible □ Drop temporary tables

□ Use physical options for temporary tables

☑ Check model □ Automatic archive

Settings set: <Name your settings set>

You can set the following options:

Option	Description
Directory	[required] Specifies the destination directory for the script file.

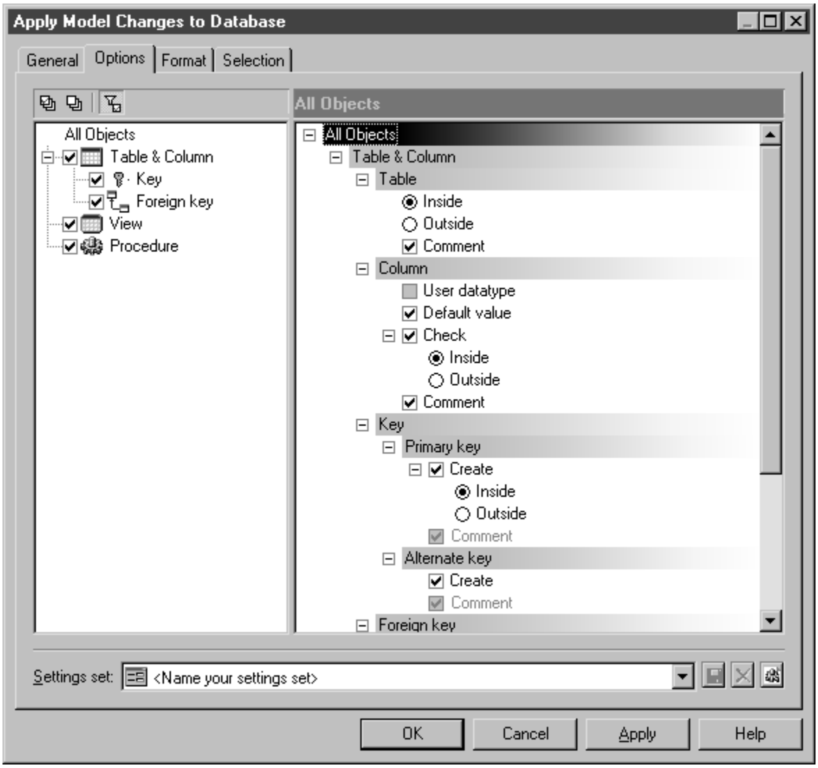
Option	Description
File name	[required] Specifies the destination filename for the script file.
One file only	Specifies that the generation script is created as a single file. By default, a separate script file is created for each table.
Generation Type	Specifies the type of generation to perform. You can choose between: ♦ Script generation - generate a script to be executed on a DBMS at a later time ♦ Direct generation – generate a script and execute it on a live database connection
Edit generation script	[available only when direct generation is selected] Opens the generation script in a text editor for review or editing before execution on a live database connection.
Obtains database schema	Select the kind of schema that the model will modify. You can choose between: ♦ Using an archive model - Modified PDM is merged with an archived PDM. ♦ Using a data source - Modified PDM is merged with a reverse engineered database schema for a live database connection. ♦ Using a script file - Modified PDM script file is merged with an existing database script file. ♦ Using a model from repository - Modified PDM is merged with a selected version of a PDM consolidated in the repository.
Backup tables	Specifies that any existing table will be copied to temporary backup tables during the modification, and then restored to the updated tables. If this option is not selected, then all existing data will be erased.
Always use create statements/ Use alter statements when possible	Select a radio button to specify whether create statements should always be used to modify database tables, or whether alter statements should be used where possible

Option	Description
Drop temporary tables	[available only when Backup Tables is selected] Specifies that the temporary backup tables are removed after script execution.
Use physical options for temporary tables	[available only when Backup Tables is selected] Specifies that the temporary backup tables are generated with their physical options.
Check model	Specifies that a model check is performed before script generation.
Automatic archive	Creates an archive version of the PDM after generation.

You can load option settings previously used for database generation via the Settings set bar at the bottom of the tab. For more information, see "Quick launch selection and settings sets" on page 441.

Apply Model Changes to Database dialog Options tab

This tab controls certain script options.



- All Objects
  - ☑ Table & Column
    - ☑ Key
    - ☑ Foreign key
  - ☑ View
  - ☑ Procedure

- All Objects
  - Table & Column
    - Table
      - ● Inside
      - ○ Outside
      - ☑ Comment
    - Column
      - □ User datatype
      - ☑ Default value
      - ☑ Check
        - ● Inside
        - ○ Outside
      - ☑ Comment
    - Key
      - Primary key
        - ☑ Create
          - ● Inside
          - ○ Outside
        - ☑ Comment
      - Alternate key
        - ☑ Create
        - ☑ Comment
    - Foreign key

Settings set: <Name your settings set>

These options are dependent on the selected DBMS and certain of them may be unavailable.

Option	Result of selection
Inside/ Outside	Specifies whether and where various constraints will be generated. Note that if alter statements are used to modify tables, then constraints may be generated outside the table even if the inside radio button is selected here.
Index Filter	Specifies which kinds of indexes to generate.
Comment	Specifies whether comments are generated.

You can load option settings previously used for database generation via the Settings set bar at the bottom of the tab. For more information, see "Quick launch selection and settings sets" on page 441.

Apply Model Changes to Database dialog Format tab

This tab controls the format of your modification script. It has the same functionality as the equivalent tab in Database Generation (see "Database Generation dialog Format tab" on page 456).

Apply Model Changes to Database dialog Selection tab

The Selection tab allows you to specify individual objects to generate. It has the same functionality as the equivalent tab in Database Generation (see "Database Generation dialog Selection tab" on page 458).

Accessing a Database

PowerDesigner allows you to display data from the database that corresponds to your model, and to send SQL queries to a connected data source.

Displaying data from a database

If the database corresponding to the PDM already exists, you can display the data that corresponds to a table, view, or reference in the PDM.

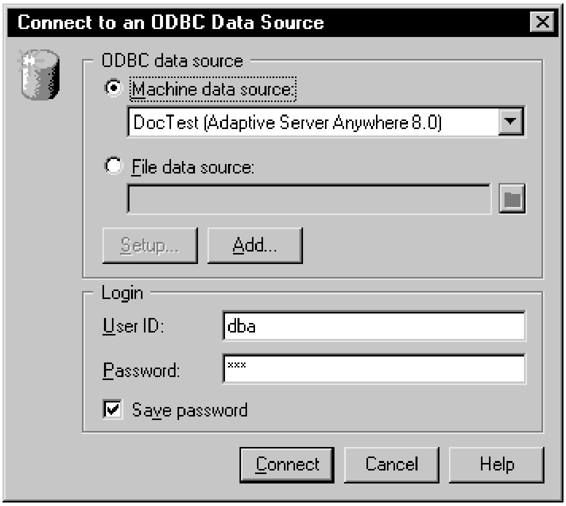
❖ To display data from a connected database

1. Right-click a table, view, or reference.

   A contextual menu is displayed.

2. Select View Data from the contextual menu.

   A dialog box asks you to identify a data source and connection parameters.



Connect to an ODBC Data Source

ODBC data source

● Machine data source:

DocTest (Adaptive Server Anywhere 8.0)

○ File data source:

Login

User ID: dba

Password: ***

☑ Save password

3. Select the Machine data source radio button.

   Select a data source from the list.

   or

   Select the File data source radio button.

   Browse to the directory containing the .DSN file.

   Select the .DSN file.

4. Type your user ID and password.

5. Click Connect and, if prompted by your data source, type additional connection parameters.

   A Query Results windows list all the database records corresponding to the selected table, view, or reference.

6. Click the Close button.

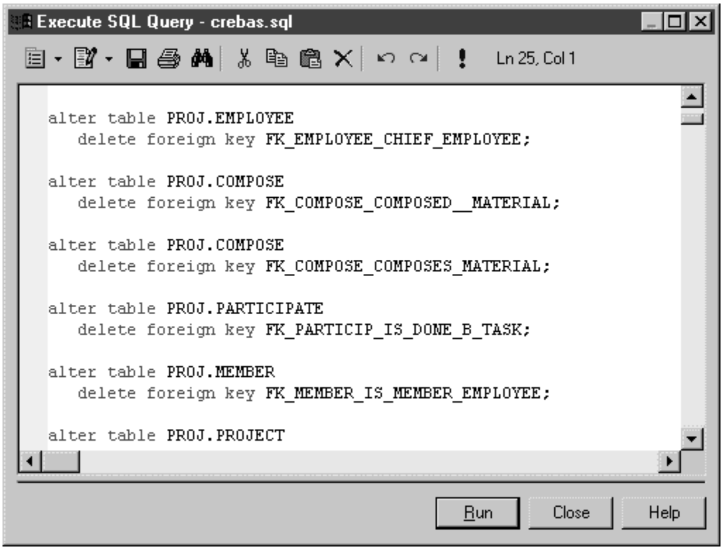
Executing SQL queries

You can send SQL queries to a database and display the results.

❖ To execute a SQL query

1. Select Database ➤ Execute SQL. If you are not already connected to a database, the Connect to Data Source window will open. Choose your connection profile and click Connect to proceed to the Execute SQL Query dialog.

2. Type one or more SQL statements in the window, and click Run to apply them to the database.



```
alter table PROJ.EMPLOYEE
   delete foreign key FK_EMPLOYEE_CHIEF_EMPLOYEE;

alter table PROJ.COMPOSE
   delete foreign key FK_COMPOSE_COMPOSED__MATERIAL;

alter table PROJ.COMPOSE
   delete foreign key FK_COMPOSE_COMPOSES_MATERIAL;

alter table PROJ.PARTICIPATE
   delete foreign key FK_PARTICIP_IS_DONE_B_TASK;

alter table PROJ.MEMBER
   delete foreign key FK_MEMBER_IS_MEMBER_EMPLOYEE;

alter table PROJ.PROJECT
```

   The query results are displayed in the Results window.

CHAPTER 10

Reverse Engineering a Database into a PDM

About this chapter

This chapter describes how to reverse engineer database objects into a PDM.

Contents

Getting Started with Reverse Engineering

Reverse engineering is the process of generating a PDM (or certain PDM objects) from an existing database schema. You can reverse engineer an existing database schema into a new PDM or into an existing PDM.

There are two ways to reverse engineer a PDM from a database schema:

Generate using	Description
Script file	The script will normally be the script used to generate the database but can also include other scripts. If you use more than one script files, make sure that the order of the files respects dependencies among objects (for example, trigger creation scripts must come after table creation scripts; and grant permission scripts must come after both table and user creation scripts. For more information, see the "Reverse Engineering from Scripts" on page 517 section.
Data source	You reverse engineer the schema for an existing database, specifying an data source, and connection information.You can select to use administrator permissions in order to be able to select the system tables that are reserved to a database admin. For more information, see the "Reverse Engineering from a Live Database" on page 520 section.

Set options appropriately

When you reverse engineer a database, whether from a script or a data source, make sure that you set the rebuild options appropriately. Click the Options tab and select or clear the checkboxes to rebuild references and/or primary keys according to your needs. By default, no rebuild options are selected.

Reverse Engineering from Scripts

PowerDesigner can reverse engineer a PDM for one or more SQL script files.

Reversing from SQL files in Eclipse
When working with the PowerDesigner Eclipse plug-in you can, in addition to the procedure below, select any SQL file in the Navigator, right-click it and select Reverse Engineer from SQL File. You will be given the option to reverse into an existing or new PDM.

❖ To reverse engineer objects one or more script files

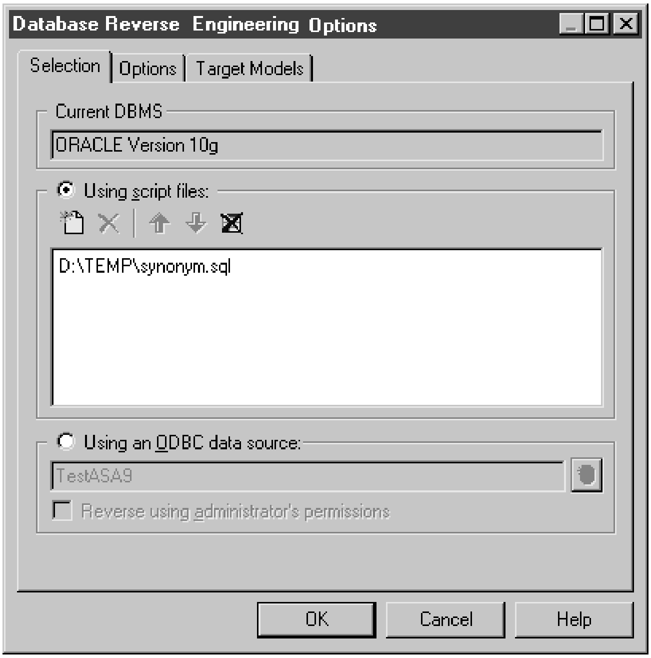
1. To reverse engineer a script into an existing PDM, select Database ➤ Update Model from Database.

   or

   To reverse engineer a script and create a new PDM, select File ➤ Reverse Engineer ➤ Database to open the New Physical Data Model dialog box. Specify a model name, choose a DBMS from the list, and then click OK.

2. When the Database Reverse Engineering Options dialog box opens, click the Using script files radio button.

   Always place trigger script files after table script files
   You can add as many script files as necessary to the list. The reverse engineering process handles files sequentially. Trigger scripts must always be executed after table scripts. This is the only constraint for ordering your files in the list, but it is essential for a successful reverse engineering of triggers. Use the Move tools to position the files correctly in the list.



3. Click the Options tab to specify any reverse engineering options. For more details, see "Reverse engineering Options tab" on page 524.

4. Click the Target Models tab to specify any external shortcuts. For more details, see "Reverse engineering Target Models tab" on page 527.

5. Click OK to begin the process of reverse engineering. When the process is complete, a confirmation message is given in the Output window. If you are reverse engineering to an existing PDM, then the Merge Models dialog box opens to help you merge the new objects into your PDM.

☞ For more information on comparing and merging two models, see the Comparing and Merging Models chapter in the *Core Features Guide* .

Script file tools

The following tools are available to help you to select script files:

Tool	Description
⊞	Add Files – Opens a dialog box to allow you to browse for scripts files. You can add as many files as necessary.
⇧	Move Up – Moves the selected file(s) up one row. This tool is grayed if the selected file(s) are at the top of the list.
⇩	Move Down - Moves the selected file(s) down one row. This tool is grayed if the selected file(s) are at the bottom of the list.
☒	Clear All - Deletes all files from the list.

Reverse Engineering from a Live Database

PowerDesigner can reverse engineer a PDM from a live database connection.

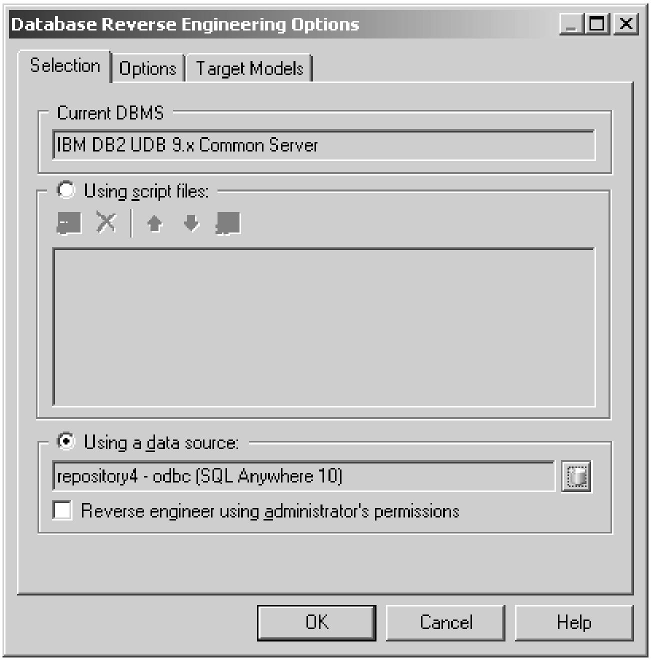
❖ To reverse engineer database objects from a live database connection

1. To reverse engineer from a live database connection into an existing PDM, select Database ➤ Update Model from Database to open the Database Reverse Engineering Options dialog box.

   or

   To reverse engineer from a live database connection and create a new PDM, select File ➤ Reverse Engineer ➤ Database to open the New Physical Data Model dialog box. Specify a model name, choose a DBMS from the list, and then click OK.

2. When the Database Reverse Engineering Options dialog box opens, click the Using a data source radio button.



Current DBMS: IBM DB2 UDB 9.x Common Server

○ Using script files:

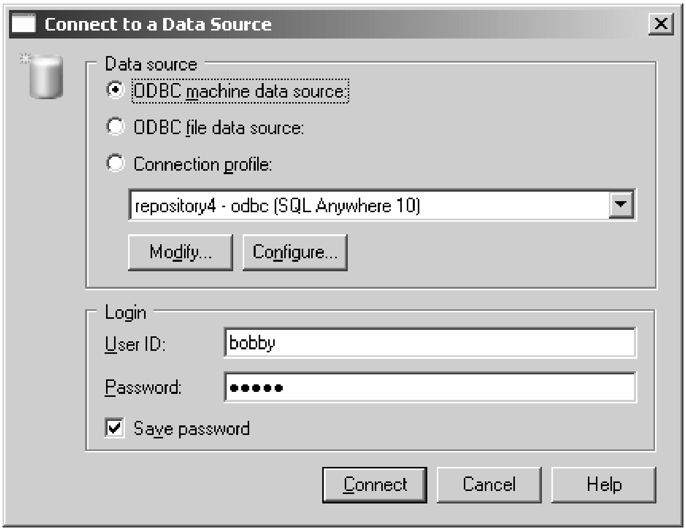
● Using a data source: repository4 - odbc (SQL Anywhere 10)

□ Reverse engineer using administrator's permissions

Data source

A data source might be predefined, or you can type the name of an existing data source. In both cases, when you click OK, a database connection dialog box opens, if you need to specify additional connection parameters. Click Connect and the Database Reverse Engineering dialog box is displayed. (Go to step 9)

3. Click the **Connect to a Data Source** tool to open the Connect to an ODBC Data Source dialog box.



Data source

● ODBC machine data source:

○ ODBC file data source:

○ Connection profile:

repository4 - odbc (SQL Anywhere 10)

Login

User ID: bobby

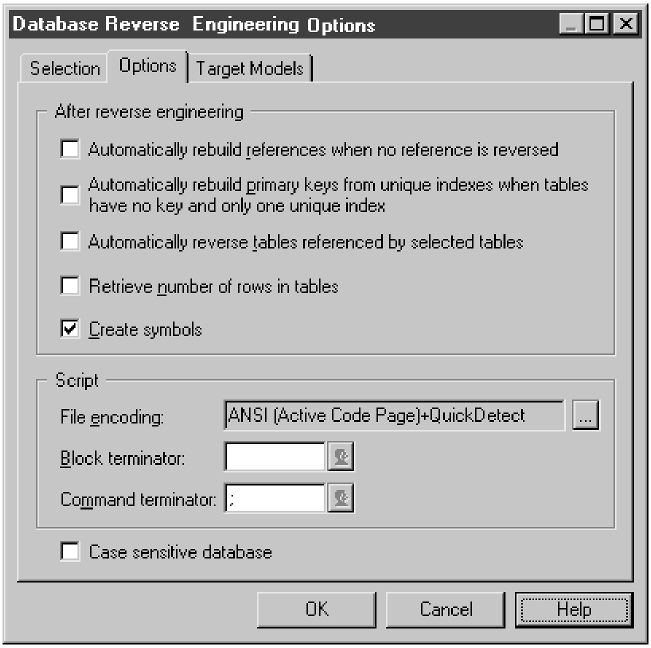
Password: •••••

☑ Save password

4. Select the appropriate source, type a user ID and a password, and then click Connect to return to the Database Reverse Engineering Options dialog box.

5. If you want to select tables reserved to the database administrator, then you must select the **Reverse using administrator’s permissions** check box.

6. Click the Options tab to specify any reverse engineering options. For more details, see “Reverse engineering Options tab” on page 524.



Database Reverse Engineering Options

After reverse engineering

☐ Automatically rebuild references when no reference is reversed

☐ Automatically rebuild primary keys from unique indexes when tables have no key and only one unique index

☐ Automatically reverse tables referenced by selected tables

☐ Retrieve number of rows in tables

☑ Create symbols

Script

File encoding: ANSI (Active Code Page)+QuickDetect

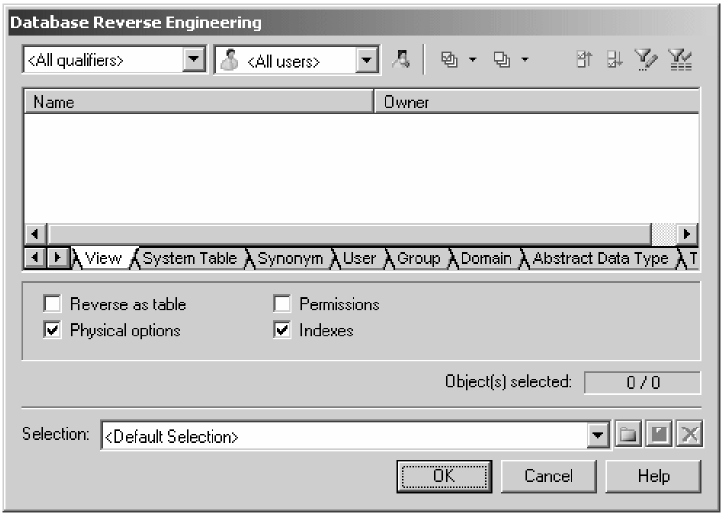
Block terminator:

Command terminator: ;

☐ Case sensitive database

7. Click the Target Models tab to specify any external shortcuts. For more details, see "Reverse engineering Target Models tab" on page 527.

8. Click OK to open the ODBC Reverse Engineering dialog box. This box allows you to specify a selection of objects to reverse engineer. Only tables and triggers are selected by default.



Database Reverse Engineering

<All qualifiers> <All users>

Name	Owner

☐ Reverse as table ☐ Permissions

☑ Physical options ☑ Indexes

Object(s) selected: 0 / 0

Selection: <Default Selection>

For more information about selecting objects, see "Database Reverse

Engineering Selection window” on page 528.

9. Click OK to begin the process of reverse engineering. When the process is complete, a confirmation message is given in the Output window. If you are reverse engineering to an existing PDM, then the Merge Models dialog box opens to help you merge the new objects into your PDM.

☞ For more information on comparing and merging two models, see the Comparing and Merging Models chapter in the *Core Features Guide* .

Reverse Engineering Options

The following sections give detailed information about the various options that you can use to control reverse engineering.

Reverse engineering Options tab

When you reverse engineer a database schema using script files or a data source, you can define rebuild options after reverse engineering.

The rebuild options automatically perform the following tasks after reverse engineering:

Reverse options	Description
Automatically rebuild references when no reference is reversed	Rebuilds references when no references are reverse engineered. The rebuild references feature starts by detecting columns with identical name and data type in different tables. A reference is created between each column belonging to a primary key and a column, with identical name and data type, that does not belong to a primary or a foreign key in another table.
Automatically rebuild primary keys from unique indexes when tables have no key and only one unique index	Rebuilds primary keys using unique indexes when tables have no key and only one unique index.
Automatically reverse tables referenced by selected tables	Reverse engineers the parents of the selected child tables in order to complement the definition of these child tables.

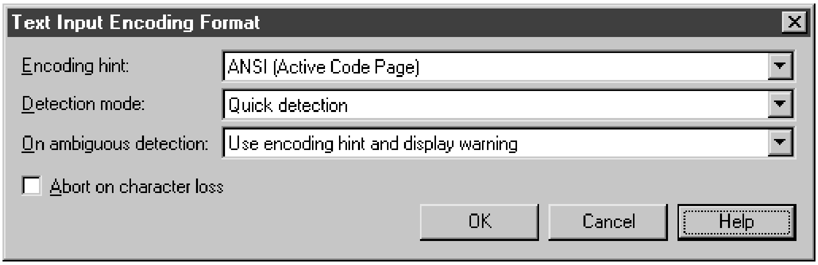
Reverse options	Description
Create symbols	Creates a symbol for each reversed object in the diagram. Otherwise, reversed objects are visible only in the browser. The layout of the symbols in the diagram will be automatically arranged. In cases where there are a large number of objects with complex interactions, the auto-layout feature is likely to create synonyms of objects to improve the diagram readability. For example, if a table has a large number of references, the auto-layout feature will create a synonym of this table in another location of the diagram in order to improve the diagram presentation.
File encoding	Specifies the default file encoding of the files to reverse engineer. Click the ellipsis to the right of the option to change the encoding (see "Reverse engineering encoding format" on page 525).
Block terminator	Specifies the end of block character for the reversed script. By default, displays the value defined in the DBMS, under Script\SQL\Syntax. You can modify this value, in which case it will be saved in the Registry for reuse in other models. You can restore the DBMS value using the Restore from DBMS tool.
Command terminator	Specifies the end of command character for the reversed script. By default, displays the value defined in the DBMS, under Script\SQL\Syntax. You can modify this value, in which case it will be saved in the Registry for reuse in other models. You can restore the DBMS value using the Restore from DBMS tool.
Case sensitive database	Specifies that the database is case sensitive and enables the case sensitive option in the model.

☞ For more information on indexes, rebuilding references and rebuilding primary keys, see chapter Building Physical Diagrams.

Reverse engineering encoding format

If the code you want to reverse engineer is written with Unicode or MBCS (Multibyte character set), you should use the encoding parameters provided to you in the File Encoding box.

If you want to change these parameters because you know which encoding is used within the sources, you can select the appropriate encoding parameter by clicking the Ellipsis button beside the File Encoding box. This opens the Text Input Encoding Format dialog box in which you can select the encoding format of your choice.


Text Input Encoding Format

Encoding hint: ANSI (Active Code Page)

Detection mode: Quick detection

On ambiguous detection: Use encoding hint and display warning

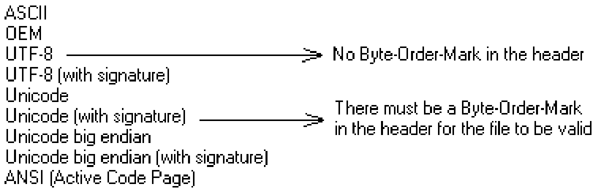
□ Abort on character loss

The Text Input Encoding Format dialog box includes the following options:

Option	Description
Encoding hint	Encoding format to be used as hint when reversing the file.
Detection mode	Indicates whether text encoding detection is to be attempted and specifies how much of each file should be analyzed. When enabled, PowerDesigner analyzes a portion of the text, and uses an heuristic based on illegal bytes sequences and/or the presence of encoding-specific tags in order to detect the appropriate encoding that should be used for reading the text. The following settings are available: ♦ No detection - for use when you know what the encoding format is ♦ Quick detection - analyzes a small part of the file. For use when you think that the encoding format will be easy to detect ♦ Full detection – analyzes the whole file. For use when you think that the number of characters that determine the encoding format is very small

Option	Description
On ambiguous detection	Specifies what action should be taken in case of ambiguity. The following settings are available: ♦ Use encoding hint and display warning - the encoding hint format is used and a warning message is displayed. ♦ Use encoding hint - the encoding hint format is used but no warning message is displayed. ♦ Use detected encoding - the encoding format detected by PowerDesigner is used
Abort on character loss	Allows you to stop reverse engineering if characters cannot be identified and are to be lost in current encoding

Here is an example on how to read encoding formats from the list:



ASCII
OEM
UTF-8
UTF-8 (with signature)
Unicode
Unicode (with signature)
Unicode big endian
Unicode big endian (with signature)
ANSI (Active Code Page)

No Byte-Order-Mark in the header
There must be a Byte-Order-Mark in the header for the file to be valid

Reverse engineering Target Models tab

External shortcuts depend on their corresponding target objects located in different models. When you need several models to design a single database, you can use shortcuts to share objects between models.

During reverse engineering, PowerDesigner allows you to create external shortcuts from target objects in target models. In the Database Reverse Engineering Options dialog box, the Target Models tab displays the list of detected target models containing target objects for shortcuts in the current model to reverse.

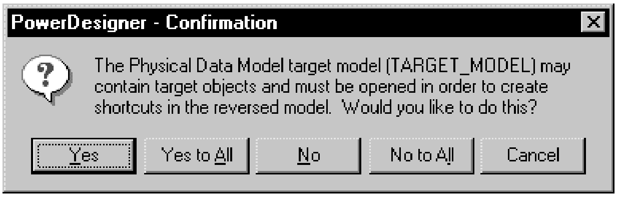
This tab is always visible in the Database Reverse Engineering Options dialog box, even if the model does not contain shortcuts.

This tab is empty when you reverse engineer into a new model. This is to let you add target models and create shortcuts instead of duplicating objects.

You can use the Target Models tab in the Database Reverse Engineering Options dialog box to manage target models using the following tools:

Tool	Tooltip	Description
	Change Target Model	Displays a standard Open dialog box to let you select another file as target model
	Open Model	Opens selected target model in current workspace
	Add Models	Opens a selection list with the models opened in the current workspace. This tool is particularly useful when you reverse engineer into a new model where the target models are not defined
✕	Delete	Deletes the target model and the shortcuts in the current model that reference the deleted target model

When you reverse engineer a model, any target models should be open in your workspace. If not, the following confirmation dialog box is displayed to let you open the target models:



PowerDesigner - Confirmation

The Physical Data Model target model (TARGET_MODEL) may contain target objects and must be opened in order to create shortcuts in the reversed model. Would you like to do this?

Reverse engineering from script

All the create statements in the script create objects, provided the script contains a full definition of the object.

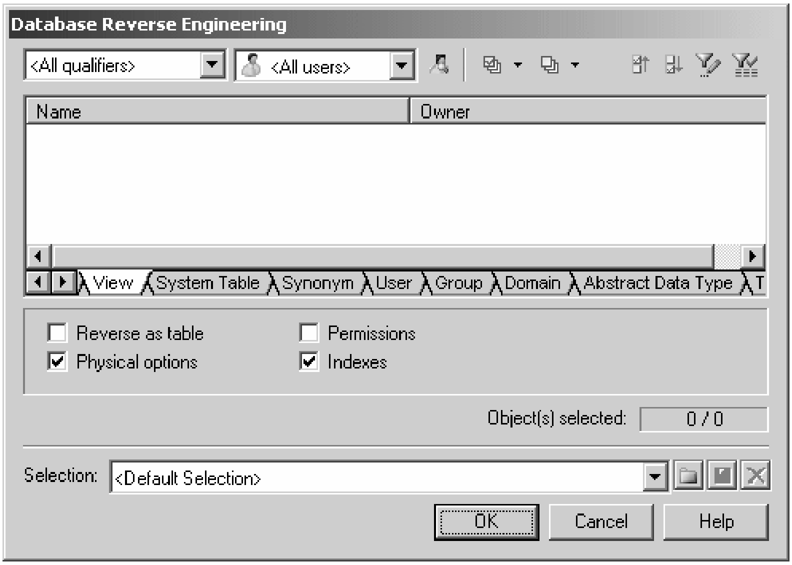
When the script only uses an object and does not define it, this object is sought among the target objects in the target models and an external shortcut is created in the reversed model.

Reverse engineering from data source

When you reverse engineer from a live database connection, external shortcuts are created for all selected objects already existing in another target model. These existing objects are deselected by default in the Selection tab of the Reverse Engineering dialog box, except the target objects corresponding to shortcuts already existing in the reversed model.

Database Reverse Engineering Selection window

When you reverse engineer a database from a live database connection, you can choose to generate a PDM for all, or just selected objects. You make this selection in the Database Reverse Engineering Selection window:



□ Reverse as table
☑ Physical options
□ Permissions
☑ Indexes

Object(s) selected: 0 / 0

Selection: <Default Selection>

The object types that you can reverse engineer are DBMS-dependent. Unavailable object types do not appear for selection.

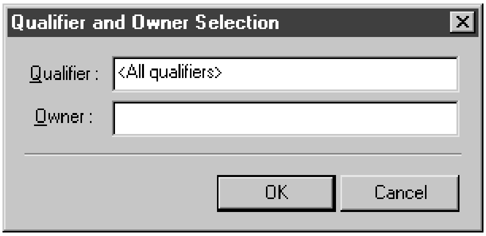
Filters

You can restrict database objects to reverse engineer by selecting an owner or a database qualifier in the top area of the window:

Filter	Description
Qualifier	A qualifier is a database, or a partition in a database, that contains one or more tables. When a qualifier is selected as a filter, it restricts the objects available for reverse engineering to the objects contained within the selected qualifier. For example, the DB2 DBMS authorizes the use of the qualifier field to select which databases are to be reverse engineered from a list.
Owner	Normally the creator of a database object is its owner. When an owner is selected as a filter; it restricts the objects available for reverse engineering to the objects owned by the selected owner.

You can filter on a qualifier and/or owner in either of the following ways:

- Select a qualifier in the Filter on object qualifier list and/or select a user ID in the Filter on object owner list.
- Click the Select Qualifier and Owner tool, and type a qualifier and/or owner in the dialog box. This method is recommended if the selected qualifier contains a large number of table owners, as opening the Owner list may take a very long time.



Note that only users that have creation rights are reverse engineered.

Selecting objects from multiple owners
To reverse engineer objects from multiple owners, you can select **All users** as a filter from the owner list. All the objects belonging to all owners appear in the list, and you can select the objects for reverse engineering regardless of their owner.

Objects and options

You access different object types to select by clicking on the appropriate sub-tabs.

Tables and triggers
When you select tables containing triggers from the Table tab, the corresponding triggers are automatically selected in the Trigger tab.

Certain object types have attributes, or options, that appear below the object lists. These options depend on the selected object type and on the current DBMS. Unavailable options appear grayed.

User-defined and abstract data types
You can reverse engineer user-defined and abstract data types. In the generated PDM, the names of these data types appear in the List of Abstract Data Types.

Selection list

You can save your selections for re-use by entering a selection name in the list at the bottom of the window and clicking the save tool to the right of the list. Selections are saved with a .sel file extension, and are added to the list for subsequent use. You can change the folder in which the files are saved by clicking the folder tool to the right of the list.

☞ For more information on selection windows, see “Adding an item from a selection list” section in the Objects chapter of the *Core Features Guide* .

Reverse engineering a Microsoft Access 97 database

PowerDesigner and MS Access 97 use .DAT files to exchange information. These files are reversed into the PDM. The access.mdb database uses or creates .DAT files to reverse Access databases.

You can define the database reverse parameters from the access.mdb database window.

❖ To reverse an MS Access database into a PowerDesigner PDM

1. Double-click ACCESS.MDB in the PowerDesigner \tools directory.
2. Select Reverse Engineer Access Database File to PowerDesigner Script.
3. Type the Access database name in the Select Database box.
4. Type the .DAT file to create in the PowerDesigner File.
5. Click the Create button.
6. Select DBMS ➤ Reverse Engineering Databases in PowerDesigner.
7. Select the newly generated script file to reverse.
8. Click OK.

Optimizing live database reverse engineering queries

Live database reverse engineering has been optimized in order to improve performance. All queries run according to an optimization process rule. This process uses the following registry keys:

- **RevOdbcMinCount** defines a number of selected objects for reverse engineering. The default number is 100
- **RevOdbcMinPerct** defines a percentage of selected objects for reverse engineering. The default percentage is 10

These keys do not exist by default, you have to create and edit them in the Registry under:

```
Current User \Software\Sybase\PowerDesigner <version>\
        FolderOptions\Physical Objects
```

During reverse engineering, PowerDesigner compares the total number of current objects for reverse engineering to the value of **RevOdbcMinCount**.

If the total number of listed items is lower than the value of RevOdbcMinCount A global reverse query is executed.

If the total number of listed items is higher than the value of RevOdbcMinCount The process uses key **RevOdbcMinPerct**.

