Data Modeling

SAP® Sybase® PowerDesigner®

16.5 SP02

Windows

DOCUMENT ID: DC38058-01-1652-01

LAST REVISED: May 2013

Contents

PART I

Building Data Models

The chapters in this part explain how to model your data systems in SAP® Sybase® PowerDesigner®.

CHAPTER 1 Getting Started with Data Modeling

A data model is a representation of the information consumed and produced by a system, which lets you analyze the data objects present in the 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.

Suggested Bibliography

- Graeme Simsion, Van Nostrand Reinhold, *Data Modeling Essentials*, 1994, 310 pages; paperbound; ISBN 1850328773
- James Martin, Prentice Hall, *Information Engineering*, 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).
- Joe Celko, *Joe Celko's SQL for Smarties* (Morgan Kaufmann Publishers, Inc., 1995), 467 pages; paperbound; ISBN 1-55860-323-9.

Conceptual Data Models

A *conceptual data model (CDM)* helps you analyze the conceptual structure of an information system, to identify the principal entities to be represented, their attributes, and the relationships between them. A CDM is more abstract than a logical (LDM) or physical (PDM) data model.

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 Diagrams* on page 27.

Logical Data Models

A *logical data model (LDM)* helps you analyze the structure of an information system, independent of any specific physical database implementation. An LDM has migrated entity identifiers and is less abstract than a conceptual data model (CDM), but does not allow you to model views, indexes and other elements that are available in the more concrete physical data model (PDM).

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

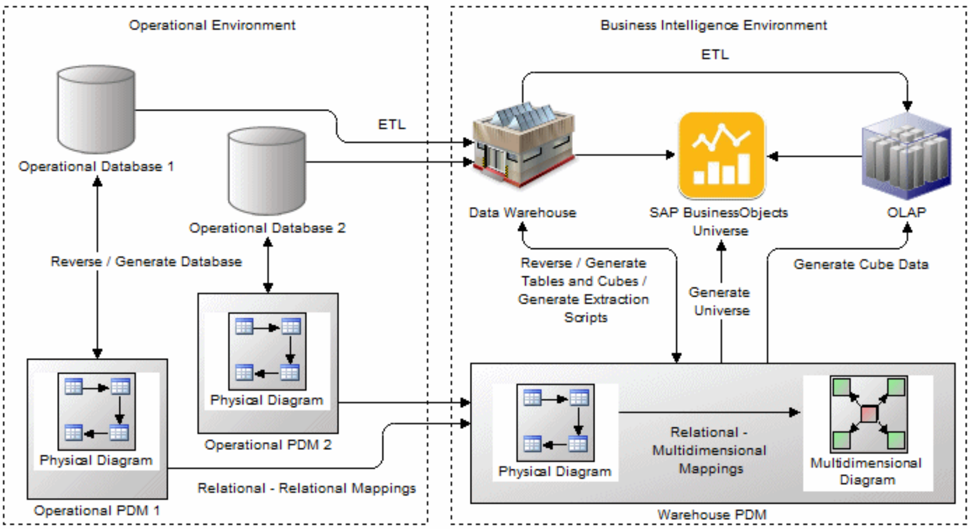
- Start with a CDM containing entities, attributes, relationships, domains, data items and business rules. If need be, you may develop the CDM in several design steps starting from a high level model to a low level CDM
- Generate an LDM. Create indexes and specify FK column names and other common features
- 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 Diagrams* on page 39.

Physical Data Models

A *physical data model (PDM)* helps you to analyze the tables, views, and other objects in a database, including multidimensional objects necessary for data warehousing. A PDM is more concrete than a conceptual (CDM) or logical (LDM) data model. You can model, reverse-engineer, and generate for all the most popular DBMSs.



PowerDesigner provides you with tools for modeling your operational and business intelligence environments:

- Operational/relational environment - modeled in physical diagrams (see *Chapter 3, Physical Diagrams* on page 73). The physical analysis may follow a 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 environment:
  - Data warehouse or data mart database tables - can be modeled in physical diagrams and mapped to their source operational tables to generate data extraction scripts.
  - Data warehouse cubes (in ROLAP or HOLAP environments) - can be modeled in multidimensional diagrams (see *Chapter 4, Multidimensional Diagrams* on page 215) and mapped to their source warehouse tables.
  - SAP® BusinessObjects™ Universes - can be generated from warehouse PDMs for direct consumption or for editing in BusinessObjects environments (see *Generating a BusinessObjects Universe* on page 301).
  - OLAP cubes - can be modeled in multidimensional diagrams and mapped to their source operational or warehouse tables to generate cube data.

PowerDesigner provides support for a wide range of database families through DBMS definition files (`*.xdb`, located in `Resource Files\DBMS` inside your installation directory), which customize the metamodel to support the specific syntax of a DBMS, through extended attributes, objects, and generation templates. To view and edit the resource file for your DBMS, select **Database > Edit Current DBMS**. For detailed information about working with these files, see *Customizing and Extending PowerDesigner > DBMS Definition Files*.

Creating a Data Model

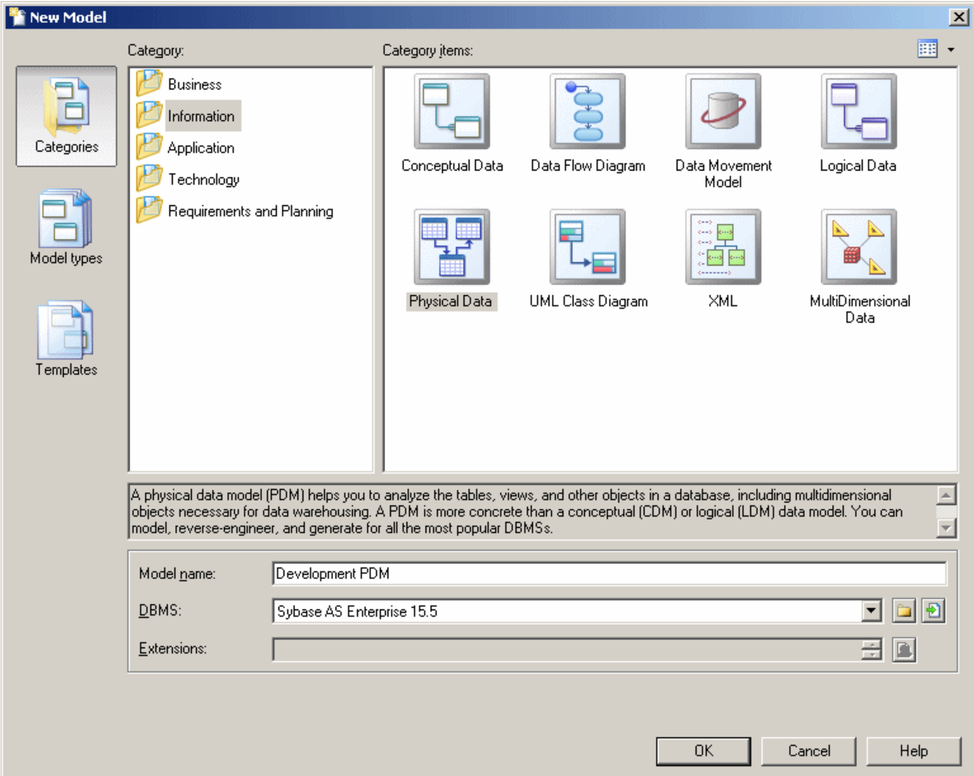
You create a new data model by selecting **File > New Model**.

Note: In addition to creating a data model from scratch with the following procedure, you can also:

- create a CDM by importing an ERwin model (.ERX) or by generating it from another PowerDesigner model.
- create an LDM by generating it from another PowerDesigner model.
- create a PDM by reverse-engineering it from an existing database (see *Reverse Engineering a Database into a PDM* on page 312) or generating it from another PowerDesigner model.

The New Model dialog is highly configurable, and your administrator may hide options that are not relevant for your work or provide templates or predefined models to guide you through model creation. When you open the dialog, one or more of the following buttons will be available on the left hand side:

- **Categories** - which provides a set of predefined models and diagrams sorted in a configurable category structure.
- **Model types** - which provides the classic list of PowerDesigner model types and diagrams.
- **Template files** - which provides a set of model templates sorted by model type.



1. Select **File > New Model** to open the New Model dialog.
2. Click a button, and then select a category or model type (**Conceptual Data Model, Logical Data Model or Physical Data Model**) in the left-hand pane.
3. Select an item in the right-hand pane. Depending on how your New Model dialog is configured, these items may be first diagrams or templates on which to base the creation of your model.

   Use the **Views** tool on the upper right hand side of the dialog to control the display of the items.
4. Enter a model name. The code of the model, which is used for script or code generation, is derived from this name using the model naming conventions.
5. [PDM only] Select a target DBMS , which customizes PowerDesigner's default modifying environment with target-specific properties, objects, and generation templates.

By default, PowerDesigner creates a link in the model to the specified file. To copy the contents of the resource and save it in your model file, click the **Embed Resource in Model** button to the right of this field. Embedding a file in this way enables you to make changes specific to your model without affecting any other models that reference the shared resource.

6. [optional] Click the **Select Extensions** button and attach one or more extensions to your model.
7. Click **OK** to create and open the data model .

Note: Sample data models are available in the Example Directory.

Data Model Properties

You open the model property sheet by right-clicking the model in the Browser and selecting **Properties**.

Each data model has the following model properties:

Property	Description
Name/Code/Comment	Identify the model. The name should clearly convey the model's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the model. By default the code is auto-generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Filename	Specifies the location of the model file. This box 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 display preferences of the Title node.
DBMS	[PDM only] Specifies the model target.

Property	Description
Database	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.
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

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/Code/ Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
DBMS	DBMS for the database
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

The following tabs are also available:

- **Options** - Lists the physical options associated with the database (see *Physical Options (PDM)* on page 88).
- **Script** - Specifies begin and end scripts to bookend the database creation script.
- **Rules** - Specifies the business rules associated with the database.

Using a Database in a Physical Option

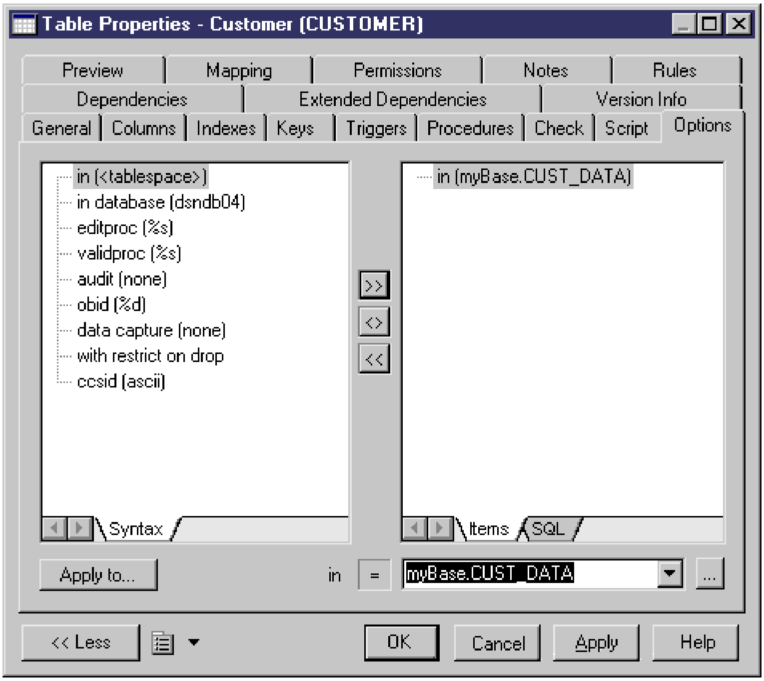
You can use a database in a physical option.

1. Open the property sheet of an object with physical options.
2. Click the Options tab, select the in database (...) optionand 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.

Customizing your Modeling Environment

The PowerDesigner data model provides various means for customizing and controlling your modeling environment.

Setting CDM/LDM Model Options

You can set CDM/LDM model options by selecting **Tools > Model Options** or right-clicking the diagram background and selecting **Model Options**.

You can set the following options on the **Model Settings** page:

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 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 (see *Requirements Modeling*).
Enforce non-divergence	Specifies that attributes attached to a domain must remain synchronized with the selected properties (see *Controlling Non-Divergence from a Domain* on page 167).
Use data type full name	Specifies that the complete data type is displayed in entity symbols.
Default data type	Specifies the default data type to apply to domains and attributes if none is selected for them.
External Shortcut Properties	Specifies the properties that are stored for external shortcuts to objects in other models for display in property sheets and on symbols. By default, **All** properties appear, but you can select to display only **Name/Code** to reduce the size of your model. **Note:** This option only controls properties of external shortcuts to models of the same type (PDM to PDM, EAM to EAM, etc). External shortcuts to objects in other types of model can show only the basic shortcut properties.

Option	Description
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 23
Unique code	Requires that data items or relationships have unique codes
Allow n-n relationships	[LDM only] Allows n-n relationships to be displayed.
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.

For information about controlling the naming conventions of your models, see *Core Features Guide > Modeling with PowerDesigner > Objects > Naming Conventions*.

Assertion Template

The assertion template is a GTL template used to automatically generate sentences from the role names you specify on the **Cardinalities** tab of relationship property sheets. To review or edit the template, select **Tools > Model Options > Assertion Template**.

The PowerDesigner Generation Template Language (GTL) is used to generate text from the objects, properties, and relationships defined in the PowerDesigner metamodel and in extensions to it.

The GTL code in the template extracts various properties of the relationship object and the entities it connects to generate the assertion statements. The mandatory property and cardinalities are evaluated in each direction in order to generate the appropriate wording around the entity and role names.

You can edit the assertion template as necessary, to change the wording or to reference other properties. To reference extended attributes or other extensions, you must specify the extension file for the template to use in the **Assertion Extension** list.

A sample extension file, `Relationship Assertion with Plural Entity Names`, is provided, which provides support for using plural entity names in assertions. For information about attaching this or any other xem to your model, see *Extending Your Modeling Environment* on page 20

For detailed information about working with GTL, see *Customizing and Extending PowerDesigner > Customizing Generation with GTL* .

Migration Settings (LDM)

To set migration settings, select **Tools > Model Options**, and select the Migration settings sub-category under **Model Settings**.

These options control the migration of identifiers along relationships:

Option	Description
Migrate attribute properties	Enables the domain, the checks or the rules to be kept when an attribute is migrated.

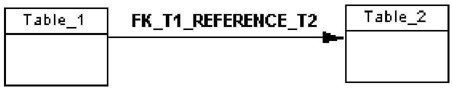
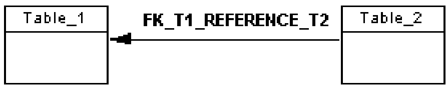
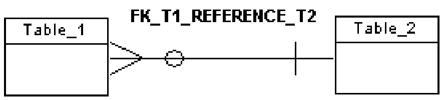
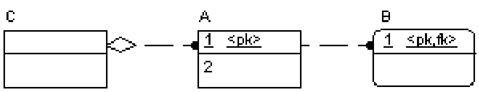
Option	Description
Foreign attribute name	Specifies the naming convention for migrated foreign identifiers. You can select one of the default templates from the list or enter your own using the following variables: • %PARENT% - Name/Code of the parent entity • %ATTRIBUTE% - Name/Code of the parent attribute • %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: • Always use template. • Only use template in case of conflict.

Setting PDM Model Options

You can set PDM model options by selecting **Tools** > **Model Options** or right-clicking the diagram background and selecting **Model Options**.

You can set the following options on the **Model Settings** page:

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 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 (see *Requirements Modeling*).