- If the percentage of reversed items is lower than the percentage defined in RevOdbcMinPerct, then the same query is executed for each object.
- If the percentage of reversed items is higher than the percentage defined in RevOdbcMinPerct, then a global query is executed.

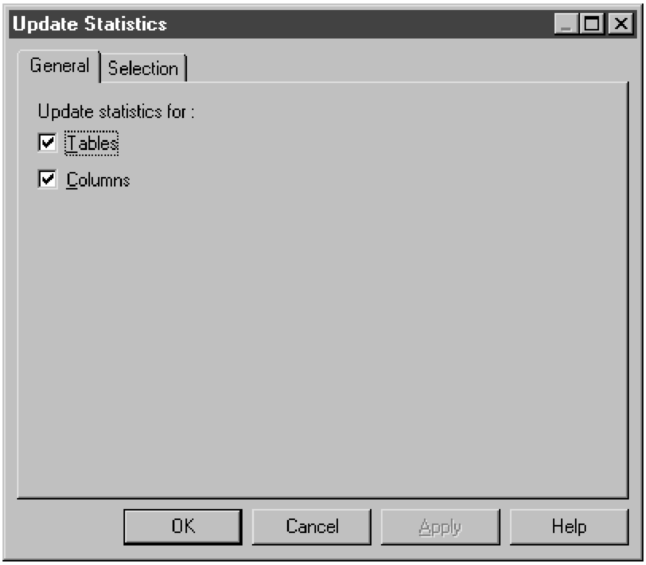
Reverse Engineering Database Statistics

You can reverse engineer statistics for an existing database, such as the number of distinct or null values in a column or the average length of a character field. These can provide helpful information when optimizing a design.

You can reverse engineer the statistics as part of the general reverse engineering process by selecting the Statistics checkbox in the Database Reverse Engineering window (see "Reverse Engineering from a Live Database" on page 520), or update them at any other time, using the dedicated Update Statistics window.

❖ To update database statistics

1. Select Tools ➤ Update Statistics to open the Update Statistics window (if PowerDesigner is not presently connected to a database via a live database connection, you will be required to connect):



Update Statistics

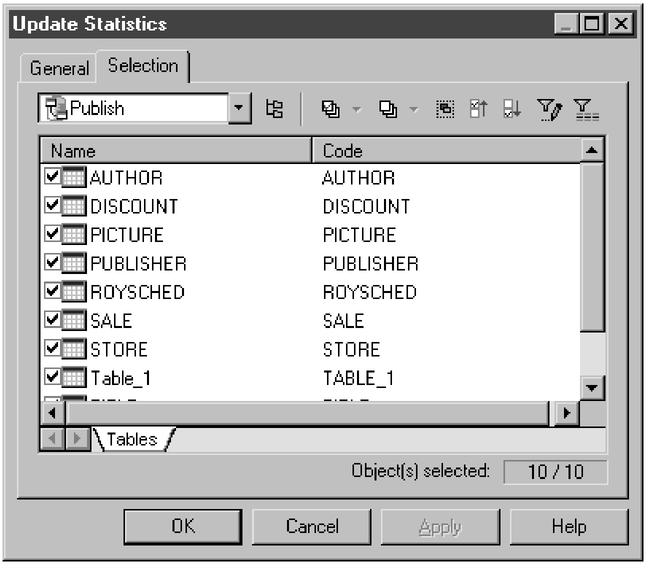
Update statistics for :

☑ Tables

☑ Columns

2. On the General tab, select or clear the checkboxes to specify whether you want to update statistics for tables and/or columns.

3. [optional] Click the Selection tab and select or clear checkboxes to specify for which tables you want to update statistics:



Update Statistics

Name	Code
☑ AUTHOR	AUTHOR
☑ DISCOUNT	DISCOUNT
☑ PICTURE	PICTURE
☑ PUBLISHER	PUBLISHER
☑ ROYSCHED	ROYSCHED
☑ SALE	SALE
☑ STORE	STORE
☑ Table_1	TABLE_1

Object(s) selected: 10 / 10

4. Click OK to begin the update. Progress appears in the Output window. For large updates, a progress dialog box opens, allowing you to cancel the update at any time.

   When the process is complete, you can view the updated statistics in the property sheets of your tables and columns.

CHAPTER 11

DBMS-Specific Features

About this chapter

PowerDesigner can be used with a number of DBMS with specific features often requiring special procedures for certain tasks. This chapter describes these DBMS-specific features and associated tasks.

☞ For more information on DBMS definition file contents, see the DBMS Resource File Reference chapter in the *Customizing and Extending PowerDesigner* manual.

Contents

Working with PowerDesigner's DBMS-Specific Features

This chapter provides information about the DBMS-specific features in PowerDesigner.

PowerDesigner provides customized support for a wide range of database families through DBMS-specific resource files. These files have a .xdb extension, and are located in the "Resource Files\DBMS" directory inside your PowerDesigner installation directory.

Resource files support, among others, three kinds of customization for PowerDesigner model objects:

- Assignment of standard objects to the specific syntax of a DBMS
- Definition of Extended Attributes for certain objects to allow for DBMS-specific object properties
- Definition of additional DBMS-specific model objects.

You can view and edit the resource file for your selected DBMS in the Resource Editor by selecting Database ➤ Edit Current DBMS.

☞ For more information about working with resource files and the Resource File Editor, see the following chapters in the *Customizing and Extending PowerDesigner* manual:

- Resource Files and the Public Metamodel
- DBMS Resource File Reference
- Extending your models with Profiles

IBM DB2 for z/OS (formerly OS/390)

This section describes features specific to the IBM DB2 for z/OS family of databases.

Deprecated versions
The DBMSs for IBM DB2 v5.x are deprecated.

The following table lists DB2 objects and their equivalents in PowerDesigner:

DB2	PowerDesigner
Bufferpool	Storage
Database Partition Group	Extended Object <<DatabasePartitionGroup>>
Distinct Type	Domain
Function	Procedure of “Function” type
Index Extension	Extended Object <<IndexExtension>>
Method	Abstract Data Type Procedure
Type	Abstract Data Type
SuperView	SubView of a View

Trusted contexts

Using a trusted context in an application can improve security by placing accountability at the middle-tier, reducing over granting of privileges, and auditing of end-user’s activities.

Trusted contexts are supported for DB2 v9.x and higher for z/OS. PowerDesigner models trusted contexts as extended objects with a stereotype of <<TrustedContext>>.

Creating a trusted context

You can create a trusted context in any of the following ways:

- Select Model ➤ Trusted Contexts to access the List of Trusted Contexts, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤ Trusted Context.

Trusted context properties

You can modify an object’s properties from its property sheet. To open a trusted context property sheet, double-click its Browser entry in the Trusted Contexts folder.

The following extended attributes are available on the DB2 tab:

Name	Description
Enable	Specifies that the trusted context is created in the enabled state. Scripting name: Enable
Authorization	Specifies that the context is a connection that is established by the authorization ID that is specified by authorization-name. Scripting name: Authorization
Default role	Specifies the default role that is assigned to a user in a trusted connection when the user does not have a role in the trusted context. If empty, then a No Default Role is assumed. Scripting name: DefaultRole
As object owner	Specifies that the role is treated as the owner of the objects that are created using a trusted connection based on the trusted context. Scripting name: WithRoleAsObjectOwner
Default security label	Specifies the default security label for a trusted connection based on the trusted context. Scripting name: DefaultSecurityLabel
Attributes	Specifies one or more connection trust attributes that are used to define the trusted context. Scripting name: Attributes
With use for	Specifies who can use a trusted connection that is based on the trusted context. Scripting name: WithUseFor

Auxiliary tables

Auxiliary tables are used to store large object (LOB) data, such as graphics, video, etc, or to store rarely-accessed data in order to improve the performance of the base table.

Auxiliary tables are supported for IBM DB2 v9.x and higher for z/OS.

PowerDesigner models auxiliary tables as extended objects with a stereotype of <<Auxiliary Table>>.

Creating an auxiliary table

You can create an auxiliary table in any of the following ways:

- Select Model ➤ Auxiliary Table to access the List of Auxiliary Tables, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤ Auxiliary Table.

Auxiliary table properties

You can modify an object's properties from its property sheet. To open an auxiliary table property sheet, double-click its Browser entry in the Auxiliary Tables folder.

The following extended attributes are available on the DB2 tab:

Name	Description
Database	Specifies the database in which the LOB data will be stored. Scripting name: Database
Tablespace	Specifies the table space in which the auxiliary table is created. Scripting name: Tablespace
Table	Specifies the table that owns the LOB column. Scripting name: Table
Column	Specifies the name of the LOB column in the auxiliary table. Scripting name: Column
Partition	Specifies the partition of the base table for which the auxiliary table is to store the specified column. Scripting name: Partition

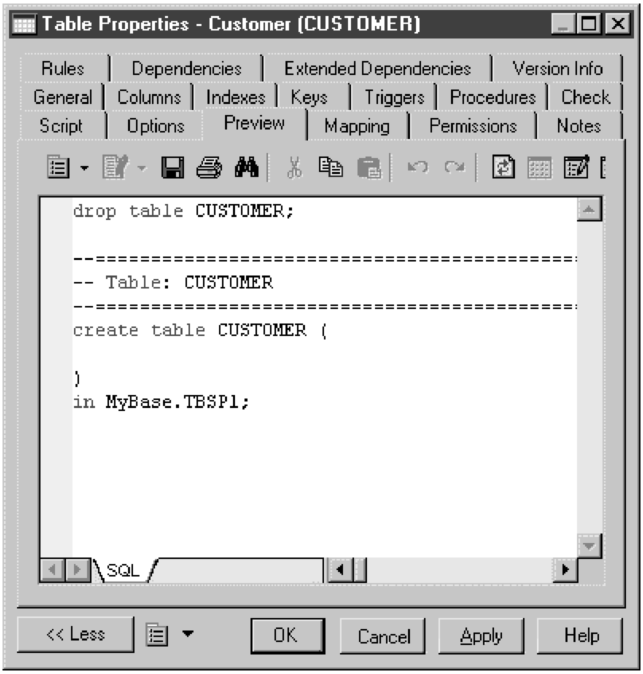
Tablespace prefix

In IBM databases for OS/390, the physical options for a table can specify the tablespace in which a table resides, as well as the database name.

You declare a tablespace in a database and assign a table to a tablespace on the Physical Options (Common) tabs of their property sheets.

If the tablespace is not declared in any database, then the tablespace is not prefixed by any database name.

When you preview your table creation code, you can verify that the tablespace is prefixed by the name of the database.



```
drop table CUSTOMER;

--============================================
-- Table: CUSTOMER
--============================================
create table CUSTOMER (

)
in MyBase.TBSP1;
```

IBM DB2 extended attributes

The following extended attributes are defined by default in the IBM DB2 DBMS.

Column

The following extended attributes are available on the DB2 tab:

Name	Description
Field procedure name	Defines the procedure that will be used as generator/cryptor of values. Scripting name: ExtFieldProcName
Character subtype	[up to v6.x] Specifies a subtype for a character string column (column with a CHAR,VARCHAR,or LONG VARCHAR data type). The subtype can proceed from the list defined in extended attribute type T_ForData. Scripting name: ExtData

Name	Description
Generated value	[v7.x and higher] Indicates that DB2 generates values for the column using the computed column function. If you select Always, the server will send an error message if you try to type a value in the column. If you select By Default, the server uses the computed column value or the value typed for the column. Scripting name: ExtGeneratedAs
Character subtype	[v7.x and higher] Specifies a subtype for a character string column. Scripting name: ExtSubtypeData

Domain

The following extended attributes are available on the DB2 tab:

Name	Description
Character Subtype	[v6.x and higher] Specifies a subtype for a character string column. Scripting name: ExtSubtypeData

IBM DB2 for Common Server

This section describes features specific to the IBM DB2 for Common Server family of databases.

Deprecated versions
The DBMSs for IBM DB2 v5.x are deprecated.

To see the comparison table between DB2 objects and their equivalents in PowerDesigner, see "IBM DB2 for z/OS (formerly OS/390)" on page 537.

Database partition groups

Database partition groups are supported for DB2 v9.x and higher for Common Server.

A partition group is a logical layer that provides for the grouping of one or more database partitions. A partition can belong to more than one partition group. When a database is created, DB2 creates three default partition groups, which cannot be dropped.

Creating a database partition group

You can create a database partition group in any of the following ways:

- Select Model ➤ Database Partition Groups to access the List of Database Partition Groups, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤ Database Partition Group.

Database partition group properties

You can modify an object's properties from its property sheet. To open a database partition group property sheet, double-click its diagram symbol or its Browser entry in the Database Partition Groups folder.

The following extended attributes are available on the DB2 tab:

Property	Description
Database partitions	Specifies the database partitions that are in the partition group. When empty, the group includes all database partitions defined in the database at the time of its creation. Scripting name: DBPartitionNumList

Index extensions

Index extensions are supported for DB2 v9.x and higher, and are used with indexes on tables that have columns of a structured or distinct type.

The following options are available on the DB2 tab:

Property	Description
Owner	Specifies the index extension schema. Scripting name: Owner
Parameters	Specifies a list of parameters (with data types) that is passed to the index extension at CREATE INDEX time to define the actual behavior of this index extension. Scripting name: IndexExtensionParameters
Key generation function	Specifies how the index key is generated using a user-defined table function. Multiple index entries may be generated for a single source key data value. Scripting name: KeyGenerationFunction
Parameter	Specifies parameters for the key generation function. Scripting name: KeyGenerationFunctionParameters
Search methods	Specifies the list of method details of the index search. Each detail consists of a method name, the search arguments, a range producing function, and an optional index filter function. Scripting name: SearchMethods
Source key parameters	Specifies the parameter (and its data type) that is associated with the source key column. Scripting name: SourceKeyParameters
Target key parameters	Specifies the target key parameters that are the output of the key generation function specified on the GENERATE KEY USING clause. Scripting name: TargetKeyParameters

IBM DB2 extended attributes

The following extended attributes are defined by default in the IBM DB2 DBMS.

Abstract Data Types

The following extended attributes are available on the DB2 tab (v9.x and higher):

Name	Description
Inline length	Indicates the maximum size (in bytes) of a structured type column instance to store inline with the rest of the values in the row of a table. Instances of a structured type or its subtypes, that are larger than the specified inline length, are stored separately from the base table row, similar to the way that LOB values are handled. Scripting name: InlineLength
Without comparison	Indicates that there are no comparison functions supported for instances of the structured type. Scripting name: WithoutComparison
Cast (ref as source) function	Defines the name of the system-generated function that casts a reference type value for this structured type to the data type representation type. A schema name must not be specified as part of function name (SQLSTATE 42601). The cast function is created in the same schema as the structured type. If the clause is not specified, the default value for function name is the name of the representation type. Scripting name: RefAsSourceCastFunction
Cast (source as ref) function	Defines the name of the system-generated function that casts a value with the data type representation type to the reference type of this structured type. A schema name must not be specified as part of the function name (SQLSTATE 42601). The cast function is created in the same schema as the structured type. If the clause is not specified, the default value for function name is the structured type name. A matching function signature must not already exist in the same schema (SQLSTATE 42710). Scripting name: SourceAsRefCastFunction
With function access	Indicates that all methods of this type and its subtypes, including methods created in the future, can be accessed using functional notation. This clause can be specified only for the root type of a structured type hierarchy (the UNDER clause is not specified) (SQLSTATE 42613). This clause is provided to allow the use of functional notation for those applications that prefer this form of notation over method invocation notation. Scripting name: WithFunctionAccess

Name	Description
Ref using	Defines the built-in data type used as the representation (underlying data type) for the reference type of this structured type and all its subtypes. This clause can only be specified for the root type of a structured type hierarchy (UNDER clause is not specified) (SQLSTATE 42613). The type cannot be a LONG VARCHAR, LONG VARGRAPHIC, BLOB, CLOB, DBCLOB, DATALINK, or structured type, and must have a length less than or equal to 32 672 bytes (SQLSTATE 42613). If this clause is not specified for the root type of a structured type hierarchy, then REF USING VARCHAR(16) FOR BIT DATA is assumed. Scripting name: RepType
Length/ precision	Specifies the precision for representation type. Scripting name: RepPrecision

Abstract Data Type Attributes

The following extended attributes are available on the DB2 tab (v9.x and higher) with the LOB data type:

Name	Description
Compact	Specifies COMPACT options for LOB data type columns. Scripting name: Compact
Logged	Specifies LOGGED options for LOB data type columns. Scripting name: Logged

Abstract Data Type Procedures

The following extended attributes are available on the DB2 tab (v9.x and higher):

Name	Description
Inherit isolation level	Specifies whether or not a lock request can be associated with the isolation-clause of the statement when the method inherits the isolation level of the statement that invokes the method. The default is INHERIT ISOLATION LEVEL WITHOUT LOCK REQUEST. Scripting name: IsolationLevel
Method is external	Indicates that the CREATE METHOD statement is being used to register a method, based on code written in an external programming language. Scripting name: ExternalMethod

Name	Description
External name	Identifies the name of the user-written code which implements the method being defined. Scripting name: ExternalName
Transform group	Indicates the transform group that is used for user-defined structured type transformations when invoking the method. A transform is required since the method definition includes a user-defined structured type. Scripting name: TransformGroup

Columns

The following extended attributes are available on the DB2 tab:

Name	Description
Lob option	[up to v8.x] Specifies options for LOB data type columns. Scripting name: ExtLobOption
For bit data	Specifies that the content of the column is to be treated as bit (binary) data. This is only applicable on columns with a character datatype. Scripting name: ExtForBitData
Expression	[v7.x and higher] Specifies that the definition of the column is based on an expression. Scripting name: ExtGenExpr
Always Generate value	[v7.x and higher] When set to True (generated always), indicates that DB2 will always generate a value for the column when a row is inserted into the table or whenever the result value of the generation expression may change. When set to False (generated by default), indicates that DB2 will generate a value for the column when a row is inserted into the table, unless a value is specified. Scripting name: ExtGenAlways
Compact	Specifies COMPACT options for LOB data type columns. Scripting name: Compact
Logged	Specifies LOGGED options for LOB data type columns. Scripting name: Logged

Tables

The following extended attributes are available on the DB2 tab:

Name	Description
Ptcfree	Indicates what percentage of each tab to leave as free space during load or reorganization. Scripting name: ExtTablePctFree

Tablespaces

The following extended attributes are available on the DB2 tab:

Name	Description
Type	Specifies the tablespace type, as defined in the extended attribute type ExtTablespaceTypeList. Scripting name: ExtTablespaceType

Views

The following extended attributes are available on the DB2 tab (v9.x and higher):

Name	Description
View is based on a type	Specifies that the columns of the view are based on the attributes of the structured type identified by type-name. Scripting name: ADTView
Structured type	Specifies the abstract data type that the view is based on. Scripting name: ViewType
Super view	Specifies the view that the current view is a subview of. The superview must be an existing view and must be defined using a structured type that is the immediate supertype of the current view type. Scripting name: SuperView
Identifier column	Defines the object identifier column for the typed view. Scripting name: OIDColumn
Unchecked	Defines the object identifier column of the typed view definition to assume uniqueness even though the system cannot prove this uniqueness. Scripting name: Unchecked
Additional options	Defines additional options that apply to columns of a typed view. Scripting name: RootViewOptions

Name	Description
With row movement	Specifies that an updated row is to be moved to the appropriate underlying table, even if it violates a check constraint on that table. Scripting name: WithRowMovement
Check option	Specifies the constraint that every row that is inserted or updated through the view must conform to the definition of the view. Scripting name: CheckOption

Informix SQL

This section describes features specific to the Informix SQL family of databases.

Informix SQL extended attributes

The following extended attributes are defined by default in the Informix SQL DBMS.

Columns

The following extended attributes are available on the Informix tab:

Name	Description
Serial Start	Defines the initial value of the column with a SERIAL datatype. Scripting name: ExtSerialStart

Ingres

This section describes features specific to the Ingres family of databases.

Ingres extended attributes

The following extended attributes are defined by default in the Ingres DBMS.

Columns

The following extended attributes are available on the Extended Attributes tab:

Name	Description
NotDefault	Indicates the column needs a value. This generates the "not default" clause in the sql statement. Scripting name: NotDefault

Users

The following extended attributes are available on the Ingres tab:

Name	Description
Default group	Specifies the default group the user belongs to. Scripting name: DefaultGroup
Expiration date	Specifies an optional expiration date associated with each user. Any valid date can be used. Once the expiration date is reached, the user is no longer able to log on. If the expire_date clause is omitted, the default is noexpire_date. Scripting name: ExpireDate
External password	Allows a user's password to be authenticated externally to Ingres. The password is passed to an external authentication server for authentication. Scripting name: ExternalPassword
Limiting security label	Allows a security administrator to restrict the highest security label with which users can connect to Ingres when enforcing mandatory access control (MAC). Scripting name: LimitingSecurityLabel
Profile	Allows a profile to be specified for a particular user. If the profile clause is omitted, the default is noprofile. Scripting name: Profile

Interbase

This section describes features specific to the Interbase family of databases.

Interbase extended attributes

The following extended attributes are defined by default in the Interbase DBMS.

Indexes

The following extended attributes are available on the Interbase tab:

Name	Description
Row sort	Defines that the default value of the index (ascending or descending) is defined on the index and not on the column. Scripting name: ExtAscDesc

Sequences

The following extended attributes are available on the Interbase tab:

Name	Description
First value	Specifies the sequence first value for Interbase generator. Scripting name: ExtStartWith
Increment value	Specifies the sequence increment value for Interbase generator. Scripting name: ExtIncrement

Microsoft Access

This section describes features specific to the MS Access family of databases.

Deprecated versions
The DBMS for Microsoft Access 95 & 97 is deprecated.

MS Access extended attributes

The following extended attributes are defined by default in the Microsoft Access DBMS.

Columns

The following extended attributes are available on the Access 2000 tab:

Name	Description
Allow Zero Length	Specifies whether a zero-length string (“”) is a valid entry in a table column. Applies only to Text, Memo, and Hyperlink table fields. Scripting name: ExtAllowZeroLength

Microsoft SQL Server

This section describes features specific to the Microsoft SQL Server family of databases.

Horizontal partitioning

Horizontal positioning in MS SQL Server 2005 is a method for making large tables and indexes more manageable by dividing them horizontally and spreading them across more than one filegroup in a database.

PowerDesigner supports horizontal partitioning through the partition function and partition scheme objects.

Partition functions

A partition function specifies how a table or index can be partitioned. PowerDesigner models partition functions as extended objects with a stereotype of <<PartitionFunction>>.

Creating a partition function

You can create a partition function in any of the following ways:

- Select Model ➤ Partition Functions to access the List of Partition Functions, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤ Partition Function.

Partition function properties

You can modify an object's properties from its property sheet. To open a partition function property sheet, double-click its diagram symbol or its Browser entry in the Partition Functions folder.

The following extended attributes are available on the Microsoft tab:

Name	Description
Input Parameter Type	Specifies the data type of the column used for partitioning. All data types are valid, except text, ntext, image, xml, timestamp, varchar(max), nvarchar(max), varbinary(max), alias data types, or CLR user-defined data types. Scripting name: InputParameterType

Name	Description
Interval Side	Specifies to which side of each boundary value interval the boundary_value [,. . . n] belongs. You can choose between: ♦ left [default] ♦ right Interval values are sorted by the Database Engine in ascending order from left to right. Scripting name: IntervalSide
Boundary Values	Specifies the boundary values for each partition of a partitioned table or index. All values must be separated by commas. Scripting name: BoundaryValues

Partition schemes

A partition scheme maps the partitions produced by a partition function to a set of user-defined filegroups. PowerDesigner models partition schemes as extended objects with a stereotype of <<PartitionScheme>>.

Creating a partition scheme

You can create a partition scheme in any of the following ways:

- Select Model ➤ Partition Schemes to access the List of Partition Schemes, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤ Partition Scheme.

Partition scheme properties

You can modify an object's properties from its property sheet. To open a partition scheme property sheet, double-click its diagram symbol or its Browser entry in the Partition Schemes folder.

The following extended attributes are available on the Microsoft tab:

Name	Description
Partition Function	Specifies the partition function using the scheme. Partitions created by the partition function are mapped to the filegroups specified in the partition scheme. Scripting name: PartitionFunction
All Partitions	Specifies that all partitions map to the filegroup specified by the File Groups property. Scripting name: AllPartitions
File Groups	Specifies the names of the filegroups to hold the partitions specified by the partition function. If [PRIMARY] is specified, the partition is stored on the primary filegroup. If ALL is specified, only one filegroup name can be specified. Scripting name: Filegroups

Partitioning a table or an index

To partition a table or an index, specify a partition scheme and column on the Microsoft tab of its property sheet.

Common Language Runtime (CLR) integration

CLR integration means that stored procedures, triggers, and user-defined types, functions, and aggregate functions can be written for SQL Server in any .NET language, such as VB .NET or C#.

PowerDesigner supports CLR integration with assemblies, aggregate functions, CLR types, procedures, functions, and triggers.

CLR assemblies

An assembly is a DLL file used to deploy functions, stored procedures, triggers, user-defined aggregates, and user-defined types that are written in one of the managed code languages hosted by the Microsoft .NET Framework common language runtime (CLR), instead of in Transact-SQL. PowerDesigner models assemblies as extended objects with a stereotype of <<Assembly>>.

Creating an assembly

You can create an assembly in any of the following ways:

- Select Model ➤ Assemblies to access the List of Assemblies, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤

Assembly.

Assembly properties

You can modify an object's properties from its property sheet. To open an assembly property sheet, double-click its diagram symbol or its Browser entry in the Assemblies folder.

The following extended attributes are available on the Microsoft tab:

Name	Description
Authorization	Specifies the name of a user or role as the owner of the assembly. Scripting name: Authorization
File name	Specifies the local path or network location where the assembly that is being uploaded is located, and also the manifest file name that corresponds to the assembly. Can be entered as a fixed string or an expression evaluating to a fixed string. Scripting name: FileName
Permission set	Specifies a set of code access permissions that are granted to the assembly when it is accessed by SQL Server. You can choose between: ♦ SAFE ♦ UNSAFE ♦ EXTERNAL_ACCESS Scripting name: PermissionSet
Visibility	Specifies that the assembly is visible for creating common language runtime (CLR) functions, stored procedures, triggers, user-defined types, and user-defined aggregate functions against it. You can choose between: ♦ On ♦ Off Scripting name: Visibility
Unchecked data	By default, ALTER ASSEMBLY fails if it must verify the consistency of individual table rows. This option allows postponing the checks until a later time by using DBCC CHECKTABLE. Scripting name: UncheckedData

CLR aggregate functions

An aggregate function performs a calculation on a set of values and returns a single value. Traditionally, Microsoft SQL Server has supported only built-in aggregate functions, such as SUM or MAX, that operate on a set of input scalar values and generate a single aggregate value from that set. SQL Server integration with the Microsoft .NET Framework common language runtime (CLR) now allows developers to create custom aggregate functions in managed code, and to make these functions accessible to Transact-SQL or other managed code. PowerDesigner models aggregate functions as extended objects with a stereotype of <<Aggregate>>.

Creating an aggregate function

You can create an aggregate function in any of the following ways:

- Select Model ➤ Aggregates to access the List of Aggregates, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤ Aggregate.

Aggregate function properties

You can modify an object's properties from its property sheet. To open an aggregate function property sheet, double-click its diagram symbol or its Browser entry in the Aggregates folder.

The following extended attributes are available on the Microsoft tab:

Name	Description
Owner	Specifies the name of a schema as the owner of the aggregate function. Scripting name: Owner
Assembly	Specifies the assembly to bind with the aggregate function. Scripting name: Assembly
Class name	Specifies the name of the class in the assembly that implements the aggregate function. If the class name is not specified, SQL Server assumes it is the same as the aggregate name. Scripting name: Class
Parameter name	Specifies the name of the input parameter. Scripting name: InputParameterName

Name	Description
Type	Specifies the type of the input parameter. All scalar data types or CLR user-defined types can be used, except text, ntext, and image. Scripting name: InputParameterType
Return type	Specifies the return type of the aggregate function. All scalar data types or CLR user-defined types can be used as return type, except text, ntext, and image. Scripting name: ReturnType

CLR user-defined types

The introduction of user-defined types (UDTs) in SQL Server 2005 allows you to extend the scalar type system of the server, enabling storage of CLR objects in a SQL Server database. UDTs can contain multiple elements and can have behaviors, differentiating them from the traditional alias data types which consist of a single SQL Server system data type.

Because UDTs are accessed by the system as a whole, their use for complex data types may negatively impact performance. Complex data is generally best modeled using traditional rows and tables. UDTs in SQL Server 2005 are well suited to the following:

- Date, time, currency, and extended numeric types
- Geospatial applications
- Encoded or encrypted data

PowerDesigner models user-defined types as abstract data types.

Creating a user-defined type

To create a user-defined type, you must have already created an assembly (see "CLR assemblies" on page 555) and have an OOM containing an appropriate class open in the workspace, in order to specify the supertype.

❖ To create a user-defined type

1. Create an abstract data type by:
   - Select Model ➤ Abstract Data Types to access the List of Abstract Data Types, and click the Add a Row tool.
   - Right-click the model or package in the Browser, and select New ➤ Abstract Data Type.
2. On the General Tab of its property sheet, select CLR from the list of Types.

3. Click the Select Object tool to the right of the Class field, in order to specify a supertype.
4. Click the Microsoft tab and select an assembly from the list to bind to the type.
5. Click OK to close the property sheet.

User-defined type properties

You can modify an object's properties from its property sheet. To open a user-defined type property sheet, double-click its diagram symbol or its Browser entry in the Abstract Data Types folder.

In addition to the standard abstract data type properties, a user-defined type has the following additional properties available on the Microsoft tab:

Name	Description
Assembly	Specifies the assembly to bind with the abstract data type. Scripting name: Assembly
Mandatory	Specifies whether the type can hold a null value. Scripting name: Mandatory

CLR procedures and functions

In Microsoft SQL Server 2005, you can write user-defined functions in any Microsoft .NET Framework programming language, such as Microsoft Visual Basic .NET or Microsoft Visual C#. PowerDesigner models CLR procedures and functions as standard procedures that use a CLR template, and are linked to a method from an associated OOM.

Creating a CLR procedure or function

To create a CLR procedure or function, you must have already created an assembly (see "CLR assemblies" on page 555) and you must have an OOM open in the workspace, in order to specify an associated class method.

❖ To create a CLR procedure or function

1. Create a standard procedure or function.
2. On the Definition Tab of its property sheet, select CLR Procedure or CLR Function from the template list. A Class method field will be displayed to the right of the template list.
3. Click the Select Method tool to the right of the Class method field, in order to specify the associated method.
4. Click the Microsoft tab and select an assembly from the list to bind to the procedure or function.

5. Click OK to close the property sheet.

CLR procedure or function properties

You can modify an object's properties from its property sheet. To open a CLR procedure or function property sheet, double-click its diagram symbol or its Browser entry in the Procedures folder.

The following extended attributes are available on the Microsoft tab:

Name	Description
Assembly	Specifies the assembly where the class method is defined. Scripting name: Assembly

CLR triggers

The Microsoft SQL Server integration with the .NET Framework common language runtime (CLR), allows you to use any .NET Framework language to create CLR triggers. PowerDesigner CLR triggers as standard triggers that use a CLR template, and are linked to a method from an associated OOM.

Creating a CLR trigger

To create a CLR trigger, you must have already created an assembly (see "CLR assemblies" on page 555) and you must have an OOM open in the workspace, in order to specify an associated class method.

❖ To create a CLR trigger

1. Create a standard trigger.

2. On the Definition Tab of its property sheet, select CLR Trigger from the template list. A Class method field will be displayed to the right of the template list.

3. Click the Select Method tool to the right of the Class method field, in order to specify the associated method.

4. Click the Microsoft tab and select an assembly from the list to bind to the trigger.

5. Click OK to close the property sheet.

CLR trigger properties

You can modify an object's properties from its property sheet. To open a CLR trigger property sheet, double-click its Browser entry.

The following extended attributes are available on the SQL Server tab:

Name	Description
Assembly	Specifies the assembly where the class method is defined. Scripting name: Assembly

Encryption

SQL Server 2005 uses a new security infrastructure that supports hierarchical encryption and key management.

PowerDesigner supports encryption with certificates and asymmetric and symmetric keys.

Certificates

A public key certificate, usually just called a certificate, is a digitally-signed statement that binds the value of a public key to the identity of the person, device, or service that holds the corresponding private key. Certificates are issued and signed by a certification authority (CA). The entity that receives a certificate from a CA is the subject of that certificate. PowerDesigner models certificates as extended objects with a stereotype of <<Certificate>>.

Creating a certificate

You can create a certificate in any of the following ways:

- Select Model ➤ Certificates to access the List of Certificates, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤ Certificate.

Certificate properties

You can modify an object's properties from its property sheet. To open a certificate property sheet, double-click its diagram symbol or its Browser entry in the Certificates folder.

The following extended attributes are available on the Microsoft tab:

Name	Description
Authorization	Specifies the name of a user as the owner of the certificate. Scripting name: Authorization
Assembly	Specifies a signed assembly that has already been loaded into the database. Scripting name: Assembly

Name	Description
Assembly File	Specifies the complete path, including file name, to a DER encoded file that contains the certificate. The path name can be a local path or a UNC path to a network location. The file will be accessed in the security context of the SQL Server service account. This account must have the required file system permissions. Scripting name: AssemblyFile
Executable	If the EXECUTABLE option is used, the file is a DLL that has been signed by the certificate. Scripting name: Executable
File	Specifies the complete path, including file name, to the private key. The private key path name can be a local path or a UNC path to a network location. The file will be accessed in the security context of the SQL Server service account. This account must have the necessary file system permissions. Scripting name: PrivateKeyFile
Encryption password (private key)	Specifies the password that will be used to encrypt the private key. Scripting name: PrivateKeyEncryptionPassword
Decryption password	Specifies the password required to decrypt a private key that is retrieved from a file. Scripting name: PrivateKeyDecryptionPassword
Subject	Specifies the value of the subject field in the metadata of the certificate as defined in the X.509 standard. Scripting name: Subject
Encryption password	Use this option only if you want to encrypt the certificate with a password. Scripting name: EncryptionPassword
Star date	Specifies the date on which the certificate becomes valid. If not specified, StartDate will be set equal to the current date. Scripting name: StartDate
Expiry date	Specifies the date on which the certificate expires. If not specified, ExpiryDate will be set to a date one year after StartDate. Scripting name: ExpiryDate

Name	Description
Active for begin dialog	Specifies that the certificate is available to the initiator of a Service Broker dialog conversation. Scripting name: ActiveForBeginDialog

Asymmetric keys

An asymmetric key is made up of a private key and the corresponding public key. Each key can decrypt data encrypted by the other. Asymmetric encryption and decryption are relatively resource-intensive, but they provide a higher level of security than symmetric encryption. An asymmetric key can be used to encrypt a symmetric key for storage in a database. PowerDesigner models OBJECTS as extended objects with a stereotype of <<AsymmetricKey>>.

Creating an asymmetric key

You can create an asymmetric key in any of the following ways:

- Select Model ➤ Asymmetric Keys to access the List of Asymmetric Keys, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤ Asymmetric Key.

Asymmetric key properties

You can modify an object’s properties from its property sheet. To open an asymmetric key property sheet, double-click its diagram symbol or its Browser entry in the Asymmetric Keys folder.

The following extended attributes are available on the Microsoft tab:

Name	Description
Authorization	Specifies the name of a user as the owner of the asymmetric key. Scripting name: Authorization
Assembly	Specifies the name of an assembly from which to load the public key. Scripting name: Assembly
Assembly file	Specifies the path of a file from which to load the key. Scripting name: AssemblyFile

Name	Description
Executable	If the EXECUTABLE option is used, the file attribute specifies an assembly file from which to load the public key, otherwise the file attribute specifies the path of a strong name file from which to load the key pair. Scripting name: Executable
Algorithm	Specifies the algorithm used to encrypt the key. Scripting name: Algorithm
Encryption password	Specifies the password with which to encrypt the private key. If this clause is not present, the private key will be encrypted with the database master key. Scripting name: EncryptionPassword

Symmetric keys

A symmetric key is one key that is used for both encryption and decryption. Encryption and decryption by using a symmetric key is fast, and suitable for routine use with sensitive data in the database. PowerDesigner models symmetric keys as extended objects with a stereotype of <<SymmetricKey>>.