Option	Function
External Shortcut Properties	Specifies the properties that are stored for external shortcuts to objects in other models for display in property sheets and on symbols. By default, **All** properties appear, but you can select to display only **Name/Code** to reduce the size of your model. **Note:** This option only controls properties of external shortcuts to models of the same type (PDM to PDM, EAM to EAM, etc). External shortcuts to objects in other types of model can show only the basic shortcut properties.
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.

For information about controlling the naming conventions of your models, see *Core Features Guide > Modeling with PowerDesigner > Objects > Naming Conventions*.

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.

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 (see *Controlling Non-Divergence from a Domain* on page 167).
Default data type	Specifies the default data type to be applied to columns and domains if none is selected for them.
Column / Domain: Mandatory by default	Specifies that columns or domains are created, by default, as mandatory and that they may must, therefore contain non-null values.

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.

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 / Auto-migrate columns	Enable the reuse of columns in child tables as foreign key columns and the migration of primary key columns to child tables during the creation of references (see *Automatic Reuse and Migration of Columns* on page 178).
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 joins are created (see *Automatic Reuse and Migration of Columns* on page 178).

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 Properties* on page 175.
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

Other Object Model Options

To set model options for tables and views, indexes, join indexes, procedures, sequences, triggers, and database packages select **Tools > Model Options**, and select the appropriate sub-category under **Model Settings**.

You can set the following options for these objects:

Option	Function
Default owner	Specifies a default owner for the specified object from the list of users (see *Creating a User* on page 150). 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.

Option	Function
Ignore identifying owner	[tables and views] 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.
Rebuild automatically triggers	[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 If this option is not selected, you can manually instruct PowerDesigner to rebuild triggers at any time by selecting **Tools > Rebuild Objects > Rebuild Triggers**.

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 data model display preferences, select **Tools > Display Preferences** or right-click the diagram background and select **Display Preferences** from the contextual menu.

For detailed information about customizing and controlling the attributes and collections displayed on object symbols, see *Core Features Guide > Modeling with PowerDesigner > Diagrams, Matrices, and Symbols > Display Preferences*.

Viewing and Editing the DBMS Definition File

Each PDM is linked to a definition file that extends the standard PowerDesigner metamodel to provide objects, properties, data types, and generation parameters and templates specific to the language being modeled. Definition files and other resource files are XML files located in the `Resource Files` directory inside your installation directory, and can be opened and edited in the PowerDesigner Resource Editor.

Warning! The resource files provided with PowerDesigner inside the `Program Files` folder cannot be modified directly. To create a copy for editing, use the **New** tool on the resource file list, and save it in another location. To include resource files from different locations for use in your models, use the **Path** tool on the resource file list.

To open your model's definition file and review its extensions, select **Database > Edit Current DBMS**.

For detailed information about the format of these files, see *Customizing and Extending PowerDesigner > DBMS Definition Files*.

Note: 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 DBMS

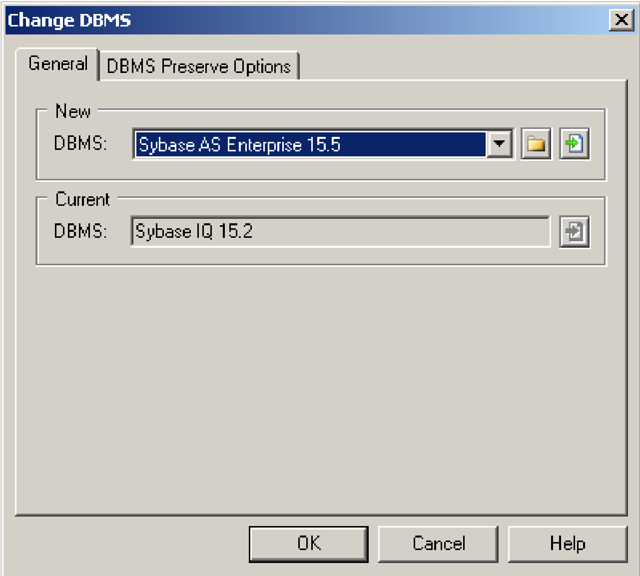
You can change the **DBMS** being modeled in your PDM at any time.

If you change the DBMS being modeled, the model will 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.
- Any objects not supported by the new DBMS will be deleted
- Certain objects, whose behavior is heavily DBMS-dependent may lose their values.

Note: You may be required to change the DBMS if you open a model and the associated definition file is unavailable.

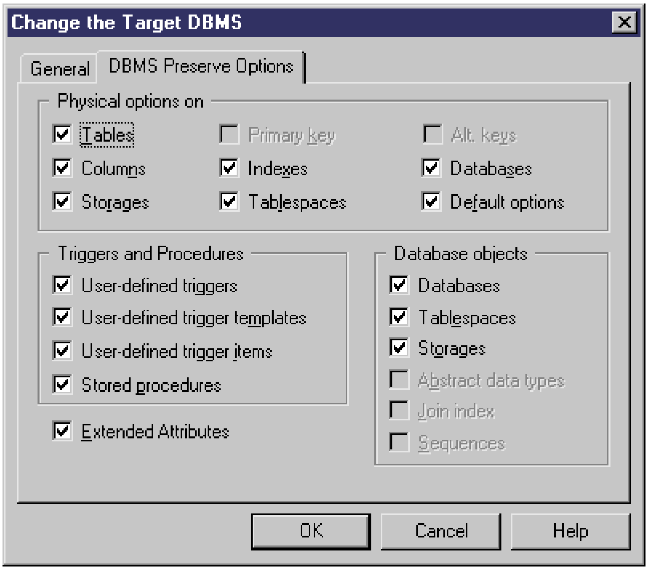
1. Select **Database > Change Current DBMS**:



2. Select a **DBMS** from the list.

   By default, PowerDesigner creates a link in the model to the specified file. To copy the contents of the resource and save it in your model file, click the **Embed Resource in Model** button to the right of this field. Embedding a file in this way enables you to make changes specific to your model without affecting any other models that reference the shared resource.

3. [optional] Click the **DBMS Preserve Options** tab, and select the check boxes for the objects and options that you want to preserve:
   - Triggers and stored procedures – 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.
   - DBMS-specific objects - databases, storages, tablespaces, abstract data types, sequences.
   - Extended attributes - which are defined for a particular DBMS.



Note: If you are changing DBMS 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 DBMSs are disabled.

4. Click **OK**.

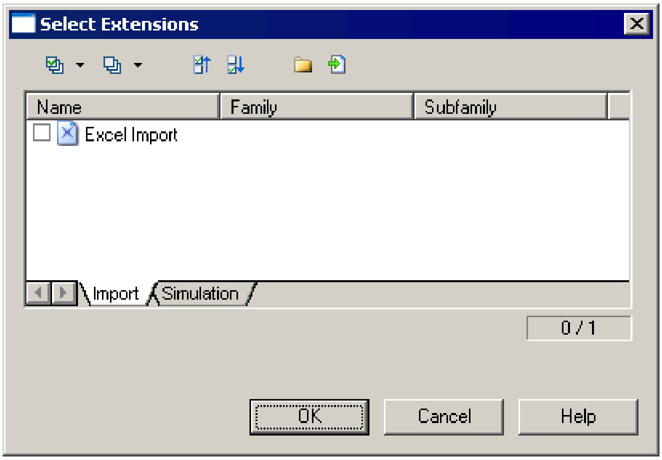
   A message box opens to tell you that the DBMS has been changed.
5. Click **OK** to return to the model.

Extending your Modeling Environment

You can customize and extend PowerDesigner metaclasses, parameters, and file generation with extensions, which can be stored as part of your model or in separate extension files (*.xem) for reuse with other models.

To access extension defined in a `*.xem` file, simply attach the file to your model. You can do this when creating a new model by clicking the **Select Extensions** button at the bottom of the New Model dialog, or at any time by selecting **Model > Extensions** to open the List of Extensions and clicking the **Attach an Extension** tool.

In each case, you arrive at the Select Extensions dialog, which lists the extensions available, sorted on sub-tabs appropriate to the type of model you are working with:



To get started extending objects, see *Core Features Guide > Modeling with PowerDesigner > Objects > Extending Objects*. For detailed information about working with extensions, see *Customizing and Extending PowerDesigner > Extension Files*.

Linking Objects with Traceability Links

You can create traceability links to show any kind of relationship between two model objects (including between objects in different models) via the **Traceability Links** tab of the object's property sheet. These links are used for documentation purposes only, and are not interpreted or checked by PowerDesigner.

For more information about traceability links, see *Core Features Guide > Linking and Synchronizing Models > Getting Started with Linking and Syncing > Creating Traceability Links*.

CHAPTER 2 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 Diagrams* on page 27) or an LDM (see *Logical Diagrams* on page 39) to analyze your system and then generate a PDM (see the *Chapter 3, Physical Diagrams* on page 73) 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.

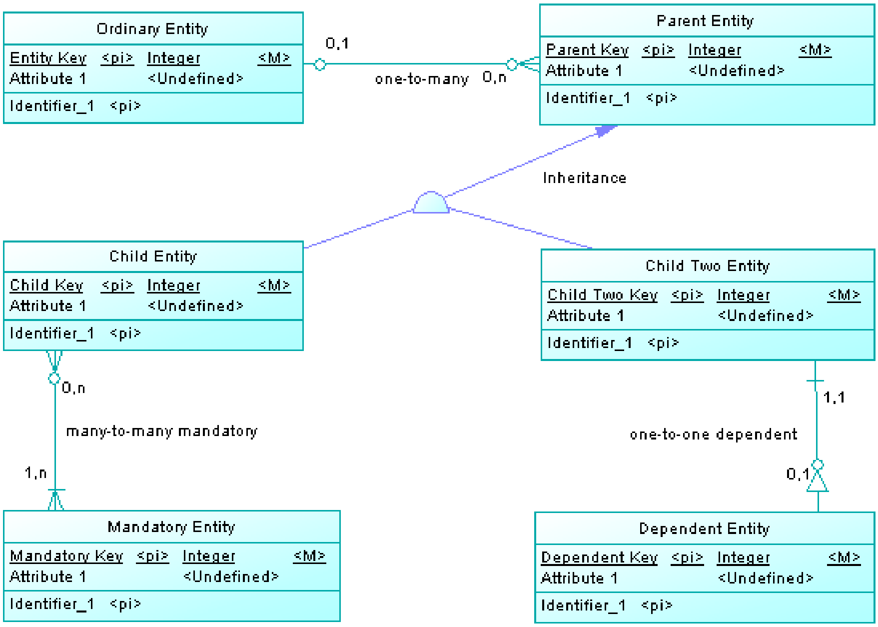
For more information about intermodel generation, see *Chapter 7, Generating Other Models from a Data Model* on page 323.

Supported CDM/LDM Notations

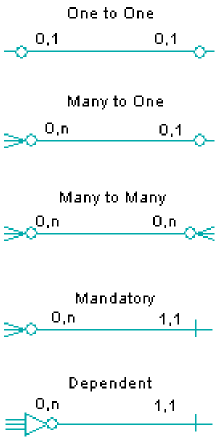
PowerDesigner supports the most popular data modeling notations in the CDM and LDM. You can choose your notation by clicking **Tools > Model Options** and selecting it in the **Notation** list.

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:



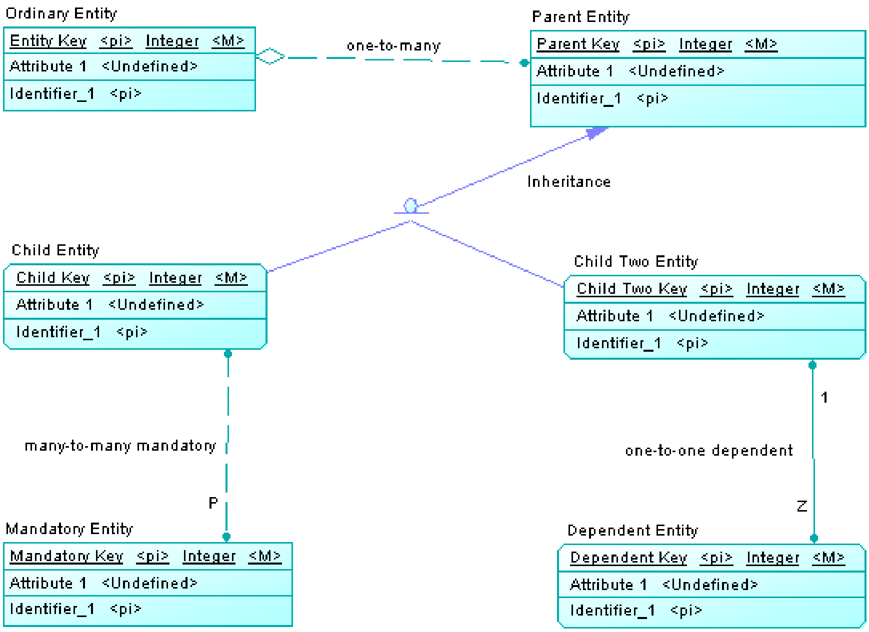
(Note that the Merise notation uses associations instead of relationships):

Inheritance symbols indicate if they are complete and if they have mutually exclusive children:

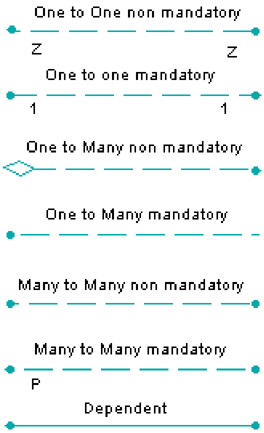
Complete	Mutually exclusive	Symbol
No	No	
Yes	No	
No	Yes	
Yes	Yes	

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:

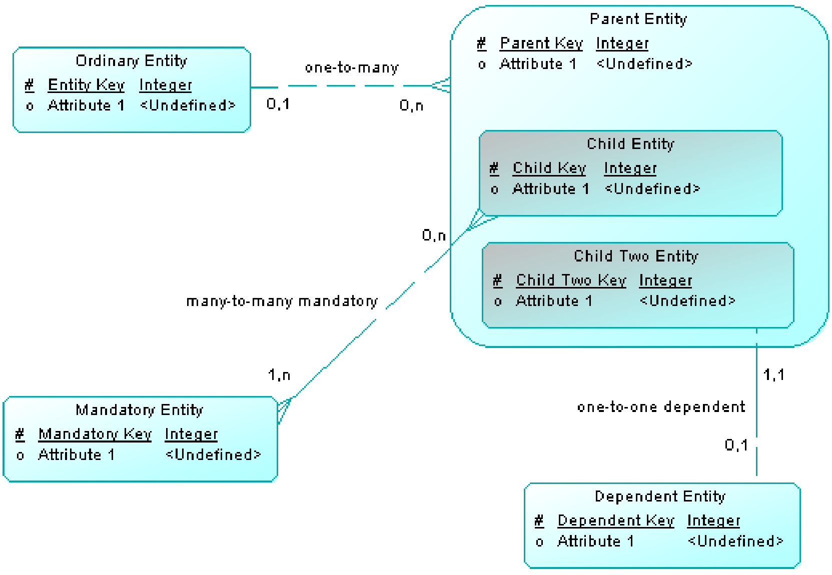


Inheritance symbols indicate if the inheritance is complete:

Complete	Symbol
Yes	
No	

Barker Notation

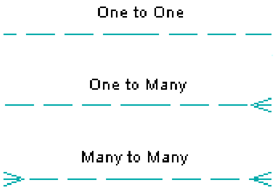
In the Barker notation, entities are drawn with round corners, and inheritances are displayed by placing children inside the parent entity.