Creating a symmetric key

You can create a symmetric key in any of the following ways:

- Select Model ➤ Symmetric Keys to access the List of Symmetric Keys, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤ Symmetric Key.

Symmetric key properties

You can modify an object's properties from its property sheet. To open a symmetric key property sheet, double-click its diagram symbol or its Browser entry in the Symmetric Keys folder.

The following extended attributes are available on the Microsoft tab:

Name	Description
Authorization	Specifies the name of a user or role as the owner of the key. Scripting name: Authorization
Certificate	Specifies the name of the certificate that will be used to encrypt the symmetric key. Scripting name: Certificate

Name	Description
Password	Specifies a password from which to derive a TRIPLE_DES key with which to secure the symmetric key. Password complexity will be checked. You should always use strong passwords. Scripting name: Password
Symmetric key	Specifies a symmetric key to be used to encrypt the key that is being created. Scripting name: SymmetricKey
Asymmetric key	Specifies an asymmetric key to be used to encrypt the key that is being created. Scripting name: AsymmetricKey
Key source	Specifies a pass phrase from which to derive the key. Scripting name: KeySource
Algorithm	Specifies the algorithm used to encrypt the key Scripting name: Algorithm
Identity value	Specifies an identity phrase from which to generate a GUID for tagging data that is encrypted with a temporary key. Scripting name: IdentityValue

Full text search

SQL Server 2005 supports full-text queries against a table's plain character data. PowerDesigner supports this feature through the full text catalog and full text index objects.

Full-text catalogs

A full-text catalog contains zero or more full-text indexes. PowerDesigner models full-text catalogs as extended objects with a stereotype of <<FullTextCatalog>>.

Creating a full-text catalog

You can create a full-text catalog in any of the following ways:

- Select Model ➤ Full-Text Catalogs to access the List of Full Text Catalogs, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤ Full Text Catalog.

Full-text catalog properties

You can modify an object’s properties from its property sheet. To open a full-text catalog property sheet, double-click its diagram symbol or its Browser entry in the Full Text Catalogs folder.

The following extended attributes are available on the Microsoft tab:

Name	Description
Authorization	Specifies the name of a user or role as the owner of the full text catalog. Scripting name: Authorization
File group	Specifies the name of the SQL Server filegroup (or storage) of which the new catalog will be part. Scripting name: FileGroup
Path	Specifies the root directory for the catalog. Scripting name: Path
Accent sensitivity	Specifies whether the catalog is accent sensitive for full text indexing. Scripting name: AccentSensitivity
Default	Specifies that the catalog is the default catalog. Scripting name: Default

Full-text indexes

A full-text index stores information about significant words and their location within a given column. This information is used to quickly compute full-text queries that search for rows with particular words or combinations of words. PowerDesigner models full-text indexes as table indexes with an index type set to “Full Text”.

Creating a full-text index

❖ To create a full-text index

1. Create an index by opening the property sheet of a table, clicking the Indexes tab and clicking the Add a Row tool.

2. Open the property sheet of the new index by clicking the properties tool, and then select FULLTEXT from the list of Types on the General tab.

3. Click the Select Class tool to the right of the Class field, in order to specify a supertype.

4. Click the Microsoft tab and select a catalog from the list and then specify the type of change tracking required.

5. Click OK to confirm your changes and close the property sheet.

Full-text index properties

You can modify an object's properties from its property sheet. To open a full-text index property sheet, double-click its Browser entry.

In addition to the standard index properties, a full-text index has the following additional properties available on the Microsoft tab:

Name	Description
Catalog	Specifies the full text catalog where the full text index is defined. Scripting name: FullTextCatalog
Change tracking	Specifies whether or not SQL Server maintains a list of all changes to the indexed data. You can choose between: ♦ manual ♦ auto ♦ off ♦ off, no population Scripting name: ChangeTracking

XML indexing

SQL Server 2005 provides improvements in indexing XML data. PowerDesigner supports these new features through the XML index object.

XML indexes

PowerDesigner models XML indexes as table indexes with an index type set to "XML".

Creating an XML index

❖ To create an XML index

1. Create an index by opening the property sheet of a table, clicking the Indexes tab and clicking the Add a Row tool.

2. Open the property sheet of the new index by clicking the properties tool, and then select XML from the list of Types on the General tab.

3. Click the Microsoft tab and specify any additional options.

4. Click OK to confirm your changes and close the property sheet.

XML index properties

You can modify an object's properties from its property sheet. To open an XML index property sheet, double-click its Browser entry.

The following extended attributes are available on the Microsoft tab:

Name	Description
Primary	Specifies that this is the primary xml index. Scripting name: XMLPrimary
Primary index	Specifies the primary XML index to use in creating a secondary XML index. Scripting name: PrimaryXMLIndex
Secondary XML index type	Specifies the type of the secondary XML index. Scripting name: SecondaryXMLIndexType
Fill factor	Specifies a percentage that indicates how full the Database Engine should make the leaf level of each index page during index creation or rebuild. Scripting name: FillFactor
Max degree of parallelism	Overrides the max degree of parallelism configuration option for the duration of the index operation. Use MAXDOP to limit the number of processors used in a parallel plan execution. The maximum is 64 processors. Scripting name: MaxDop
Pad index	Specifies index padding. Scripting name: PadIndex
Statistics no recompute	Specifies whether distribution statistics are recomputed. Scripting name: StatisticsNoRecompute
Drop existing	Specifies that the named, preexisting clustered, nonclustered, or XML index is dropped and rebuilt. Scripting name: DropExisting
Sort in temporary database	Specifies whether to store temporary sort results in tempdb. Scripting name: SortInTempDB
Allow row locks	Specifies whether row locks are allowed. Scripting name: AllowRowLocks

Name	Description
Allow page locks	Specifies whether page locks are allowed. Scripting name: AllowPageLocks

XML data types

The new XML data type allows you to store XML documents and fragments in a SQL Server 2005 database. PowerDesigner supports this through new column properties and the XML schema collection object.

XML schema collections

An XML schema collection provides the following:

- Validation constraints - Whenever a typed XML instance is assigned to or modified, SQL Server validates the instance.
- Data type information about the instance data - Schemas provide information about the types of attributes and elements in the XML data type instance. The type information provides more precise operational semantics to the values. For example, decimal arithmetic operations can be performed on a decimal value, but not on a string value. Because of this, typed XML storage can be made significantly more compact than untyped XML.

PowerDesigner models XML schema collections as extended objects with a stereotype of <<XMLSchemaCollection>>.

Creating an XML schema collection

You can create a XML schema collection in any of the following ways:

- Select Model ➤ XML Schema Collections to access the List of XML Schema Collections, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤ XML Schema Collection.

XML schema collection properties

You can modify an object's properties from its property sheet. To open a XML schema collection property sheet, double-click its diagram symbol or its Browser entry in the XML Schema Collections folder.

The following extended attributes are available on the Microsoft tab:

Name	Description
Owner	Specifies the name of a user, role, or schema as the owner of the schema collection. Scripting name: Owner
XML model	Specifies a PowerDesigner XML model to link to the schema. Scripting name: XMLModel
Content	Specifies the content of the xml schema. By default this field contains the %xmlModelContent% template, which represents the content of the linked XML model. Scripting name: Content

XML table columns

You can create columns with a type of XML and store XML instances in them.

Creating an XML table column

❖ To create an XML table column

1. Create a column by opening the property sheet of a table, clicking the Columns tab and clicking the Add a Row tool.
2. Open the property sheet of the new column by clicking the properties tool, and then select XML from the list of Data types on the General tab.
3. Click the Microsoft tab and specify an XLM schema collection and content type.
4. Click OK to confirm your changes and close the property sheet.

XML table column properties

You can modify an object's properties from its property sheet. To open an XML table column property sheet, double-click its Browser entry.

The following extended attributes are available on the Microsoft tab:

Name	Description
XML schema collection	Specifies an XML schema collection for the type. Scripting name: XMLSchemaCollection

Name	Description
Content type	Specifies the nature of the content to be stored in the column. You can choose between: ♦ CONTENT – [default] the data can contain multiple top-level elements. ♦ DOCUMENT – the data can contain only one top-level element. Scripting name: ContentType

Database mirroring

SQL Server 2005 supports database mirroring, in which the principal server sends, in real-time, blocks of its database log records to the mirror instance which, in the event of failover, can be made available within a few seconds.

PowerDesigner supports database mirroring with the following model objects:

- Databases (Mirroring tab)
- Endpoints

Databases

For information about creating a database object, see "Creating a database" section in the Getting Started with Data Modeling chapter.

The following extended attributes are available on the Mirroring tab:

Name	Description
Enable mirroring	Enables mirroring for the database. Scripting name: EnableMirroring
Partner/ Witness	Specifies the role that the database will play in the mirroring relationship. You can choose between: ♦ Partner – the database is either a principal or mirror database. ♦ Witness – the database acts as a witness to a mirroring relationship. A SET WITNESS clause affects both copies of the database, but can only be specified on the principal server. If a witness is set for a session, a quorum is required to serve the database, regardless of the SAFETY setting. Scripting names: Partner, Witness

Name	Description
Options	Specifies mirroring options for the database. You can choose between: ♦ <None> ♦ server ♦ off ♦ failover ♦ force_service_allow_data_loss ♦ resume ♦ safety full ♦ safety off ♦ suspend ♦ timeout Scripting name: MirrorOptions
Server	For partner mirroring, specifies the server network address of an instance of SQL Server to act as a failover partner in a new database mirroring session. For witness mirroring, specifies an instance of the Database Engine to act as the witness server for a database mirroring session. Scripting name: MirrorServer
Time-out	[if partner is selected] Specifies the time-out period in seconds. The time-out period is the maximum time that a server instance waits to receive a PING message from another instance in the mirroring session before considering that other instance to be disconnected. Scripting name: TimeOut

End points

An end point encapsulates a transport protocol and a port number, and enables SQL Server to communicate over the network. PowerDesigner models end points as extended objects with a stereotype of <<EndPoint>>.

Creating an end point

You can create an end point in any of the following ways:

- Select Model ➤ End Points to access the List of End Points, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤ End

Point.

End point properties

You can modify an object's properties from its property sheet. To open an end point property sheet, double-click its Browser entry.

The following extended attributes are available on the Microsoft tab:

Name	Description
Owner	Specifies the owner of the endpoint. Scripting name: Owner
State	Specifies the state of the endpoint at creation. You can choose between: ♦ started ♦ stopped ♦ disabled Scripting name: State
Protocol: Name	Specifies the transport protocol to be used by the endpoint. You can choose between: ♦ http ♦ tcp Scripting name: Protocol
Protocol: Argument	Allows you to enter arguments for the chosen protocol. Scripting name: ProtocolArgument
Language: Name	Specifies the type of content to be sent. You can choose between: ♦ soap ♦ tsql ♦ service_broker ♦ database_mirroring Scripting name: Language
Language: Argument	Allows you to enter arguments for the chosen language. Scripting name: LanguageArgument

Service Broker

The service broker manages a queue of services. Applications that use Service Broker communicate by sending messages to one another as part of a conversation. The participants in a conversation must agree on the name and content of each message. PowerDesigner supports service broker through the following objects:

- Message types - define the type of data that a message can contain.
- Contracts - define which message types an application uses to accomplish a particular task.
- Queues - store messages.
- Event notifications - execute in response to a DDL statements and SQL Trace events by sending information about these events to a Service Broker service.
- Services - are specific tasks or sets of tasks.

Message types

Message types define the type of data that a message can contain. You create identical message types in each database that participates in a conversation.

Message types specify the type of XML validation that SQL Server performs for messages of that type. For arbitrary or binary data, the message type can specify that SQL Server performs no validation. PowerDesigner models message types as extended objects with a stereotype of <<MessageType>>.

Creating a message type

You can create a message type in any of the following ways:

- Select Model ➤ Message Types to access the List of Message Types, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤ Message Type.

Message type properties

You can modify an object’s properties from its property sheet. To open a message type property sheet, double-click its Browser entry.

The following extended attributes are available on the Microsoft tab:

Name	Description
Authorization	Specifies a database user or role as the owner of the message type. If the current user is dbo or sa, this may be the name of any valid user or role. Otherwise, it must be the name of the current user, a user that the current user has IMPERSONATE permission for, or a role to which the current user belongs. By default, the message type belongs to the current user. Scripting name: Owner
Validation	Specifies how the Service Broker validates the message body for messages of this type. You can choose between: ♦ none [default] – no validation performed ♦ empty – message must contain no data ♦ well_formed_xml – message must contain well-formed XML ♦ valid_xml with schema collection – message must conform to the specified XML schema Scripting name: Validation
Schema	Specifies the name of the schema to be used for validating the message contents. Scripting name: SchemaCollectionName

Contracts

Contracts define the message types used in a Service Broker conversation and also determine which side of the conversation can send messages of that type. Each conversation follows a contract. The initiating service specifies the contract for the conversation when the conversation begins. The target service specifies the contracts that the target service accepts conversations for. PowerDesigner models contracts as extended objects with a stereotype of <<Contract>>.

You create an identical contract in each database that participates in a conversation.

Creating a contract

You can create a contract in any of the following ways:

- Select Model ➤ Contracts to access the List of Contracts, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤ Contract.

Contract properties

You can modify an object's properties from its property sheet. To open a contract property sheet, double-click its Browser entry.

The following extended attributes are available on the Microsoft tab:

Name	Description
Authorization	Specifies a database user or role as the owner of the contract. If the current user is dbo or sa, this may be the name of any valid user or role. Otherwise, it must be the name of the current user, a user that the current user has IMPERSONATE permission for, or a role to which the current user belongs. By default, the contract belongs to the current user. Scripting name: Owner

The MessageTypes tab lists the message types included in the contract via intermediary "message contract" objects. You can reuse an existing message contract or create a new one, using the tools on this tab.

Once you have added or created a message contract, double-click its entry to open its property sheet.

Message contracts

Message contracts are intermediary objects that are used to include a single message in multiple contracts. Message contracts are modeled as extended objects with a stereotype of <<MessageContract>>.

Creating a message contract

You can create a message contract in any of the following ways:

- Use the tools on the MessageTypes tab of a contract property sheet (see "Contracts" on page 575).
- Select Model ➤ Message Contracts to access the List of Message Contracts, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤ Message Contract.

Message contract properties

You can modify an object's properties from its property sheet. To open a message contract property sheet, double-click its Browser entry.

The following extended attributes are available on the Microsoft tab:

Name	Description
Sent by	Specifies which endpoint can send a message of the indicated message type. Contracts document the messages that services can use to have specific conversations. Each conversation has two endpoints: the initiator endpoint, the service that started the conversation, and the target endpoint, the service that the initiator is contacting. Scripting name: Sender
Message type	Specifies the message type of the contract. Scripting name: MessageType

Queues

When a message arrives for a service, Service Broker places the message on the queue associated with the service. PowerDesigner models queues as extended objects with a stereotype of <<Queue>>.

Creating a queue

You can create a queue in any of the following ways:

- Select Model ➤ Queues to access the List of Queues, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤ Queue.

Queue properties

You can modify an object's properties from its property sheet. To open a queue property sheet, double-click its Browser entry.

The following extended attributes are available on the Microsoft tab:

Name	Description
Owner	Specifies the owner of the queue. Scripting name: Owner
Status	Specifies that the queue is available. This is the default. If a queue is unavailable, no messages can be added to or removed from it. If you create a queue as unavailable, then no messages can be added to it until it is made available with an ALTER QUEUE statement. Scripting name: Status

Name	Description
Retention	Specifies that all messages sent or received on conversations using this queue are retained in the queue until the conversations have ended. This allows you to retain messages for auditing purposes, or to perform compensating transactions if an error occurs. The default is to not retain messages in the queue in this way. Scripting name: Retention
Activation	Specifies that a stored procedure is required to activate message processing for the queue. Scripting name: Activation
Status (activation)	Specifies that Service Broker activates the associated stored procedure when the number of procedures currently running is less than MAX_QUEUE_READERS and when messages arrive on the queue faster than the stored procedures receive messages. This is the default. Scripting name: ActivationStatus
Procedure	Specifies the name of the stored procedure to activate to process messages in this queue. Scripting name: ActivationProcedureName
MaxQueueReaders	Specifies the maximum number of instances of the activation stored procedure that the queue can start at the same time. Must be set to between 0 and 32767. Scripting name: ActivationMaxQueueReaders
Execute as	Specifies the user under which the activation stored procedure runs. SQL Server must be able to check the permissions for this user at the time that the queue activates the stored procedure. You can choose between: ♦ SELF - the stored procedure executes as the current user. (The database principal executing this CREATE QUEUE statement.) ♦ OWNER - the stored procedure executes as the owner of the queue. Scripting name: ActivationExecuteAs
File group	Specifies the SQL Server filegroup on which to create the queue. Scripting name: FileGroup

Event Notifications

An event notification sends information about a database or server event to a service broker service. Event notifications are created only by using Transact-SQL statements. PowerDesigner models event notifications as extended objects with a stereotype of <<EventNotification>>.

Creating an event notification

You can create an event notification in any of the following ways:

- Select Model ➤ Event Notifications to access the List of Event Notifications, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤ Event Notification.

Event notification properties

You can modify an object's properties from its property sheet. To open an event notification property sheet, double-click its Browser entry.

The following extended attributes are available on the Microsoft tab:

Name	Description
Applies on	Specifies the scope of the event notification. You can choose between: ♦ database – the notification fires whenever the specified event in the FOR clause occurs anywhere in the instance of SQL Server. ♦ server - the notification fires whenever the specified event in the FOR clause occurs in the current database. ♦ queue - the notification fires whenever the specified event in the FOR clause occurs in the current queue. Can be specified only if FOR QUEUE_ACTIVATION or FOR BROKER_QUEUE_DISABLED is also specified. Scripting name: AppliesOn
Queue	Specifies the queue to which the event notification applies. Available only if Applies on is set to "queue". Scripting name: Queue

Name	Description
With fan in	Instructs SQL Server to send only one message per event to any specified service for all event notifications that: ♦ are created on the same event. ♦ are created by the same principal (as identified by SID). ♦ specify the same service and broker_instance_-specifier. ♦ specify WITH FAN_IN. Scripting name: WithFanIn
Events	Specifies the name of the event type that causes the event notification to execute. Can be a Transact-SQL DDL, SQL Trace, or Service Broker event type. Scripting name: Events
Service	Specifies the target service that receives the event instance data. SQL Server opens one or more conversations to the target service for the event notification. This service must honor the same SQL Server Events message type and contract that is used to send the message. See "Services" on page 580. Scripting name: Service
Instance	Specifies a service broker instance against which broker_service is resolved. Use 'current database' to specify the service broker instance in the current database. Scripting name: Instance

Services

Services are specific tasks or set of tasks. Service Broker uses the name of the service to route messages, deliver messages to the correct queue within a database, and enforce the contract for a conversation. PowerDesigner models services as extended objects with a stereotype of <<Service>>.

Creating a service

You can create a service in any of the following ways:

- Select Model ➤ Services to access the List of Services, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤ Service.

Service properties

You can modify an object's properties from its property sheet. To open a service property sheet, double-click its Browser entry.

The following extended attributes are available on the Microsoft tab:

Name	Description
Authorization	Specifies the owner of the service. Scripting name: Owner
Queue	Specifies the queue that receives messages for the service. The queue must exist in the same database as the service. Scripting name: Queue

The Contracts tab lists the contracts with which the service is associated.

Routes

Routes appear in the routing table for the database. For outgoing messages, Service Broker determines routing by checking the routing table in the local database. For messages on conversations that originate in another instance, including messages to be forwarded, Service Broker checks the routes in msdb. PowerDesigner models routes as extended objects with a stereotype of <<Route>>.

Creating a route

You can create a route in any of the following ways:

- Select Model ➤ Routes to access the List of Routes, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤ Route.

Route properties

You can modify an object's properties from its property sheet. To open a route property sheet, double-click its Browser entry.

The following extended attributes are available on the Microsoft tab:

Name	Description
Owner	Specifies the owner of the route. Scripting name: Owner
Remote service	Specifies the name of the remote service to which the route points. Scripting name: Service

Name	Description
Broker instance	Specifies the database that hosts the target service. Scripting name: BrokerInstance
Lifetime	Specifies the amount of time, in seconds, that SQL Server retains the route in the routing table. Scripting name: Lifetime
Address	Specifies the network address for the route. The next_hop_address specifies a TCP/IP address in the following format: TCP://{ dns_name | netbios_name | ip_address } : port_number Scripting name: Address
Mirror address	Specifies the network address for a mirrored database with one mirrored database hosted at the next_hop_address. The next_hop_mirror_address specifies a TCP/IP address in the following format: TCP://{ dns_name | netbios_name | ip_address } : port_number Scripting name: MirrorAddress

Remote service bindings

Remote service bindings create a binding that defines the security credentials to use to initiate a conversation with a remote service. PowerDesigner models remote service bindings as extended objects with a stereotype of <<RemoteServiceBinding>>.

Creating a remote service binding

You can create a remote service binding in any of the following ways:

- Select Model ➤ Remote Service Bindings to access the List of Remote Service Bindings, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤ Remote Service Binding.

Remote service binding properties

You can modify an object's properties from its property sheet. To open a remote service binding property sheet, double-click its Browser entry.

The following extended attributes are available on the Microsoft tab:

Name	Description
Owner	Specifies the owner of the binding. Scripting name: Owner
Remote service	Specifies the remote service to bind to the user identified in the WITH USER clause. Scripting name: RemoteService
User	Specifies the database principal that owns the certificate associated with the remote service identified by the TO SERVICE clause. Scripting name: User
Anonymous	Specifies that anonymous authentication is used when communicating with the remote service. Scripting name: Anonymous

Resource Governor

Resource Governor, available in SQL Server 2008, lets you limit resource requests by workloads for CPU time and memory to optimize their allocation. PowerDesigner supports Resource Governor through the following objects:

- Workload groups – are containers for sets of similar session requests.
- Resource pools – represent the physical resources of the server.

Workload groups

A workload group serves as a container for session requests that are similar, to allow the aggregate monitoring of resource consumption and the application of a uniform policy to all the requests in the group. A group defines the policies for its members. PowerDesigner models workload group sas extended objects with a stereotype of <<WorkloadGroup>>.

Creating a workload group

You can create a workload group binding in any of the following ways:

- Select Model ➤ Workload groups to access the List of Workload Groups, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤ Workload Group.

Workload group properties

You can modify an object’s properties from its property sheet. To open a workload group property sheet, double-click its Browser entry.

The following extended attributes are available on the Microsoft tab:

Name	Description
Importance	Specifies the relative importance of a request in the workload group. Scripting name: Importance
Request maximum memory	Specifies the maximum amount of memory that a single request can take from the pool. Scripting name: RequestMaxMemoryGrantPercent
Request maximum CPU	Specifies the maximum amount of CPU time, in seconds, that a request can use. Scripting name: RequestMaxCpuTimeSec
Memory grant request timeout	Specifies the maximum time, in seconds, that a query can wait for a memory grant (work buffer memory) to become available. Scripting name: RequestMemoryGrantTimeoutSec
Maximum degree of parallelism	Specifies the maximum degree of parallelism (DOP) for parallel requests. Scripting name: MaxDop
Maximum requests	Specifies the maximum number of simultaneous requests that are allowed to execute in the workload group. Scripting name: GroupMaxRequests
Resource pool	Associates the workload group with the specified resource pool. Scripting name: ResourcePool

Resource pools

A resource pool represents the physical resources of the server. PowerDesigner models resource pools as extended objects with a stereotype of <<ResourcePool>>.

Creating a resource pool

You can create a resource pool in any of the following ways:

- Select Model ➤ Resource Pools to access the List of Resource pools, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤ Resource Pool.

Resource pool properties

You can modify an object's properties from its property sheet. To open a resource pool property sheet, double-click its Browser entry.

The following extended attributes are available on the Microsoft tab:

Name	Description
CPU percent Min	Specifies the guaranteed average CPU bandwidth for all requests in the resource pool when there is CPU contention. The value is an integer, with a default setting of 0. Scripting name: MinCpuPercent
CPU percent Max	Specifies the maximum average CPU bandwidth that all requests in resource pool will receive when there is CPU contention. The value is an integer, with a default setting of 100. Scripting name: MaxCpuPercent
Memory percent Min	Specifies the minimum amount of memory reserved for this resource pool that can not be shared with other resource pools. The value is an integer, with a default setting of 0. Scripting name: MinMemoryPercent
Memory percent Max	Specifies the total server memory that can be used by requests in this resource pool. The value is an integer, with a default setting of 100. Scripting name: MaxMemoryPercent

Synonyms

PowerDesigner supports SQL Server 2005 synonyms through the standard synonym object.

Synonyms can be created for the following types of objects:

- Assembly (CLR) Stored Procedure
- Assembly (CLR) Table-valued Function
- Assembly (CLR) Scalar Function
- Assembly Aggregate (CLR) Aggregate Functions
- Replication-filter-procedure
- Extended Stored Procedure

- SQL Scalar Function
- SQL Table-valued Function
- SQL Inline-table-valued Function
- SQL Stored Procedure
- View
- Table

For more information about synonyms, see the “Building Physical Diagrams” chapter.

SQL Server 2000 Analysis Services

The OLAP Services feature from SQL Server v7.0 is called **Analysis Services** in SQL Server 2000. PowerDesigner supports analysis services through an add-in.

For information about analysis services in SQL Server 2005, see “Microsoft SQL Server 2005 Analysis Services” on page 592.

Analysis Services provide the following capabilities:

- The **Analysis server** that manages, stores multidimensional information and serves client application requests for OLAP data. The server stores cube metadata (cube definition specifications) in a repository. Completed cubes can be stored in a variety of storage modes: multidimensional database files (MOLAP), tables in a relational database (ROLAP), or a hybrid of multidimensional database files and relational tables (HOLAP).

- A metadata **repository** that contains definitions of OLAP data objects such as cubes and their elements.

- The **PivotTable Service,** which is an OLE DB for OLAP provider that connects client applications to the Analysis server and manages offline cubes.

- An object model called **Decision Support Objects** (DSO), that provides support for the Analysis Manager user interface and for custom applications that manage OLAP metadata and control the server. DSO uses hierarchically arranged groups of objects to define basic elements of OLAP data. PowerDesigner creates and manipulates DSO objects to manage metadata for OLAP data.

Source data for multidimensional cubes resides in relational databases where the data has been transformed into a star or snowflake schema typically used in OLAP data warehouse systems. Analysis Services can work with many relational databases that support connections using ODBC or OLE DB.

☞ For more information on SQL Server Analysis Services, see your DBMS documentation.

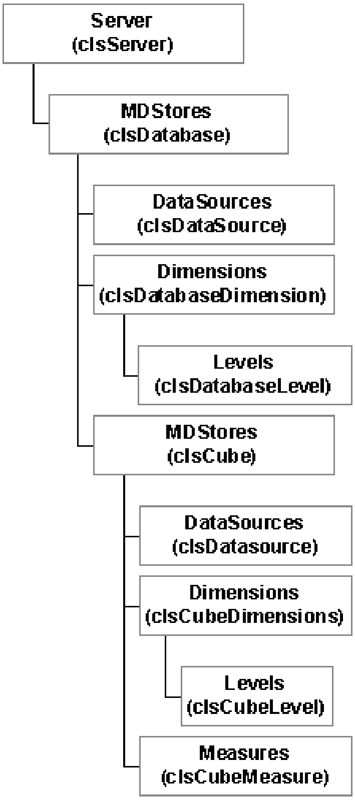
❖ To enable PowerDesigner support for Analysis Services

1. Select Tools ➤ General Options, and select the Add-ins category in the left-hand pane.
2. Select the Microsoft Analysis Services add-in (PdMsOlap.dll) and then click OK to install it and return to the model.

☞ For more information, see "Managing add-ins" in the Models chapter of the *Core Features Guide* .

PowerDesigner and the DSO metamodel

DSO uses hierarchically arranged groups of objects to define basic elements of Analysis Services data storage, as implemented by the Analysis server.



The following table lists the mappings between the objects contained within the DSO and PowerDesigner PDM metamodels:

DSO Object	PowerDesigner PDM Object
clsDatabase	Model (Each model corresponds to a DSO Database.)
clsDataSource	Data source
ClsDatabaseDimension	Dimension (As in the DSO model, PowerDesigner dimensions are shared among cubes.)
clsCube	Cube (Cubes managed by PowerDesigner are only local cubes.)

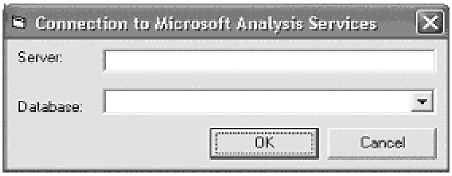
DSO Object	PowerDesigner PDM Object
clsCube	Fact (A Fact corresponds to a DSO cube in order to store measures.)
clsCubeMeasure	Measure
clsDatabaseDimension	Dimension hierarchy (Each dimension hierarchy is generated as a DSO Database Dimension. Attributes of a dimension hierarchy define levels of the corresponding DatabaseDimension.)
clsDatabaseLevel clsCubeLevel	Dimension attribute (Attributes of a dimension or dimension hierarchy define levels in a database dimension.)
clsCubeDimension	Cube dimension association (In DSO, when the name of a Cube Dimension corresponds to the name of a Database Dimension, the Cube Dimension is automatically associated with the Database Dimension to be shared between cubes.)

Generating cubes

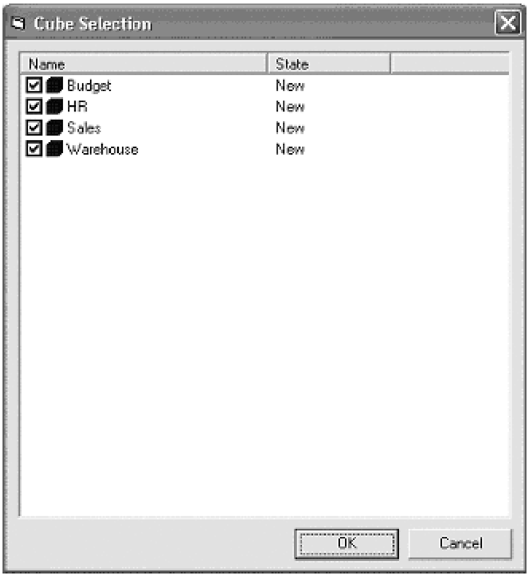
The Microsoft Analysis Services add-in lets you generate cubes.

❖ To generate cubes

1. Select Tools ➤ Microsoft Analysis Services ➤ Generate Cubes to open the connection dialog box.



2. Enter a name for the server and database, and then click OK to open the Cube Selection dialog box, which lists all the available cubes. The state column indicates if the cube has already been generated. Cubes already generated are deselected by default.



Name	State
☑ Budget	New
☑ HR	New
☑ Sales	New
☑ Warehouse	New

3. Select the cubes you want to generate, and then click OK.

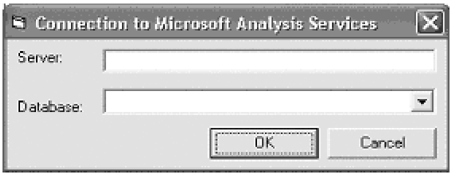
   The selected cubes are generated. If a cube already exists in the database, it is dropped before being recreated. If a dimension already exists, the selected cube reuses it. To be fully generated, a cube must have a complete mapping to a table before being generated.

Reverse Engineering cubes

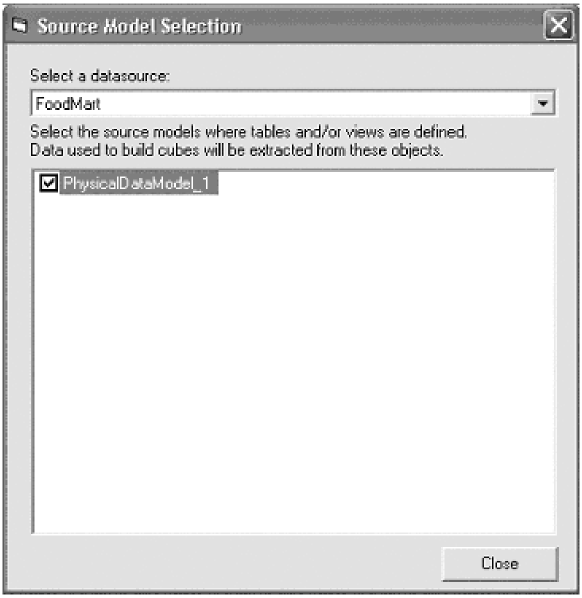
The Microsoft Analysis Services add-in lets you reverse engineer cubes.

❖ To reverse engineer cubes

1. Select Tools ➤ Microsoft Analysis Services ➤ Reverse Engineer Cubes to open the connection dialog box.



2. Enter a name for the server and database, and then click OK to open the Source Model Selection dialog box, which lists the models linked to the selected data source.



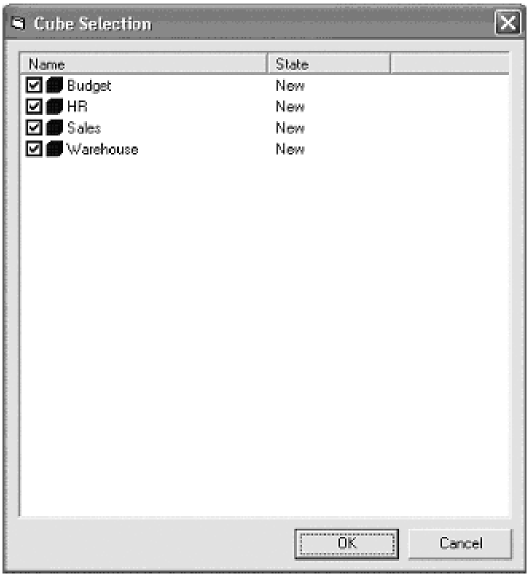
Select a datasource:

FoodMart

Select the source models where tables and/or views are defined.
Data used to build cubes will be extracted from these objects.

☑ PhysicalDataModel_1

3. Select the appropriate source models and then click OK to open the Cube Selection dialog box, which lists all the available cubes. The state column indicates if the cube already exists in the current model. Cubes already existing are deselected by default.



Name	State
☑ Budget	New
☑ HR	New
☑ Sales	New
☑ Warehouse	New

4. Select the cubes you want to reverse engineer, and then click OK.

The selected cubes are created or updated in the current model. If a dimension or a cube already exists, it is updated.

Microsoft SQL Server 2005 Analysis Services

PowerDesigner allows you to retrieve multiple dimension objects in a PDM in order to build cubes, and to create a new multiple-dimension diagram. From this diagram, you can generate cubes to a Microsoft SQL Server 2005 Analysis Server (SSAS).

SQL Server 2005 Management Tools required
In order to use the analysis services add-in to generate and reverse-engineer cubes, you must have installed the SQL Server 2005 Management Tools client component.

❖ To enable PowerDesigner support for Analysis Services

1. Select Tools ➤ General Options, and select the Add-ins category in the left-hand pane.

2. Select the Microsoft SQL Server 2005 Analysis Services add-in (PowerDesigner.AddIn.Pdm.SQLServer.dll) and then click OK to install it and return to the model.

☞ For more information, see “Managing add-ins” in the Models chapter of the *Core Features Guide* .

Generating cubes

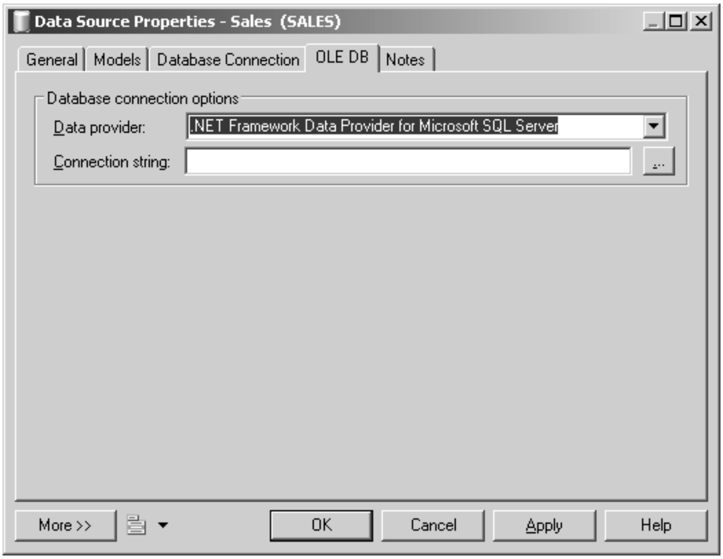
The Microsoft SQL Server 2005 Analysis Services add-in enables the generation of cubes.

Before generating cubes, you must define a data source with an OLE DB connection that will specify from where the cubes will be populated.

❖ To specify a data source for a cube

1. Create a data source in your PDM from the List of data sources or by right-clicking the model in the browser and selecting New ➤ Data Source from the contextual menu.

2. Select the OLE DB tab and specify the kind of data provider.



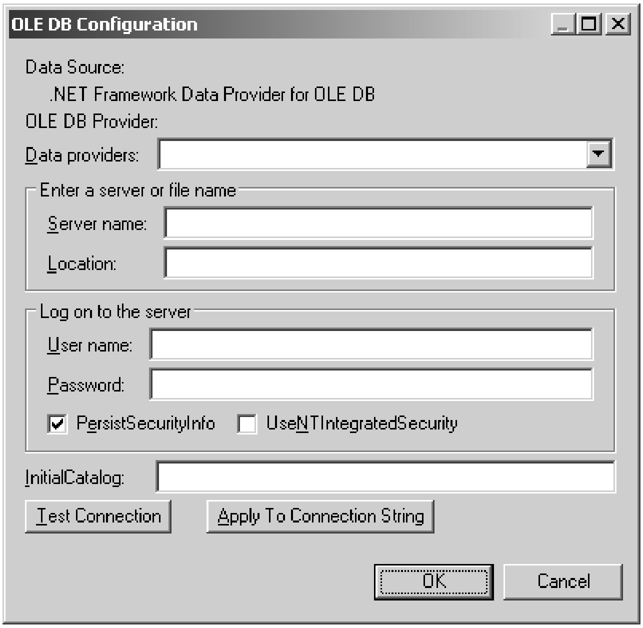
Data Source Properties - Sales (SALES)

Database connection options

Data provider: .NET Framework Data Provider for Microsoft SQL Server

Connection string:

3. Click the ellipsis tool to the right of the connection string field to open the provider-specific configuration dialog.



OLE DB Configuration

Data Source:
.NET Framework Data Provider for OLE DB

☑ PersistSecurityInfo □ UseNTIntegratedSecurity

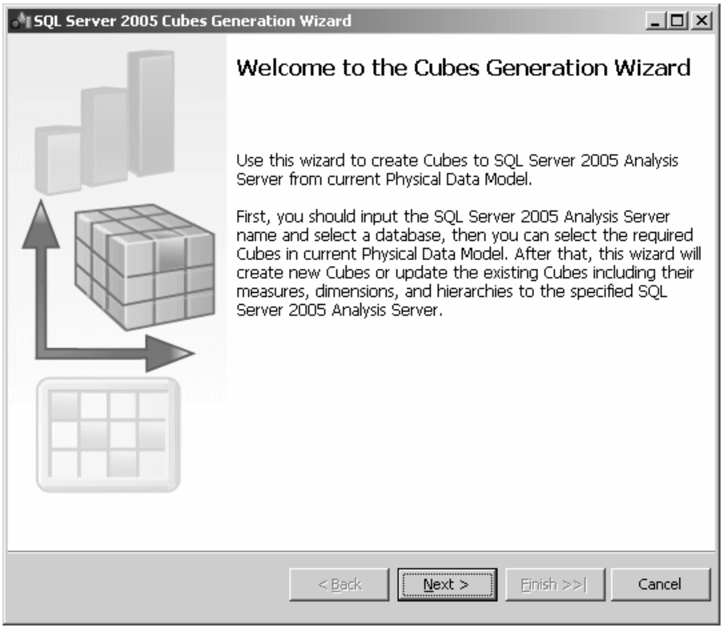
4. Complete the parameters appropriately, click Apply to Connection String, and then Test Connection. Then click Ok to return to the data source property sheet.

5. Click OK to return to your model.

When you have created the appropriate data sources, you can proceed with generating your cubes.

❖ To generate cubes

1. Select Tools ➤ Microsoft SQL Server 2005 Analysis Services ➤ Generate Cubes to open the wizard.

   SQL Server 2005 Cubes Generation Wizard

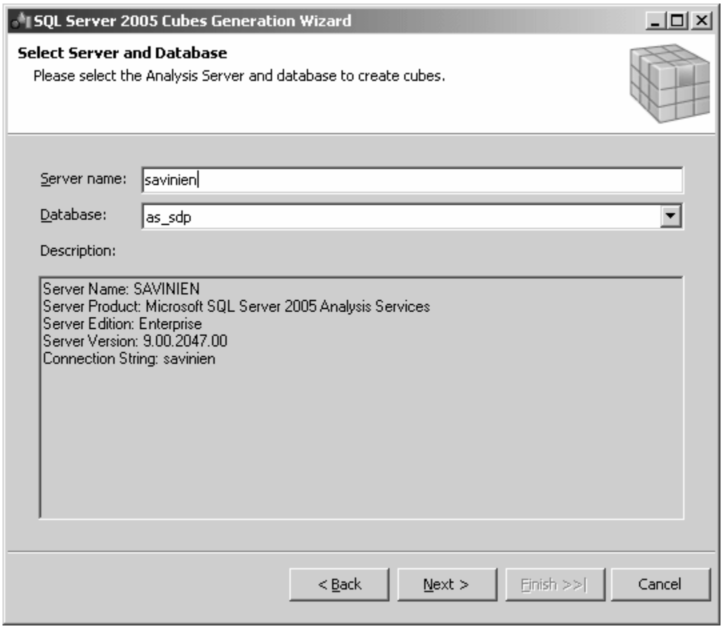
   Welcome to the Cubes Generation Wizard

   Use this wizard to create Cubes to SQL Server 2005 Analysis Server from current Physical Data Model.

   First, you should input the SQL Server 2005 Analysis Server name and select a database, then you can select the required Cubes in current Physical Data Model. After that, this wizard will create new Cubes or update the existing Cubes including their measures, dimensions, and hierarchies to the specified SQL Server 2005 Analysis Server.

   Click Next to continue.

2. Enter a server name, and select the database you want to generate to:



SQL Server 2005 Cubes Generation Wizard

Select Server and Database

Please select the Analysis Server and database to create cubes.

Server name: savinien

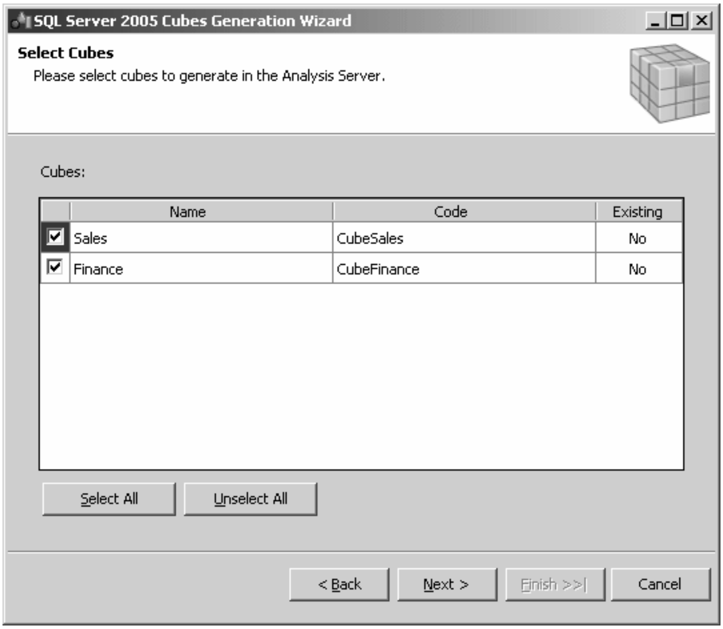
Database: as_sdp

Description:

```
Server Name: SAVINIEN
Server Product: Microsoft SQL Server 2005 Analysis Services
Server Edition: Enterprise
Server Version: 9.00.2047.00
Connection String: savinien
```

Click Next to continue.

3. The Select Cubes page lists the cubes available in the model, along with whether they currently exist in the database. Select the cubes you want to generate:



SQL Server 2005 Cubes Generation Wizard

Select Cubes

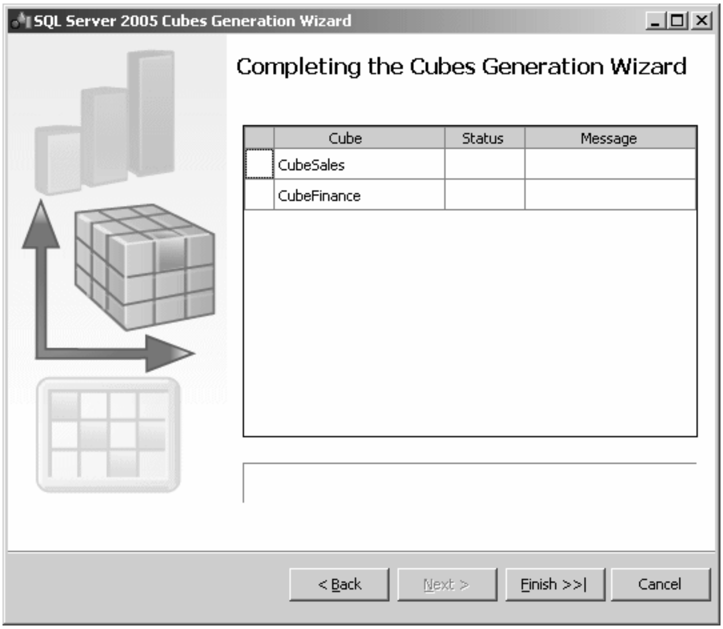
Please select cubes to generate in the Analysis Server.

Cubes:

	Name	Code	Existing
☑	Sales	CubeSales	No
☑	Finance	CubeFinance	No

Click Next to continue.

4. The Generate Cubes page lists the cubes to be generated:



Completing the Cubes Generation Wizard

Cube	Status	Message
CubeSales		
CubeFinance		

Click Finish to begin generation. Progress is displayed in the wizard, which will close automatically after successful completion.

If a cube already exists in the database, it is dropped and recreated. If a related dimension already exists, it is reused. To fully generate a cube, your model must include a complete mapping to a table.

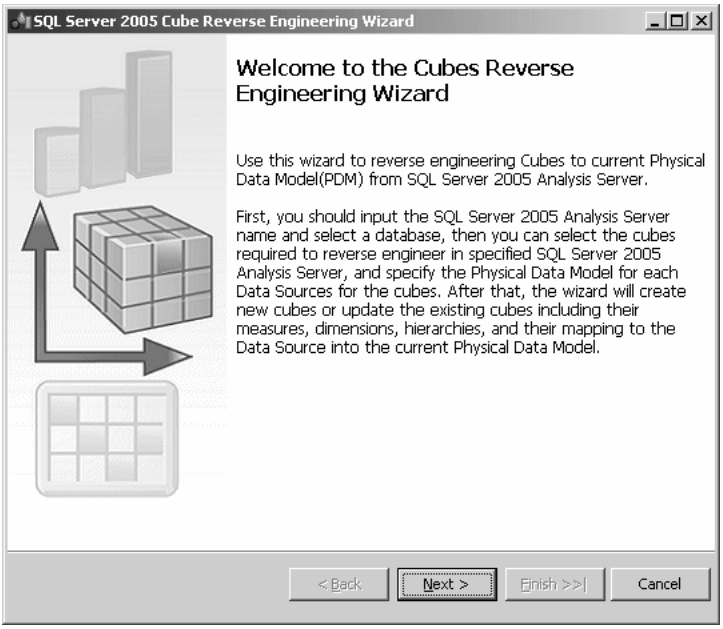
Reverse engineering Microsoft SQL Server 2005 cubes

The Microsoft SQL Server 2005 Analysis Services add-in enables the reverse engineering of cubes.

Before reverse-engineering cubes, you should create one or more PDMs to model the tables which provide its data. As part of the reverse-engineering process, PowerDesigner will create links from the reversed cubes to these tables.

❖ To reverse engineer cubes

1. Select Tools ➤ Microsoft SQL Server 2005 Analysis Services ➤ Reverse Engineer Cubes to open the wizard.



SQL Server 2005 Cube Reverse Engineering Wizard

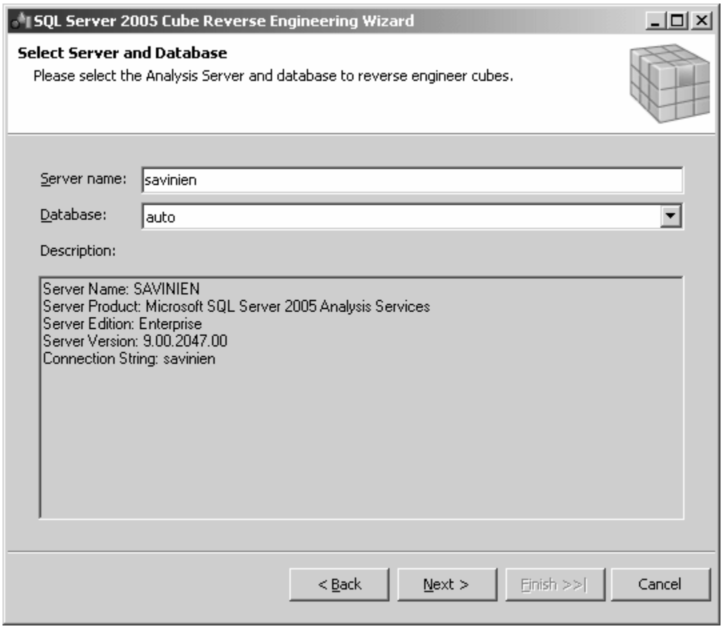
Welcome to the Cubes Reverse Engineering Wizard

Use this wizard to reverse engineering Cubes to current Physical Data Model(PDM) from SQL Server 2005 Analysis Server.

First, you should input the SQL Server 2005 Analysis Server name and select a database, then you can select the cubes required to reverse engineer in specified SQL Server 2005 Analysis Server, and specify the Physical Data Model for each Data Sources for the cubes. After that, the wizard will create new cubes or update the existing cubes including their measures, dimensions, hierarchies, and their mapping to the Data Source into the current Physical Data Model.

Click Next to continue.

2. Enter a server name, and select the database you want to reverse from:



Select Server and Database

Please select the Analysis Server and database to reverse engineer cubes.

Server name: savinien

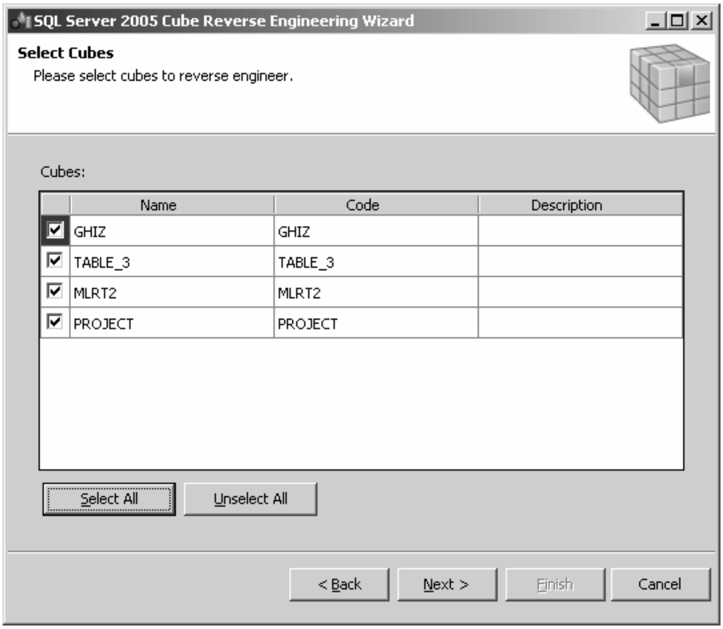
Database: auto

Description:

Server Name: SAVINIEN
Server Product: Microsoft SQL Server 2005 Analysis Services
Server Edition: Enterprise
Server Version: 9.00.2047.00
Connection String: savinien

Click Next to continue.

3. The Select Cubes page lists the available cubes. Select the cubes you want to reverse engineer and then click Next to continue:



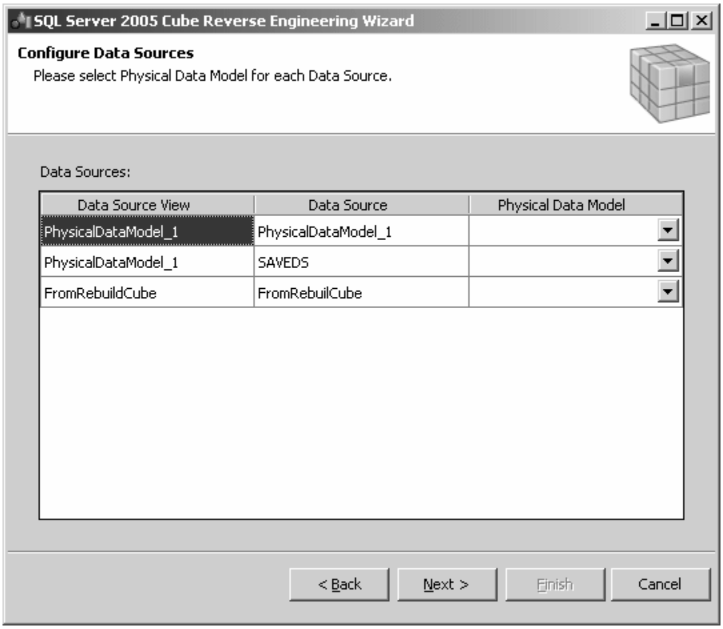
Select Cubes

Please select cubes to reverse engineer.

Cubes:

	Name	Code	Description
☑	GHIZ	GHIZ	
☑	TABLE_3	TABLE_3	
☑	MLRT2	MLRT2	
☑	PROJECT	PROJECT	

4. The Configure Data Sources page lists the data sources that are required to populate the selected cubes. For each source, select the Physical Data Model in which the tables are modeled, and then click Next to continue:



SQL Server 2005 Cube Reverse Engineering Wizard

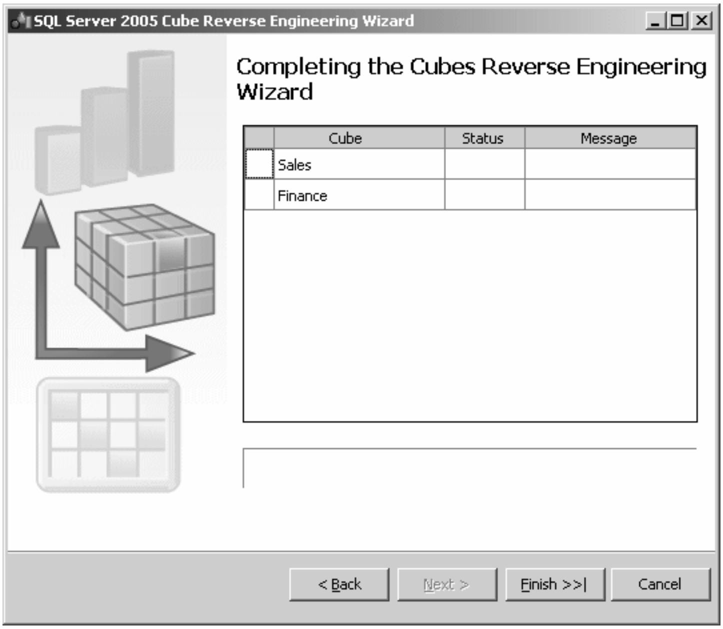
Configure Data Sources

Please select Physical Data Model for each Data Source.

Data Sources:

Data Source View	Data Source	Physical Data Model
PhysicalDataModel_1	PhysicalDataModel_1	
PhysicalDataModel_1	SAVEDS	
FromRebuildCube	FromRebuilCube	

5. The Reverse Engineer Cubes page lists the cubes to be reversed:



SQL Server 2005 Cube Reverse Engineering Wizard

Completing the Cubes Reverse Engineering Wizard

	Cube	Status	Message
	Sales		
	Finance		

Click Finish to begin reverse-engineering. Progress is displayed in the wizard, which will close automatically after successful completion.

Miscellaneous additional features

The following additional features are also available for SQL Server 2005:

- User Schemas – Use the schema stereotype to specify that a user is actually a schema, belonging to another user (the “principal”).
- WithOption – Use the withoptions type to enable access to additional physical options when working with views.
- Support for multiple databases during live database reverse engineering.

MS SQL Server extended attributes

The following extended attributes are defined by default in the MS SQL Server DBMS.

Columns

The following extended attributes are available on the Microsoft tab:

Name	Description
Do not validate check constraints during replication	Specifies that “NOT FOR REPLICATION” keywords are used to prevent the CHECK constraint from being enforced during the distribution process used by replication. Scripting name: ExtCkcNotForReplication
Default constraint name	Contains the name of the constraint that is used to apply a default value to the column. If empty, the “constraint” keyword is not generated. Scripting name: ExtDeftConstName
Identity seed and increment	Is a string composed of two integer values separated by a comma. First value is the seed value of the identity column, meaning the value to be assigned to the first row in the table. Second value is the increment to add to the seed value for successive rows in the table. Scripting name: ExtIdentitySeedInc
Identity value not replicated	Indicates that the IDENTITY property should not be enforced when a replication login inserts data into the table. Scripting name: ExtIdtNotForReplication

Name	Description
Not null constraint name	Contains the name of the constraint that is used to apply a mandatory property of the column. If empty, the “constraint” keyword is not generated. Scripting name: ExtNullConstName
Row global unique identifier	[v2000 and higher] Indicates that the new column is a row global unique identifier column. Only one unique identifier column per table can be designated as the ROWGUIDCOL column. Scripting name: ExtRowGuidCol
Collation name	[v2005 and higher] A single string that specifies the collation name for a SQL collation. Scripting name: ExtCollation
XML schema collection	[v2000 and higher] Applies only to the XML data type for associating an XML schema collection with the type. Scripting name: XMLSchemaCollection
Content type	[v2005 and higher] - CONTENT: Specifies that each instance of the XML data type in column_name can contain multiple top-level elements. CONTENT applies only to the XML data type and can be specified only if xml_schema_collection is also specified. If not specified, CONTENT is the default behavior. - DOCUMENT: Specifies that each instance of the XML data type in column_name can contain only one top-level element. DOCUMENT applies only to the XML data type and can be specified only if xml_schema_collection is also specified. Scripting name: ContentType

Cubes

The following extended attributes are available on the Microsoft tab:

Name	Description
Options	[v2000] You can choose between the following: ♦ PASSTHROUGH: causes the SELECT clause to be passed directly to the source database without modification by PivotTable Service. If PASSTHROUGH is not specified, PivotTable Service parses the query and formulates a set of queries equivalent to the original that is optimized for the source database and index structures. This set of queries is often more efficient than the specified. ♦ DEFER_DATA: causes the query to be parsed locally and executed only when necessary to retrieve data to satisfy a user request. DEFER_DATA is used to specify that a local cube has to be defined in the ROLAP storage mode. ♦ ATTEMPT_DEFER: causes PivotTable Service to attempt to parse the query and defer data loading if successful, or, if the query cannot be parsed, to process the specified query immediately as if PASSTHROUGH had been specified. ♦ ATTEMPT_ANALYSIS: causes PivotTable Service to attempt to parse the query and formulate an optimized set of queries. If the query cannot be parsed, PivotTable Services processes the query immediately as if PASSTHROUGH had been specified. Scripting name: Options
Storage mode	[v2005 and higher] Specifies the storage mode for the cube. Scripting name: StorageMode
Visible	[v2005 and higher] Determines the visibility of the Cube. Scripting name: Visible

Dimensions

The following extended attributes are available on the Microsoft tab:

Name	Description
Hidden	[v2000] Indicates whether the dimension is hidden from clients. Scripting name: IsHidden

Name	Description
Options	[v2000] Dimension options to manage member uniqueness and specify their storage. You can choose between: ♦ UNIQUE_NAME: Member names are unique within the dimension. ♦ UNIQUE_KEY: Member keys are unique within the dimension. ♦ NOTRELATEDTOFACTTABLE: Indicates that non-leaf members cannot be associated with fact table data. ♦ ALLOWSIBLINGSWITHSAMENAME: Determines whether children of a single member in a hierarchy can have identical names. Scripting name: Options
Subtype	[v2000] Indicates the subtype of a dimension. You can choose between: ♦ PARENT_CHILD:Indicates that the dimension is a parent-child dimension. ♦ LINKED: Indicates that the cube is linked to another cube on a remote Analysis server. ♦ MINING: Indicates that the dimension is based on the content of an OLAP data-mining model that has been processed for a cube. Scripting name: SubType
Template	[v2000] Contains a template string that is used to generate captions for system-generated data members. Scripting name: Template

Name	Description
Time	[v2000] Indicates that a dimension refers to time (year, month, week, day, and so on). You can choose between: ♦ TIME: Year, month, week, day, and so on. The only valid levels in a time dimension are those defined in the LevelTypes enumeration. The following values post-fixed by an asterisk (*) are additional values that can be used by the add-in but do not exist in the MDX syntax. You can choose between a dimension that contains: ♦ ACCOUNT: (*) an account structure with parent-child relationships. ♦ BILLOFMATERIALS (*): a material/component breakdown. The parent-child relationship implies a parent composed of its children. ♦ CHANNEL (*): a distribution channel. ♦ CURRENCY (*): currency information. ♦ CUSTOMERS (*): customer information. The lowest level represents individual customers. ♦ GEOGRAPHY (*): a geographic hierarchy. ♦ ORGANIZATION (*): the reporting structure of an organization. ♦ PRODUCTS (*): product information. The lowest level represents individual products. ♦ PROMOTION (*): marketing and advertising promotions. ♦ QUANTITATIVE (*): quantitative elements (such as example, income level, number of children, and so on). ♦ RATES (*): different types of rates (for example, buy, sell, discounted. and so on). ♦ SCENARIO (*): different business scenarios. Scripting name: TimeDef
Type	[v2005 and higher] Provides information about the contents of the dimension. Scripting name: Type
Storage mode	[v2005 and higher] Determines the storage mode for the parent element. Scripting name: StorageMode
AttributeAllMemberName	[v2005 and higher] Contains the caption, in the default language, for the All member of the dimension. Scripting name: AttributeAllMemberName

Name	Description
WriteEnabled	[v2005 and higher] Indicates whether dimension writebacks are available (subject to security permissions). Scripting name: WriteEnabled

Dimension Attributes

The following extended attributes are available on the Microsoft tab:

Name	Description
Rollup expression	[v2000] Contains a Multidimensional Expressions (MDX) expression used to override the default roll-up mode. Scripting name: CustomRollupExpr
Format key	[v2000] Name of the column or expression that contains member keys. Scripting name: FormatKey
Format name	[v2000] Name of the column or expression that contains member names. Scripting name: FormatName
Hide values	[v2000] Options to hide level members. You can choose between: ♦ BLANK_NAME: Hides a level member with an empty name. ♦ PARENT_NAME: Hides a level member when the member name is identical to the name of its parent. ♦ ONLY_CHILD_AND_BLANK_NAME: Hides a level member when it is the only child of its parent and its name is null or an empty string. ♦ ONLY_CHILD_AND_PARENT_NAME: Hides a level member when it is the only child of its parent and is identical to the name of its parent. Scripting name: HideValues
Hidden	[v2000] Indicates whether the level is hidden from client applications. Scripting name: IsHidden

Name	Description
Options	[v2000] Options about member uniqueness, ordering and data source. You can choose between: ♦ UNIQUE: Indicates that the members of a level are unique. ♦ UNIQUE_NAME: Indicates that their member name columns uniquely identify the level members. ♦ UNIQUE_KEY: Indicates that their member key columns uniquely identify the level members. ♦ NOTRELATEDTOFACTTABLE: Indicates that the level members cannot be associated with fact table data. ♦ SORTBYNAME: Indicates that level members are ordered by their names. ♦ SORTBYKEY: Indicates that level members are ordered by their keys. ♦ SORTBYPROPERTY <property names>: Indicates that members are ordered by their property <property names>. Scripting name: Options
Root values	[v2000] Determines how the root member or members of a parent-child hierarchy are identified. You can choose between: ♦ ROOT_IF_PARENT_IS_BLANK: Only members with a null, a zero, or an empty string in their parent key column are treated as root members. ♦ ROOT_IF_PARENT_IS_MISSING: Only members with parents that cannot be found are treated as root members. ♦ ROOT_IF_PARENT_IS_SELF: Only members having themselves as parents are treated as root members. ♦ ROOT_IF_PARENT_IS_BLANK _OR_SELF_OR_MISSING: Members are treated as root members if they meet one or more of the conditions specified by ROOT_IF_PARENT_IS_BLANK, ROOT_IF_PARENT_IS_SELF, or ROOT_IF_PARENT_IS_MISSING. Scripting name: RootValues

Name	Description
Type	[v2000 and higher] Identifies the specific type of level. You can choose between: ♦ ALL: Indicates the top (All) level of a dimension (the one that precalculates all the members of all lower levels). ♦ YEAR: a level that refers to years (Time dimension only). ♦ QUARTER: a level that refers to (calendar) quarters (Time dimension only). ♦ MONTH: a level that refers to months (Time dimension only). ♦ WEEK: a level that refers to weeks (Time dimension only). ♦ DAY: a level that refers to days (Time dimension only). ♦ DAYOFWEEK: a level that refers to days of the week (Time dimension only). ♦ DATE: a level that refers to dates (Time dimension only). ♦ HOUR: a level that refers to hours (Time dimension only). ♦ MINUTE: a level that refers to minutes (Time dimension only). ♦ SECOND: Indicates that a level refers to seconds (Time dimension only). Scripting name: Type
MembersWithData	[v2005 and higher] Determines whether to display data members for non-leaf members in the parent attribute. Scripting name: MembersWithData
OrderBy	[v2005 and higher] Describes how to order the members contained in the attribute. Scripting name: OrderBy
MemberNamesUnique	[v2005 and higher] Determines whether member names under the parent element must be unique. Scripting name: MemberNamesUnique
IsAggregatable	[v2005 and higher] Specifies whether the values of the DimensionAttribute element can be aggregated. Scripting name: IsAggregatable
AttributeHierarchyEnabled	[v2005 and higher] Determines whether an attribute hierarchy is enabled for the attribute. Scripting name: AttributeHierarchyEnabled

Name	Description
AttributeHierarchyVisible	[v2005 and higher] Determines whether the attribute hierarchy is visible to client applications. Scripting name: AttributeHierarchyVisible

Databases

The following extended attributes are available on the Microsoft tab:

Name	Description
Primary	Specifies that the associated file specification list defines the primary file. Scripting name: Primary
File	Gets or sets the file specification. Scripting name: FileListFileSpec
Filegroup	Gets or sets the first filegroup name. Scripting name: FilelistFilegroup
File (filegroup)	Gets or sets the Filegroup specification. Scripting name: FileGroupFileSpec
Log on	Gets or sets the log file specification. Scripting name: LogOnFileSpec
Collation name	[v2000 and higher] Specifies the default collation for the database. Collation name can be either a Windows collation name or a SQL collation name. Scripting name: Collate
Attach	Specifies that a database is attached from an existing set of operating system files. Scripting name: ForAttach

Name	Description
With	[v2005 and higher] Controls Service Broker options on the database. Service Broker options can only be specified when the FOR ATTACH clause is used. ♦ ENABLE_BROKER: Specifies that Service Broker is enabled for the specified database. ♦ NEW_BROKER: Creates a new service_broker_guid value in both sys.databases and the restored database and ends all conversation endpoints with clean up. The broker is enabled, but no message is sent to the remote conversation endpoints. ♦ ERROR_BROKER_CONVERSATIONS: Ends all conversations with an error stating that the database is attached or restored. The broker is disabled until this operation is completed and then enabled. Scripting name: ForAttachWith
Attach rebuild log	[v2005 and higher] Specifies that the database is created by attaching an existing set of operating system files. Scripting name: ForAttachRebuildLog
Database chaining	[v2005 and higher] When ON is specified, the database can be the source or target of a cross database ownership chain. When OFF, the database cannot participate in cross database ownership chaining. The default is OFF. Scripting name: WithDbChaining
Trust worthy	[v2005 and higher] When ON is specified, database modules (for example, views, user-defined functions, or stored procedures) that use an impersonation context can access resources outside the database. When OFF, database modules in an impersonation context cannot access resources outside the database. The default is OFF. Scripting name: WithTrustworthy
Snapshot of	[v2005 and higher] Specifies the name of the new database snapshot. Scripting name: AsSnapshotOf

Name	Description
Load	[up to v2000] Indicates that the database is created with the "dbo use only" database option turned on, and the status is set to loading. Scripting name: ForLoad

For information about the extended attributes available on the Mirroring tab, see "Database mirroring" on page 571.