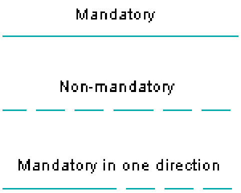
Only attributes are listed and a symbol specifies whether each attribute is a key, a mandatory or an optional attribute as follows:

- # Primary
- * Mandatory
- o Optional

Relationship symbols indicate the cardinality as follows:



The line style specifies if a relationship is mandatory:

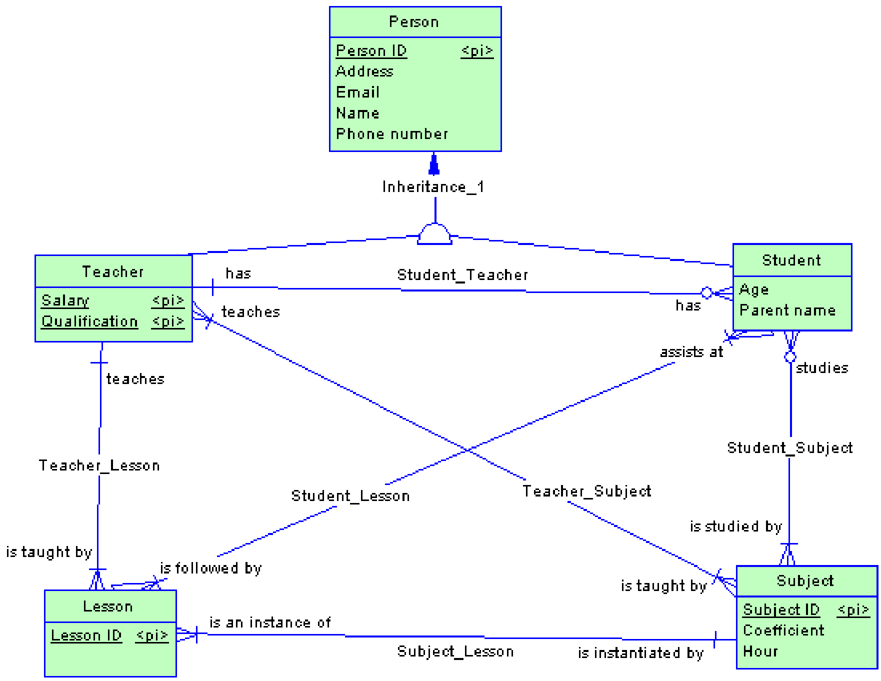


Conceptual Diagrams

A *conceptual data diagram* provides a graphical view of the conceptual structure of an information system, and helps you identify the principal entities to be represented, their attributes, and the relationships between them.

Note: To create a conceptual diagram in an existing CDM, right-click the model in the Browser and select **New** > **Conceptual Diagram**. To create a new model, select **File** > **New Model**, choose Conceptual Data Model as the model type and **Conceptual Diagram** as the first diagram, and then click **OK**.

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).



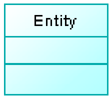
In addition:

- 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

PowerDesigner supports all the objects necessary to build conceptual diagrams.

Object	Tool	Symbol	Description
Domain	[none]	[none]	Set of values for which a data item is valid. See *Domains (CDM/LDM/PDM)* on page 162.
Data Item	[none]	[none]	Elementary piece of information. See *Data Items (CDM)* on page 42.

Object	Tool	Symbol	Description
Entity			Person, place, thing, or concept that is of interest to the enterprise. See *Entities (CDM/LDM)* on page 45.
Entity Attribute	[none]	[none]	Elementary piece of information attached to an entity. See *Attributes (CDM/LDM)* on page 50.
Identifier	[none]	[none]	One or many entity attributes, whose values uniquely identify each occurrence of the entity. See *Identifiers (CDM/LDM)* on page 52.
Relationship			Named connection or relation between entities (ER modeling methodology). See *Relationships (CDM/LDM)* on page 53.
Inheritance			Relationship that defines an entity as a special case of a more general entity. See *Inheritances (CDM/LDM)* on page 67.
Association			Named connection or association between entities (Merise modeling methodology). See *Associations and Association Links (CDM)* on page 62.
Association Link			Link that connects an association to an entity. See *Associations and Association Links (CDM)* on page 62.

Example: Building a Data Dictionary in a CDM

PowerDesigner supports the definition and maintenance of an enterprise data dictionary in a CDM. A data dictionary defines the data items, entities and attributes of the enterprise, and by managing it in a CDM and linking it (through generation or through the mapping editor) with your data and other models, you can ensure consistency of use and benefit from sophisticated impact analysis and "where used" reporting.

Data dictionaries ensure consistency of use by providing a single authoritative definition for all common data elements used across the enterprise. They are used to standardize data content, context, and definitions and to achieve consistency and reusability while increasing the quality of the data used throughout the organization. By clearly defining and delineating the objects that comprise the enterprise and its systems, they enable:

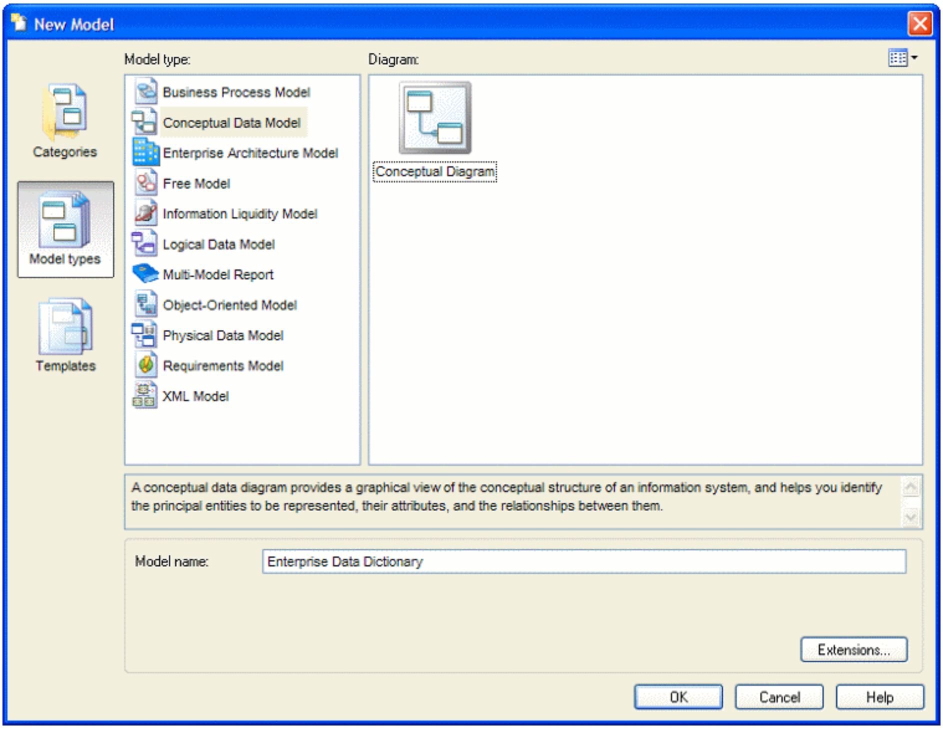
- easier integration and communication between systems

- more standardized messaging between applications
- higher quality business intelligence and analytics
- better understanding between all subject matter experts
- more agile response to change and more complete impact analysis

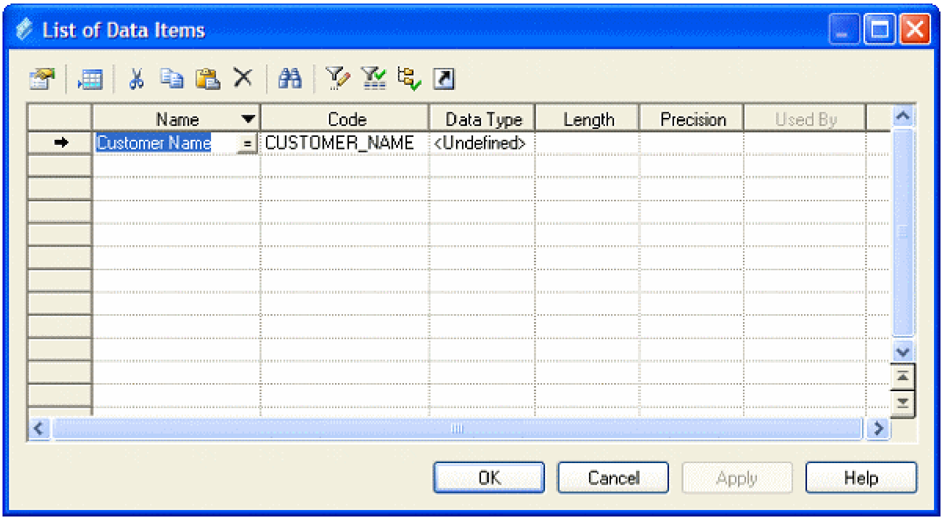
A data dictionary defined in a PowerDesigner CDM provides:

- a unique list of entities and data items
- data items as descriptions of data artifacts
- entities connected to data items through attributes
- entity-to-entity relationships
- traceability from the data dictionary to logical and physical data models and other models
- impact analysis and "where used" reporting capabilities

1. Select **File** > **New** to open the New Model dialog, select to create a new CDM and give it an appropriate name, for example, `Enterprise Data Dictionary`.



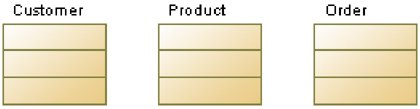
2. Select **Model** > **Data Items** to open the List of Data Items and enter some concepts that you want to define. Each data item is an elementary piece of information, which represents a fact or a definition defined using business terms.



Some examples of data items are `Customer Name`, `Order Description`, and `Zip Code`. Data items exist independently of any containing entity, which is important in a data dictionary as you are seeking to define atomic business data and terms, independent of how they may ultimately be used by entities. For more information about defining data items, see *Data Items (CDM)* on page 42.

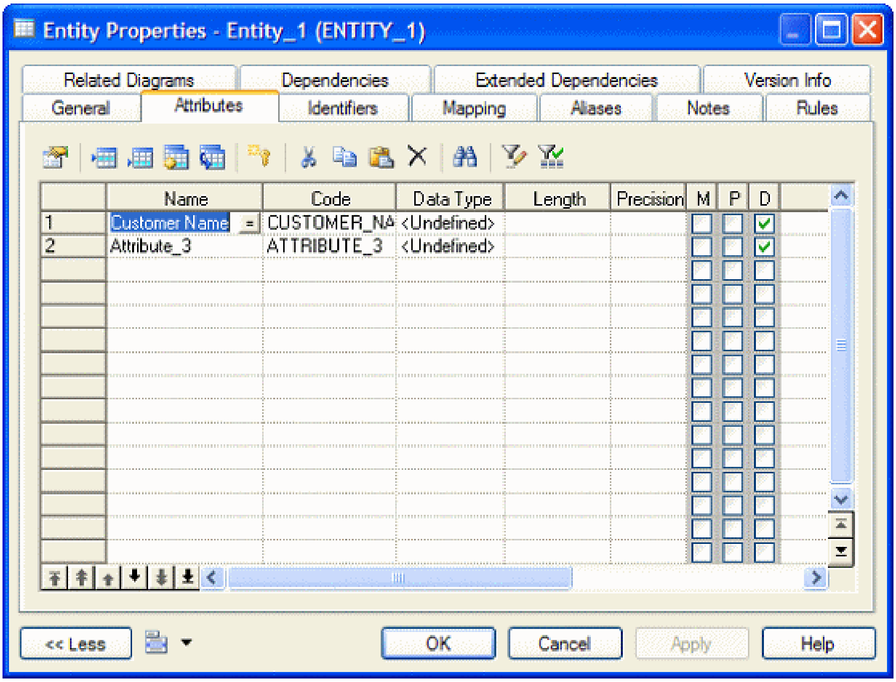
3. Select **Model > Entities** to open the List of Entities and enter some of the entities that you want to define. Entities represent more complex business structures composed of one or more attributes (which are associated with data items).

   Some examples of entities are `Customer`, `Product`, `Order`. When you create entities, a symbol for each one will be created in the CDM diagram. While such a graphical representation is not strictly necessary for the purposes of creating a data dictionary, you may find this diagram useful to help you visualize the content and structure of business concepts.



   For more information about defining entities, see *Entities (CDM/LDM)* on page 45.

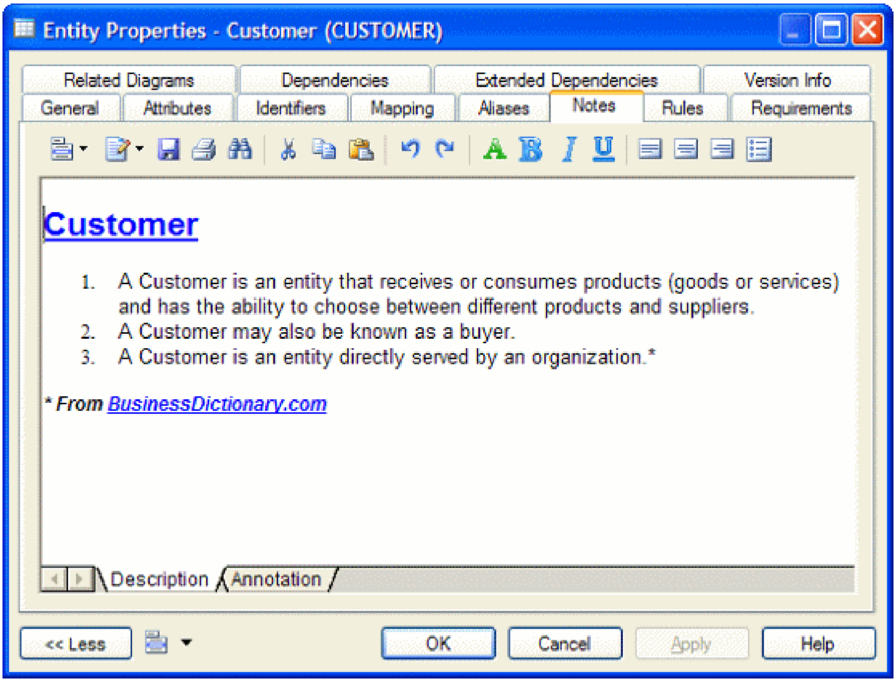
4. Double-click an entity in the Browser or diagram to open its property sheet, and click the **Attributes** tab. Entity attributes provide the link between an entity and a data item:



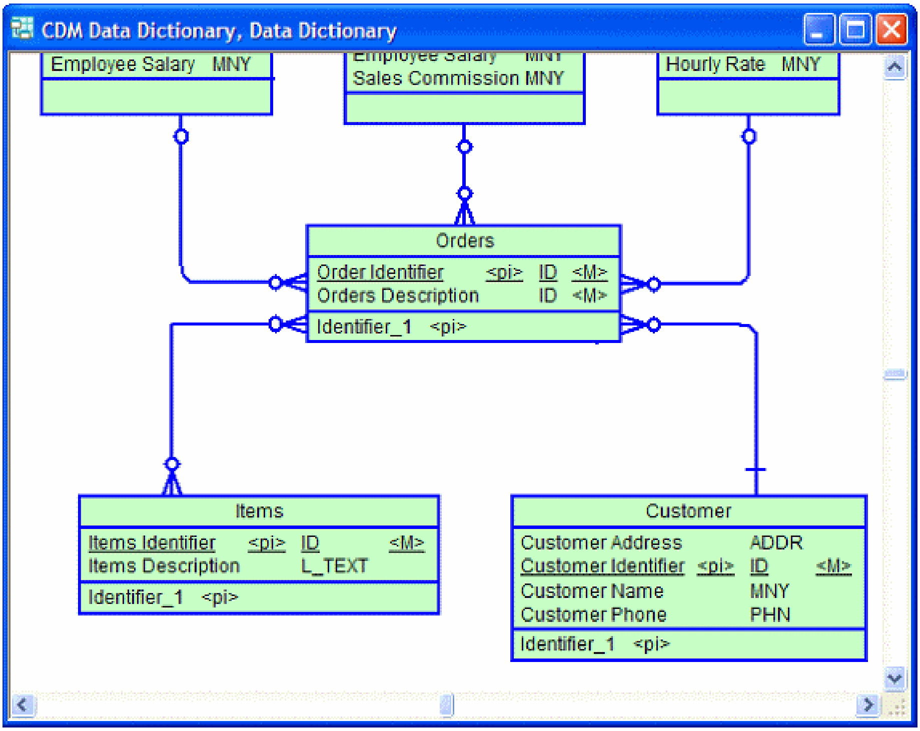
Create a new attribute by reusing an existing data item by clicking the **Reuse Data Item** tool and selecting the data item that you want to reuse. By default, PowerDesigner allows you to reuse a data item for more than one entity attribute so that, for example, you can define a `Zip Code` data item once, and reuse it in whatever entities contain addresses. If you then update the data item, your changes will simultaneously cascade down to all the entity attributes that use it. This is a great way to enforce consistency throughout the data dictionary model.