Data Sources

The following extended attributes are available on the OLE DB tab:

Name	Description
Data provider	Specifies the data provider. You can choose between: ♦ .NET Framework Data Provider for Microsoft SQL Server ♦ .NET Framework Data Provider for Oracle ♦ Native Data Provider for OLE DB Scripting name: DataProvider
Connection string	Specifies the connection string. Scripting name: ConnectionString

The following extended attributes are available on the configuration:

Name	Description
Server name	Specifies the server name. Scripting name: ServerName
Authentication	[only for SQL Server] Specifies the Windows Authentication and SQL Server Authentication types. Scripting name: AuthenticationType
User name	Specifies the User name. Scripting name: UserName
Password	Specifies the password. Scripting name: Password
Initial catalog	[only for SQL Server and OLE DB] Specifies the Initial catalog. Scripting name: InitialCatalog

Name	Description
Database File	[only for SQL Server] Specifies a Microsoft SQL Server database file if you select an MSSQL connection. Scripting name: MSSQLDatabaseFile
Logical name	[only for SQL Server] Specifies the logical name of the selected database file. Scripting name: LogicalName
Data providers	[only for OLE DB] Specifies the data provider. Scripting name: DataProvider
Location	[only for OLE DB] Specifies the location for OLEDB. Scripting name: Location
Persist security info	[only for OLE DB] Specifies that security information be persistent. Scripting name: PersistSecurityInfo
Use Windows NT Integrated Security	[only for OLE DB] Specifies whether to use windows NT Integrated Security or not. Scripting name: UseNTIntegratedSecurity

Dimension Hierarchies

The following extended attributes are available on the Microsoft tab:

Name	Description
Hidden	[v2000] Indicates whether the hierarchy is hidden from client applications. Scripting name: IsHidden
AllMemberName	[v2005 and higher] Contains the caption in the default language for the All member of a Hierarchy element. Scripting name: AllMemberName
MemberNamesUnique	[v2005 and higher] Determines whether member names under the parent element must be unique. Scripting name: MemberNamesUnique
AllowDuplicateNames	[v2005 and higher] Determines whether duplicate names are allowed in a Hierarchy element. Scripting name: AllowDuplicateNames

Fact Measures

The following extended attributes are available on the Microsoft tab:

Name	Description
Format	[v2000] Format used to display the values of the cube measure. Scripting name: Format
Cube measure function type	[v2000] A value corresponding to the type of aggregate function used by the cube measure. Scripting name: Function
Hidden	[v2000] Indicates whether the measure is visible to the client. Scripting name: IsHidden
Member calculating order	[v2000] Order in which the calculated member will be solved when calculated members intersect each other. Scripting name: SolveOrder
Source column data type	[v2000] Returns an OLE DB enumeration constant that identifies the SourceColumn (in the fact table) data type. Scripting name: Type
Aggregate-Function	[v2005 and higher] Defines the common prefix to be used for aggregation names throughout the associated parent element. Scripting name: AggregateFunction
BindingType	[v2005 and higher] Defines the binding type for the measure. Scripting name: BindingType
Visible	[v2005 and higher] Determines the visibility of the Fact Measure. Scripting name: Visible
FormatString	[v2005 and higher] Describes the display format for a CalculationProperty or a Measure element. Scripting name: FormatString

Indexes

The following extended attributes are available on the Microsoft tab:

Name	Description
Filegroup	Specifies the name of the filegroup. Scripting name: FileGroup
Partition scheme	[v2005 and higher] Specifies the name of the partition scheme. Scripting name: PartitionScheme
Column	[v2005 and higher] Specifies the partitioned column. Scripting name: PartitionSchemeColumn
Fill factor	Specifies a percentage that indicates how full the Database Engine should make the leaf level of each index page during index creation or rebuild. Scripting name: FillFactor
Max degree of parallelism	[v2005 and higher] Overrides the max degree of parallelism configuration option for the duration of the index operation. Use MAXDOP to limit the number of processors used in a parallel plan execution. The maximum is 64 processors. Scripting name: MaxDop
Pad index	Specifies index padding. Scripting name: PadIndex
Statistics no recompute	Specifies whether distribution statistics are recomputed. Scripting name: StatisticsNoRecompute
Drop existing	Specifies that the named, preexisting clustered, nonclustered, or XML index is dropped and rebuilt. Scripting name: DropExisting
Online	[v2005 and higher] Specifies whether underlying tables and associated indexes are available for queries and data modification during the index operation. Scripting name: Online
Sort in temporary database	[v2005 and higher] Specifies whether to store temporary sort results in tempdb. Scripting name: SortInTempDB
Allow row locks	[v2005 and higher] Specifies whether row locks are allowed. Scripting name: AllowRowLocks

Name	Description
Allow page locks	[v2005 and higher] Specifies whether page locks are allowed. Scripting name: AllowPageLocks
Ignore dup key	Specifies the error response to duplicate key values in a multiple row insert operation on a unique clustered or unique nonclustered index. Scripting name: IgnoreDupKey

If the index is not a cluster index, then the Include tab is displayed, allowing you to specify the columns with which it is associated.

Keys

The following extended attributes are available on the Microsoft tab:

Name	Description
Filegroup	Specifies the name of the filegroup. Scripting name: FileGroup
Fill Factor	Specifies how full SQL Server should make each index page used to store the index data. Scripting name: FillFactor

References

The following extended attributes are available on the Microsoft tab:

Name	Description
Do not validate foreign key constraint during replication	Specifies that “NOT FOR REPLICATION” keywords are used to prevent the FOREIGN KEY constraint from being enforced during the distribution process used by replication. Scripting name: ExtFkNotForReplication

Tables

The following extended attributes are available on the Microsoft tab:

Name	Description
File group	Specifies the name of the filegroup. Scripting name: FileGroup
Partition scheme	[v2005 and higher] Specifies the name of the partition scheme. Scripting name: PartitionScheme

Name	Description
Column	[v2005 and higher] Specifies the name of the partitioned column. Scripting name: PartitionSchemeColumn
Text/Image on	Specifies the name of the filegroup where text and image are stored. Scripting name: TextImageOn
Do not validate check constraints during replication	Specifies that “NOT FOR REPLICATION” keywords are used to prevent the TABLE CHECK constraint from being enforced during the distribution process used by replication. Scripting name: ExtCktNotForReplication

Users

The following extended attributes are available on the Microsoft tab with the stereotype <<schema>> (v2005 and higher):

Name	Description
Schema owner	Specifies the name of the database-level principal user that will own the schema. This principal may own other schemas, and may not use the current schema as its default schema. Scripting name: SchemaOwner

Views

The following extended attributes are available on the Microsoft tab:

Name	Description
Encryption option	Defines the encryption option of the view, respecting the view creation syntax. Scripting name: WithOption

MySQL

This section describes features specific to the MySQL family of databases.

Deprecated versions

The DBMSs for MySQL v3.22 and 3.23 are deprecated.

Note that when developing for MySQL and using double quotes as a delimiter, it is necessary to set the sql_mode to ANSI_QUOTES:

```
SET sql_mode='ANSI_QUOTES'
```

MySQL extended attributes

The following extended attributes are defined by default in the MySQL DBMS.

Columns

The following extended attributes are available on the MySQL tab:

Name	Description
Character set	Set of symbols and encodings. Scripting name: CharSet
Collation	Set of rules for comparing characters in a character set. Scripting name: Collate
National	A way to indicate that a CHAR column should use UTF8 character set. Scripting name: National
Unsigned	Indicates negative values are not allowed for the column. Scripting name: Unsigned
Retrieve with leading zeros	When displayed, the default padding of spaces is replaced with zeros. For example, for a column declared as INT(5) ZEROFILL, a value of 4 is retrieved as 00004. If you specify ZEROFILL for a numeric column, MySQL automatically adds the UNSIGNED attribute to the column. Scripting name: ZeroFill

Indexes

The following extended attributes are available on the MySQL tab:

Name	Description
Full text index	Indicates that the index is a full text index. Scripting name: FullText

Keys

The following extended attributes are available on the MySQL tab:

Name	Description
Unique key	When set to True, indicates that the key is unique. False implies that the key allows duplicate values. Scripting name: ExtUnique

Models

The following extended attributes are available on the MySQL tab:

Name	Description
Database type	Indicates the type of the database, as specified in the extended attribute type DatabaseType. Scripting name: DatabaseType

References

The following extended attributes are available on the MySQL tab:

Name	Description
Reference match type	Indicates the reference match type, as specified in the extended attribute type ReferenceMatchType. Scripting name: ReferenceMatch

NonStop SQL

This section describes features specific to the NonStop SQL family of databases.

NonStop SQL extended attributes

The following extended attributes are defined by default in the NonStop SQL DBMS.

Columns

The following extended attributes are available on the Extended Attributes tab:

Name	Description
[none]	Specifies an extended type for columns, listed in extended attribute type TExtType. Scripting name: ExtType

Oracle

This section describes features specific to the Oracle family of databases:

Deprecated versions
The DBMSs for Oracle v8 and v8i (8.1.5) are deprecated.

Special variables for triggers

You can use the following variables with Oracle:

♦ `TRGBODY`

♦ `TRGDESC`

Object and SQLJ object data types

Oracle v8 and higher allows you to specify a table type of “Object”, and to base the table on an object or SQLJ object abstract data type, so that the table uses the properties of the ADT and the ADT attributes become table columns.

☞ For more information on ADTs, see your Oracle documentation.

❖ To create an object (or SQLJ object) data type

1. Select Model ➤ Abstract Data Types to open the List of Abstract Data Types, and click the Add a Row tool. Enter a name for the new ADT, and click the Properties tool to open its property sheet.

2. Select OBJECT or SQLJ_OBJECT from the Type list.

   Additional Attributes and Procedures tabs are displayed in the ADT property sheet.

3. Click the Attributes tab, and create as many attributes, specifying a name, a code, and a data type for each.

4. Click OK to close the property sheet and return to your model.

Now that you have created your new data type, you can base tables on it.

❖ To base a table on an object (or SQLJ object) data type

1. Open a table property sheet, and select Object in the Type list.

2. Select the new object (or SQLJ object) data type in the Based on list.

3. Click OK to close the property sheet and return to your model.

Bitmap join indexes

A bitmap join index is a bitmap index described through a join query. It is defined on a base table, and stores the row ids from the base table along with the indexed columns from the joined tables.

☞ For more information about bitmap join indexes, see your Oracle documentation.

You can design a bitmap join index either automatically or manually.

Automatically creating bitmap join indexes through rebuilding

You can automatically generate a bitmap join index for each fact table and the dimension tables that it references. Each generated bitmap join index consists of the references that link a fact table to all the dimension tables located on a single axis proceeding from the fact table.

A reference between two fact tables does not generate any bitmap join index. A bitmap join index is constrained and can only be defined for tables that are organized in a connected tree.

❖ To rebuild bitmap join indexes for all references linked to a fact table

1. Select Tools ➤ Rebuild Objects ➤ Rebuild Join Indexes to open the Rebuild Join Indexes dialog box, and select one of the following modes:
   - ♦ Delete and Rebuild - all existing indexes are deleted before join index rebuild.
   - ♦ Preserve - preserves all existing join indexes in the PDM.
2. Click the Selection tab, select one or more fact tables in the list, and then click OK.

   A confirmation box asks if you want to continue.
3. Click Yes to generate a bitmap join index for each fact table.

Displaying automatically generated bitmap join indexes
Automatically generated bitmap join indexes appear in the list of join indexes. To display the list, select Model ➤ Join Indexes.

Manually creating bitmap join indexes

You can manually create bitmap join indexes from the list of join indexes or via the base table property sheet.

❖ To create a bitmap join index in the list of join indexes

1. Select Model ➤ Join Indexes to open the List of Join Indexes.
2. Click the Add a Row tool and enter a bitmap join index name in the Name column.
3. Click the Properties tool to open the new bitmap join index property sheet.
4. Select a base table on the General tab, and then click the References tab.
5. Click the Add References tool to open a selection window, which lists the available references depending on the selected base table. Select one or more references in the list, and then click OK.

   The selected reference is displayed in the References list.
6. Click the Columns tab, and then click the Add Columns tool to open a selection window, which lists the available columns depending on the selected references. Select one or more columns in the list, and then click OK.

   The selected columns are displayed in the Columns list.
7. Click OK to complete the creation of the bitmap join index and return to the model.

You can create a join index or a bitmap join index from the Join Index tab of a table property sheet. The bitmap join index will be based on the table.

❖ To create a bitmap join index from the table property sheet

1. Open the property sheet of a table and click the Join Index tab.
2. Click the Insert a Row or the Add a Row tool to create a new bitmap join index associated with the table.
3. Click the Properties tool to display the new bitmap join index property sheet. The Base table box is initialized with the selected table.

Bitmap join index properties

A bitmap join index has the following properties:

Property	Description
Name	The name of the item which should be clear and meaningful, and should convey the item’s purpose to non-technical users.
Code	The technical name of the item used for generating code or scripts, which may be abbreviated, and should not generally include spaces.
Comment	Additional information about the bitmap join index.
Stereotype	Sub-classification among bitmap join indexes.
Owner	Name of the user who created the bitmap join index.
Base table	Name of the table that stores the bitmap join index.

The following tabs are also available:

- Columns tab - The bitmap join index stores information about the columns used for the index. These columns are displayed in the Columns tab of the bitmap join index property sheet.
- Options tab - You can define physical options for bitmap join indexes generation. These options override the default physical options defined in the model. You can choose to generate these options by selecting the Physical Options check box in the Join Index groupbox in the Keys and Indexes tab of the Generation dialog box.

Adding a column to a bitmap join index

You can define the list of columns involved in the bitmap join index. These columns proceed from the different dimension tables linked to the base table.

When you create a bitmap join index manually, you have to select columns using the list of columns in the bitmap index property sheet.

When you create a bitmap join index using the rebuild join index feature, the list of columns is initialized with all columns of the tables involved in the join except foreign keys.

❖ To add a column to a bitmap join index

1. Open the bitmap join index property sheet and make sure the bitmap join index has a base table.
2. Click the Columns tab and then click the Add Columns tool.

   A selection dialog box is displayed, which lists columns to include in the join depending on the selected references.

3. Select one or more columns in the list and click OK.

   The columns appear in the list of bitmap join index columns.

4. Click OK.

Adding a reference to a bitmap join index

You can add a reference to any bitmap join index. You do this, for example, when you create a new reference that you want to include in an existing bitmap join index.

❖ To add a reference to a bitmap join index

1. Select Model ➤ Join Indexes to open the List of Join Indexes.

2. Select a bitmap join index in the list and then click the Properties tool to open its property sheet.

3. Click the References tab, and then click the Add References tool.

   A selection dialog box is displayed. It displays available references depending on the selected base table.

4. Select one or several references in the list.

5. Click OK to add the new reference to the References list:

6. Click OK to close the bitmap join index property sheet.

Database packages

In Oracle, packages encapsulate related procedures, functions, and associated cursors and variables together as a unit in the database. Packages usually have two parts, a specification and a body. The **specification** is the interface with your applications; it declares the types, variables, constants, exceptions, cursors, and subprograms available for use. The **body** fully defines cursors and subprograms, and so implements the specification.

Packages provide advantages in the following areas:

- ♦ **Encapsulation** of related procedures and variables in a single named, stored unit in the database. This provides for better organization during the development process and makes privilege management easier.

- ♦ **Separation** of public and private procedures, variables, constants, and cursors.

- ♦ Better **performance** since entire package is loaded into memory when an object from the package is called for the first time.

Creating a database package

❖ To create a database package

1. Select Model ➤ Database Packages to open the List of Database Packages.
2. Click the Add a Row tool to add a database package to the list, and then click the Properties tool to display its property sheet.
3. Type a name and a code for the database package.
4. [optional] Select an Owner and/or Privilege for the database package.
5. Click OK to complete the creation of the database package.

A database package has the following properties:

Property	Description
Name	The name of the item which should be clear and meaningful, and should convey the item's purpose to non-technical users.
Code	The technical name of the item used for generating code or scripts, which may be abbreviated, and should not generally include spaces.
Comment	Additional information about the database package.
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined.
Owner	Name of database package owner. You choose an owner from a list of users. A database package can only have one owner at a time.
Privilege	Lets you specify whether the functions and procedures in the database package execute with the privileges and in the schema of the user who owns it (definer), or with the privileges and in the schema of CURRENT_USER (invoker).

The following tabs are also available:

- Procedures tab – Lists the procedures associated with the database package (see "Database package procedures" on page 627).
- Variables tab - Lists the variables associated with the database package (see "Database package variables" on page 628).

- Cursors tab - Lists the cursors associated with the database package (see “Database package cursors” on page 629).
- Exceptions tab – Lists the exceptions associated with the database package (see “Database package exceptions” on page 630).
- Types tab - Lists the types associated with the database package (see “Database package types” on page 630).
- Initialization tab - Lets you define initialization code for the database package body. Typically initialization holds statements that initialize database package variables. Initialization takes place after database package creation and compilation in the server.

Inserting procedures in a database package

A database package is a set of related procedures. When you create a database package, you have to declare the procedures it contains.

PowerDesigner differentiates between:

- A procedure created in the model.
- A procedure created in a database package. This procedure only exists in the database package and disappears when you delete the package. In this section, we call it **package procedure**. You can create a package procedure from a model procedure using the copy feature.

❖ To insert procedures in a database package

1. Open the property sheet of the database package and click the Procedures tab.
2. Click the Insert a Row tool if you want to create procedures in the database package.

   or

   Click the Create from Procedure tool if you want to duplicate existing procedures in the database package. Select the procedures and click OK.

   The procedures appear in the list.
3. Select a procedure in the list and click the Properties button to open its property sheet.
4. Specify any appropriate properties.
5. Click OK to close the property sheet and return to your model.

Database package procedures

A package procedure is created in a database package. If you delete the database package you also delete the procedures it contains.

A package procedure has the following properties:

Property	Description
Name	The name of the item which should be clear and meaningful, and should convey the item's purpose to non-technical users.
Code	The technical name of the item used for generating code or scripts, which may be abbreviated, and should not generally include spaces.
Comment	Additional information about the package procedure.
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined.
DB Package	Name of the database package to which the procedure belongs.
Type	Allows you to choose between procedure and function.
Return data type	Allows you to define the return data type of a function.
Pragma	Allows you to type a compiler directive, that is, a string for specifying compilation parameters for the procedure.
Public	Allows you to declare the procedure in the package specification and to permit use from outside the database package. A private procedure (check box deselected) is only defined in the package body.

The following tabs are also available:

- Parameters tab – Lists the input and output parameters required by the procedure (see "Database package parameters" on page 631).
- Definition tab - Lets you define package procedures. Package procedures are not built using the structure of templates defined in the DBMS. You have to type the entire package procedure definition. To do so, you can

use operators and functions to insert script items into the cursor definition.

For example, the definition of the CREDIT package procedure is the following:

```
CREATE PROCEDURE credit (Account_number NUMBER, Amount IN
         NUMBER) AS
BEGIN
UPDATE accounts
SET balance = balance + amount
WHERE account_id = acc_no;
END;
```

Database package variables

Variables can be declared within a package. A variable can be used in a SQL or PL/SQL statement to capture or provide a value when one is needed.

For example, you can define the variable in_stock with a boolean data type to verify if a product is available or not.

Variables have the following properties:

Property	Description
Name	The name of the item which should be clear and meaningful, and should convey the item's purpose to non-technical users.
Code	The technical name of the item used for generating code or scripts, which may be abbreviated, and should not generally include spaces.
Comment	Additional information about the variable.
DB Package	Name of the database package to which the variable belongs.
Data Type	Data type of the variable. You can use the Question Mark button to display the list of Standard Data Types.
Mandatory	If selected, indicates that the not null clause is set on the variable, thus making it mandatory.
Length	Allows you to define the variable length.
Precision	Number of places after the decimal point, for data values that can take a decimal point.
Default value	Default value of the variable.

Property	Description
Constant	Indicates that the variable is a constant. A constant has a value assigned. For example: Credit_Limit constant REAL := 500 000;
Public	Allows you to declare the variable in the package specification and to permit use from outside the database package. A private variable (check box deselected) is only defined in the package body.

Database package cursors

A cursor is a multi-row query, which lets you name a work area and access its stored information.

Cursors have the following properties:

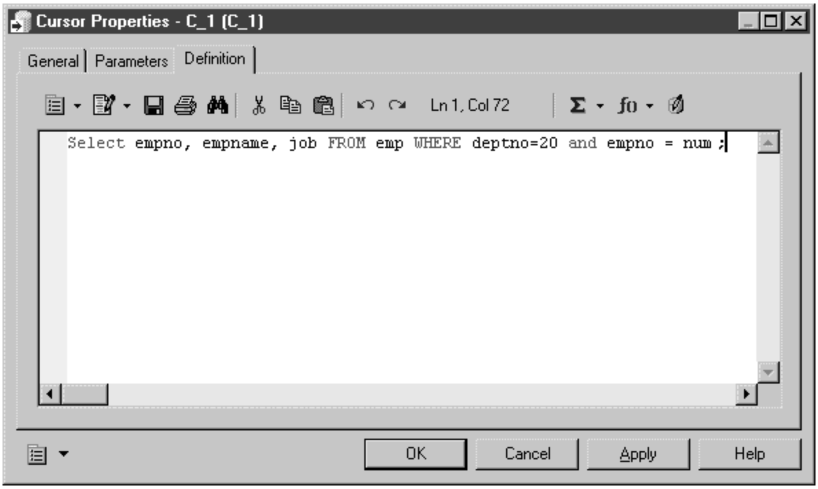
Property	Description
Name	The name of the item which should be clear and meaningful, and should convey the item's purpose to non-technical users.
Code	The technical name of the item used for generating code or scripts, which may be abbreviated, and should not generally include spaces.
Comment	Additional information about the cursor.
DB Package	Name of the database package to which the cursor belongs.
Return Data Type	Allows you to define the data type of a cursor result value.
Public	Allows you to declare the cursor in the package specification and to permit use from outside the database package. A private cursor (check box deselected) is only defined in the package body.

The following tabs are also available:

- Parameters tab – Lists the input and output parameters required by the cursor (see "Database package parameters" on page 631).
- Definition tab - lets you define the cursor. You can use operators and functions to insert script items into the cursor definition.

For example, the following cursor allows locating in table emp, the employee number, name, and function in a given department and for a given employee number:

```
Select empno, empname, job FROM emp WHERE deptno=20 and empno =
        num ;
```



```
Select empno, empname, job FROM emp WHERE deptno=20 and empno = num ;
```

Database package exceptions

PL/SQL allows you to explicitly handle internal and user-defined error conditions, called exceptions, that arise during processing of PL/SQL code.

Exceptions have the following properties:

Properties	Description
Name	The name of the item which should be clear and meaningful, and should convey the item's purpose to non-technical users.
Code	The technical name of the item used for generating code or scripts, which may be abbreviated, and should not generally include spaces.
Comment	Additional information about the exceptions.
DB Package	Name of the database package to which the exception belongs.
Pragma	Allows you to type a compiler directive, that is, a string for specifying compilation parameters for the exception.

Database package types

A type is a user-defined composite datatype that encapsulates a data structure along with the functions and procedures needed to manipulate the data. You can also define subtypes of object types: a subtype contains all the attributes and methods of the parent type, it can contain additional attributes and can override methods from the type.

Types have the following properties:

Property	Description
Name	The name of the item which should be clear and meaningful, and should convey the item's purpose to non-technical users.
Code	The technical name of the item used for generating code or scripts, which may be abbreviated, and should not generally include spaces.
Comment	Additional information about the type.
DB Package	Name of the database package to which the type belongs.
Type	Allows you to declare the type as type or subtype.
Public	Allows you to declare the type in the package specification and to permit use from outside the database package. A private type (check box deselected) is only defined in the package body.

The following tabs are also available:

- Definition - Used to declare the type contents.

The following example defines the type bank_account:

```
CREATE TYPE Bank_Account AS OBJECT (
acct_number INTEGER(5),
balance REAL,
status VARCHAR2(10),
);
```

Database package parameters

Database package procedures (see "Database package procedures" on page 627) and cursors (see "Database package cursors" on page 629) can use input and output parameters.

For example, in a CREDIT procedure, you could define the parameters Account Number and Amount.

Parameters have the following properties:

Property	Description
Name	The name of the item which should be clear and meaningful, and should convey the item's purpose to non-technical users.
Code	The technical name of the item used for generating code or scripts, which may be abbreviated, and should not generally include spaces.
Comment	Additional information about the parameter.
Parent	Specifies the database package parent of the parameter. You can see the database package property sheet by clicking the Properties tool at the right of the field.
Data type	Data type of the parameter. You can use the Question Mark button to display the list of Standard Data Types.
Default Value	Default value of the parameter.
Parameter type	Type of the parameter.

Rebuilding database package procedures

You can rebuild database package procedure dependencies (along with other procedure dependencies) by selecting Tools ➤ Rebuild Objects ➤ Rebuild Procedures Dependencies. For more information, see "Rebuilding procedure dependencies" in the Building Triggers and Procedures chapter.

Generating database packages

You can define generation parameters for database packages in the Database Package tab of the Triggers and Procedures Generation dialog box.

The following parameters are available:

Parameter	Resulting generation command

Create database package	Create database package, as defined in the Oracle DBMS:
	`create package %DBPACKAGE% [authid`
	`   %DBPACKAGEPRIV%`
	`   ][[%R%?[is][as]:as]`
	`%DBPACKAGESPEC%`
	`end [%DBPACKAGE%]/`
	`[create package body %DBPACKAGE% AS`
	`%DBPACKAGEBODY%`
	`[BEGIN`
	`%DBPACKAGEINIT%] ]`
	`end [%DBPACKAGE%]`
	`]`

Drop database package	Drop database package, as defined in the Oracle DBMS: `drop package %DBPACKAGE%`
Permission	Generate the permission statement for a given user during database package creation.

☞ For more information on the create and drop statements for database packages, see the DBMS Resource File Reference chapter in the *Customizing and Extending PowerDesigner* manual.

Reverse engineering database packages

You can select database packages to reverse engineer from the Database Package tab of the Database Reverse Engineering dialog box.

When you reverse engineer a database package, the sub-objects (variable, procedure, cursor, exception, and type) are created from the specification and the body of the database package.

Reverse engineering tablespace physical options

In Oracle, you cannot reverse engineer tablespace physical options via a live database connection without a login “System”.

If you do not have a login “System”, the list of tablespaces is reversed without the physical options of the objects.

If you want to cancel the reverse engineering of tablespace physical options, you have to clear the query `SqlAttrQuery` in the Tablespace category in the Oracle DBMS.

Oracle dimensions

In Oracle, the columns of the dimension table are called **levels** of the dimension hierarchy. These levels can have **dependent columns**. For example, in the dimension Time, the level Quarter has a dependent column called Number of business days.

The following mapping is established between dimensions in Oracle and PowerDesigner:

Oracle object	PowerDesigner object
Dimension	Dimension
Hierarchy	Dimension hierarchy
Level	Dimension attribute used in a hierarchy
Attribute	Dimension attribute used as detail attribute

Oracle dimension creation and deletion orders can be generated and reverse engineered in PowerDesigner. Both generation and reverse engineering can be done by script or through a live database connection. For more information, see the Generating a Database from a PDM and the Reverse Engineering a Database into a PDM chapters.

☞ For more information on dimensions, see the Building Multidimensional Diagrams chapter.

Transparent Data Encryption (TDE)

Oracle 10gR2 provides Transparent Data Encryption (TDE), encryption that is transparent for the user.

When encrypting a column, Oracle creates an encryption key for the parent table and encrypts text data in the column with a user-specified encryption algorithm. The table key is encrypted using a master key and placed in the data dictionary. The master key is stored in a secure location called a wallet, which can be a file on the database server. When a user enters data into an encrypted column, Oracle retrieves the master key from the wallet, decrypts the table key from the data dictionary, and uses it to encrypt the new data.

Creating a wallet password

In order to access the master key used to encrypt the table keys, you must create a master password to open the wallet.

❖ To create a wallet password

1. Right-click the model in the Browser, and select Properties from the contextual menu to open its property sheet.
2. Click the Oracle tab, and enter your wallet password in the Password Encryption field.
3. Click OK to close the model property sheet. The password will be used to create alter statements for opening and closing the wallet.

Creating an encrypted table column

You can create one or more encrypted column in one or more tables. You can specify the encryption algorithm to be used, but all columns in a particular table must use the same algorithm. If you create a second encrypted column in a table, and specify a different algorithm, the last specified algorithm will be used for all columns in the table.

❖ To encrypt a table column

1. Create a column and open its property sheet.
2. On the General tab, specify any of the following types, which support encryption:
   - CHAR, NCHAR, VARCHAR2, and NVARCHAR2
   - DATE and TIMESTAMP
   - INTERVAL DAY TO SECOND and YEAR TO MONTH
   - NUMBER
   - RAW
3. Click the Oracle tab and select the Encryption checkbox.
4. Select an encryption algorithm from the list particular
5. [optional] Select the With salt checkbox in order to add some random bits to the encryption key.
6. Click OK to complete the column definition.

Clusters

A cluster is a schema object that contains data from one or more tables, which have one or more columns in common. Oracle Database stores together all the rows from all the tables that share the same cluster key.

PowerDesigner models clusters as extended objects with a stereotype of <<Cluster>>.

Upgrading clusters from v10gR2 and earlier
Clusters in Oracle v10gR2 and earlier are modeled as indexes with the Cluster check box selected. To upgrade such clusters to v11 or higher, you must generate a new PDM with the appropriate DBMS target from your original model. Simply changing the target DBMS will result in the loss of any existing clusters

Creating a cluster

You can create a cluster in any of the following ways:

- Select Model ➤ Clusters to access the List of Clusters, and click the Add a Row tool
- Right-click the model or package in the Browser, and select New ➤ Cluster

Cluster properties

You can modify an object's properties from its property sheet. To open a cluster property sheet, double-click its Browser in the Clusters folder.

The following extended attributes are available on the General tab:

Name	Description
Owner	Specifies the owner of the cluster

In addition, the following tabs are available:

- Columns – lists the columns associated with the cluster
- Indexes – lists the indexes defined for the cluster

Database Links

A database link is a schema object in one database that enables you to access objects on another database.

Database links are supported for Oracle 11g and higher. PowerDesigner models database links as extended objects with a stereotype of <<Database Link>>.

Creating a database link

You can create a database link in any of the following ways:

- Select Model ➤ Database links to access the List of Database links, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤ Database link.

Database link properties

You can modify an object's properties from its property sheet. To open a database link property sheet, double-click its Browser in the Database links folder.

The following extended attributes are available on the Oracle tab:

Name	Description
Shared	Specifies the use of a single network connection to create a public database link that can be shared among multiple users. If selected, you must also specify a user name and password for the target instance on the remote server. Scripting names: Shared, AuthenticatedBy, AuthenticationPassword
Connect to	Specifies the user name and password used to connect to the remote database using a fixed user database link.You need to specify CURRENT_USER to create a current user database link. The current user must be a global user with a valid account on the remote database. If you do not specify a value, then the database link uses the user name and password of each user who is connected to the database. Scripting names: Username, Password
Service name	Specifies the service name of a remote database. If you specify only the database name, then Oracle Database implicitly appends the database domain to the connect string to create a complete service name. Scripting name: ServiceName
Physical data model	Specifies the PowerDesigner model that contains the objects of the remote database. Use the buttons to the right of the field to create, delete, select, or view the property sheet of the model. Scripting name: LinkModel

Materialized view logs

When DML changes are made to master table data, Oracle Database stores rows describing those changes in the materialized view log and then uses the materialized view log to refresh materialized views based on the master table.

Materialized view logs are supported for Oracle 11g and higher. PowerDesigner models materialized view logs as extended objects with a stereotype of <<Materialized view log>>.

Creating a materialized view log

You can create a materialized view log as follows:

- Open the property sheet of the table to which you want to attach the log, select the Oracle tab, and click the Create button in the Materialized view log groupbox.

Materialized view log properties

You can modify an object's properties from its property sheet. To open a materialized view log property sheet, double-click its Browser entry or click the Properties button on its parent table Oracle tab.

The General tab displays the master table name and the comment. The following properties are available on the Partitions tab:

Name	Description
Type	Specifies the method for paritioning the table. You can choose between: ♦ Range/Composite - Partitions the table on ranges of values from the column list. ♦ Hash - Partitions the table using the hash method. ♦ List - Partitions the table on lists of literal values from column. ♦ Reference - Equipartitions the table being created (the child table) by a referential constraint to an existing partitioned table (the parent table). ♦ System - Partitions the table by the partitions specified. When you select a type, additional options are displayed, to allow you to specify the appropriate parameters.

Oracle extended attributes

The following extended attributes are defined by default in the Oracle DBMS.

Abstract Data Types Attributes

The following extended attributes are available for attributes of abstract data types of type OBJECT or SQLJ_OBJECT on the Oracle tab:

Name	Description
Declare REF	Generates a REF modifier on attribute to declare references, which hold pointers to objects. Scripting name: RefAttribute

Columns

The following extended attributes are available on the Oracle tab:

Name	Description
Deferred option of check constraint	Defines the deferred option of a column constraint check. It is used in the definition or create and add items statements. Scripting name: ExtColumnDeferOption
Name of not null constraint	[v8i and higher] Defines the name of the not null constraint for a column. Scripting name: ExtNotNullConstraintName
Deferred option of not null constraint	[v8i and higher] Defines the deferred option of a column not null constraint. It is used in "create" and "add" statement items definition. An empty value means "Not deferrable". Scripting name: ExtNotNullDeferOption
Encrypted	[v10gR2 and higher] Specifies if column is encrypted. Scripting name: Encrypted
Algorithm	[v10gR2 and higher] Specifies the algorithm used for encryption. Scripting name: Algorithm
With salt	[v10gR2 and higher] Specifies if encryption adds salt to encoded data. Scripting name: EncryptionWithSalt
Identified by Password	[v10gR2 and higher] Identifies by password. Scripting name: IdentifiedByPassword

Database Packages

The following extended attributes are available on the Oracle tab:

Name	Description
Add serially_reusable pragma on package specification	[v9i and higher] When set to True, defines that the pragma serially_reusable clause must be applied on the database package specification. Scripting name: IsSpecPragma
Add serially_reusable pragma on package body	[v9i and higher] When set to True, defines that the pragma serially_reusable clause must be applied on the database package body declaration. Scripting name: IsPragma

References

The following extended attributes are available on the Oracle tab:

Name	Description
Deferred option of foreign key constraint	Defines the deferred option of a reference. It is used in the definition of create and add items statements. Scripting name: ExtReferenceDeferOption
Exceptions into	Specifies a table into which Oracle places the ROWIDs of all rows violating the constraint. Scripting name: ExceptionsInto
Rely	[v8i and higher] Specifies whether an enabled constraint is to be enforced. Specify RELY to enable an existing constraint without enforcement. Specify NORELY to enable and enforce an existing constraint. Scripting name: Rely
Disable	Disables the integrity constraint. Scripting name: Disable
Validate	Checks that all old data also obeys the constraint. Scripting name: Validate

Tables

The following extended attributes are available on the Oracle tab:

Name	Description
Type	Defines if the table is of type global temporary or not. Scripting name: ExtTableType

The following extended attributes are available (for v11g and higher) on the XML properties tab when the table type is set to XML:

Name	Description
Object properties	Specifies that the properties of object tables are essentially the same as those of relational tables. However, instead of specifying columns, you specify attributes of the object. Scripting name: XmlTypeObjProperty
Storage type	Specifies that XMLType columns can be stored in LOB, object-relational, or binary XML columns. Scripting name: XMLTypeStorage

Name	Description
Basic file	Use this clause to specify the traditional LOB storage. Scripting name: BasicFile
Secure file	Use this clause to specify high-performance LOB. Scripting name: SecureFile
LOB segment name	Specify the name of the LOB data segment. You cannot use LOB_segname if you specify more than one LOB_item. Scripting name: LOBSegname
LOB parameters	Use this clause to specify various elements of LOB parameters. Scripting name: LOBParameters

Tablespaces

The following extended attributes are available on the Oracle tab:

Name	Description
Size specification	[v10g and higher] Specifies whether the tablespace is a bigfile or smallfile tablespace. This clause overrides any default tablespace type setting for the database. You can choose from the following settings: ♦ bigfile - contains only one datafile or tempfile. The maximum size of the single datafile or tempfile is 128 terabytes (TB) for a tablespace with 32K blocks and 32TB for a tablespace with 8K blocks. ♦ smallfile - a traditional Oracle tablespace. Scripting name: SizeSpecification
Temporary tablespace	Use this option to create a locally managed temporary tablespace, which is an allocation of space in the database that can contain transient data that persists only for the duration of a session. This transient data cannot be recovered after process or instance failure. Scripting name: Temporary
Undo tablespace	Use this option to create an undo tablespace. When you run the database in automatic undo management mode, Oracle Database manages undo space using the undo tablespace instead of rollback segments. This clause is useful if you are now running in automatic undo management mode but your database was not created in automatic undo management mode. Scripting name: Undo

Views

The following extended attributes are available on the Oracle tab:

Name	Description
Super view object	[v9i and higher] Used in the UNDER clause to specify the superview the current object view is based on. Scripting name: ExtObjSuperView
Object view key	[v8i and higher] Specifies the attributes of the object type that will be used as a key to identify each row in the object view. Scripting name: ExtObjOIDList
Object view type	[v8i and higher] Defines the type of the object view. Scripting name: ExtObjViewType
Force	When set to TRUE, allows you to create the view regardless of the existence of the base tables or the owner privileges on these tables. Scripting name: ExtViewForce

PostgreSQL

This section describes features specific to the PostgreSQL family of databases.

PostgreSQL base and composite type domains

PowerDesigner supports the following special PostgreSQL data types:

- Base Type
- Composite Type

❖ To create a base or composite type

1. Select Model ➤ Domains to open the List of Domains.
2. Click the Add a Row tool to create a new domain, and then click the Properties tool to open its property sheet.
3. Select either BaseType or CompositeType from the Stereotype list and click Apply .
4. An additional tab called either Base Type or Composite Type will be displayed, allowing you to specify type-specific extended attributes. See the relevant section in the "PostgreSQL extended attributes" on page 643 section, for details of these properties.