You can also create data items in this list by clicking the **Insert a Row** or **Add a Row** tool to add a new line in the list and entering an appropriate name. PowerDesigner will create the attribute and an associated data item. You can also create a new attribute by creating a copy of an existing data item. Click the **Add Data Item** tool and select the data item that you want to copy. Any changes made through this attribute or directly to this copy of the data item will only affect this attribute and no others.

5. Double-click one of your entity symbols (or its Browser entry) to open its property sheet so that you can provide a precise definition for it. The **Comment** field on the **General** tab is intended for a simple, short description in unformatted text, while the **Description** field on the **Notes** tab allows for fully formatted text, and is intended to contain the complete, detailed definition from the business:



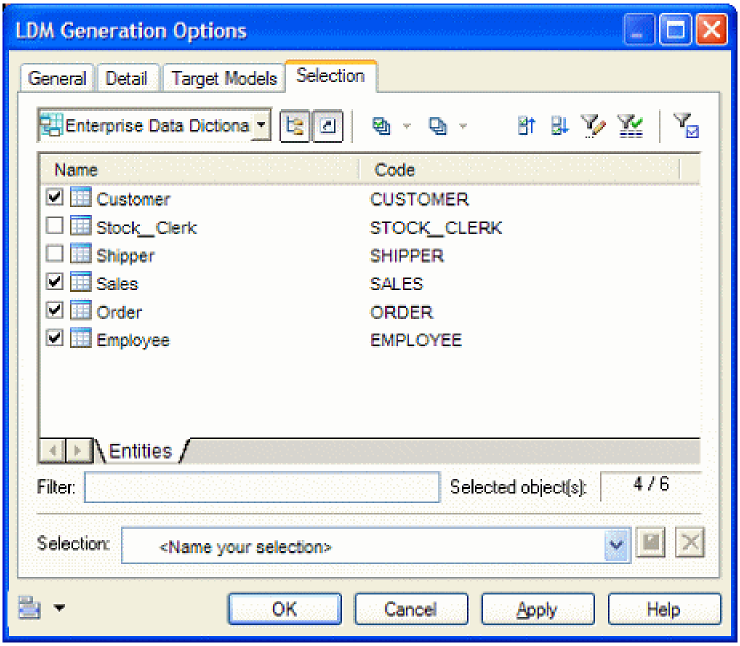
6. [optional] Select the **Relationship** tool in the pallet and create relationships between the entities in your data dictionary. Click and hold in one entity, then drag the cursor to a second entity and release the mouse button. Draw other relationships as necessary and then right-click anywhere in the diagram to drop the tool. Double-click a relationship line to open its property sheet and specify properties such as role name and cardinality.



For detailed information about defining relationships, see *Relationships (CDM/LDM)* on page 53.

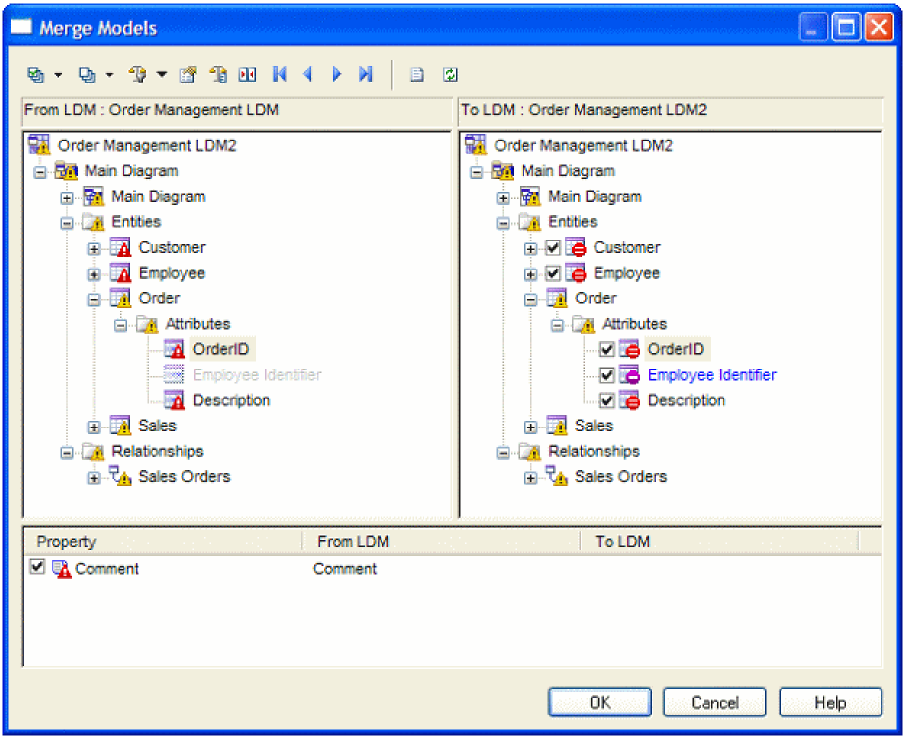
7. The purpose of a data dictionary is to map the concepts that it defines to the concepts, logical entities, and physical tables that make up the implementation of these ideas in the enterprise. PowerDesigner provides two complementary methods for connecting the data dictionary with your other models:

   - Generation - If you have no existing PDM, you can generate a new model from your data dictionary. Click **Tools** > **Generate Physical Data Model** to open the Generate dialog, select the **Generate new...** option, and specify a name for the model to generate. Click the **Selection** tab and select the concepts you want to generate to the new model, and then click **OK**.



You can review the links created between the data dictionary and your other models in the Generation Links Viewer (select **Tools > Generation Links > Derived Models**).

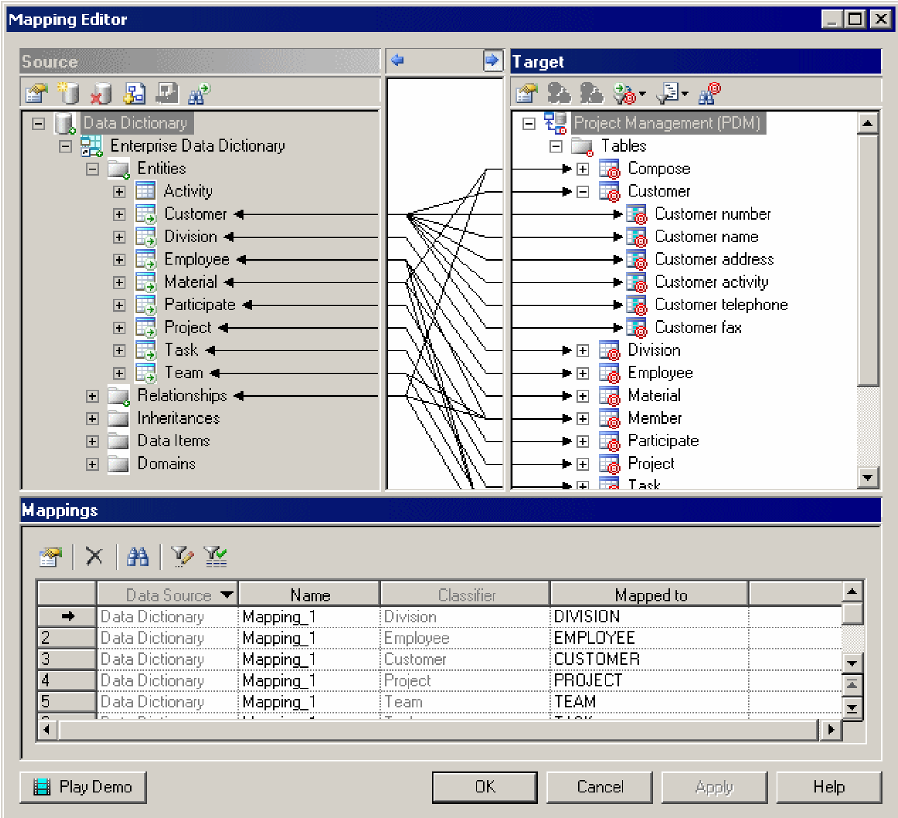
You can regenerate whenever necessary to propagate updates or additions in the data dictionary to your other models. The Merge Models dialog (see *Core Features Guide > Modeling with PowerDesigner > Comparing and Merging Models*) will appear, which lets you review and approve (or reject) the changes that will be propagated from the data dictionary to the model.



For detailed information about generating models, see *Chapter 7, Generating Other Models from a Data Model* on page 323.

- Mapping Editor - If you have an existing PDM or other model it may be more appropriate to map your data dictionary concepts to your PDM objects using the Mapping Editor, which provides a finer degree of control and a simple drag and drop interface.

  Open the model containing the objects you want to link with your data dictionary and select **Tools > Mapping Editor**. In the Data Source Creation Wizard, enter `Data Dictionary` in the **Data Source** field, select `Conceptual Model` in the **Model type** list, and click **Next**. Select your data dictionary CDM and click **Next**. Select the **Create default mapping** option to instruct PowerDesigner to auto-create mappings where possible based on shared names, and click **Finish** to open your model and the data dictionary in the Mapping Editor:

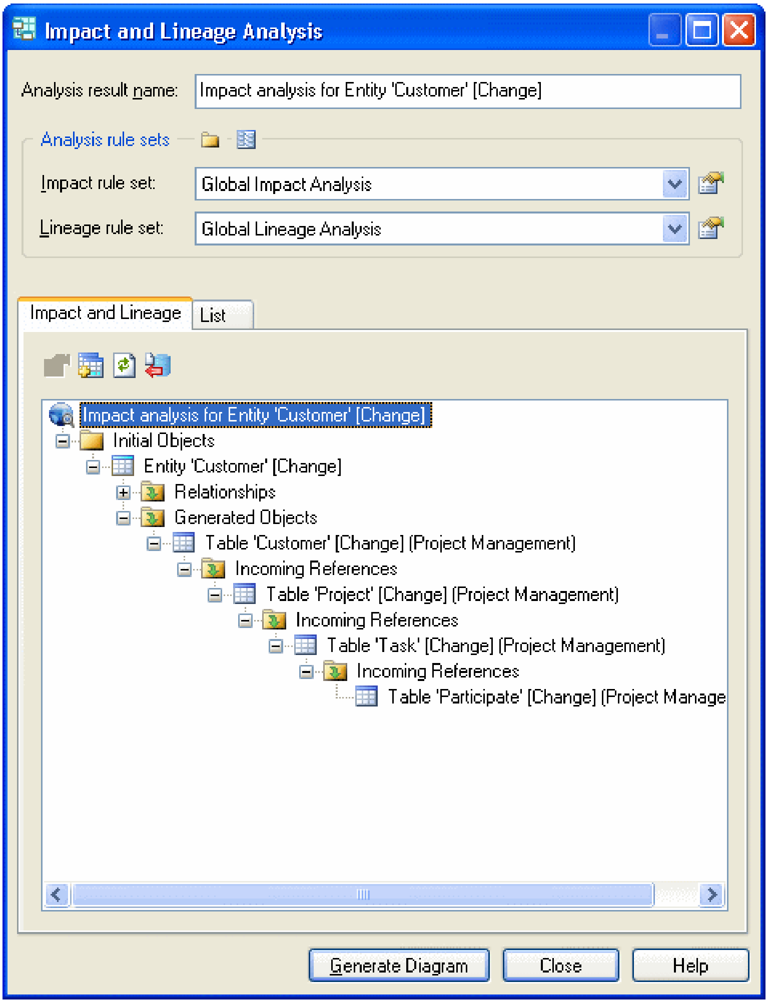


You can create additional mappings as necessary by dragging and dropping entities and attributes from the data dictionary onto objects in the target model. Note that mappings created in this way will not automatically propagate changes.

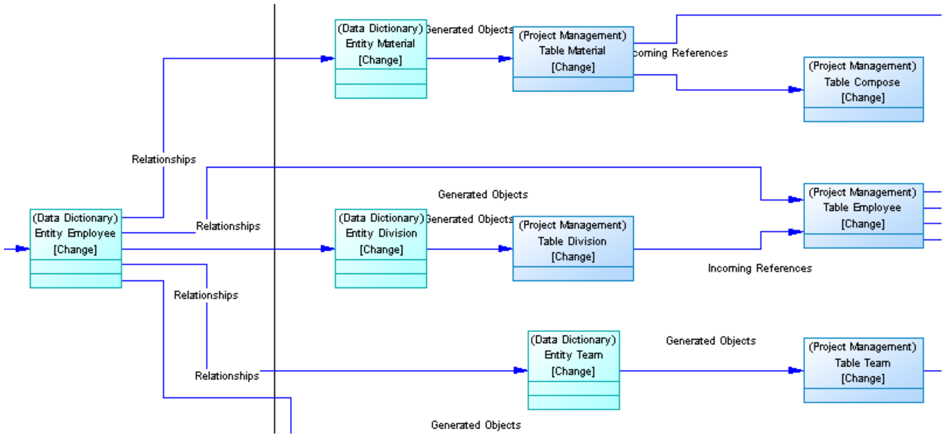
For detailed information about using the Mapping Editor, see *Core Features Guide > Linking and Synchronizing Models > Object Mappings*.

8. Once the data dictionary is established and linked to the other models used in the enterprise to define the information architecture, you will need to manage changes to it. New concepts will be added and existing elements updated due to refinements in understanding the business or changes to business operations. Some elements may also be removed (though this will probably be rare). Maintaining your data dictionary in a PowerDesigner CDM enables you to leverage sophisticated impact analysis tools to help you understand the time, cost and risk associated with proposed changes.

To launch an impact analysis, select one or more objects in a diagram or the Browser and select **Tools > Impact and Lineage Analysis**:



You can edit the rule sets used to control the analysis and manually adjust the tree view by right-clicking items. Once the analysis view contains the level of detail you want, click the **Generate Diagram** button to create an impact analysis diagram. This diagram, which can be saved and compared to other impact analysis snapshots, shows the connections that link your dictionary concepts through intermediate objects and models to the physical objects that implement them, providing a graphical "where used" report:



The diagram helps you plan the implementation of a change, as everything defined in the diagram will require further assessment to ensure the change does not invalidate any specific work we have done at the implementation level.

For detailed information about working with impact analysis, see *Core Features Guide > Linking and Synchronizing Models > Impact and Lineage Analysis*.