PostgreSQL extended attributes

The following extended attributes are defined by default in the PostgreSQL DBMS.

Database

The following extended attributes are available on the PostgreSQL tab:

Name	Description
Template	The name of the template from which to create the new database, or DEFAULT to use the default template. Scripting name: Template
Encoding	Character set encoding to use in the new database. Specify a string constant (e.g., 'SQL_ASCII'), or an integer encoding number, or DEFAULT to use the default encoding. Scripting name: Encoding

Domain (Base Type)

The following extended attributes are available on the PostgreSQL tab:

Name	Description
Array delimiter	Delimiter character for the array. Scripting name: ExtTypeDelimiter
Array Element type	Specifies the type of the array elements. Scripting name: ExtTypeElement
Input function	Name of a function, created by CREATE FUNCTION, which converts data from its external form to the internal form of the type. Scripting name: ExtTypeInput
Length	Literal value which specifies the internal length of the new type. Scripting name: ExtTypeLength
Output function	Name of a function, created by CREATE FUNCTION, which converts data from its internal form to a form suitable for display. Scripting name: ExtTypeOutput
By Value	Indicates that operators and functions which use this data type should be passed an argument by value rather than by reference. Scripting name: ExtTypePassedByValue
Receive function	Name of a function, created by CREATE FUNCTION, which converts data of this type from a form suitable for transmission from another machine to internal form. Scripting name: ExtTypeReceive
Send function	Name of a function, created by CREATE FUNCTION, which converts data of this type into a form suitable for transmission to another machine. Scripting name: ExtTypeSend

Domain (Composite Type)

The following extended attributes are available on the PostgreSQL tab:

Name	Description
Definition	The composite type is specified by a list of attribute names and data types. This is essentially the same as the row type of a table, but using CREATE TYPE avoids the need to create an actual table when all that is wanted is to define a type. A stand-alone composite type is useful as the argument or return type of a function. Scripting name: CompositeDefinition

Groups

The following extended attributes are available on the PostgreSQL tab (v8 and higher):

Name	Description
Group identifier (id)	The SYSID clause can be used to choose the PostgreSQL group ID of the new group. This is normally not necessary, but may be useful if you need to recreate a group referenced in the permissions of some object. Scripting name: SysId

Procedures

The following extended attributes are available on the PostgreSQL tab:

Name	Description
Language	The name of the language that the function is implemented in. May be SQL, C, internal, or the name of a user-defined procedural language. (See also extended attribute type ProcLanguageList.) Scripting name: ProcLanguage

References

The following extended attributes are available on the PostgreSQL tab (v8 and higher):

Name	Description
Deferrable	Controls whether the constraint can be deferred. A constraint that is not deferrable will be checked immediately after every command. Checking of constraints that are deferrable may be postponed until the end of the transaction. Only foreign key constraints currently accept this clause. All other constraint types are not deferrable. Scripting name: Deferrable
Foreign key constraint	If a constraint is deferrable, this clause specifies the default time to check the constraint. False means the constraint is INITIALLY IMMEDIATE, it is checked after each statement. This is the default. True means the constraint is INITIALLY DEFERRED, it is checked only at the end of the transaction. Scripting name: ForeignKeyConstraintDeferred

Tables

The following extended attributes are available on the PostgreSQL tab (v8 and higher):

Name	Description
Temporary state	If specified, the table is created as a temporary table. Temporary tables are automatically dropped at the end of a session, or optionally at the end of the current transaction. Scripting name: Temporary

Tablespaces

The following extended attributes are available on the PostgreSQL tab (v8 and higher):

Name	Description
Location	Specifies the directory that will be used for the tablespace. The directory must be specified by an absolute path name. Scripting name: TbspLocation
Owner	Specifies the name of the user who will own the tablespace. If omitted, defaults to the user executing the command. Only superusers may create tablespaces, but they can assign ownership of tablespaces to non-superusers. Scripting name: TbspOwner

Users

The following extended attributes are available on the PostgreSQL tab (v8 and higher):

Name	Description
Create database	Defines a user's ability to create databases. If TRUE, the user is allowed to create databases. Scripting name: CreateDB
Create user	If TRUE, the user is allowed to create new users. This option also turns the user into a superuser who can override all access restrictions. Scripting name: CreateUser
Encrypted password	Controls whether the password is stored encrypted in the system catalogs. Scripting name: EncryptedPassword
User identifier (id)	The SYSID clause can be used to choose the PostgreSQL user ID of the new user. This is normally not necessary, but may be useful if you need to recreate the owner of an orphaned object. Scripting name: SysId
Validity	This clause sets an absolute time after which the user's password is no longer valid. If this clause is omitted the password will be valid for all time. Scripting name: Validity

Red Brick Warehouse

This section describes features specific to the Red Brick Warehouse family of databases.

Red Brick Warehouse extended attributes

The following extended attributes are defined by default in the Red Brick Warehouse DBMS.

Columns

The following extended attributes are available on the Red Brick tab:

Name	Description
Unique	Declares that duplicate values are not allowed in the column. Declaring a column UNIQUE does not enforce uniqueness on the column; to enforce uniqueness, you must also build a BTREE index on the column. Scripting name: IsUnique

Procedures

The following extended attributes are available on the Red Brick tab:

Name	Description
Macro Type	A macro can be temporary, public, or private: see MacroTypeList for details. If neither temporary nor public is specified, a private macro is created by default. Scripting name: MacroType

Sybase AS Anywhere

This section describes features specific to the Sybase AS Anywhere family of databases

Deprecated versions
The DBMSs for Sybase AS Anywhere v7 and v8 are deprecated.

Note that from version 10, AS Anywhere is known as Sybase SQL Anywhere (see "Sybase SQL Anywhere" on page 674).

Auto-increment columns

Auto-increment columns are equivalent to identity columns in those DBMS that support identity columns.

If you switch from Sybase ASA to a DBMS that supports identity columns, the Identity checkbox will be selected for each auto-increment column. On the other hand, if you switch to Sybase ASA, identity columns will be assigned the autoincrement default value.

When you reverse engineer a script containing identity columns (using Sybase ASE-compatible syntax), these are automatically converted into auto-increment columns in Sybase ASA.

Working with proxy tables in Sybase ASA and ASE

A proxy table is used to access data in a remote table, it has all the attributes of the remote table, but it does not contain any data locally.

PowerDesigner can generate the script for a proxy table in order to run it in a Sybase ASA or ASE database. To do so, you need to attach the ProxyTables extended model definition to your model and then design proxy tables using **external shortcuts** or **replications**.

After designing the proxy tables, you can use the **build data source** feature that will create a data source for each target model of the current model. Target models are models containing the target tables of the replica or external shortcuts, they are also called **remote servers.**

Once the data sources are properly defined, you can use the **extended generation** feature to generate the proxy table and remote server creation scripts.

Understanding the ProxyTables extended model definition

For an external shortcut or replica to be used as a proxy table you need to attach the ProxyTable extended model definition to the model.

❖ To attach the ProxyTables extended model definition to an existing model

1. Select Model ➤ Extended Model Definitions.
2. Click the Import an Extended Model Definition tool.

   A selection dialog box is displayed.
3. Select the ASA Proxy Tables or ASE Proxy Tables extended model definition and click OK.

   The extended model definition is displayed in the list.
4. Click OK.

> **Select ProxyTables XEM at model creation**
> You can also select the relevant Proxy Tables XEM in the Extended Model Definitions tab of the New dialog box

The ProxyTables extended model definition contains generation templates, extended attributes, custom checks and custom methods to support the definition of external proxy tables.

You can double-click the ProxyTables extended model definition in the Browser to display its properties in the resource editor. The following extensions must be defined in the Profile category to fully support proxy tables:

- BasePackage:
  - Generation template - for generating proxy tables.
- DataSource:
  - Connection information custom check - verifies that the connection information is sufficient to connect to the database. You must specify the data source name, user login and password in the Database Connection tab of the data source property sheet.
  - GenerateAsProxyServer extended attribute - when set to true, defines the data source model as the proxy remote server.
  - [various templates] - used for proxy table generation.
- Model:
  - Proxy Servers and Tables generated files - to generate proxy server and table script files.

- Menu – provides a contextual menu for building data sources and commands in the Tools menu for rebuilding data sources and generating proxy tables.
- BuildProxyTableDataSourcesand GenerateProxyTables methods - used in the menu.
- [various templates] - required for proxy server and proxy table script generation.

♦ Shortcut:

- Data source existence custom check - verifies that data sources are defined for the shortcuts.

♦ Table:

- Proxy table is child of reference custom check - verifies that the model replica is not the child of another table via a reference link.
- [various templates] - required for proxy table, remote server and access definition creation syntax.

Creating a proxy table

You use external shortcuts and/or replica to design proxy tables in your model.

An external shortcut is a non-modifiable reference to an object in another model. For more information on shortcuts, see “Shortcuts”in the Shortcuts and Object Replications chapter of the *Core Features Guide* .

A replica is an exact copy of an object that can be updated when the original object is modified. For more information on replications, see “Replications”in the Shortcuts and Object Replications chapter of the *Core Features Guide* .

One interesting aspect of using replica, is that you can modify the replica code in order to make it different from the target table. A custom check verifies that replica are not used as child tables of a reference.

❖ To create a proxy table in a PDM

1. Select a table in a target model and drag it to the model where you want to create proxy tables using the appropriate key combination to create either an external shortcut or a replica.

2. Repeat this operation for each proxy table.

Defining the remote server of a proxy table

The remote server is the model containing the target tables of the external shortcut or replica. The remote server is defined using a data source in the proxy tables model; this data source provides access to the remote data on the server.

Note: the same data source can contain information for several models that share the same remote servers.

When you attach the ProxyTables extended model definition to the model containing proxy tables, a specific contextual menu is displayed on the physical data model item, this menu command is used for building the data source of the proxy tables.

Note: This command does not build data sources for target models that are closed in the workspace.

❖ To define the remote server of a proxy table

1. Create a new data source and set the GenerateAsProxyServer extended attribute to True.
2. Add the target models in the Models tab of the data source property sheet.

   You can also right-click the model that contains replica and/or shortcuts in the Browser and select the Build Proxy Tables Data Sources command. A data source is automatically created for each target model.
3. Double-click a data source in the Browser to display its property sheet.
4. Click the Database Connection tab, and define the data source name, login and password.
5. Click OK.
6. Repeat steps 2 to 5 for each data source.

Generating the remote server and proxy tables creation scripts

You can generate the remote server and proxy tables creation scripts in order to run them in the database. Generation must be started from the model containing proxy tables.

The ProxyTables extended model definition contains the creation script syntax for ASA or ASE.

❖ To generate the remote server and proxy tables creation scripts

1. Select Tools ➤ Proxy Tables ➤ Generate Proxy Tables to open the Generation dialog box, and click the Options tab.
2. Set a value for the UserReplica and UserShorcut options to allow you to generate the proxy tables corresponding to replica and/or external shortcuts.
3. Set the Generate proxy servers option to one of the following values:
   - True – to generate proxy servers. You can deselect any proxy servers you do not want to generate.
   - False – to not generate proxy servers
4. Click OK to begin generation.

   The generated script is displayed in the Result dialog box.
5. [optional] Double-click the generated SQL file or click the Edit button to open the script in a text editor.
6. Run the script on your database in order to create the proxy tables.

Sybase AS Anywhere extended attributes

The following extended attributes are defined by default in the Sybase AS Anywhere DBMS.

Table

The following extended attributes are available on the Sybase tab:

Name	Description
Global temporary table	Defines if table is a global temporary table or not. Scripting name: ExtGlobalTemporaryTable

Web Services

The following extended attributes are available on the Sybase tab (v9 and higher):

Name	Description
Port number	Specifies the web service port number. Scripting name: PortNumber
Server name	Specifies the web service server name. Scripting name: ServerName
Name prefix (dish services only)	Specifies the web service name prefix (dish services only). Scripting name: Prefix

Web Operations

The following extended attributes are available on the Sybase tab (v9 and higher) when the service type is not dish:

Name	Description
URL	Determines whether URI paths are accepted and, if so, how they are processed. Scripting name: Url

Sybase AS Enterprise

This section describes features specific to the Sybase AS Enterprise family of databases.

Deprecated versions
The DBMSs for Sybase AS Enterprise v11.0 and v11.5-11.9 are deprecated.

Proxy tables

For information about using proxy tables, see "Working with proxy tables in Sybase ASA and ASE" on page 649.

Encryption Keys

Encryption keys are supported for ASE v12.5.3a and higher. PowerDesigner models encryption keys as extended objects with a stereotype of <<EncryptionKey>>.

Adaptive Server authentication and access control mechanisms ensure that only properly identified and authorized users can access data. You can encrypt data at the column level, thus restricting your security measures to only sensitive data, and minimizing processing overhead.

Encrypting columns in Adaptive Server is more straightforward than using encryption in the middle tier, or in the client application. You use SQL statements to create the encryption keys and specify columns for encryption. Adaptive Server handles key generation and storage. Encryption and decryption of data occurs automatically and transparently as you write and read the data in encrypted columns. No application changes are required, and there is no need to purchase third-party software.

Creating an encryption key

You can create an encryption key in any of the following ways:

- Select Model ➤ Encryption Keys to access the List of Encryption Keys, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤ Encryption Key.

Encryption key properties

You can modify an object's properties from its property sheet. To open an encryption key property sheet, double-click its Browser entry in the Encryption Keys folder.

The following extended attributes are available on the Sybase tab:

Name	Description
Owner	[v12.5.3a and higher] Specifies the owner of the encryption key. Scripting name: Owner
Key length	[v12.5.3a and higher] Specifies the size in bits of the key to be created. Valid key lengths for AES are 128, 192 and 256 bits. Scripting name: KeyLength
Algorithm	[v12.5.3a and higher] Specifies the algorithm used to generate the encryption key. Currently, Advanced Encryption Standard (AES) is the only algorithm supported. Scripting name: Algorithm
Initialization vector	[v12.5.3a and higher] Controls the use of an initialization vector when encrypting. When an initialization vector is used by the encryption algorithm, the ciphertext of two identical pieces of plaintext will be different, which would prevent the cryptanalyst from detecting patterns of data but would render the data on disk useless for indexing or matching without decryption. Scripting name: InitVector
Padding of datatypes	[v12.5.3a and higher] Specifies the use of padding of datatypes whose length is less than one block. Padding can be used instead of an initialization vector to randomize the ciphertext. It is only suitable for columns whose plaintext length is less than half the block length. For the default AES algorithm the block length is 16 bytes. Scripting name: Pad
Password phrase	[v15.0.2 and higher] Specifies a default key for use on all encrypted columns which do not have a keyname specified in create table or alter table. This is a database specific default key for use with tables in the same database. The default key is stored in the database sysencryptkeys table, the same as non-default keys. Scripting name: PasswordPhrase

Name	Description
Default encryption key	[v12.5.3a and higher] Allows the System Security Officer to create a default key for use on all encrypted columns which do not have a keyname specified in create table or alter table. This is a database specific default key for use with tables in the same database. The default key is stored in the database sysencryptkeys table, the same as non-default keys. Scripting name: Default

Sybase AS Enterprise extended attributes

The following extended attributes are defined by default in the Sybase AS Enterprise DBMS.

Columns

The following extended attributes are available on the Sybase tab:

Name	Description
Store Java-SQL column in row	[v12.0 and higher] Specifies whether a Java-SQL column is stored separate from the row (set to False) or in storage allocated directly in the row (set to True). Scripting name: InRow
Computed column is materialized	Specifies that the computed column is materialized. Scripting name: Materialized
Encrypted	[v12.5.3a and higher] Specifies that the column is encrypted. Scripting name: Encrypted
Encryption Key	[v12.5.3a and higher] Specifies an encryption key name. Scripting name: EncryptionKey

Keys

The following extended attributes are available on the Sybase tab:

Name	Description
Key index is descending	Specifies if the index created for a constraint is to be created in descending order for each column. Scripting name: DescKey

Model

The following extended attributes are available on the Encryption tab (v12.5.3a and higher):

Name	Description
Encryption password	Global encryption password. Scripting name: EncryptionPassword

Web Services

The following extended attributes are available on the Sybase tab (v15.0 and higher):

Name	Description
Port number	Specifies the web service port number. Scripting name: PortNumber
Server name	Specifies the web service server name. Scripting name: ServerName
Database name	Specifies the database name used in the URL to access the web service. Scripting name: DatabaseName

Web Operations

The following extended attributes are available on the Sybase tab (v15.0 and higher):

Name	Description
Alias	Specifies the name of the user-defined database alias. Scripting name: Alias
Secure	Security option. clear indicates that HTTP is used to access this Web service. ssl indicates HTTPS is used to access this Web service Scripting name: Secure

Sybase AS IQ

This section describes features specific to the Sybase AS IQ family of databases.

Deprecated versions
The DBMSs for Sybase AS IQ v12.0 and v12.4.3 are deprecated.

Sybase Adaptive Server IQ (AS IQ) is a high-performance decision support server designed specifically for data warehousing.

Sybase AS IQ provides benefits that allow an interactive approach to decision support including:

- Intelligent query processing using index-only access plans to process only the data needed to satisfy any type of query
- Truly interactive, ad hoc query performance on a uniprocessor as well as on parallel systems. An ad hoc query is a query about which the system has no prior knowledge and for which no explicit tuning is required. Ad hoc queries are distinguished from standard or production reports, where only pre-defined variables, such as dates, are used to generate pre-defined reports on a regular basis.
- Fully-flexible schema support
- Efficient query execution without query-specific tuning by the System Administrator (under most circumstances)
- Fast aggregations, counts, comparisons of data
- Stored procedures
- Entire database and indexing stored in less space than raw data
- Numerous index types
- Join indexes to improve overall query performance. **Join indexes** represent links between columns in different tables. They enable much faster query processing on multiple indexes in a very large database, and are usually applied instead of ad hoc queries.

IQ indexes

Before creating IQ indexes, you should consider the implications of various types of indexes on the database server memory and disk space.

The set of indexes you define for any given column can have dramatic impact on the speed of query processing. There are four main criteria for choosing indexes:

- Number of unique values
- Types of queries
- Disk space usage
- Data types

It is best to consider all criteria in combination, rather than individually. To decide on indexes, look closely at the data in each column.

Try to anticipate the number of unique and total values, the query results users will want from it, and whether the data will be used in ad hoc joins or join indexes.

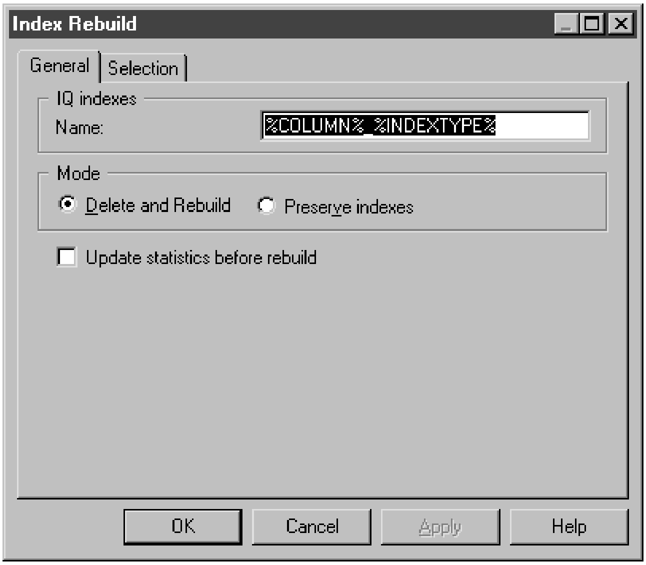
☞ For more information about choosing index types, see chapter Adaptive Server IQ Indexes in the Adaptive Server IQ Administration and Performance Guide.

Rebuilding IQ indexes

As you develop a PDM or modify an existing one, you may change data types, alter the percentage of distinct values or change the number of values in tables. You must then rebuild the IQ indexes to reflect these changes.

❖ To rebuild IQ indexes

1. Select Tools ➤ Rebuild Objects ➤ Rebuild Indexes to open the Rebuild Indexes dialog box:



Name: %COLUMN%_%INDEXTYPE%

● Delete and Rebuild ○ Preserve indexes

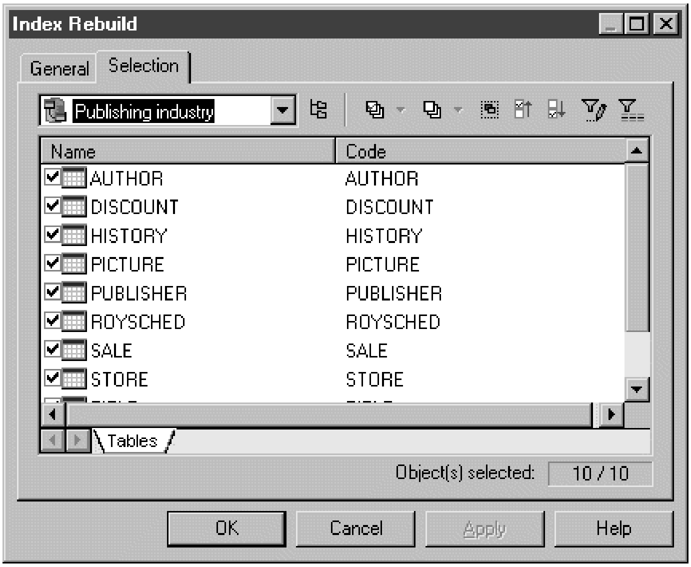
□ Update statistics before rebuild

2. Select a default name to generates IQ indexes. You can use the following variables:
   - %COLUMN% - *Column name*
   - %INDEXTYPE% - Type of index to be rebuilt
   - %TABLE% - Name or code of table (based on display preferences)

3. Specify a mode to use. You can choose between:
   - Delete and Rebuild - All existing indexes are deleted before index rebuild
   - Preserve Indexes - Preserves all existing indexes

4. [optional] Select the Update statistics before rebuild checkbox to update such statistics as the number of records in a table and the number of distinct values in a column before performing the rebuild. Selecting this option can help with optimizing the rebuild.

5. [optional] Click the Selection tab and select or clear checkboxes to specify for which tables you want to rebuild indexes.



Publishing industry

Name	Code
☑ AUTHOR	AUTHOR
☑ DISCOUNT	DISCOUNT
☑ HISTORY	HISTORY
☑ PICTURE	PICTURE
☑ PUBLISHER	PUBLISHER
☑ ROYSCHED	ROYSCHED
☑ SALE	SALE
☑ STORE	STORE

Object(s) selected: 10 / 10

6. Click OK, and then Yes to confirm the rebuilding of your indexes.

Index types

When you rebuild indexes, PowerDesigner determines the index type based on information contained from the table statistics. It uses the number field, which indicates the estimated number of records per table, and the percentage of distinct values to compute the number of unique values. If the user has not specified a number of rows for the table, PD assumes that the table will include at least 1 row of data.

Usually the rebuild process creates a FASTPROJECTION index for all columns. Otherwise, the following rules are applied:

Criteria	Index type
If no statistics are provided and the column has an undefined data type	No index is created
Low number of unique values in a column Column used in join predicate	LOWFAST
High number of unique values in a column No COUNT DISTINCT, SELECT DISTINCT, or GROUP BY queries required	HIGHNON-GROUP

Criteria	Index type
Column used in join predicate High number of unique values in a column (more that 1000) Anticipate COUNT DISTINCT, SELECT DISTINCT, or GROUP BY queries Column must enforce uniqueness	HIGHGROUP
Column without numeric datatype	No index is created
Column with date type	DATE
Column with time type	TIME
Column with datetime or smalldatetime type	DTTM

For example (IQ v12.5, Table A contains 1500 rows

Column	% Distinct values	Unique values	Rebuild indexes generates
Col_1 integer	100	1500	HG index
Col_2 integer	50	750	LF index
Col_3 integer	0	0	no index
Col_4 char (10)	100	1500	no index
Col_5 char (10)	50	750	LF index

IQ join indexes

A join index is a special type of index used in Sybase AS IQ, which represents a full outer join of two or more tables. The query engine may use this full outer join as a starting point for queries that include left outer, right outer, and inner joins. A full outer join is one where all rows from both the left and right specified tables are included in the result, with NULL returned for any column with no matching value in the corresponding column.

Join indexes are defined from references. You can create a join index for any set of columns that your users commonly join to resolve queries.

While some references are based on keys, Sybase AS IQ allows you to create user-defined references to include the exact join required by your foreseen queries.

You can define a join index:

- Automatically, by creating join indexes from all references linked to a fact table
- Manually, by creating a join index in the list of join indexes

Facts and Dimensions

A fact corresponds to the focus of a decision support investigation. For example, Sales, Revenue, and Budget are facts. A table containing facts is set as a fact table in the table property sheet.

A dimension defines the axis of the investigation of a fact. For example, Product, Region, and Time are the axes of investigation of the Sales fact. A table containing dimensions is linked to a fact table and is set as a dimension table in the table property sheet.

☞ For more information about setting table properties, see the Building Physical Diagrams chapter.

Creating a join index from the table property sheet

You can create a join index or a bitmap join index from the Join Index tab of the table property sheet. The join index is automatically based on the current table.

❖ To create a join index from the table property sheet

1. Open the property sheet of a table and click the Join Index tab.
2. Click the Insert a Row or the Add a Row tool to add a join index.
3. [optional] Click the Properties tool to open the join index property sheet and check that the Base table box is initialized with the selected table.

Creating a join index in the list of join indexes

If you need to create more join indexes than those automatically created by the rebuild process, you can use the list of join indexes.

Several references can be made at once
The following procedure shows how to create a join index in its simplest form: using one reference at a time. Once you become familiar with join index creation, you can select several references at once to reflect a more complex query plan.

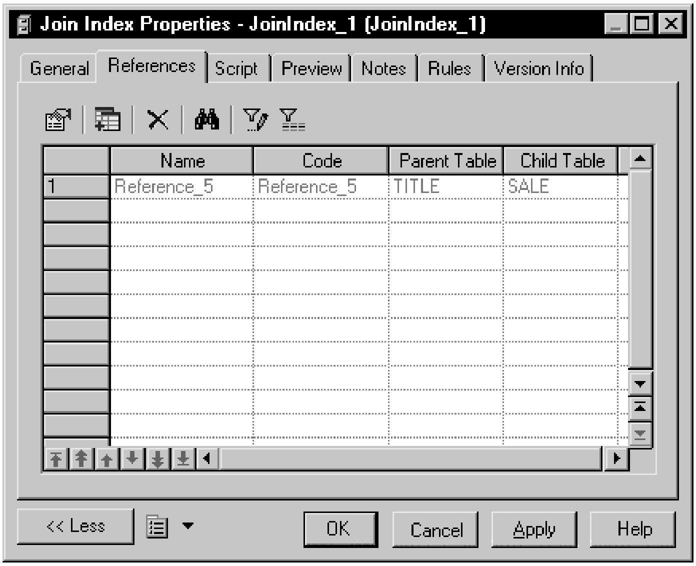
❖ To create a join index in the list of join indexes

1. Select Model ➤ Join Indexes to open the List of Join Indexes.

2. Click the Add a Row tool, type a name in the Name column, and then click the Properties tool to open the property sheet of the new join index.

3. Click the References tab to display the list of references of the join index. The list is empty when you create a new join index.

4. Click the Add References tool to open a selection box listing all the available references in the PDM, select a reference, and then click OK to add it to the Join Index references list:



	Name	Code	Parent Table	Child Table
1	Reference_5	Reference_5	TITLE	SALE

5. Click OK in each of the dialog boxes to complete the creation of the join index.

Join index properties

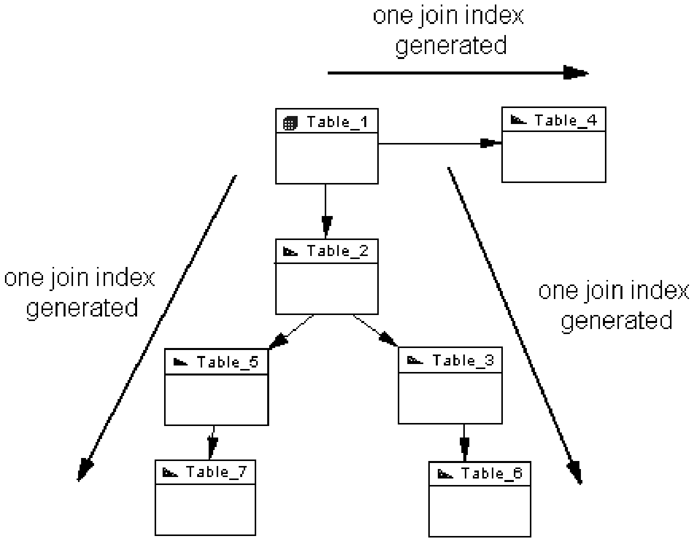
A join index has the following properties:

Property	Description
Name	The name of the item which should be clear and meaningful, and should convey the item's purpose to non-technical users.
Code	The technical name of the item used for generating code or scripts, which may be abbreviated, and should not generally include spaces.
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined.
Comment	Descriptive label for the join index.
Base table	Name of the table that stores the join index. You can use the tools to the right of the list to create an object, browse the complete tree of available objects or view the properties of the currently selected object.

Rebuilding join indexes for all references linked to a fact table

You can automatically generate a join index for each selected fact table and the dimension tables that it references. The rebuild process only applies to fact tables selected in the Selection tab of the Join Index Rebuild tab.

Each automatically generated join index consists of the references that link a fact table to all the dimension tables located on a single axis proceeding from the fact table.



A reference between two fact tables does not generate any join index.

A join index is constrained and can only be defined for tables that are organized in a connected tree.

Parameter	Result of selection
Delete and Rebuild	When selected all existing indexes are deleted before join index rebuild.
Preserve	When selected, preserves all existing join indexes in the PDM.

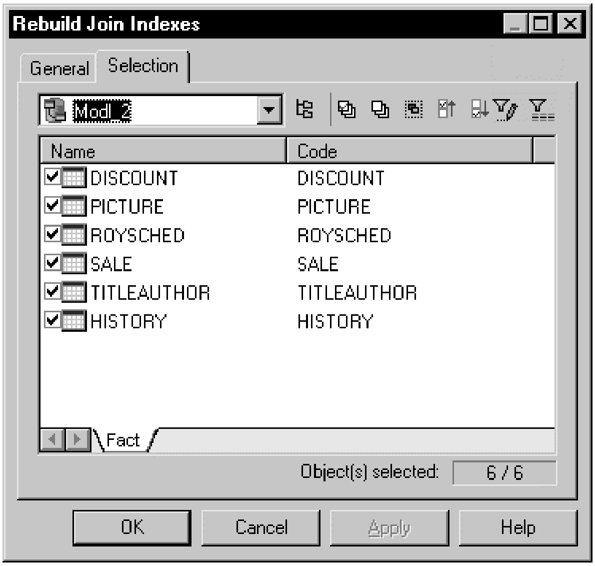
The same feature applies to Oracle bitmap join indexes.

☞ For more information on Oracle bitmap join indexes, see section "Automatically creating bitmap join indexes through rebuilding" on page 621.

❖ To rebuild join indexes for all references linked to a fact table

1. Select Tools ➤ Rebuild Objects ➤ Rebuild Join Indexes to open the Rebuild Join Indexes dialog:

2. Click the Selection tab, and select one or more fact tables from the list:



Rebuild Join Indexes

Mod_2

Name	Code
☑ DISCOUNT	DISCOUNT
☑ PICTURE	PICTURE
☑ ROYSCHED	ROYSCHED
☑ SALE	SALE
☑ TITLEAUTHOR	TITLEAUTHOR
☑ HISTORY	HISTORY

Object(s) selected: 6 / 6

3. Click OK, and then Yes to confirm rebuilding of the join indexes.

   A join index is automatically generated for each fact table.

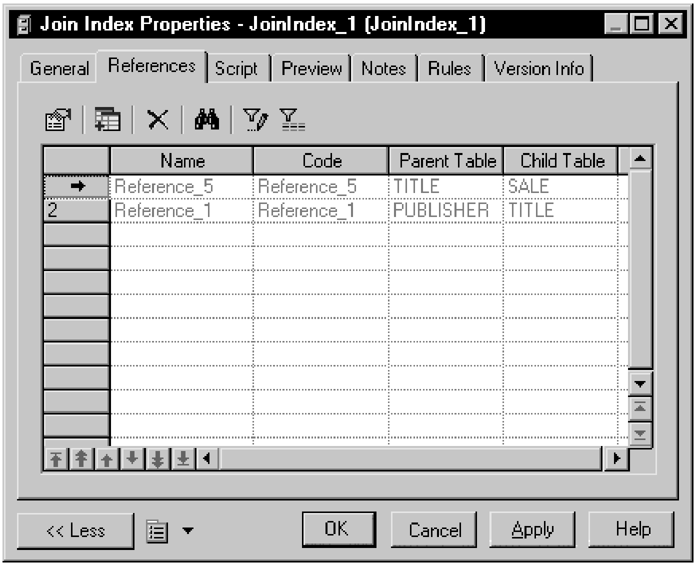
Displaying automatically generated join indexes
Automatically generated join indexes appear in the list of join indexes. To display the list, select Model ➤ Join Indexes.

Adding references to a join index

You can add a reference to any join index. You do this, for example, when you create a new reference that you want to include in an existing join index.

❖ To add a reference to a join index

1. Select Model ➤ Join Indexes to open the List of Join Indexes.
2. Select a join index in the list, and then click the Properties tool to open its property sheet.
3. Select the References tab, and click the Add References tool to open a selection box listing all the available references in the PDM. Select the appropriate reference in the list and click OK to add it to the join index:



	Name	Code	Parent Table	Child Table
→	Reference_5	Reference_5	TITLE	SALE
2	Reference_1	Reference_1	PUBLISHER	TITLE

4. Click OK in each of the dialog boxes to complete the addition of the reference to the join index.

Generating IQ data movement scripts

PowerDesigner provides the capability to generate data movement scripts to populate your AS IQ data warehouse from your other databases. The script can be used to:

- Generate a flat file for loading to the AS IQ data warehouse

- Create Insert Location statements for use with a proxy data base (for ASE and ASA only)

To create a data movement script, you must:

- Add the Data Movement IQ XEM to your AS IQ model

- Specify your data movement options

- [optional] Create a data source linked to a model of the database from which you want to draw the data to be moved

- [optional] Specify mappings between the tables in your data source and your AS IQ database

- Generate the data movement script

❖ To add the Data Movement IQ XEM to your AS IQ model

1. Select Model ➤ Extended Model Definitions to open the List of Extended Model Definitions.

2. Click the Import an Extended Model Definition tool, select the Data Movement IQ and click OK to add this XEM to the model.

3. Click OK to close the List of Extended Model Definitions and return to the model.

❖ To create a data source to populate your IQ data warehouse

1. Create a PDM to model your source database, and ensure that it is open in your workspace.

2. In your AS IQ PDM, right-click the model name in the Browser and select New ➤ Data Source.

3. Enter a name for the source and then click the Models tab.

4. Click the Add Models tool, and select your source model.

5. Click the Database Connection tab, and complete the fields to enable a connection to your source database.

6. Complete the fields on the Data Movement tab and click OK.

Data Source properties Data Movement tab

The following fields are available on the Data Source properties sheet Data Movement tab:

Property	Description
Remote Server Name	Specifies the name of the remote server used in the interface file for IQ server.
Remote Database Name	Specifies the name of the remote database.
Data Source Name	Specifies the label given to the data source in the sql.ini file.
Dump file directory	Specifies the directory where the 'dump' file (external flat file), that contains the data to be imported, will be created.
Local user name	Specifies the database user name.

❖ To specify data movement options

1. Right-click the model item in the Browser and select Properties from the contextual menu.
2. Click the Data Movement tab and enter the appropriate values for the model as a whole.
3. [optional] To override these global data movement options for a specific table, open its property sheet and enter table-specific values on the Data Movement tab. This tab also allows you to specify a table-specific dump file for importing into the table

Model properties Data Movement tab

The following fields are available on the Model properties sheet Data Movement tab:

Property	Description
Field Delimiter	Specifies the delimiter to be used between fields in the dump file.
Row Delimiter	Specifies the delimiter to be used between rows in the dump file.
Maximum Image or Text Size	Specifies the maximum length of an image (or text) record, to which it will be truncated if necessary.
Load file directory	Specifies the directory where the load file is located.