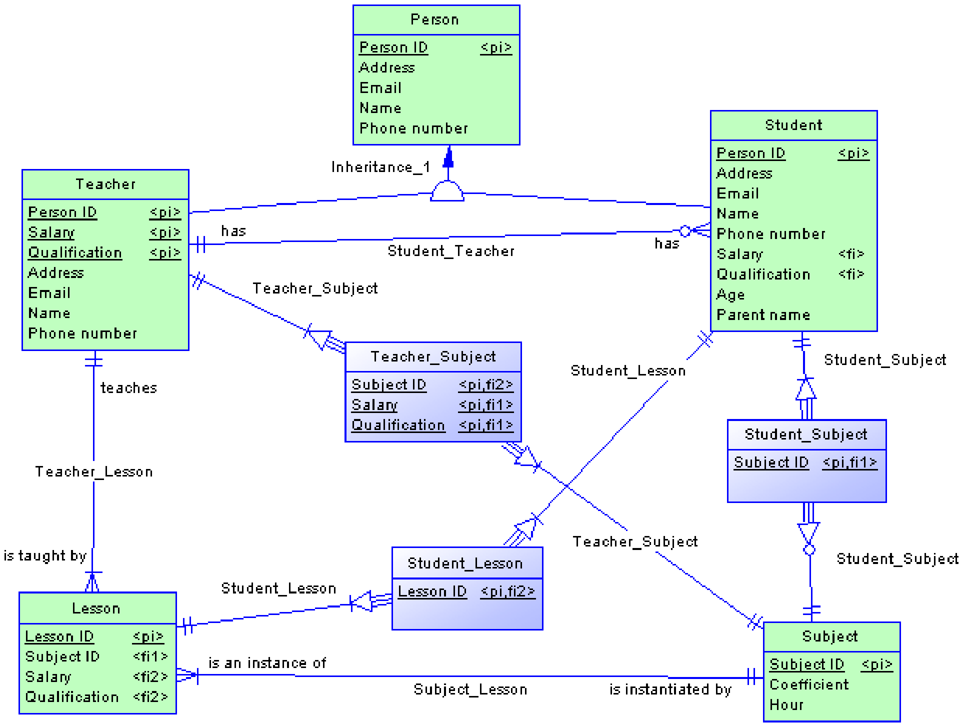
9. Share your data dictionary with your modeling team and ensure that the latest version is always available to them, by checking it into your PowerDesigner repository library as a reference model (see *Core Features Guide > Administering PowerDesigner > Deploying an Enterprise Glossary and Library*).
10. Share your data dictionary with other members of your organization through the PowerDesigner Portal (see *Core Features Guide > Storing, Sharing and Reporting on Models > The PowerDesigner Portal*) or by publishing it to HTML or RTF (see *Core Features Guide > Storing, Sharing and Reporting on Models > Reports*).

Logical Diagrams

A *logical data diagram* provides a graphical view of the structure of an information system, and helps you analyze the structure of your data system through entities and relationships, in which primary identifiers migrate along one-to-many relationships to become foreign identifiers, and many-to-many relationships can be replaced by intermediate entities.

Note: To create a logical diagram in an existing LDM, right-click the model in the Browser and select **New > Logical Diagram**. To create a new model, select **File > New Model**, choose Logical Data Model as the model type and **Logical Diagram** as the first diagram, and then click **OK**.

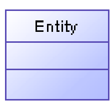
The following logical diagram represent the same system as that in our CDM example (see *Conceptual Diagrams* on page 27).

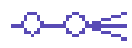
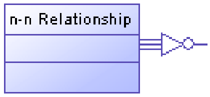


Primary identifiers have migrated along one-to-many relationships to become foreign identifiers, and many-to-many relationships are replaced with an intermediary entity linked with one-to-many relationships to the extremities.

Logical Diagram Objects

PowerDesigner supports all the objects necessary to build logical diagrams.

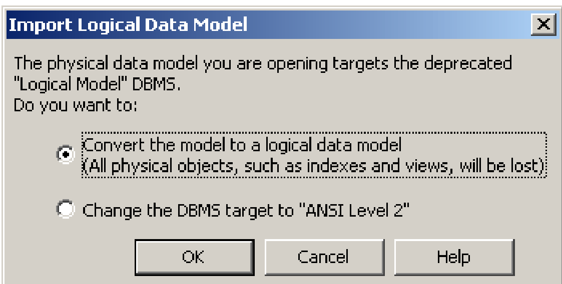
Object	Tool	Symbol	Description
Domain	[none]	[none]	Set of values for which a data item is valid. See *Domains (CDM/LDM/PDM)* on page 162.
Entity			Person, place, thing, or concept that is of interest to the enterprise. See *Entities (CDM/LDM)* on page 45.
Entity Attribute	[none]	[none]	Elementary piece of information attached to an entity. See *Attributes (CDM/LDM)* on page 50.

Object	Tool	Symbol	Description
Identifier	[none]	[none]	One or many entity attributes, whose values uniquely identify each occurrence of the entity. See *Identifiers (CDM/LDM)* on page 52.
Relationship			Named connection or relation between entities (ER modeling methodology). See *Relationships (CDM/LDM)* on page 53.
n-n Relationship			[LDM only] Named cardinality represented with an intermediary entity. See *Relationships (CDM/LDM)* on page 53.
Inheritance			Relationship that defines an entity as a special case of a more general entity. See *Inheritances (CDM/LDM)* on page 67.

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.

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



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

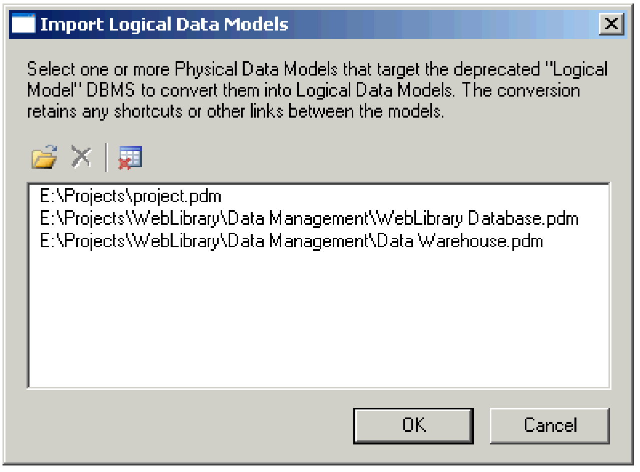
4. Click OK to open the model.

Note: 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 *Core Features Guide > Linking and Synchronizing Models > Generating Models and Model Objects*).

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.

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



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 45) in order to create an entity attribute (see *Attributes (CDM/LDM)* on page 50), 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 from the Browser or **Model** menu. Data items are automatically created when you create entity attributes.

- Select **Model > Data Items** to access the List of Data Items, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New > Data Item**.
- Create an entity attribute (see *Attributes (CDM/LDM)* on page 50). A data item will be automatically created.

For general information about creating objects, see *Core Features Guide > Modeling with PowerDesigner > Objects*.

Data Item Properties

To view or edit a data item's properties, double-click its Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

Property	Description
Name/Code/ Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.

Property	Description
Data type/ Length/ Precision	Specifies the form of data to be stored, such as numeric, alphanumeric, or Boolean, and, where appropriate, the maximum number of characters or numerals that can be stored, and the maximum number of places after the decimal point. Click the ellipsis button to choose from the list of standard data types (see *PowerDesigner Standard Data Types* on page 165).
Domain	Specifies the domain associated with the object (see *Domains (CDM/LDM/PDM)* on page 162). Use the tools to the right of this field to create or browse to a domain, or to open the property sheet of the selected domain.
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

The following tabs are also available:

- **Standard Checks** - Specifies constraints to control the range and format of permitted data (see *Setting Data Profiling Constraints* on page 94)
- **Additional Checks** - Displays an editable SQL statement, initialized with the standard checks, which can be used to generate more complex constraints (see *Specifying Advanced Constraints* on page 97).
- **Rules** - Lists the business rules associated with the object (see *Business Rules (CDM/ LDM/PDM)* on page 184).

Controlling Uniqueness and Reuse of Data Items

You can control naming restraints and reuse for data items with CDM model options, by selecting **Tools** > **Model Options** .

Option	When selected	When cleared
Unique code	Each data item must have a unique code. If you try to select this option and some existing data items are already sharing a code, the following error will be displayed: Unique Code option could not be selected because two data items have the same code: *data_item_code* To be able to select the option, you must first assign unique codes to all data items.	Multiple 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. **Note:** 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.
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

For more information about CDM model options, see *Setting CDM/LDM Model Options* on page 10.

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 from the Toolbox, Browser, or **Model** menu.

- Use the **Entity** tool in the Toolbox.
- Select **Model > Entities** to access the List of Entities, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New > Entity**.

For general information about creating objects, see *Core Features Guide > Modeling with PowerDesigner > Objects*.

Entity Properties

To view or edit an entity's properties, double-click its diagram symbol or Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

Property	Description
Name/Code/ Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
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. When modeling in the Barker notation (see *Supported CDM/LDM Notations* on page 23), only leaf subtypes can be generated as PDM tables, and so this option is disabled on Barker supertype property sheets.

Property	Description
Parent Entity	[read-only] Specifies the parent entity. Click the Properties tool at the right of the field to open the parent property sheet.
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

The following tabs are also available:

- Attributes - lists the attributes associated with the entity (see *Attributes (CDM/LDM)* on page 50).
- Identifiers - lists the attributes associated with the entity (see *Identifiers (CDM/LDM)* on page 52).
- Rules - lists the business rules associated with the entity (see *Business Rules (CDM/LDM/PDM)* on page 184).
- Subtypes – [Barker only] lists the subtypes that inherit from the entity.

Copying Entities

You can make a copy of an entity within the same model or between models. When you copy an entity, you create a new entity with a new name and code, attributes, and identifiers. Model options control whether you create new data items or reuse the data items that are attached to the original entity.

1. Select an entity in the CDM/LDM, and then select **Edit > Copy** (or press **Ctrl+C**).
2. Select the diagram or model to where you want to copy the entity and select **Edit > Paste** (or press **Ctrl+V**).

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

   Note: When copying an entity to the same model, a new entity with a new name and code, attributes, and identifiers is always created, but the creation of new data items is controlled by data item model options (see *Setting CDM/LDM Model Options* on page 10). Select:

   - **Allow reuse** - to attach the original data items to the new entity attributes. If this option is not selected, the original data items will be copied and these copies will be attached to the new entity attributes.
   - **Unique code** - to force all data items to have unique codes (though two or more data items can have the same name). If neither this option nor **Allow reuse** is selected, then duplicate data items will be created with the same names and codes.

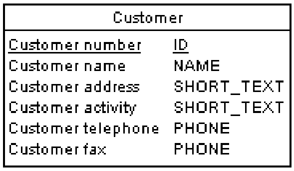
Displaying Attributes and Other Information on an Entity Symbol

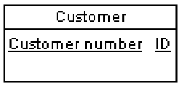
To set display preferences for entities, select **Tools > Display Preferences**, and select the Entity sub-category in the left-hand Category pane.

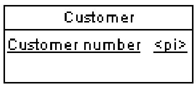
Entity

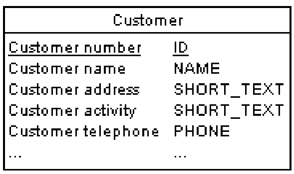
By default the following properties can be displayed on entity symbols:

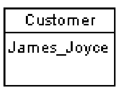
<table>
<tr><th>Preference</th><th>Display description</th></tr>
<tr><td>Attributes</td><td>Specifies whether Attributes are displayed on entity symbols. If selected, you can choose between displaying:

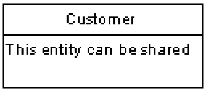
All attributes - All attributes:


Primary attributes - Only primary identifier attributes:


Identifying attributes - All identifier attributes:


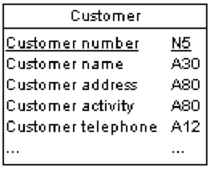
Display limit - Number of attributes shown depends on defined value. For example, if set to 5:


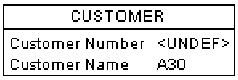
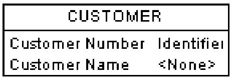
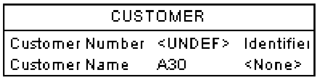
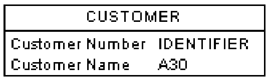
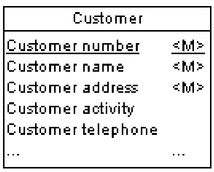
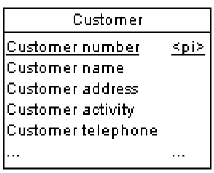
</td></tr>
<tr><td>Identifiers</td><td>All identifier attributes for the entity are listed at the bottom of the entity symbol:

</td></tr>
<tr><td>Stereotype</td><td>Stereotype of the entity.</td></tr>
</table>

Preference	Display description
Comment	Comment of the entity. When selected, all other check boxes are deselected, except for Stereotype: 

Entity Attributes

By default the following properties can be displayed for entity attributes:

Preference	Display description
Data type	Data type for each entity attribute: 
Domain or data type	Domain for each entity attribute. You can only display domains when the Data type check box is selected.

Preference	Display description
Domain	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:  • Domains - Displays only the domain, if any:  • Data types and Domain - Displays both data type and domain, if any:  • Data types and Replace by domains - Displays either data type or domain, if any, and domain if both are present: 
Mandatory	*<M>* indicators are displayed next to each mandatory attribute: 
Identifier indicators	*<pi>* indicators are displayed next to primary identifiers and *<ai>* indicators next to non-primary identifiers: 
Stereotype	Displays the stereotype of the entity attributes

Note: For information about selecting other properties to display, see *Core Features Guide > Modeling with PowerDesigner > Diagrams, Matrices, and Symbols > Display Preferences*.

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 from the **Attributes** tab in the property sheet of an entity, association, or inheritance.

You can use the following tools, available on the **Attributes** tab:

Tool	Description
	**Add a Row** – Creates a new attribute and associated data item. If you have enabled the **Allow Reuse** model option (see *Setting CDM/LDM Model Options* on page 10), 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 it 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.
	**Reuse Data Item** (CDM) - 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.

For general information about creating objects, see *Core Features Guide > Modeling with PowerDesigner > Objects*.

Attribute Properties

To view or edit an attribute's properties, double-click its Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

Property	Description
Name/Code/ Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
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 (see *Data Items (CDM)* on page 42). 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/ Length/ Precision	Specifies the form of data to be stored, such as numeric, alphanumeric, or Boolean, and, where appropriate, the maximum number of characters or numerals that can be stored, and the maximum number of places after the decimal point. Click the ellipsis button to choose from the list of standard data types (see *PowerDesigner Standard Data Types* on page 165).
Domain	Specifies the domain associated with the object (see *Domains (CDM/LDM/PDM)* on page 162). Use the tools to the right of this field to create or browse to a domain, or to open the property sheet of the selected domain.
Primary Identifier	[entity attributes only] Specifies that the attribute is the primary identifier of the entity.
Displayed	[entity and association attributes] Displays the attribute in the object symbol.
Mandatory	Specifies that every object occurrence must assign a value to the attribute. Identifiers (see *Identifiers (CDM/LDM)* on page 52) are always mandatory.
Foreign identifier	[LDM only, read-only] Specifies that the attribute is the foreign identifier of the entity.

The following tabs are also available:

- **Standard Checks** - Specifies constraints to control the range and format of permitted data (see *Setting Data Profiling Constraints* on page 94)
- **Additional Checks** - Displays an editable SQL statement, initialized with the standard checks, which can be used to generate more complex constraints (see *Specifying Advanced Constraints* on page 97).

- **Rules** - Lists the business rules associated with the object (see *Business Rules (CDM/LDM/PDM)* on page 184).

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 identifier from the property sheet of an entity.

- 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 *Core Features Guide > Modeling with PowerDesigner > Objects*.

Identifier Properties

To view or edit an indentifier's properties, double-click its Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

Property	Description
Name/Code/ Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
Entity	Specifies the name of the entity to which the identifier belongs.
Primary identifier	Specifies that the identifier is a primary identifier.
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

The following tabs are also available:

- Attributes - lists the attributes (see *Attributes (CDM/LDM)* on page 50) associated with the identifier: Click the Add Attributes tool to add an attribute.

Relationships (CDM/LDM)

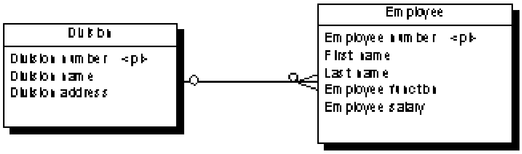
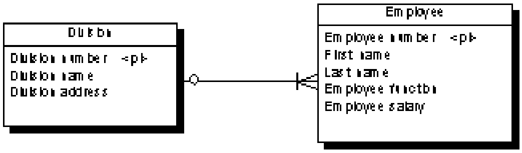
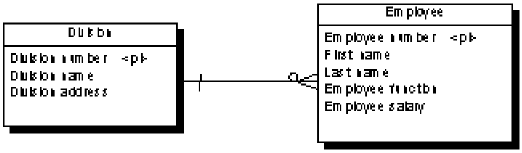
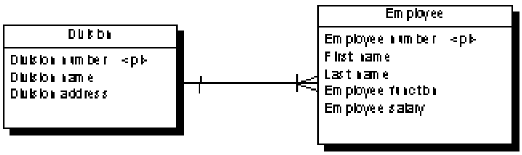
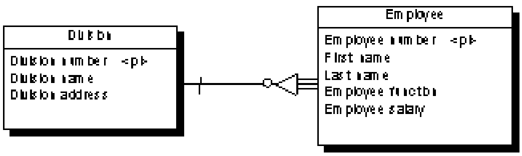
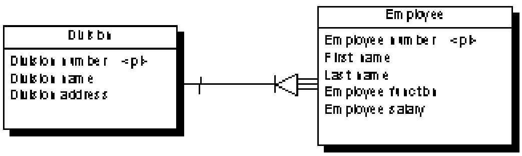
A relationship is a link between entities. For example, in a model that manages human resources, the `Member` relationship links the `Employee` and `Team` entities and expresses that each employee works in a team, and each team has employees.

For example, *the employee Martin works in the Marketing team* is one occurrence of the `Member` relationship.

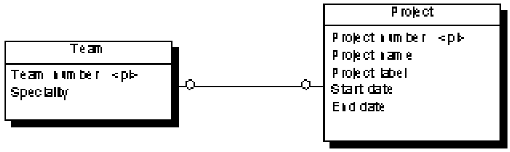
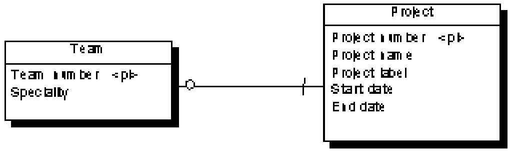
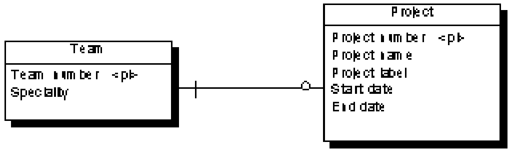
When you generate a PDM from a CDM or LDM, relationships are generated as references.

Note: Relationships are used to link entities in the ER, Barker, and IDEF1X methodologies, while Merise uses associations (see *Associations and Association Links (CDM)* on page 62). PowerDesigner lets you use relationships or associations exclusively, or combine the two methodologies in the same model. The following examples use the ER format. For more information about the other notations, see *Supported CDM/LDM Notations* on page 23.

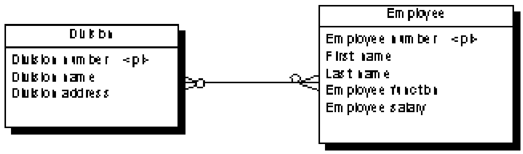
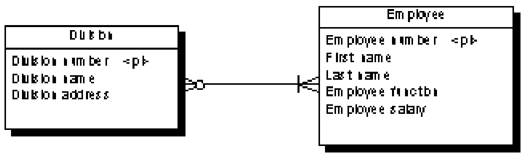
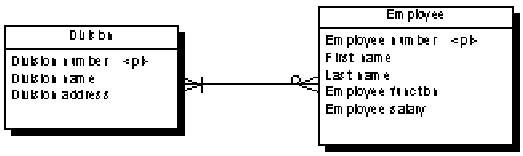
A one-to-many relationship links one instance of the first entity to multiple instances of the second entity. Additional properties can make one or both sides of this relationship mandatory and define identification rules:

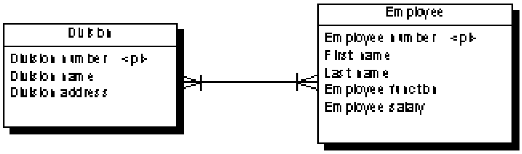
One-to-many relationship	Description
 	Each division may have zero or more employees Each employee may belong to zero or one division Each division must have one or more employees Each employee may belong to zero or one division
	Each division may have zero or more employees Each employee must belong to one and only one division
	Each division must have one or more employees Each employee must belong to one and only one division
	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
	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

A one-to-one relationship links one instance of the first entity with one instance of the second entity:

One-to-one relationship	Description
	Each team works on zero or one project Each project is managed by zero or one team
	Each team works on one and one project only Each project is managed by zero or one team
	Each team works on zero or one project Each project is managed by one and one team only

A many-to-many relationship links multiple instances of the first entity to multiple instances of the second entity. This type of relationship is not permitted, by default, in the LDM (see *Enabling Many-to-many Relationships in an LDM* on page 60):

Many-to-many relationship	Description
	Each division may have zero or more employees Each employee may belong to zero or more divisions
	Each division must have one or more employees Each employee may belong to zero or more divisions
	Each division may have zero or more employees Each employee must belong to one or more divisions

Many-to-many relationship	Description
	Each division must have one or more employees Each employee must belong to one or more divisions

Creating a Relationship

You can create a relationship from the Toolbox, Browser, or **Model** menu.

- Use the **Relationship** tool in the Toolbox. 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 a package) in the Browser, and select **New > Relationship**.

For general information about creating objects, see *Core Features Guide > Modeling with PowerDesigner > Objects*.

Relationship Properties

To view or edit a relationship's properties, double-click its diagram symbol or Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

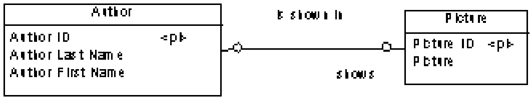
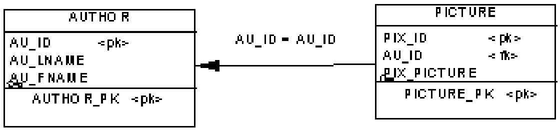
Property	Description
Name/Code/ Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
Entity1 Entity2	Specifies the two entities linked by the relationship. Use the tools to the right of the list to create, browse for, or view the properties of the currently selected entity.

Property	Description
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.
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

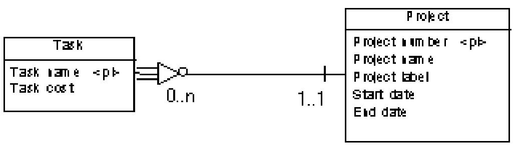
Cardinalities Tab

The **Cardinalities** tab allows you to specify the nature of the relationship between the two entities. The following properties are available:

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 (<1..1>) - One instance of entity A can correspond to only one instance of entity B. • One-to-many (<1..n>) - One instance of entity A can correspond to more than one instance of entity B. • Many-to-one (<n..1>) - More than one instance of entity A can correspond to the same one instance of entity B. • Many-to-many (<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 60. For information about the termination points of the relationships in each of the supported notations, see *Supported CDM/LDM Notations* on page 23.

Property	Description
Dominant role	[one-to-one relationships only] Specifies one direction of the relationship as dominant. If you define a dominant direction, the one-to-one relationship generates one reference in a PDM, with the dominant entity as the parent table. If you do not define a dominant direction, the one-to-one relationship generates two references. In the following example, the author is the dominant entity:  In a PDM, this relationship generates a reference with Author as the parent table, and its primary key migrated to the Picture table as a foreign key: 

In addition, this tab contains a groupbox for each direction of the relationship, containing the following properties:

Property	Description
Role name	Text that describes the relationship of EntityA to EntityB, and which is used to generate the assertion statements displayed at the top of this tab. You should use the infinitive phrase that describes the relationship of one entity to the other. For example, `Each Order may` **`contain`** `one or more line.`, and `Each line must` **`belong to`** `one and only one Order.` To modify the sentences generated from your role names, edit your model's assertion template (see *Assertion Template* on page 12).
Dependent	Specifies that the entity is dependent on and partially identified by the other entity. In the following example, the task entity is dependent on the project entity. Each task is a part of a project and each project can contain zero or more tasks: 

Property	Description
Mandatory	Specifies that each instance of the entity requires at least one instance of the other entity. 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. Implied by dependent
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

Joins Tab (LDM)

The **Joins** tab lists the joins defined between parent and child entity attributes. Joins can link primary, alternate, or foreign identifiers, or any user-specified attributes.

On this tab, you can either:

- Select an identifier from the parent entity in the **Parent** field on which to base the join to autopopulate the list with its associated parent and child attributes. If necessary, you can modify the specified child attributes.
- Specify `<None>` in the **Parent** field and specify your own attribute pairs on which to base the join using the following tools:

Tool	Description
	Reuse Attributes - Create a join by matching parent and child attributes that share the same code.
	Migrate Attributes - First specify attributes in the **Parent Attribute** column and then click this tool to migrate them to foreign identifier attributes in the child table. If the attributes do not exist in the child table, they are created.
	Cancel Migration - Remove any attributes migrated to the child table.
	Insert a Row - Inserts a row before the selected row in the list to specify another attribute to join on.
	Add a Row - Adds a row at the end of the list to specify another attribute to join on.

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.

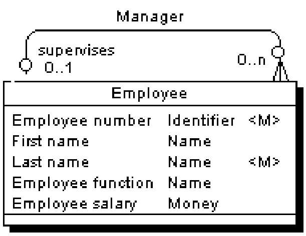
1. Select **Tools** > **Model Options**.
2. Select the **Allow n-n relationships** check box in the Relationship groupbox, and then click **OK** to return to the model.

Note: When generating an LDM from a CDM, you can authorize the generation of many-to-many relationships by clicking the **Configure Model Options** button on the **General** tab of the generation dialog, and selecting the **Allow n-n relationships** option.

Creating a Reflexive Relationship

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

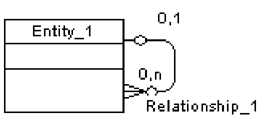
In the following example, the reflexive relationship *Supervise* expresses that an employee (Manager) can supervise other employees.



Note: 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.

1. Click the **Relationship** tool in the Toolbox.
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.



Note: 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.

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

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.

Changing a Relationship into an Associative Entity

You can transform a relationship between two entities into an associative entity linked by two relationships, and then attach entity attributes to the associative entity that you could not attach to the relationship.

1. Right-click a relationship symbol and select **Change to Entity**.

   The original relationship is split in two and an associative entity is created between the two new relationships, taking the name and code of the original relationship.
2. Open the property sheet of the associative entity or one of the new relationships to modify their properties as appropriate.

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.

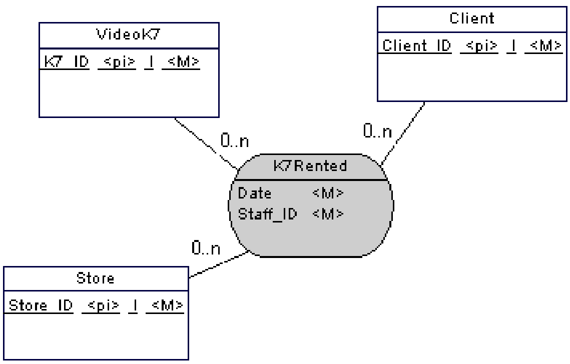
Associations and Association Links (CDM)

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.

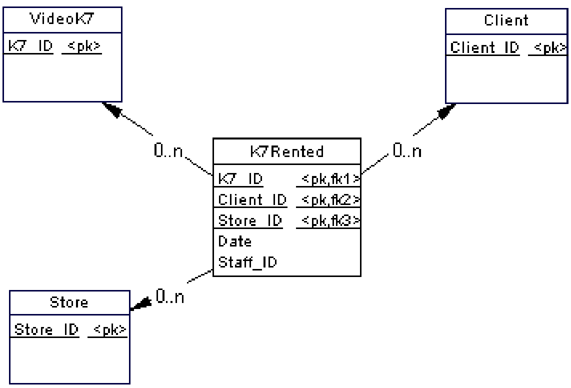
Each instance of an association corresponds to an instance of each entity linked to the association.

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

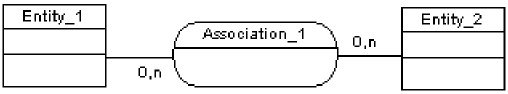
An association is connected to an entity by an association link, which symbolizes the role and the cardinality between an association and an entity.

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.

1. Click the **Association Link** tool in the Toolbox.
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 from the Toolbox, Browser, or **Model** menu.

- Use the **Association** tool in the Toolbox..
- Select **Model > Associations** to access the List of Associations, and click the **Add a Row** tool.
- Right-click the model (or a 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 *Core Features Guide > Modeling with PowerDesigner > Objects*.

Association Properties

To view or edit an association's properties, double-click its diagram symbol or Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

Property	Description
Name/Code/Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
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.
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

Association Link Properties

To view or edit an association link's properties, double-click its diagram symbol or Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

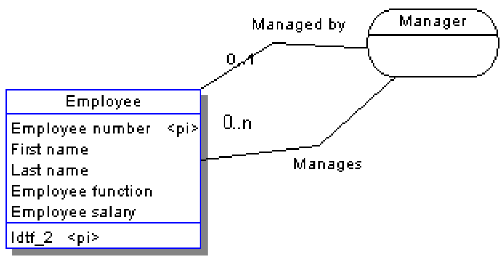
Property	Description
Entity	Specifies the entity connected by the association link. Use the tools to the right of the list to create, browse for, or view the properties of the currently selected entity.
Association	Specifies the association connected by the association link.
Role	Specifies the label indicating the role of the association link.
Identifier	Indicates if the entity is dependent on the other entity.

Property	Description
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)`
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

Creating a Reflexive Association

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

1. Click the **Association Link** tool in the Toolbox.
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:



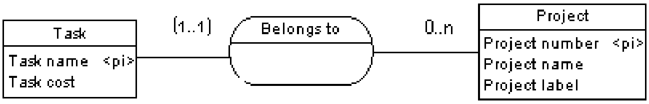
In the example above, the reflexive association *Manager* expresses that an employee (Manager) can manage other employees.

Defining a Dependent Association

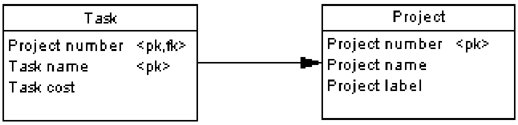
In a dependent association, one entity is partially identified by another. Each entity must have an identifier. In some cases, however, the attributes of an entity are not sufficient to identify an occurrence of the entity. For these entities, their identifiers incorporate the identifier of another entity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

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.



Note: The same association can not have two identifier association links.

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.

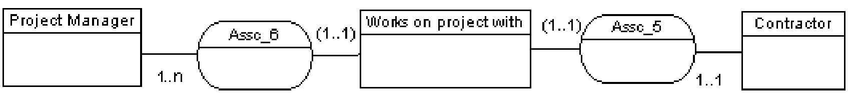
   The cardinality of the association link is enclosed in parenthesis to indicate that the association link is an identifier.

Changing an Association into an Associative Entity

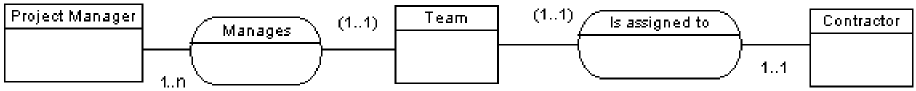
You can transform an association into an associative entity linked by two associations. The associative entity gets the name and code of the association. The two new associations handle cardinality properties.

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:



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

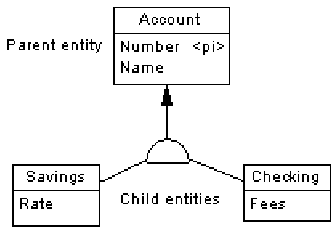
The tools used for creating association attributes on this tab are the same as those for creating entity attributes.