Table properties Data Movement tab

If the Data Movement Method generation option is set to Insert Location, a Data Movement tab is available on each Table properties sheet, containing the following fields:

Property	Description
Dump file name	Specifies the name of the ‘dump’ file (external flat file) that contains the data to be imported.
Field Delimiter	Specifies the delimiter to be used between fields in the dump file.
Row Delimiter	Specifies the delimiter to be used between rows in the dump file.
Maximum Image or Text Size	Specifies the maximum length of an image (or text) record, to which it will be truncated if necessary.

❖ To specify mappings between the tables in your data source and your AS IQ database

1. Select Tools ➤ Mapping Editor to open the Mapping Editor.
2. Create the necessary mappings and then click OK. For detailed information about using the Mapping Editor, see the Creating Mappings chapter in the *Core Features Guide* .

❖ To generate the data movement script

1. Select Tools ➤ Extended Generation to open the Generation window.
2. Specify a directory in which to generate your data movement files.
3. [optional] Click the Selection tab and specify for which Tables and/or Data Sources you want to generate a data movement script.
4. Click the Options tab and specify your data movement script generation options. You can set the following options:
   - ♦ Use Mappings – specifies whether any previously-created mappings should be used for the data movement
   - ♦ Data Movement Method – specifies which kind of script to generate. You can choose between:
     - Insert Location – [IQ or ASE only] PowerDesigner will create a loadscript for connecting the source database to the IQ server. Note that if the data source is not an IQ or ASE database, then no loadscript will be generated.

- External File – PowerDesigner will create a dump file from the source database together with a loadscript to upload it to the IQ server.

5. [optional] Click the Generated Files tab to review the names and locations of the files to be generated.

6. Click OK to begin the generation of the data movement script.

Sybase AS IQ events

PowerDesigner supports Sybase AS IQ events for v12.7 and higher. For details, see "SQL Anywhere events" on page 674.

Sybase AS IQ extended attributes

The following extended attributes are defined by default in the Sybase AS IQ DBMS.

Columns

The following extended attributes are available on the Sybase tab (v12.4.3 and higher):

Name	Description
Number of distinct value (Iq unique)	Defines the cardinality of the column (to optimize the indexes internally). Scripting name: ExtIqUnicity

Tables

The following extended attributes are available on the Sybase tab (v12.4.3 and higher):

Name	Description
Global temporary table	Specifies that the table is a global temporary table. Scripting name: ExtGlobalTemporaryTable

Web Services

The following extended attributes are available on the Sybase tab (v12.6 and higher):

Name	Description
Port number	Specifies the web service port number. Scripting name: PortNumber
Server name	Specifies the web service server name. Scripting name: ServerName
Name prefix	[DISH service type] Specifies a name prefix. Only SOAP services whose names begin with this prefix are handled. Scripting name: Prefix

Web Operations

The following extended attributes are available on the Sybase tab (v12.6 and higher) when the service type is not dish:

Name	Description
URL	Determines whether URI paths are accepted and, if so, how they are processed. Scripting name: Url

Sybase SQL Anywhere

This section describes features specific to the Sybase SQL Anywhere family of databases.

Note that Sybase SQL Anywhere is the new name for Sybase AS Anywhere (see "Sybase AS Anywhere" on page 649).

SQL Anywhere events

This section also applies to Sybase AS IQ events.

Events allow you to automate and schedule actions. PowerDesigner models events as extended objects with a stereotype of <<Event>>.

You can create an event in any of the following ways:

- Select Model ➤ Events to access the List of Events, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤ Event.

You can modify an object's properties from its property sheet. To open an event property sheet, double-click its diagram symbol or its Browser entry in the Events folder.

The following extended attributes are available on the Sybase tab:

Name	Description
Event is scheduled	Specifies that the server carries out a set of actions according to a schedule of times. If selected, this option disables the "Event is triggered" option. Scripting name: ScheduledEvent
Schedule definition	Enter the schedule of event trigger times here. Click the New button to launch a dedicated editor window. Scripting name: SchedulesText
Event is triggered	Specifies that the server carries out a set of actions when a predefined type of system event occurs. This option is the default and, if selected, disables the "Event is scheduled" option. Scripting name: TypedEvent

Name	Description
Event type	The event-type is one of the listed set of system-defined event types. The event types are case insensitive. To specify the conditions under which this event-type triggers the event, use the WHERE clause. Scripting name: EventType
Trigger condition	Determines the condition under which an event is fired. For example, to take an action when the disk containing the transaction log becomes more than 80% full, use the following triggering condition: WHERE event_condition('LogDiskSpacePercentFree') < 20 The argument to the event_condition function must be valid for the event type. You can use multiple AND conditions to make up the WHERE clause, but you cannot use OR conditions or other conditions. Scripting name: TriggerCondition
Handler	Each event has one handler. The actions of an event handler are committed if no error is detected during execution, and rolled back if errors are detected. Scripting name: Handler
Enable	By default, event handlers are enabled. When DISABLE is specified, the event handler does not execute even when the scheduled time or triggering condition occurs. A TRIGGER EVENT statement does not cause a disabled event handler to be executed. Scripting name: Enable
At (databases)	If you want to execute events at remote or consolidated databases in a SQL Remote setup, you can use this clause to restrict the databases at which the event is handled. By default, all databases execute the event. Scripting name: Database

Sybase SQL Anywhere extended attributes

The following extended attributes are defined by default in the Sybase SQL Anywhere DBMS.

Columns

The following extended attributes are available on the Sybase tab:

Name	Description
Column is compressed	Specifies whether this column is stored in a compressed format. Scripting name: Compressed

Tables

The following extended attributes are available on the Sybase tab:

Name	Description
PCTFREE	Specifies the percentage of free space you want to reserve for each table page. The free space is used if rows increase in size when the data is updated. If there is no free space in a table page, every increase in the size of a row on that page requires the row to be split across multiple table pages, causing row fragmentation and possible performance degradation. The value percent-free-space is an integer between 0 and 100. The former specifies that no free space is to be left on each page (each page is to be fully packed). A high value causes each row to be inserted into a page by itself. If PCTFREE is not set, 200 bytes are reserved in each page. Scripting name: PctFree
Temporary table	Specifies either temporary table is a global or a local temporary table. Scripting name: TemporaryTable

Indexes

The following extended attributes are available on the Sybase tab:

Name	Description
Virtual index	The VIRTUAL keyword is primarily for use by the Index Consultant. A virtual index mimics the properties of a real physical index during the evaluation of query plans by the Index Consultant and when the PLAN function is used. You can use virtual indexes together with the PLAN function to explore the performance impact of an index, without the often time consuming and resource consuming effects of creating a real index. Scripting name: Virtual

Web Services

The following extended attributes are available on the Sybase tab:

Name	Description
Port number	Specifies the web service port number. Scripting name: PortNumber
Server name	Specifies the web service server name. Scripting name: ServerName
Name prefix	[DISH service type] Specifies a name prefix. Only SOAP services whose names begin with this prefix are handled. Scripting name: Prefix

Web Operations

The following extended attributes are available on the Sybase tab when the service type is not dish:

Name	Description
URL	Determines whether URI paths are accepted and, if so, how they are processed. Scripting name: Url

Teradata

This section describes features specific to the Teradata family of databases.

Transform groups

A transform is a mechanism for creating an external representation of the UDT that is used when exporting and importing data between the client and the Teradata server. This mechanism allows most Teradata client utilities and open APIs to transparently move data to and from a UDT without the need for special logic or metadata.

Transforms usually appear as a named pair of functions or methods (usually referred to as To-SQL and From-SQL to indicate the direction of data flow to and from the database) called a transform group. A transform group is required if the type is to be used in a table.

Transform groups are supported for Teradata v2r6 and higher. PowerDesigner models transform groups as extended objects with a stereotype of <<TransformGroup>>.

Creating a transform group

You can create a transform group in any of the following ways:

- Select Model ➤ Transform Groups to access the List of Transform Groups, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select New ➤ Transform Group.

Transform group properties

You can modify an object's properties from its property sheet. To open a transform group property sheet, double-click its Browser entry in the Transform Groups folder.

The following extended attributes are available on the Teradata tab (vV2R6 and higher):

Name	Description
UDT	Specifies the name of the user-defined type associated with the transform group. Scripting name: UDT
To sql with	Specifies the function name and parameters to be used as the tosql routine for this transform group, and whether or not it is specific. Scripting names: ToName, ToParms, ToSpecific
From sql with	Specifies the method or function name and parameters to be used as the fromsql routine for this transform group, and whether or not it is specific and/or instantiable. Scripting names: FromType, FromName, FromParms, FromSpecific, FromInstance, FromUDT

Teradata database permissions

You can define multiple databases in a PDM for Teradata, and also define **permissions** on the database object.

☞ For more information on permissions, see the “Building a Database Access Structure” chapter.

Primary indexes in Teradata

In Teradata, users tend to use indexes rather than key constraints.

❖ To create a primary index

1. Open the property sheet of an index from the Indexes tab of a table, or from the List of Indexes available by selecting Model ➤ Indexes.

2. Click the Teradata tab and select the Primary Index checkbox.

3. Click OK to close the index property sheet.

   When a primary index is based on a key, it is automatically unique. You can make this primary index non-unique by detaching the index from the key. To do so, select <None> in the Columns Definition list in the Columns tab of the index property sheet, and set the PrimaryIndex extended attribute of the index to True.

   Once defined, you can decide to generate indexes or keys in the SQL script, and you can also decide to generate them inside or outside the table creation script.

Teradata extended attributes

The following extended attributes are defined by default in the Teradata DBMS.

Abstract data types

The following extended attributes are available on the Teradata tab if the type is distinct (vV2R6 and higher):

Name	Description
Predefined data type	Indicates that character column comparison uses character case (upper and lower) to raise differences. Scripting name: PredefinedDataType

Abstract data type procedures

The following extended attributes are available on the Teradata tab if the type is distinct (vV2R6 and higher):

Name	Description
Return data type	Specifies the name of the data type returned by the method, which can be either a predefined data type or a UDT. Scripting name: ReturnDataType
Self as result	Specifies that the method is type-preserving. If so, then the data type specified in the RETURNS clause for the method must have the same name as UDT_name. Scripting name: SelfAsResult
As locator	Specifies that BLOB and CLOB types must be represented by a locator. The Teradata Database does not support in-memory LOB parameters: an AS LOCATOR phrase must be specified for each LOB parameter and return value. Scripting name: ReturnAsLocator
Character set	Specifies the CHARACTER SET clause for character data type. Scripting name: ReturnCharSet
Cast data type	Specifies a computed attribute that show the datatype and its length and precision. Scripting name: CastDataTypeDisplay
As locator	Specifies that BLOB and CLOB types must be represented by a locator. Scripting name: CastAsLocator

Name	Description
Specific method name	Specifies the specific name of the method whose signature is being added to the type definition for UDT_name. Scripting name: SpecificMethodName
Parameter style	Specifies the parameter style for the method defined by this signature. Scripting name: ParameterStyle
Returns null on null input	Specifies that the method defined by this signature is not called if any of the arguments passed to it is null. Instead, it returns a null. Scripting name: ReturnsNullOnNullInput
Deterministic	Specifies that the result of invoking the method defined by this signature is deterministic. Scripting name: Deterministic
Language	Specifies the language (either C or C++) used to write the source code for the method defined by this signature. Scripting name: Language

Columns

The following extended attributes are available on the Teradata tab (vV2R6 and higher):

Name	Description
Case specific	Specifies that character column comparison is case-sensitive. Scripting name: CaseSpecific
Character set	Specifies the character set to be used. Scripting name: CharacterSet
System generated only	Specifies that identity column values are always system-generated. You cannot insert values into, nor update, an identity column defined as GENERATED ALWAYS. If not selected, identity column values are system-generated unless the user does not enter a non-null value. Scripting name: ExtGenAlways

Name	Description
Compressed values	Compresses specified values and nulls in one or more columns of a table to zero space. When the data in a column matches a value specified in the COMPRESS phrase, then that value is stored only once in the table header regardless of how many times it occurs as a field value for the column, thus saving disk storage space. Attribute must be enclosed in parenthesis when it is composed of multiple values. Scripting name: Compress

Databases

The following extended attributes are available on the Teradata tab (vV2R6 and higher):

Name	Description
Account	Specifies the account ID identifiers. Scripting name: Account
After journal	Specifies the type of image to be maintained by default for data tables created in the new database. Scripting name: AfterJournal
Default journal table	Specifies the default table that is to receive the journal images of data tables created in the new database. Scripting name: DefaultJournalTable
Fallback	Specifies whether to create and store a duplicate copy of each table created in the new database. Scripting name: Fallback
Owning database	Specifies the name of the immediate owning user or database. The default is the user name associated with the current session. Scripting name: FromDatabaseName
Journal	Specifies the number of before change images to be maintained by default for each data table created in the new database. Scripting name: Journal
Permanent space	Specifies the number of bytes to be reserved for permanent storage of the new user database. The space is taken from unallocated space in the database of the immediate owner. Scripting name: PermanentSpace

Name	Description
Spool space	Specifies the number of bytes (n) to be allocated for spool files. The default is the largest value that is not greater then the owner spool space, and that is a multiple of the number of AMPs on the system. Scripting name: SpoolSpace
Temporary space	Specifies how much space (in bytes) is to be allocated for creating temporary tables by this user. Note that temporary space is reserved prior to spool space for any user defined with this characteristic. Scripting name: TemporarySpace

Indexes

The following extended attributes are available on the Teradata tab:

Name	Description
Primary Index	Specifies that the index is the primary index. Scripting name: PrimaryIndex
All	Specifies that a NUSI should retain row ID pointers for each logical row of a join index (as opposed to only the compressed physical rows). Scripting name: AllIndex
Index has name	Specifies that the index will be generated with its name (as Teradata allows index with no name). Scripting name: NamedIndex

Tables

The following extended attributes are available on the Teradata tab (vV2R6 and higher):

Name	Description
On commit action	Specifies the action to take with the contents of a global temporary table when a transaction ends: ♦ DELETE ROWS - clears the temporary table of all rows. ♦ PRESERVE ROWS - retains the rows in the table after the transaction is committed. Scripting name: CommitRowAction

Name	Description
Type	Specifies whether the table to be created is a global temporary table or a volatile table: ♦ GLOBAL TEMPORARY - a temporary table definition is created and stored in the data dictionary for future materialization. You can create global temporary tables by copying a table WITH NO DATA, but not by copying a table WITH DATA. ♦ VOLATILE - specifies that a volatile table be created, with its definition retained in memory only for the course of the session in which it is defined. Scripting name: GlobalTemporary
Duplicate row control	Controls the treatment of duplicate rows. If there are uniqueness constraints on any column or set of columns in the table definition, then the table cannot have duplicate rows even if it is declared as MULTISET. Some client utilities have restrictions with respect to MULTISET tables. Scripting name: SetOrMultiset

Users

The following extended attributes are available on the Teradata tab :

Name	Description
Owner	Specifies the database (or user) that owns the current user. Scripting name: DBOwner
Permanent	Specifies the number of bytes to be reserved for permanent storage of the new user database. The space is taken from unallocated space in the database of the immediate owner. Scripting name: PermanentSpace
Spool	Specifies the number of bytes (n) to be allocated for spool files. The default is the largest value that is not greater then the owner spool space, and that is a multiple of the number of AMPs on the system. Scripting name: SpoolSpace
Temporary	Specifies how much space (in bytes) is to be allocated for creating temporary tables by this user. Note that temporary space is reserved prior to spool space for any user defined with this characteristic. Scripting name: TemporarySpace

Name	Description
Account	Specifies the account ID identifiers. Scripting name: Account
Fallback	Specifies whether to create and store a duplicate copy of each table created in the new database. Scripting name: Fallback
Journal	Specifies the number of before change images to be maintained by default for each data table created in the new database. Scripting name: Journal
After journal	Specifies the type of image to be maintained by default for data tables created in the new database. Scripting name: AfterJournal
Default table	Specifies the default table that is to receive the journal images of data tables created in the new database. Scripting name: DefaultJournalTable
Database	Specifies the default database name. Scripting name: DefaultDatabase
Role	Specifies the default role for the user. Scripting name: DefaultRole
Character set	Specifies the default character data type. Scripting name: DefaultCharacterSet
Collation	Specifies the default collation for this user. Scripting name: Collation
Time zone	Specifies the default time zone displacement for the user. Scripting name: TimeZone
Date format	Specifies the default format for importing and exporting DATE values for the user. Scripting name: DateForm
Profile name	Specifies a profile to the user. Scripting name: Profile
Startup string	Specifies a startup string. Scripting name: Startup

Views

The following extended attributes are available on the Teradata tab (vV2R6 and higher):

Name	Description
Lock type	Specifies the type of lock to be placed. Scripting name: LockType
Locked object class	Specifies the type (class) of the object to be locked. Scripting name: LockedClass
Locked object	Specifies the name of the object to be locked. Scripting name: LockedObjt
No wait	Specifies that if the indicated lock cannot be obtained, the statement should be aborted. Scripting name: NoWait

CHAPTER 12

Writing SQL Statements in PowerDesigner

About this appendix

This appendix provides a short introduction to writing SQL scripts in PowerDesigner and lists the variables and macros available for use in such scripts.

Contents

Introduction

PowerDesigner allows you to associate your own custom SQL statements with many database objects.

You can write triggers and procedures, and also add begin and end scripts to many database objects.

☞ For more information, see:

- ♦ The “Building Triggers and Procedures” chapter
- ♦ The “Customizing scripts” section in the “Generating a Database from a PDM” chapter

Instead of hard coding the names of tables, columns, and other objects in your SQL statements, you can use the PowerDesigner Generation Template Language (GTL) or the PDM variables and macros to obtain these values from the model.

While you can perform many tasks using the PDM variables and macros, the GTL is more powerful, as it allows you to access any information about any object in the model.

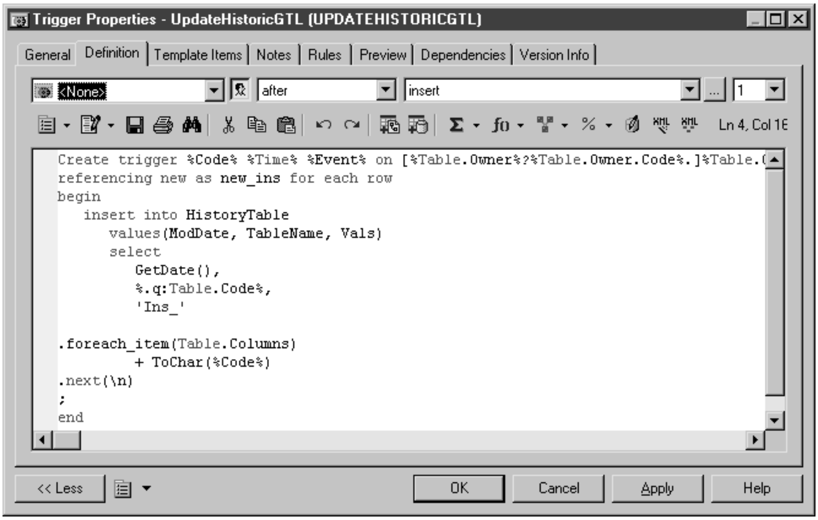
☞ For more detailed information about the GTL, see the Customizing Generation with GTL chapter of the *Customizing and Extending PowerDesigner* manual.

Writing SQL with the PowerDesigner GTL

You can use the PowerDesigner Generation Template Language (GTL) to write SQL statements.

In the following example, a trigger written using GTL is attached to the Example table, and writes the contents of any insertion to HistoryTable.

See "Writing SQL with the PDM Variables and Macros" on page 691 for an example of the same trigger written using the PowerDesigner macros and variables:

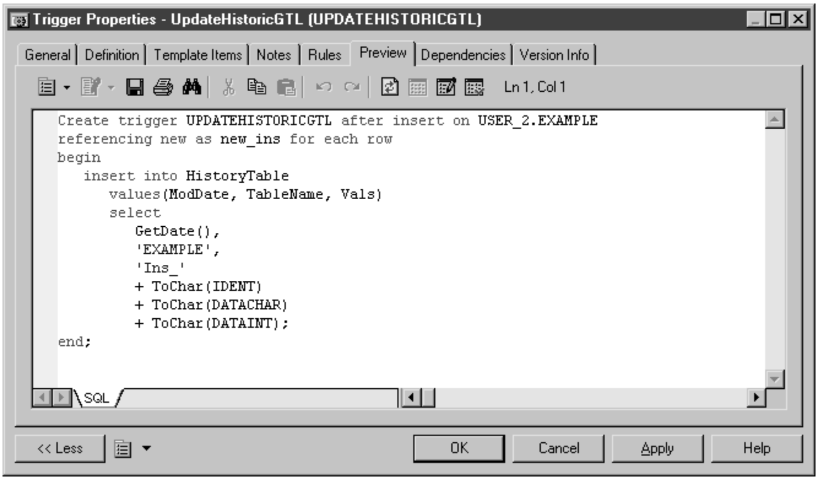


<None> after insert 1

```
Create trigger %Code% %Time% %Event% on [%Table.Owner%?%Table.Owner.Code%.]%Table.(
referencing new as new_ins for each row
begin
   insert into HistoryTable
      values(ModDate, TableName, Vals)
      select
         GetDate(),
         %.q:Table.Code%,
         'Ins_'

.foreach_item(Table.Columns)
         + ToChar(%Code%)
.next(\n)
;
end
```

The actual trigger code can be viewed on the Preview tab:



```
Create trigger UPDATEHISTORICGTL after insert on USER_2.EXAMPLE
referencing new as new_ins for each row
begin
   insert into HistoryTable
      values(ModDate, TableName, Vals)
      select
         GetDate(),
         'EXAMPLE',
         'Ins_'
         + ToChar(IDENT)
         + ToChar(DATACHAR)
         + ToChar(DATAINT);
end;
```

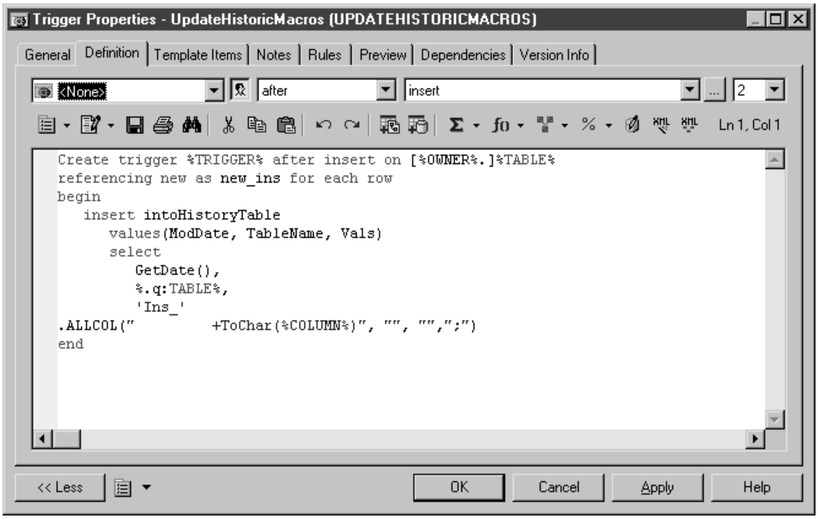
☞ For more information, see the Customizing Generation with GTL chapter of the *Customizing and Extending PowerDesigner* manual.

Writing SQL with the PDM Variables and Macros

You can use the PDM Variables and Macros to write SQL statements.

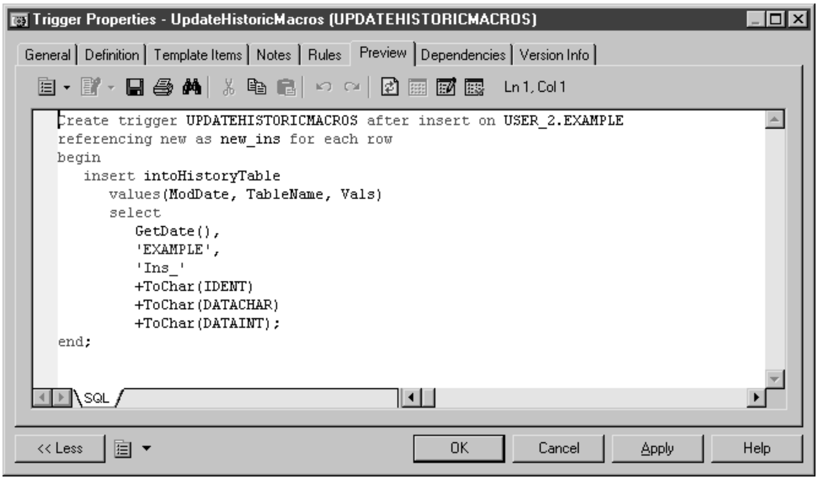
In the following example, a trigger written using the PDM variables and macros is attached to the Example table, and writes the contents of any insertion to HistoryTable.

See "Writing SQL with the PowerDesigner GTL" on page 689 for an example of the same trigger written using the PowerDesigner GTL:



```
Create trigger %TRIGGER% after insert on [%OWNER%.]%TABLE%
referencing new as new_ins for each row
begin
   insert intoHistoryTable
      values(ModDate, TableName, Vals)
      select
         GetDate(),
         %.q:TABLE%,
         'Ins_'
.ALLCOL("         +ToChar(%COLUMN%)", "", "",";")
end
```

The actual trigger code can be viewed on the Preview tab:



```
Create trigger UPDATEHISTORICMACROS after insert on USER_2.EXAMPLE
referencing new as new_ins for each row
begin
   insert intoHistoryTable
      values(ModDate, TableName, Vals)
      select
         GetDate(),
         'EXAMPLE',
         'Ins_'
         +ToChar(IDENT)
         +ToChar(DATACHAR)
         +ToChar(DATAINT);
end;
```

☞ For lists of the available variables and macros, see “PDM Macros” on page 693, “PDM Variables” on page 704, and “PowerDesigner Formatting Variables” on page 716.

PDM Macros

You can use predefined macros in trigger templates, template items, triggers, and procedures. Macros perform specific functions.

AKCOLN

Description

Repeats a statement for each alternate key in a table

Syntax

.AKCOLN("*statement*","*prefix*","*suffix*","*last_suffix*", "**condition**"**)**

Argument	Description
statement	Statement to repeat for each column
prefix	Prefix for each new line
suffix	Suffix for each new line
last suffix	Suffix for the last line
condition	Alternate key code (if condition argument is left empty the macro returns a statement for each alternate key in the table)

Example

In a trigger for the table TITLEAUTHOR, the following macro:

```
message .AKCOLN("’%COLUMN% is an alternate key column’","", "",
         "", "AKEY1")
```

generates the following trigger script:

```
  message ’TA_ORDER is an alternate key column’,
```

Column variable only

For columns, the macro AKCOLN only accepts the variable %COLUMN%.

ALLCOL

Description

Repeats a statement for each column in a table

Syntax

.ALLCOL("*statement*","*prefix*","*suffix*","*last_suffix*"**)**

Argument	Description
statement	Statement to repeat for each column
prefix	Prefix for each new line
suffix	Suffix for each new line
last suffix	Suffix for the last line

Example

In a trigger for the table AUTHOR, the following macro:

```
.ALLCOL("%COLUMN% %COLTYPE%","",",",";")
```

generates the following trigger script:

```
AU_ID char(12),
AU_LNAME varchar(40),
AU_FNAME varchar(40),
AU_BIOGRAPH long varchar,
AU_ADVANCE numeric(8,2),
AU_ADDRESS varchar(80),
CITY varchar(20),
STATE char(2),
POSTALCODE char(5),
AU_PHONE char(12);
```

DEFINE

Description

Defines a variable and initializes its value.

Syntax

.DEFINE "*variable*" "*value*"

Argument	Description
variable	Variable name (without % signs)
value	Variable value (may include another variable surrounded by % signs)

Example

In a trigger for the table AUTHOR, the following macro:

```
.DEFINE "TRIGGER" "T_%TABLE%"
message 'Error: Trigger(%TRIGGER%) of table %TABLE%'
```

generates the following trigger script:

```
message 'Error: Trigger(T_AUTHOR) of table AUTHOR';
```

DEFINEIF

Description

Defines a variable and initializes its value if the test value is not null

Syntax

.DEFINEIF "*test_value*" "*variable*" "*value*"

Argument	Description
test_value	Value to test
variable	Variable name (without % signs)
value	Variable value (may include another variable surrounded by % signs)

Example

For example, to define a variable for a default data type:

```
%DEFAULT%
.DEFINEIF "%DEFAULT%" "_DEFLT"" "%DEFAULT%"
Add %COLUMN% %DATATYPE% %_DEFLT%
```

ERROR

Description

Handles errors

Syntax

.ERROR (*errno***, "***errmsg***")**

Argument	Description
errno	Error number
errmsg	Error message

Example

```
.ERROR(-20001, "Parent does not exist, cannot insert child")
```

FKCOLN

Description

Repeats a statement for each foreign key column in a table

Syntax

.FKCOLN("*statement***","***prefix***","***suffix***","***last_suffix***")**

Argument	Description
statement	Statement to repeat for each column
prefix	Prefix for each new line
suffix	Suffix for each new line
last suffix	Suffix for the last line

Example

In a trigger for the table TITLEAUTHOR, the following macro:

```
message .FKCOLN("’%COLUMN% is a foreign key column’","",",",";")
```

generates the following trigger script:

```
message ’AU_ID is a foreign key column,
TITLE_ISBN is a foreign key column;’
```

Column variable only
For columns, the macro FKCOLN only accepts the variable %COLUMN%.

FOREACH_CHILD

Description

Repeats a statement for each parent-to-child reference in the current table fulfilling a condition

Syntax

.FOREACH_CHILD ("*condition*")

"*statement*"

.ENDFOR

Argument	Description
condition	Reference condition (see below)
statement	Statement to repeat

Condition	Selects
UPDATE RESTRICT	Restrict on update
UPDATE CASCADE	Cascade on update
UPDATE SETNULL	Set null on update
UPDATE SETDEFAULT	Set default on update

Condition	Selects
DELETE RESTRICT	Restrict on delete
DELETE CASCADE	Cascade on delete
DELETE SETNULL	Set null on delete
DELETE SETDEFAULT	Set default on delete

Example

In a trigger for the table TITLE, the following macro:

```
.FOREACH_CHILD("DELETE RESTRICT")
--  Cannot delete parent "%PARENT%" if children still exist in
         "%CHILD%"
.ENDFOR
```

generates the following trigger script:

```
--  Cannot delete parent "TITLE" if children still exist in
         "ROYSCHED"
--  Cannot delete parent "TITLE" if children still exist in
         "SALE"
--  Cannot delete parent "TITLE" if children still exist in
         "TITLEAUTHOR"
```

FOREACH_COLUMN

Description

Repeats a statement for each column in the current table fulfilling a condition

Syntax

.FOREACH_COLUMN ("*condition*")

"*statement*"

.ENDFOR

Argument	Description
condition	Column condition (see below)
statement	Statement to repeat

Condition	Selects
empty	All columns
PKCOLN	Primary key columns
FKCOLN	Foreign key columns
AKCOLN	Alternate key columns
NMFCOL	Non-modifiable columns (columns that have Cannot Modify selected as a check parameter)
INCOLN	Triggering columns (primary key columns, foreign key columns; and non-modifiable columns)

Example

In a trigger for the table TITLE, the following macro:

```
.FOREACH_COLUMN("NMFCOL")
-- "%COLUMN%" cannot be modified
.ENDFOR
```

generates the following trigger script:

```
-- "TITLE_ISBN" cannot be modified
-- "PUB_ID" cannot be modified
```

FOREACH_PARENT

Description

Repeats a statement for each child-to-parent reference in the current table fulfilling a condition

Syntax

.FOREACH_PARENT ("*condition*")

"*statement*"

.ENDFOR

Argument	Description
condition	Reference condition (see below)
statement	Statement to repeat

Condition	Selects references defined with . . .
empty	All references
FKNULL	Non-mandatory foreign keys
FKNOTNULL	Mandatory foreign keys
FKCANTCHG	Non-modifiable foreign keys

Example

In a trigger for the table SALE, the following macro:

```
.FOREACH_PARENT("FKCANTCHG")
--  Cannot modify parent code of "%PARENT%" in child "%CHILD%"
.ENDFOR
```

generates the following trigger script:

```
--  Cannot modify parent code of "STORE" in child "SALE"
--  Cannot modify parent code of "TITLE" in child "SALE"
```

INCOLN

Description

Repeats a statement for each primary key column, foreign key column, alternate key column, or non-modifiable column in a table.

Syntax

.INCOLN("*statement*","*prefix*","*suffix*","*last_suffix*")

Argument	Description
statement	Statement to repeat for each column
prefix	Prefix for each new line
suffix	Suffix for each new line
last suffix	Suffix for the last line

Example

In a trigger for the table TITLE, the following macro:

```
.INCOLN("%COLUMN% %COLTYPE%","",",",";")
```

generates the following trigger script:

```
TITLE_ISBN char(12),
PUB_ID char(12);
```

JOIN

Description

Repeats a statement for column couple in a join.

Syntax

.JOIN("*statement*","*prefix*","*suffix*","*last_suffix*"**)**

Argument	Description
statement	Statement to repeat for each column
prefix	Prefix for each new line
suffix	Suffix for each new line
last suffix	Suffix for the last line

Example

In a trigger for the table TITLE, the following macro:

```
.FOREACH_PARENT()
where .JOIN("%PK%=%FK%", " and", "", ";")
message 'Reference %REFR% links table %PARENT% to %CHILD%'
 .ENDFOR
```

generates the following trigger script:

```
message 'Reference TITLE_PUB links table PUBLISHER to TITLE
```

Primary key, alternate key, and foreign keys variables only
For columns, the macro JOIN only accepts the variables %PK%, %AK%, and %FK%.

NMFCOL

Description

Repeats a statement for each non-modifiable column in a table.
Non-modifiable columns have Cannot Modify selected as a check parameter.

Syntax

.NMFCOL("*statement*","*prefix*","*suffix*","*last_suffix*"**)**

Argument	Description
statement	Statement to repeat for each column
prefix	Prefix for each new line
suffix	Suffix for each new line
last suffix	Suffix for the last line

Example

In a trigger for the table TITLE, the following macro:

```
.NMFCOL("%COLUMN% %COLTYPE%","",",",";")
```

generates the following trigger script:

```
TITLE_ISBN char(12),
PUB_ID char(12);
```

PKCOLN

Description

Repeats a statement for each primary key column in a table

Syntax

.PKCOLN("*statement*","*prefix*","*suffix*","*last_suffix*")

Argument	Description
statement	Statement to repeat for each column
prefix	Prefix for each new line
suffix	Suffix for each new line
last suffix	Suffix for the last line

Example

In a trigger for the table TITLEAUTHOR, the following macro:

```
message .PKCOLN("'%COLUMN% is a primary key column'","",",",";")
```

generates the following trigger script:

```
message 'AU_ID is a primary key column',
        'TITLE_ISBN is a primary key column';
```

Column variable only
For columns, the macro PKCOLN only accepts the variable %COLUMN%.

CLIENTEXPRESSION and SERVEREXPRESSION

Description

Uses the client and/or server expression of a business rule in the trigger template, template item, trigger, and procedure script.