For more information, see *Creating an attribute* on page 50.

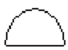
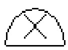
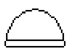
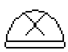
Inheritances (CDM/LDM)

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

Note: There is no separate inheritance object in the Barker notation (see *Supported CDM/LDM Notations* on page 23), as inheritances are represented by placing one entity symbol on top of another. Barker inheritances are always complete and mutually exclusive, and the supertype lists its subtypes on the **Subtypes** tab (see *Entity Properties* on page 45). Only leaf subtypes can be generated as PDM tables, and the **Generate** option is disabled on Barker supertype property sheets.

Creating an Inheritance

You can create an inheritance from the Toolbox, Browser, or **Model** menu.

- Use the Inheritance tool in the diagram Toolbox (see *Creating an Inheritance with the Inheritance Tool* on page 68).
- 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.

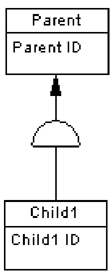
For general information about creating objects, see *Core Features Guide > Modeling with PowerDesigner > Objects*.

Creating an Inheritance with the Inheritance Tool

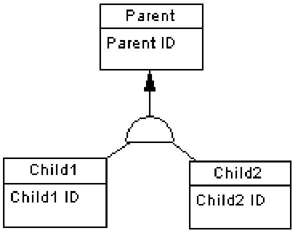
You can use the inheritance tool to create inheritances between entities and to join additional children to an inheritance.

1. Select the **Inheritance** tool in the Toolbox.
2. Click and hold inside the child entity and then drag to the parent entity and release the mouse button.

   The link is created between the two entities with a half-circle symbol in the middle with the arrow pointing to the parent entity.



3. [optional] To add further child entities to the inheritance link, click and hold inside the child entity and then drag to the inheritance half circle and release the mouse button:



4. [optional] Double-click the half circle or one of the links to open the inheritance property sheet, and enter any appropriate properties.

Inheritance Properties

To view or edit an inheritance's properties, double-click its diagram symbol or Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

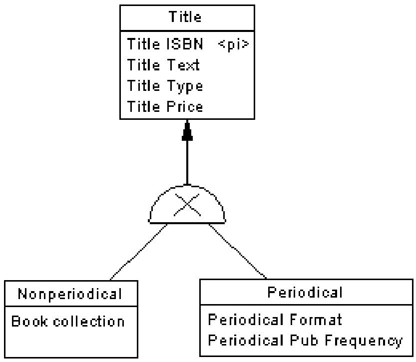
Property	Description
Name/Code/ Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
Parent	Specifies the name of the parent entity. Use the tools to the right of the list to create, browse for, or view the properties of the currently selected entity.
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.
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

Generation Tab

This tab allows you to specify how the inheritance structure will be generated to a PDM, including which attributes will be inherited.

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. If one or more child entities are not generated, the parent will take on their attributes and references.
	• **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. You must additionally 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: For LDM inheritances, primary identifiers of a parent entity always migrate to all child entities, even if the children are not selected for generation, and any changes you make on this tab will have an immediate effect on the inheritance of attributes in the LDM.
	Note: You can control the generation of individual child tables using the **Generate** option in the property sheet of each child entity (see *Entity Properties* on page 45).

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. For information about the tools on this tab, see *Creating an Attribute* on page 50. In the example below, the `TITLE` entity has two non-generated children, `NON-PERIODICAL` and `PERIODICAL`, and a specifying entity attribute `PERIODICAL` is defined for the inheritance link to differentiate between the two child entities.  In the PDM, the child entity attributes will generate columns in the table TITLE, and the specifying entity will generate a boolean PERIODICAL column, which indicates whether an instance of TITLE is a periodical.

The following tabs are also available:

- **Children** - lists the child entities of the inheritance. Use the **Add Children** and **Delete** tools to modify the contents of the list.

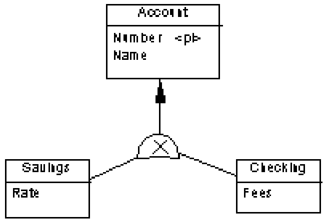
Making Inheritance Links Mutually Exclusive

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.

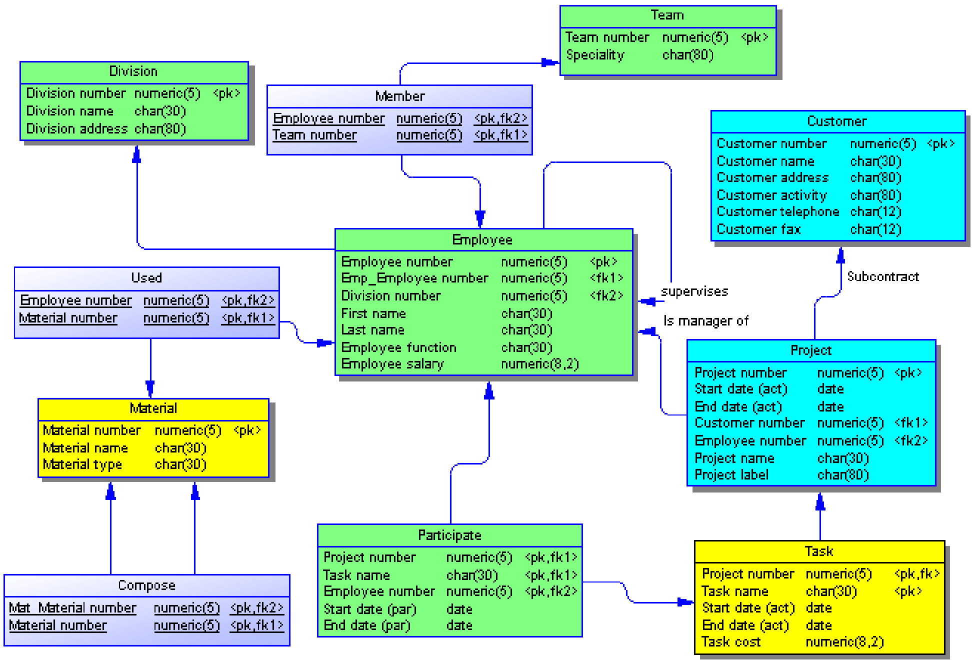

Account
Number <pi>
Name
Savings
Rate
Checking
Fees

Physical Diagrams

A *physical data diagram* provides a graphical view of your database structure, and helps you analyze its tables (including their columns, indexes, and triggers), views, and procedures, and the references between them.

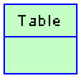
Note: To create a physical diagram in an existing PDM, right-click the model in the Browser and select **New > Physical Diagram**. To create a new model, select **File > New Model**, choose Physical Data Model as the model type and **Physical Diagram** as the first diagram, and then click **OK**.

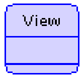
In the following example, the Employee table is shown in relation to the Team, Division, Material, Task, and Project tables:



Physical Diagram Objects

PowerDesigner supports all the objects necessary to build physical diagrams.

Object	Tool	Symbol	Description
Table			Collection of rows (records) that have associated columns (fields). See *Tables (PDM)* on page 76.
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 91.
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 105.
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 105.
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 105.
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 108.
Default	[none]	[none]	[certain DBMSs] A default value for a column. See *Defaults (PDM)* on page 160.
Domain	[none]	[none]	Defines valid values for a column. See *Domains (CDM/LDM/PDM)* on page 162.
Sequence	[none]	[none]	[certain DBMSs] Defines the form of incrementation for a column. See *Sequences (PDM)* on page 169.
Abstract data type	[none]	[none]	[certain DBMSs] User-defined data type. See *Abstract Data Types (PDM)* on page 171.
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 174.

Object	Tool	Symbol	Description
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 112.
View Reference			Link between a table and a view. See *View References (PDM)* on page 182.
Trigger	[none]	[none]	A segment of SQL code associated with a table or a view. See *Triggers (PDM)* on page 119.
Procedure			Precompiled collection of SQL statements stored under a name in the database and processed as a unit. See *Stored Procedures and Functions (PDM)* on page 140.
Database	[none]	[none]	The database of which the PDM is a representation. See *Database Properties (PDM)* on page 8.
Storage	[none]	[none]	A partition on a storage device. See *Configuring Tablespaces and Storages* on page 198.
Tablespace	[none]	[none]	A partition in a database. See *Configuring Tablespaces and Storages* on page 198.
User	[none]	[none]	A person who can log in or connect to the database. See *Users, Groups, and Roles (PDM)* on page 149.
Role	[none]	[none]	A predefined user profile. See *Users, Groups, and Roles (PDM)* on page 149.
Group	[none]	[none]	Defines privileges and permissions for a set of users. See *Users, Groups, and Roles (PDM)* on page 149.
Synonym	[none]	[none]	An alternative name for various types of objects. See *Synonyms (PDM)* on page 157.
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)* on page 201.
Web operation	[none]	[none]	Sub-object of a Web service containing a SQL statement and displaying Web parameters and result columns. See *Web Operations (PDM)* on page 204.

Tables (PDM)

A table is used to store data in a set of columns. Each record in the table is represented as a row, which is uniquely identified by the values in its primary key column or columns.

Tables are generally defined using the following sub-objects:

- Columns - are named properties of a table that describe its characteristics (see *Columns (PDM)* on page 91).
- Primary Keys - Uniquely identify rows through the values in the column or columns with which they are associated (see *Primary, Alternate, and Foreign Keys (PDM)* on page 105). Each key can generate a unique index or a unique constraint in a target database.
- Indexes - Help improve search times by ordering the values in the column or columns with which they are associated (see *Indexes (PDM)* on page 108).
- Triggers - SQL code invoked automatically whenever there is an attempt to modify data in the tables (see *Triggers (PDM)* on page 119).

Tables are linked together by references (see *References (PDM)* on page 174).

Creating a Table

You can create a table from the Toolbox, Browser, or **Model** menu.

- Use the **Table** tool in the Toolbox.
- Select **Model > Tables** to access the List of Tables, and click the **Add a Row tool**.
- Right-click the model (or a package) in the Browser, and select **New > Table**.

For general information about creating objects, see *Core Features Guide > Modeling with PowerDesigner > Objects*.

Table Properties

To view or edit a table's properties, double-click its diagram symbol or Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

Property	Description
Name/Code/Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.

Property	Description
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
Owner	Specifies the user who is the owner of the object. This is usually its creator. Use the tools to the right of the list to create, browse for, or view the properties of the currently selected user.
Number	Specifies the estimated number of records in the table, which is used to estimate database size. This field is automatically populated during reverse engineering if you select the **Statistics** option (see *Reverse Engineering from a Live Database* on page 314). 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 and selecting **Update Statistics**. To update the statistics for all tables, select **Tools > Update Statistics** (see *Reverse Engineering Database Statistics* on page 321).
Generate	Selects the table for generation to the database.
Dimension-al type	Specifies the type of the table for purposes of creating star or snowflake schemas containing fact tables and dimensions. You can choose between: • Fact - see *Facts (PDM)* on page 219 • Dimension - see *Dimensions (PDM)* on page 222 • Exclude - PowerDesigner will not consider the table when identifying or generating multidimensional objects. You can instruct PowerDesigner to complete this field for you (see *Identifying Fact and Dimension Tables* on page 217). PowerDesigner's support for the generation of BusinessObjects universes (see *Generating a BusinessObjects Universe* on page 301) and of facts and dimensions in a multidimensional diagram (see *Generating Cubes* on page 217) depends on the value of this field.
Type	[if your DBMS supports various types of table] Specifies the type of the table. You can choose between: • Relational - Standard tables. • Object - Tables based on abstract data types (see *Linking a Table to an Abstract Data Type* on page 79). • XML - Tables storing XML documents (see *Creating an XML Table or View* on page 79).
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

Lifecycle Tab

The Lifecycle tab is available if data lifecycle modeling (see *Lifecycles (PDM)* on page 189) is supported for your DBMS. These properties can be set for all the tables governed by the lifecycle on the lifecycle property sheet **Tables** tab (see*Lifecycle Properties* on page 193).

Property	Description
Lifecycle	Specifies the lifecycle with which the table is associated. Select a lifecycle from the list or click the tools to the right of this field to create a new lifecycle or open the property sheet of the currently selected one.
Start date	Specifies the start date from which to generate the first partition. Click the **Generate Partitions** tool to the right of this field to create partitions for the table, based on the partition range and start date.
Partition range	[read only] Specifies the duration of the partitions that will be created for the table. This value is controlled by the lifecycle (see *Lifecycle Properties* on page 193).
Row growth rate (per year)/ Initial Rows	Specifies an estimate of the increase of the size of the table per year, and the number of rows to start from as a basis for the calculation of cost savings. Click the **Estimate Cost Savings** tool to the right of this field to perform the calculation.
Cost Savings	This groupbox lists the cost savings that accrue to the storage of this table's data through its association with the lifecycle. Each line in the grid represents one year of savings, which are shown as a monetary value and as a percentage of the cost of storing the data statically outside of a lifecycle.

The following tabs are also available:

- **Columns** - Lists the columns associated with the table (see *Columns (PDM)* on page 91). The following tools are available on this tab:

Tool	Description
	Insert a Row / **Add a Row** - Creates a column above the selected column or at the end of the list.
	Add Columns / **Replicate Columns** - Copies or replicates columns from another table (see *Copying or Replicating a Column from Another Table* on page 104).
	Create Index - Creates an index associated with the selected columns (see *Creating Standard, Key, or Function-Based Indexes* on page 109).
	Create Key - Creates a (by default, alternate) key associated with the selected columns (see *Creating Alternate Keys* on page 107).

- **Indexes** - Lists the indexes associated with the table (see *Indexes (PDM)* on page 108).
- **Keys** - Lists the keys associated with the table (see *Primary, Alternate, and Foreign Keys (PDM)* on page 105).
- **Triggers** - Lists the triggers associated with the table (see *Triggers (PDM)* on page 119).

- **Procedures** - Lists the procedures associated with the table (see *Stored Procedures and Functions (PDM)* on page 140).
- **Security Procedures** - [data lifecycle modeling only] Lists the procedures which control access to the table (see *Stored Procedures and Functions (PDM)* on page 140).
- **Check** - Specifies the constraints associated with the table (see *Setting Data Profiling Constraints* on page 94).
- **Physical Options** - Lists the physical options associated with the table (see *Physical Options (PDM)* on page 88).
- **Preview** - Displays the SQL code associated with the table (see *Previewing SQL Statements* on page 285).

Linking a Table to an Abstract Data Type

If your DBMS supports it, PowerDesigner allows you to base tables on abstract data types (ADT), where the table uses the properties of the ADT and the ADT attributes become table columns. To link a table to an ADT, open the table property sheet to the **General** tab, and select the ADT (of type Object, SQLJ Object, or Structured type) in the **Based On** field.

For detailed information about working with abstract data types, see *Abstract Data Types (PDM)* on page 171.

Creating an XML Table or View

If your DBMS supports it, PowerDesigner allows you to create XML tables and views. An XML table does not contain columns, and instead stores an XML document. You must associate the table with a registered XML schema to validate the XML document stored in the table, and can specify a root element for the structure stored in your table.

When you select the XML in the **Type** field, the **Column** tab is removed and the following properties are added to the **General** tab:

Property	Description
Schema	Enter the target namespace or name of an XML model (see *XML Modeling*) or use the **Select** tool to the right of the field to connect to the database and select a registered schema. The schema must be registered in the database to be used for validating XML documents.
Element	Allows you to specify a root element in the XML document. You can enter an element name or click the **Select** tool to the right of the field to select an element from an XML model open in the workspace or from the schema registered in the database If you select an element from a PowerDesigner XML model, the **Schema** property is set to the XML model target namespace.