Syntax

.CLIENTEXPRESSION(code of the business rule)

.SERVEREXPRESSION(code of the business rule)

Example

The business rule ACTIVITY_DATE_CONTROL has the following server expression:

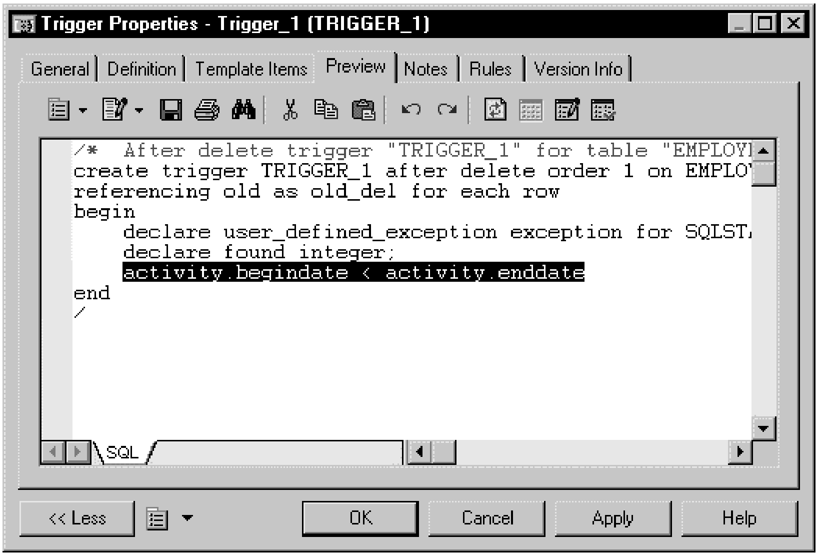
```
activity.begindate < activity.enddate
```

In a trigger based on template AfterDeleteTrigger, you type the following macro in the Definition tab of the trigger:

```
.SERVEREXPRESSION(ACTIVITY_DATE_CONTROL)
```

This generates the following trigger script:

```
activity.begindate < activity.enddate
end
```



```
/*  After delete trigger "TRIGGER_1" for table "EMPLOYI
create trigger TRIGGER_1 after delete order 1 on EMPLO
referencing old as old_del for each row
begin
    declare user_defined_exception exception for SQLST
    declare found integer;
    activity.begindate < activity.enddate
end
/
```

SQLXML

Description

Avoids typing a SQL/XML query in the definition of a trigger, a procedure or a function. Use one of the following tools:

- The **Insert SQL/XML Macro** tool opens a selection dialog box where you choose a global element from an XML model open in the workspace, mapped to a PDM, and linked with the SQL/XML extended model definition. Click OK in the dialog box and the SQLXML macro is displayed in the definition code, with the code of the XML model (optional) and the code of the global element

- The **Macros** tool, where you select **.SQLXML( )** in the list. The SQLXML macro is displayed empty in the definition code. You must fill the parentheses with the code of an XML model (optional), followed by :: and the code of a global element. The XML model, from which you choose a global element, must be open in the workspace, mapped to a PDM, and linked with the SQL/XML extended model definition

After generation, the SQLXML macro is replaced by the SQL/XML query of the global element.

Syntax

.SQLXML(code of an XML model::code of a global element)

Note: the code of an XML model is optional.

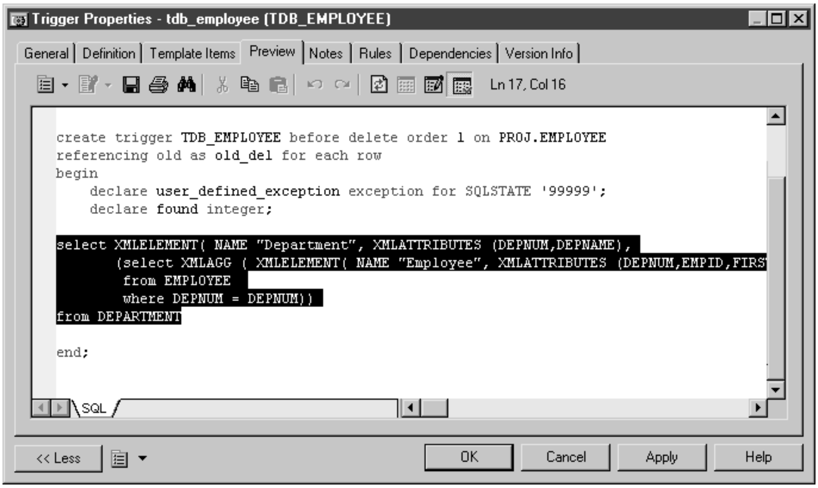
Example

In a trigger for the table EMPLOYEE, the following macro:

```
.SQLXML(CorporateMembership::DEPARTMENT)
```

generates the following trigger script:

```
select XMLELEMENT( NAME "Department", XMLATTRIBUTES
         (DEPNUM,DEPNAME),
        (select XMLAGG ( XMLELEMENT( NAME "Employee",
          XMLATTRIBUTES (DEPNUM,EMPID,FIRSTNAME,LASTNAME)) )
         from EMPLOYEE
          where DEPNUM = DEPNUM))
from DEPARTMENT
```



```
create trigger TDB_EMPLOYEE before delete order 1 on PROJ.EMPLOYEE
referencing old as old_del for each row
begin
    declare user_defined_exception exception for SQLSTATE '99999';
    declare found integer;

select XMLELEMENT( NAME "Department", XMLATTRIBUTES (DEPNUM,DEPNAME),
       (select XMLAGG ( XMLELEMENT( NAME "Employee", XMLATTRIBUTES (DEPNUM,EMPID,FIRS
        from EMPLOYEE
        where DEPNUM = DEPNUM))
from DEPARTMENT

end;
```

PDM Variables

Many objects have both a code variable and a generated code variable.

The code variable is the attribute code that is defined in the object property sheet.

The generated code variable is computed from the attribute code according to available generation options. The values for the code and generated code variables can be different in the following situations:

♦ When the Code is a reserved word or it contains invalid characters, the generated code is quoted

♦ When the code is longer than authorized by the database, the generated code is truncated

Note:

To access variables of sub-objects (columns in a table for example) you have to use loop macros or GTL macros to browse the list of sub-objects. For example, in a trigger the following macro loops on table columns and for each column with the CannotModify attribute outputs the code of the column followed by “cannot be modified”:

```
.foreach_item(Table.Columns)
.if(%CannotModify%)
-- "%Code%" cannot be modified

.endif
.next
```

☞ For more information, see “Using macros” in the chapter “Building Triggers and Procedures”.

Common variables for all named objects

The following variables can be used for all named object definitions:

Variable name	Comment
@OBJTNAME	Object name
@OBJTCODE	Object code
@OBJTLABL	Object comment
@OBJTDESC	Object description

Common variables for objects

The following variables can be used in all object definitions:

Variable name	Comment
COMMENT	Object comment. If no comment is defined the object name is used
OWNER	Generated code of the object owner, or the object parent
DBPREFIX	Database prefix of object (name of database + '.' if database defined)
QUALIFIER	Whole object qualifier (database prefix + owner prefix)
OPTIONS	SQL text defining physical options for object
CONSTNAME	Object constraint name
CONSTRAINT	Object constraint SQL body. For example(A <= 0) AND (A >= 10)
RULES	Concatenation of server expression for business rule associated with object

Table variables

The following variables can used in a table definition:

Variable name	Comment
TABLE	Generated code of the table
TNAME	Table name
TCODE	Table code
TLABL	Table comment
PKEYCOLUMNS	List of primary key columns. For example: A, B
TABLDEFN	Complete body of the table definition. It contains definition of columns, checks and keys
CLASS	Abstract data type name
CLUSTERCOLUMNS	List of columns used for the cluster

Variables for domain and column checks

The following variables can be used in domain check parameter and column check parameter definitions:

Variable name	Comment
UNIT	Standard check unit attribute

Variable name	Comment
FORMAT	Standard check format attribute
DATATYPE	Data type. For example int, char(10) or numeric(8, 2)
DTTPCODE	Data type code. For example int, char or numeric
LENGTH	Data type length. For example 0, 10 or 8
PREC	Data type precision. For example 0, 0 or 2
ISRDONLY	TRUE if the read-only attribute of standard check is selected
DEFAULT	Default value
MINVAL	Minimum value
MAXVAL	Maximum value
VALUES	List of values. For example(0, 1, 2, 3, 4, 5)
LISTVAL	SQL constraint associated with a list of values. For example C1 in (0, 1, 2, 3, 4, 5)
MINMAX	SQL constraint associated with minimum and maximum values. For example(C1 <= 0) AND (C1 >= 5)
ISMAND	TRUE if the domain or column is mandatory
MAND	Contains the keywords “null” or “not null” depending on if the attribute is mandatory or not mandatory
NULL	Contains keyword “null” if the domain or column is not mandatory
NOTNULL	Contains Keyword “not null” if the domain or column is mandatory
IDENTITY	Keyword “identity” if the domain or column is identity (Sybase specific)
WITHDEFAULT	Keyword “with default” if the domain or column is with default
ISUPPERVAL	TRUE if the upper-case attribute of standard check is selected
ISLOWERVAL	TRUE if the lower-case attribute of standard check is selected

Column variables

The following variables can be used in a column definition:

Variable name	Comment
COLUMN	Generated code of the column
COLNNO	Position of Column in List of columns of Table
COLNNAME	Column name
COLNCODE	Column code
PRIMARY	Contains Keyword “primary” if the column is a primary key column
ISPKEY	TRUE if the column is part of a primary key
FOREIGN	TRUE if the column is part of a foreign key
COMPUTE	Compute constraint text

Abstract data type variables

The following variables can be used in an abstract data type definition:

Variable name	Comment
ADT	Generated code of the abstract data type
TYPE	Type of Abstract data type. For example “array”, or “list”
SIZE	Abstract data type size
ISARRAY	TRUE if the abstract data type is of type array
ISLIST	TRUE if the abstract data type is of type list
ISSTRUCT	TRUE if the abstract data type is of type structure
ISOBJECT	TRUE if the abstract data type is of type object
ISJAVA	TRUE if the abstract data type is of type JAVA class
ADTDEF	Contains definition of the abstract data type

Abstract data type attribute variables

The following variable can be used in an abstract data type attribute definition:

Variable name	Comment
ADTATTR	Generated code of an abstract data type attribute

Domain variables

The following variable can be used in a domain definition:

Variable name	Comment
DOMAIN	Generated code of a domain (also available for columns)

Business rule variables

The following variables can be used in a business rule definition:

Variable name	Comment
RULE	Generated code of a business rule
RULENAME	Rule name
RULECODE	Rule code
RULECEXPR	Rule client expression
RULESEXPR	Rule server expression

Index variables

The following variables can be used in an index definition:

Variable name	Comment
TABLE	Generated code of the parent of an index, can be a table or a query table (view)
INDEX	Generated code of the index
INDEXNAME	Index name
INDEXCODE	Index code
UNIQUE	Contains keyword “unique” when an index is unique
INDEXTYPE	Contains the index type (DBMS dependant)
CIDXLIST	List of index columns. For example A asc, B desc, C asc
INDEXKEY	Contains keywords “primary”, “unique” or “foreign” depending on an index origin
CLUSTER	Contains keyword “cluster” when an index is a clustered index
INDXDEFN	Used for defining an index within a table definition

Index column variables

The following variables can be used in index column definitions:

Variable name	Comment
ASC	Contains keywords “ASC” or “DESC” depending on the sort order
ISASC	TRUE if the index column sort is ascending

Reference variables

The following variables can be used in a reference definition:

Variable name	Comment
REFR	Generated code of the reference
PARENT	Generated code of the parent table
PNAME	Parent table name
PCODE	Parent table name
PQUALIFIER	Parent table qualifier. See also QUALIFIER
CHILD	Generated code of a child table
CNAME	Child table name
CCODE	Child table code
CQUALIFIER	Child table qualifier. See also QUALIFIER
REFRNAME	Reference name
REFRCODE	Reference code
FKCONSTRAINT	Foreign key (reference) constraint name
PKCONSTRAINT	Constraint name of the primary key used to reference the object
CKEYCOLUMNS	List of parent key columns. For example: C1, C2, C3
FKEYCOLUMNS	List of child foreign key columns. For example: C1, C2, C3
UPDCONST	Update declarative constraint. This can be any of the following keywords Restrict Cascade Set null Set default

Variable name	Comment
DELCONST	Delete declarative constraint. This can be any of the following keywords Restrict Cascade Set null Set default
MINCARD	Minimum cardinality
MAXCARD	Maximum cardinality
POWNER	Parent table owner name
COWNER	Child table owner name
CHCKONCMMT	TRUE when check on commit is selected on a reference (ASA 6.0 specific)
JOINS	Reference joins
REFRNO	Reference number in the child table collection of references

Reference column variables

The following variables can be used in reference column definitions:

Variable name	Comment
CKEYCOLUMN	Generated code of the parent table column (primary key)
FKEYCOLUMN	Generated code of a child table column (foreign key)
PK	Generated code of a primary key column
PKNAME	Primary key column name
FK	Generated code of a foreign key column
FKNAME	Foreign key column name
AK	Alternate key column code (same as PK)
AKNAME	Alternate key column name (same as PKNAME)
COLTYPE	Primary key column data type
DEFAULT	Foreign key column default value
HOSTCOLTYPE	Primary key column data type used in procedure declaration. For example: without length

Key variables

The following variables can be used in a key definition:

Variable name	Comment
COLUMNS COLNLIST	List of key columns. For example"A, B, C"
ISPKEY	TRUE when a key is the primary key for the table
PKEY	Primary key constraint name
AKEY	Alternate key constraint name
KEY	Key constraint name
ISMULTICOLN	True if the key has more than one column
CLUSTER	Cluster keyword

Variables for views

The following variables can be used in a view definition:

Variable name	Comment
VIEW	Generated code of the view
VIEWNAME	View name
VIEWCODE	View code
VIEWCOLN	List of columns of a view. For example: "A, B, C"
SQL	SQL text of a view. For example Select * from T1
VIEWCHECK	Contains the keyword "with check option" if this option is selected in the view property sheet
SCRIPT	Complete view creation order. For example create view V1 as select * from T1

Trigger variables

The variables listed below can be used in a trigger definition. You can also use owning table variables in a trigger definition.

Variable name	Comment
ORDER	Order number of Trigger (where the current DBMS supports more than one trigger of one type)
TRIGGER	Generated code of the trigger
REFNO	Reference order number in the list of references for the table
ERRNO	Error number for standard error

Variable name	Comment
ERRMSG	Error message for standard error
MSGTAB	Name of a table containing user-defined error messages
MSGNO	Name of a column containing error numbers in a user-defined error table
MSGTXT	Name of a column containing error messages in a user-defined error table
SCRIPT	SQL script of trigger or procedure.
TRGBODY	Trigger body (only for Oracle live database reverse engineering)
TRGDESC	Trigger description (only for Oracle live database reverse engineering)
TRGDEFN	Trigger definition

Database, trigger, and procedure generation variables

The following variables can be used for database, procedure, and trigger generation:

Variable name	Comment
DATE	Generation date & time
USER	Login name of the user executing the generation
PATHSCRIPT	Path where the file script will be generated
NAMESCRIPT	Name of the file script where SQL orders will be written
STARTCMD	Description explaining how to execute a generated script
ISUPPER	TRUE if upper case generation option is set
ISLOWER	TRUE if lower case generation option is set
DBMSNAME	Name of the DBMS associated with the generated model
DATABASE	Code of the database associated with the generated model

Reverse engineering variables

The following variables can be used when reverse engineering a database into a PDM:

Variable name	Comment
R	Set to TRUE during reverse engineering
S	Allows to skip a word. The string is parsed for reverse engineering but is not generated
D	Allows to skip a numeric value. The numeric value is parsed for reverse engineering but is not generated
A	Allows to skip all text. The text is parsed for reverse engineering but is not generated
ISODBCUSER	True if the current user is the connected user
CATALOG	Catalog name that will be used in live database reverse engineering queries
SCHEMA	Schema that will be used in live database reverse engineering queries
SIZE	Data type size of a column or a domain. Used for live database reverse engineering, when a data type length is not defined in the system tables
VALUE	One value from the list of values in a column or domain
TRGTYPE	Variable used in the Create order of a trigger. Trigger type uses keywords for each trigger type, for example “BeforeInsert”, or “AfterUpdate”
TRGEVENT	Variable used in the Create order of a trigger. Trigger event uses keywords for each trigger event, for example” Insert”, “Update”, and “Delete”
TRGTIME	Variable used in the Create order of a trigger. Timing of trigger uses the keywords “Null”, “Before”, and “After”

Database synchronization variables

The following variables can be used for database generation when synchronizing a modified PDM with an existing database:

Variable name	Comment
OLDOWNER	Old owner name of the object. See also OWNER
NEWOWNER	New owner name of the object. See also OWNER
OLDQUALIFIER	Old qualifier of the object. See also QUALIFIER
NEWQUALIFIER	New qualifier for the object. See also QUALIFIER

Variable name	Comment
OLDTABL	Old code of the table
NEWTABL	New code of the table
OLDCOLN	Old code of the column
NEWCOLN	New code of the column

Database security variables

The following database security variables are available:

Variable name	Comment
PRIVLIST	List of privileges for a grant/revoke order
PERMLIST	List of permissions for a grant/revoke order
GRANTEE	Name of the user, group, or role for a grant/revoke order
ID	Name of the user
GROUP	Name of the group
ROLE	Name of the role
OBJECT	Database objects (table, view, column, and so on)
PERMISSION	SQL grant/revoke order for a database object
PRIVILEGE	SQL grant/revoke order for an ID (user, group, or role)
GRANTOPTION	Option for grant: with grant option / with admin option
REVOKEOPTION	Option for revoke: with cascade

Metadata variables

The following metadata variables are available:

Variable name	Comment
@CLSSNAME	Localized name for an object class. For example: Table, View, Column, Index
@CLSSCODE	Object class code. For example: TABL, VIEW, COLN, INDX

DBMS, database options variables

The following DBMS and database options variables are available:

Variable name	Comment
TABLESPACE	Generated code of a tablespace
STORAGE	Generated code of a storage

Variables for ASE & SQL Server

The following DBMS specific variables are available for Sybase Adaptive Server Anywhere and Microsoft SQL Server:

Variable name	Comment
RULENAME	Name of a business rule object associated with a domain
DEFAULTNAME	Name of a default object associated with a domain
USE_SP_PKEY	Use sp_primary key to create primary keys (SQL Server)
USE_SP_FKEY	Use sp_foreign key to create foreign keys (SQL Server)

Sequence variable

The following variable can be used in a sequence definition:

Variable name	Comment
SQNC	Name of sequence

Procedure variables

The following variables can be used in a procedure definition:

Variable name	Comment
PROC	Generated code of a procedure (also available for trigger when a trigger is implemented with a procedure)
FUNC	Generated code of a procedure where the procedure is a function (with a return value)

Join index variables (IQ)

The following variables can be used in a join index definition:

Variable name	Comment
JIDX	Generated code for join index
JIDXDEFN	Complete body of join index definition
REFRLIST	List of references (for live database connection)
RFJNLIST	List of reference joins (for live database connection)

PowerDesigner Formatting Variables

Variables have a syntax that can force a format on their values, as follows:

- Force values to lower-case or upper-case characters
- Truncate the length of values

When a variable does not specify formatting, its values have the same format as in the PDM.

Format code	Format of variable value in script
.L	Lower-case characters
.T	Removes blank spaces
.U	Upper-case characters
.c	Upper-case first letter and lower-case next letters
.*n*	Maximum length where n is the number of characters
n	Justifies to fixed length where n is the number of characters
-*n*	Right justify variable text to fixed length where *n* is the number of characters

You embed formatting options in variable syntax as follows:

%.format:*variable*%

For example:

```
%.L:TABLE%
```

The table below shows formatted variables and their results in a script for the table EMPLOYEE.

Template statement with variable	Resulting script statement
create trigger %TABLE%	create trigger EMPLOYEE
create trigger %.L:TABLE%	create trigger employee
create trigger %.U:TABLE%	create trigger EMPLOYEE
create trigger %.4:TABLE%	create trigger EMPL
create trigger %.4L:TABLE%	create trigger empl

CHAPTER 13

Migrating from ERwin to PowerDesigner

About this appendix

This appendix explains how to migrate from ERwin to PowerDesigner. You will learn about the differences between the two design environments and how to import your models into PowerDesigner.

Contents

Introducing the ERwin Import Process

You can easily import a model built with ERwin into PowerDesigner with no loss of metadata. PowerDesigner allows complete flexibility through reliable linking and synchronization between conceptual, physical and object-oriented model approaches, providing outstanding model clarity and flexibility.

File types

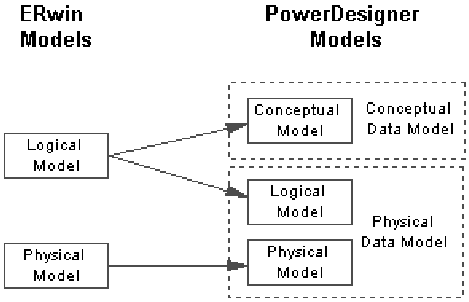
PowerDesigner supports the import of the following ERwin file types:

- ERwin v3.x (.erx)
- ERwin v4.x (.xml)
- ERwin v7.x (.xml) – the ERwin model must be saved as "Standard XML Format", and you must uncheck the "Only save minimum amount of information" check box in the ERwin Save as XML File dialog box.

Versions 4.x or 7.x are recommended, as they contain more metadata.

ERwin and PowerDesigner implement database design in different ways.

- An ERwin logical model can be imported into either:
  - a PowerDesigner conceptual data model (CDM), or
  - a PowerDesigner logical data model
- An ERwin Physical Model is imported into a PowerDesigner physical data model (PDM)



In the PowerDesigner Physical Data Model you can use two design levels: the logical model allows you to design the database structure and perform some database denormalization actions without taking into account any DBMS physical requirement. The physical model allows you to design the database taking into account the physical requirements of a given DBMS

Limitations

You cannot import the following ERwin objects:

- ERwin triggers and stored procedures (not directly possible, but see the process in "Post-import checklist" on page 725)
- ERwin reports
- ER1 files
- ERwin data sources
- ERwin target clients

While PowerDesigner can import all your object display preferences and will retain color and font information, it does not support multiple colors for columns in a single table. The default column color will be used during the import.

Preparing to Import your ERwin models

This section provides a list of suggestions for how to prepare your ERwin files and set up your PowerDesigner environment:

- We recommend that you import from an ERwin 4.x file (.xml) rather than a 3.x file (.erx), due to the greater amount of metadata available in the newer ERwin files.
- Review your ERwin model to see if any model object names are duplicated. It is good practice to avoid using duplicate names, and PowerDesigner will automatically attach a suffix to any duplicate objects that it encounters during the import process.

The Import Process

PowerDesigner provides a wizard to automatically import your ERwin files.

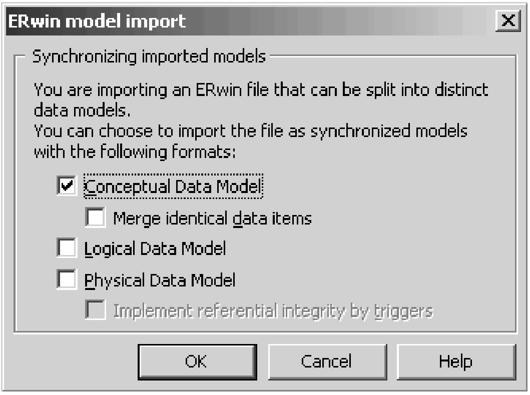
❖ To import an ERwin model into PowerDesigner

1. Select File ➤ Import ➤ ERwin File.

2. Select or browse to the directory that contains the ERwin file, select it, and then click Open.

3. If the ERwin file contains only a physical model, you will be prompted to choose whether to import references as triggers. Select Yes or No to begin the import.

   Alternatively, if the ERwin file contains a logical model or a combined logical and physical model, the ERwin model import dialog box opens:



ERwin model import

Synchronizing imported models

You are importing an ERwin file that can be split into distinct data models.
You can choose to import the file as synchronized models with the following formats:

☑ Conceptual Data Model

□ Merge identical data items

□ Logical Data Model

□ Physical Data Model

□ Implement referential integrity by triggers

The options available depend on the type of ERwin model that you are importing. The traditional approach to data modeling in PowerDesigner is to create a Conceptual Data Model synchronized with a Physical Data Model. The full set of options is as follows:

- A **Conceptual Data Model** can be created when you are importing an ERwin Logical Model. It provides a platform-independent representation of a system, giving an abstract view of its static data structures, and permitting real normalized data structures with many-to-many and inheritance relationships.

- A **Logical Data Model** can be created when you are importing an ERwin Logical Model. It allows you to resolve many-to-many and super/sub-type relationships, de-normalize your data structures, and define indexes, without specifying a particular RDBMS. In PowerDesigner, a Logical Data Model is an RDBMS-independent version of a Physical Data Model.

- A **Physical Data Model** can be created when you are importing an ERwin Physical Model. It is a representation of a real database and associated objects running on a server with complete information on the structure of the physical objects, such as tables, columns, references, triggers, stored procedures, views, and indexes.

 Select the checkbox for each type of models that you want to create.

4. If your ERwin model contains a logical model, and you want to create a conceptual data model, then you can choose to merge identical data items. This is a powerful metadata management technique that is not available in the ERwin environment.

   For example, your ERwin logical model may contain multiple entities that contain an attribute “address”. By default, PowerDesigner will create a separate data item for each of these entity attributes. However if you select the Merge identical data items checkbox, then a single data item will be created, and adjustments to it will automatically cascade down to all the associated entity attributes.

5. If your ERwin model contains a physical model, then you can choose whether to implement referential integrity using triggers.

6. Click OK to begin the import. When the process is complete, the imported models will appear in the Browser.

After Importing

This section describes what you should expect in your newly-imported models.

PowerDesigner object terminology

PowerDesigner and ERwin use a different terminology to speak about many of their model objects. The following sections compare how these two tools refer to model objects.

General PowerDesigner model objects

The import process converts general ERwin model objects into PowerDesigner objects as follows:

ERwin	PowerDesigner
Model	Model
Stored display and subject area	Diagram
Business rule	Business rule
Domain	Domain
Symbols (including symbol size and position)	Symbols (including symbol size and position)
Description	Description
Notes	Annotation
Text block	Text symbol
IE notation	Entity/Relationship notation
IDEF1X notation	IDEF1X notation
User-defined properties	Imported as extended attributes stored into a specific extended model definition embedded in the model. The name of this extended model definition is Imported Attributes, you can manage it from the resource editor. For more information, see “Extended Model Definitions” in the Resource Files and the Public Metamodel chapter of the *Customizing and Extending PowerDesigner* manual.

PowerDesigner conceptual data model objects

The import process converts ERwin logical model objects into conceptual data model (CDM) objects as follows:

ERwin logical model	PowerDesigner CDM
Attribute	Data item, entity attribute
Key group	Identifier
Entity	Entity
Relationship	Relationship
Subtype relationship	Inheritance link
Subtype category	Inheritance

PowerDesigner physical data model objects

The import process translates ERwin physical model objects into physical data model (PDM) objects as follows:

ERwin physical model	PowerDesigner PDM
Column	Column
Key	Key
Table	Table
Relationship	Reference
Index	Index
View table	View
Fact, dimension, outrigger	Table
Target database	Current DBMS
Valid value	Check parameter
Tablespace	Tablespace
Segment	Storage

Post-import checklist

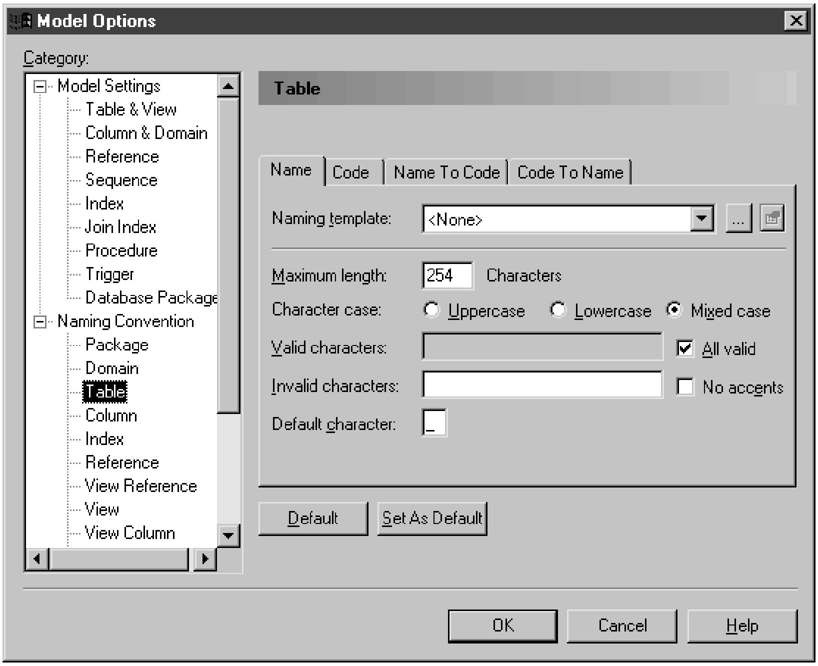
This section lists a series of recommended post-import checks and processes.

Import triggers: Triggers cannot be directly imported from ERwin. There are, however, two methods for transferring your constraint trigger information to PowerDesigner:

- **Automatically generate triggers**: To do this, select Tools ➤ Rebuild Objects ➤ Rebuild Triggers. Creating triggers in this way ensures that they will be synchronized automatically by PowerDesigner, but the actual code may be different from that which you are used to in ERwin.
- **Reverse engineer triggers**: To do this, generate the triggers from ERwin, and then reverse engineer them into PowerDesigner. Creating triggers in this way ensures that they use exactly the same code as before, but they will not be automatically synchronized by PowerDesigner.

Import procedures: Procedures cannot be directly imported from ERwin. You can, however transfer them by generating the triggers from ERwin, and then reverse engineering them into PowerDesigner.

Set up your object naming conventions and other model options: To control the naming conventions of model objects in PowerDesigner, select Tools ➤ Model Options, and then click on the object entry in the Naming Convention category in the Category pane:



Naming template: <None>

Maximum length: 254 Characters

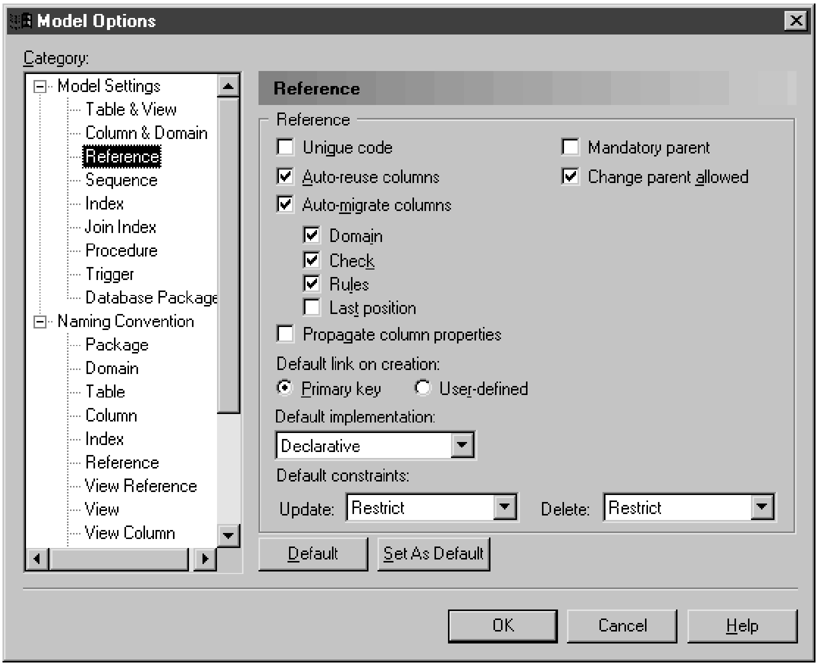
Character case: ○ Uppercase ○ Lowercase ● Mixed case

Valid characters: ☑ All valid

Invalid characters: □ No accents

Default character: _

You can control other object creation defaults by clicking on the object entry in the Model Settings category:



Reference

Reference

☐ Unique code
☑ Auto-reuse columns
☑ Auto-migrate columns
☑ Domain
☑ Check
☑ Rules
☐ Last position
☐ Propagate column properties
☐ Mandatory parent
☑ Change parent allowed

Default link on creation:
● Primary key ○ User-defined

Default implementation: Declarative

Default constraints:
Update: Restrict Delete: Restrict

Differences to Expect

This section lists certain differences that you may encounter when working with your imported ERwin model in PowerDesigner.

Why do I see errors in Check Model when my ERwin model was clean? PowerDesigner performs stricter checks than ERwin. For example, duplicate objects are not permitted in PowerDesigner, and the existence of orphaned items will generate a warning.

Why do some of my object symbols appear with numeric suffixes? If an object is required to appear more than once in a diagram (for, example, to improve readability), PowerDesigner will create a **graphical synonym** to represent it. Thus, if the table “Purchase” is displayed twice in a diagram, the two symbols will be labeled as “Purchase: 1” and “Purchase: 2”.

Getting Started Using PowerDesigner

This section lists some common tasks that former ERwin users will want to perform with PowerDesigner:

Objects

How do I find objects? All the objects in the model are listed, organized by type, in the Browser. PowerDesigner provides various methods for locating your objects:

- **To find the symbol for an object in the Browser**: Right-click the object in the Browser and select Find in Diagram from the contextual menu.
- **To find the browser entry for an object symbol**: Right-click the symbol in the diagram and select Find in Browser from the contextual menu.
- **To search for an object**: Type CTRL+F to open the Find Objects dialog box. Enter the text to search for (you can use the asterisk as a wild card) and click Find Now. Right-click any of the results choose whether to find it in the Browser or Diagram.

How do I edit objects? You can edit the name of an object by selecting its symbol in the diagram and typing F2. To edit other object properties, double-click the symbol or the object entry in the Browser and enter the necessary information in its property sheet.

How do I share objects? You can share objects between packages and models using shortcuts and replications. For more information, see the Shortcuts and Object Replications chapter in the *Core Features Guide* .

Packages/Subject Areas

How do I create subject areas? In PowerDesigner, you can create multiple views of your model by adding additional diagrams. You can also divide your model into smaller subdivisions using packages.

- **To add a diagram to your model**: Right-click the diagram background and select Diagram ➤ New Diagram ➤ [Diagram Type] from the contextual menu.
- **To convert a diagram into a package**: Right-click the diagram background and select Diagram ➤ Convert to Package. The Convert Diagram to Package wizard will open, permitting you to name the package and select objects to move into it. The package will appear in the Browser with its own diagram and associated objects. For more information about packages, see “Packages” in the Models chapter of the *Core Features Guide* .

Reports

How do I create a report? PowerDesigner provides wizards to create two different types of report:

- **To create a report about a specific type of object**: Select Report ➤ List Report Wizard and follow the wizard instructions.
- **To create a report about multiple object types or the whole model**: Select Report ➤ Report Wizard and follow the wizard instructions.

For more information about PowerDesigner reports, see the Reports chapter in the *Core Features Guide* .

Databases

How do I create a model from a database? Select File ➤ Reverse Engineer ➤ Database and complete the dialog box. For more information, see the "Reverse Engineering a Database into a PDM" chapter.

How do I update my model from a database? Select Database ➤ Reverse Engineer ➤ Database and complete the dialog box. A Merge window will open to allow you to verify the changes to be made before committing them. For more information, see the "Reverse Engineering a Database into a PDM" chapter.

How do I generate a database from my model? Select Database ➤ Generate Database and complete the dialog box. For more information, see the "Generating a Database from a PDM" chapter.

How do I update a database from my model? Select Database ➤ Apply Model Changes to Database and complete the dialog box. A Database Synchronization window will open to allow you to verify the changes to be made before committing them. For more information, see the "Generating a Database from a PDM" chapter.

Models

How do I compare two models? Select Tools ➤ Compare Models and complete the dialog box. For more information, see the Comparing and Merging Models chapter in the *Core Features Guide* .

How do I merge two models? Select Tools ➤ Merge Model and complete the dialog box. For more information, see the Comparing and Merging Models chapter in the *Core Features Guide* .

Index