Specifying Table Constraints

The table **Check** tab is initialized with the PowerDesigner `%RULES%` variable to generate validation rules specified on the **Rules** tab. You can edit the code on this tab by entering an appropriate SQL expression to supplement, modify, or replace these constraints.

You can override the default **Constraint name**. To revert to the default name, click to reset the **User-Defined** button to the right of the field:

For information about business rules, see *Business Rules (CDM/LDM/PDM)* on page 184. For information about setting column constraints, see *Setting Data Profiling Constraints* on page 94.

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 sometimes be more desirable.

PowerDesigner supports denormalization through:

- *Horizontal partitioning* - dividing a table into multiple tables containing the same columns but fewer rows.
- *Vertical partitioning* - dividing a table into multiple tables containing the same number of rows but fewer columns.
- *Table collapsing* - merging tables in order to eliminate the join between them.
- *Column denormalization* - repeating a column in multiple tables in order to avoid creating a join between them.

Horizontal and vertical partitioning involve tradeoffs in terms of performance and complexity. Though they can improve query response time and accelerate data backup and recovery, they require additional joins and unions to retrieve data from multiple tables, more complex queries to determine which table contains the requested data, and additional metadata to describe the partitioned table. Column denormalization can simplify queries but requires more maintenance and storage space as data is duplicated.

When deciding whether to denormalize, you should analyze the data access requirements of the applications in your environment and their actual performance characteristics. Often, good indexing and other solutions may more effectively address performance problems. Denormalization may be appropriate when:

- Critical queries rely upon data from more than one table.
- Many calculations need to be applied to columns before queries can be successfully answered.
- Tables need to be accessed in different ways by different kinds of users simultaneously.
- Certain columns are queried extremely frequently.

Horizontal Partitions

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

1. Select **Tools > Denormalization > Horizontal Partitioning**, or right-click a table in the diagram and select **Horizontal Partitioning** to open the Horizontal Partitioning Wizard.
2. Select the table to partition, specify whether you want to keep the original table after partitioning, and then click **Next**.
3. Create as many partition tables as necessary using the **Insert** and **Add a Row** tools (specifying an appropriate name for each, which must be unique in the model), and then click **Next**.
4. Click the **Add Columns** tool to select one or more discriminant columns to use as partition criteria (these columns will be excluded from the partitions), and then click **Next**.
5. Specify a name and code for the transformation object that will be created to preserve information about the partitioning, and then click **Finish** to create a table for each partition, taking the name of the partition. All references to the original table are created on each partition table.

   In this example, the table `Annual Sales`, which contains a very large amount of data is horizontally partitioned on the `Year` column:

Before	After		
Annual Sales OrderID INTEGER <pk> Amount NUMERIC Year DATE	Annual Sales_2010 OrderID INTEGER <pk> Amount NUMERIC	Annual Sales_2011 OrderID INTEGER <pk> Amount NUMERIC	Annual Sales_2012 OrderID INTEGER <pk> Amount NUMERIC

Note: Horizontal partitionings created in a PDM generated from another model are preserved when applying changes from the original model. The absence of discriminant columns in the target PDM is respected in the Merge dialog (see *Core Features Guide > Modeling with PowerDesigner > Comparing and Merging Models*), and changes in the source model are selected, by default, to be cascaded as appropriate to all partition tables.

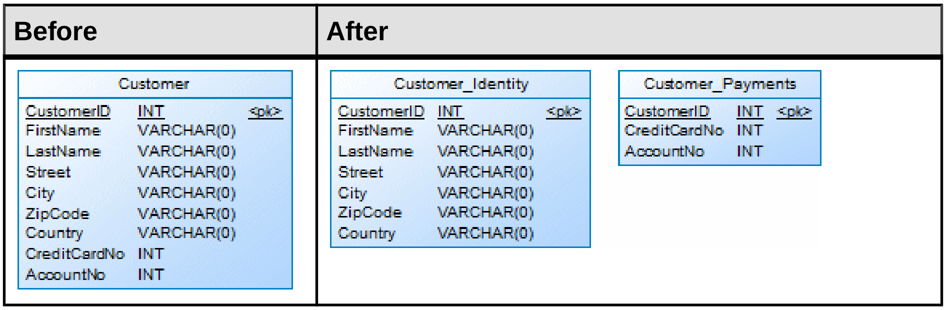
Vertical Partitions

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.

1. Select **Tools > Denormalization > Vertical Partitioning**, or right-click a table in the diagram and select **Vertical Partitioning** to open the Vertical Partitioning Wizard.
2. Select the table to partition, specify whether you want to keep the original table after partitioning, and then click **Next**.

3. Create as many partition tables as necessary using the **Insert** and **Add a Row** tools (specifying an appropriate name for each, which must be unique in the model), and then click **Next**.
4. Drag columns from under the original table in the **Available columns** pane, to the appropriate partition table in the **Columns distribution** pane, (or select source and target tables and use the **Add** and **Remove** buttons), and then click **Next**.
5. Specify a name and code for the transformation object that will be created to preserve information about the partitioning, and then click **Finish** to create a table for each partition, taking the name of the partition. All references to the original table are created on each partition table.

   In this example, the table `Customer`, is divided into two tables, each of which details one type of information about the customer:



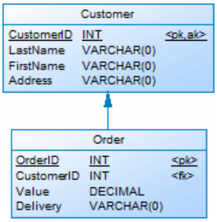
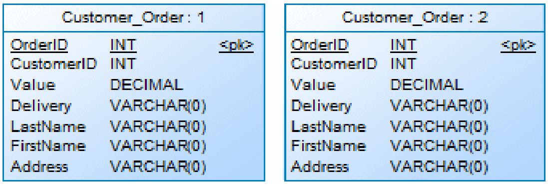
Note: Vertical partitionings created in a PDM generated from another model are preserved when applying changes from the original model. The columns absent from each partition table in the target PDM are shown but not selected in the Merge dialog (see *Core Features Guide > Modeling with PowerDesigner > Comparing and Merging Models*). Any changes in the source model are proposed, where appropriate, to each of the partition tables, and you should deselect the change for those partitions to which you do not want to apply it.

Table Collapsings

Table collapsing consists in merging tables in order to eliminate joins and to improve query performance. You can collapse tables related to each other with a reference or tables with identical primary keys.

1. Select **Tools > Denormalization > Table Collapsing**, or right-click a reference between the tables to collapse and select **Table Collapsing** to open the Table Collapsing Wizard.
2. Specify a name and code for the table to be created, and then click **Next**.
3. Click the **Add Tables** tool to select tables to collapse into the new table, specify whether you want to keep the original tables after collpsing, and then click **Next**.
4. Specify a name and code for the transformation object that will be created to preserve information about the collapsing, and then click **Finish** to collapse the selected tables into a single unified table (with graphical synonyms replacing each original table symbol in the diagram to minimize disruption of references.

In this example, the tables `Customer` and `Order` are collapsed together to eliminate the join and optimize data retrieval. The result is a single table (with 2 synonym symbols) with the primary key of the child table:

<table>
<tr><th>Before</th><th>After</th></tr>
<tr><td>
</td><td>
</td></tr>
</table>

5. [optional] Delete one of more of the synonymns. References will redirect to the remaining symbol.

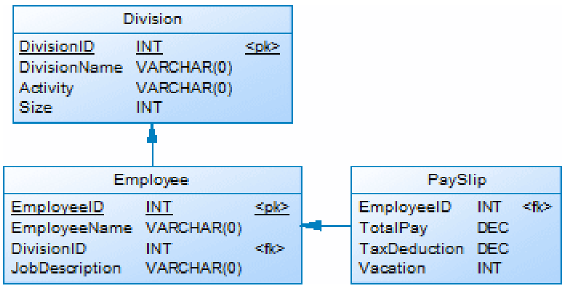
Column Denormalization

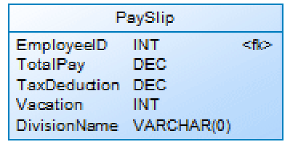
Column denormalization consists in replicating columns from one table to another to reduce the number of joins needed for frequently called queries. Though it can provide improved performance, column denormalization requires more maintenance and disk space as the data in the replicated column is stored twice.

1. Select **Tools > Denormalization > Column Denormalization**, or right-click the table to which you want to replicate columns and select **Column Denormalization** to open the Column Denormalization Wizard.
2. Specify the table to which you want to replicate columns, and then click **Next**.
3. Select one or more columns, and then click **Finish** to replicate them to the selected table.

Note: Replicas are, by default, read-only copies of objects. Any changes made to the original column are automatically propagated to the replica. This synchronization is controlled by a replication object for each replica, a list of which is available by selecting **Model > Replications**. To revert a column denormalization, simply delete the duplicated column from the target table property sheet. For detailed information about working with replicas and replications, see *Core Features Guide > Linking and Synchronizing Models > Shortcuts and Replicas*.

In this example, to obtain the division name on the pay slip of each employee without requiring a link to the `Division` table, the `DivisionName` column is replicated to the `PaySlip` table:

<table>
<tr><th>Before</th><th>After</th></tr>
<tr><td>


</td><td>


</td></tr>
</table>

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 the property sheet of this object, select **Model > Transformations** to open the List of Transformations, select the appropriate denormalization, and then click the **Properties** tool.

The **General** tab contains the following properties:

Property	Description
Name/Code/Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Partitioned table	[partitionings only] Specifies the name of the table used to create the table partitions.
Discriminant Columns	[horizontal partitionings] Specifies the name and code of the columns used as partition criteria.
Target table	[collapsings] Specifies the name of the table resulting from the collapsing of the tables.

The following tabs are also available:

- **Partitions** - [partitionings] Lists the tables associated with the partitioning. You can create or delete partition tables, and edit their properties. If you delete a partition, you are prompted to specify whether you want to delete the corresponding table.
- **Partition Columns** - [vertical partitionings] Displays the distribution of columns between the partition tables. You can drag and drop columns between tables.

- **Source Tables** - [table collapsings] Lists the tables that were collapsed. These tables will no longer exist unless you selected to keep them in the wizard.

Removing Partitionings and Table Collapsings

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

Select **Model > Transformations** to open the List of Transformations. The following tools are available for removing transformations:

Tool	Description
	**Delete** - Removes the denormalization but retains any tables created by it.
	**Cancel** [if the denormalization object is based upon a table generated from another model] Removes the denormalization and any tables created by it. You can recover the original table by regenerating it from the source model.

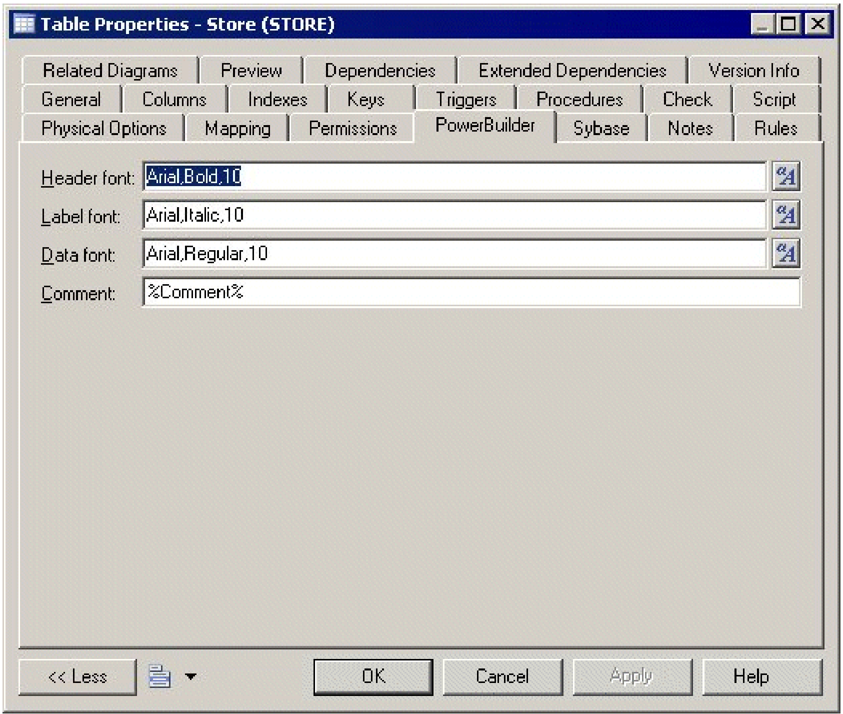
Note: You cannot move or paste a denormalization object to another model or package.

PowerBuilder DataWindow Extended Attributes

When designing tables to be used in a SAP® Sybase® 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 the modeling of this information through an extension file. To enable the PowerBuilder extensions in your model, select **Model > Extensions**, click the **Attach an Extension** tool, select the `PowerBuilder` file (on the **General Purpose** tab), and click **OK** to attach it.

When this extension file is attached, additional properties for two PowerBuilder system tables (*PBCatTbl* for tables and *PBCatCol* for columns) are available on the PowerBuilder tab of tables and columns:



To import the PowerBuilder extended attributes contained in your database to your PDM, select **Tools > PowerBuilder > Reverse Extended Attributes**, click the **Connect to a Data Source** tool, select a machine or file data source and click **Connect**. Select the tables you want to reverse-engineer, and click **OK**.

To update the PowerBuilder extended attribute system tables in your database, select **Tools > PowerBuilder > Generate Extended Attributes**, click the **Connect to a Data Source** tool, select a machine or file data source and click **Connect**. Select the tables you want to generate, and click **OK**. Reversed extended attributes are compared with the translated default values in the PowerBuilder extension file. If these attributes match, the reversed value is replaced by the default value from the extension file.

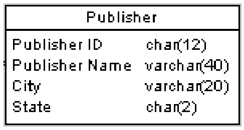
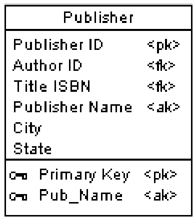
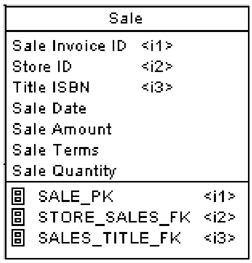
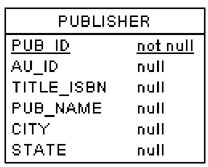
Displaying Column, Domain, and Data Type Information on a Table Symbol

To set display preferences for tables, select **Tools > Display Preferences**, and select the Table sub-category in the left-hand Category pane.

Columns

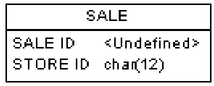
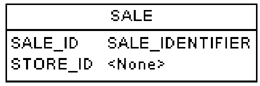
Keys and indexes are represented by indicators in the table symbol. Each key and index indicator is assigned a number. You can use these numbers to keep track of the different groups of alternate keys, foreign keys, and indexes in your model.

By default, the following information about columns can be displayed on table symbols.

Preference	Displays	Example
Data types	Data type for each column	
Replace by domains	Domain codes for each column attached to a domain	
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 the Display Domain and Data Type section below 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	
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	
NULL/NOT NULL	Column indicator: null, not null, identity, or with default (DBMS-dependent)	

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	
Domains	Only the domain, if it exists	
Data types and Domains	Both data type and domain, if they exist	
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.	

Note: For information about selecting other properties to display, see *Core Features Guide > Modeling with PowerDesigner > Diagrams, Matrices, and Symbols > Display Preferences*.

Physical Options (PDM)

Physical options are DBMS-specific parameters that specify how an object is optimized or stored in a database, and are included at the end of the object's `Create` statement. Physical options are defined in the DBMS definition file, and may be available for tables, columns, indexes, tablespaces, and other objects. You can specify default physical options for all objects of a particular type and for individual objects (overriding the default, if one is specified).

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. Changes made on either of these tabs will be reflected on the other:

- **Physical Options (Common)** – this tab is displayed by default (along with the **Partition** tab, if applicable), and lists the most commonly-used physical options as a standard property sheet tab. Select or enter values for the appropriate options and click **OK**
- **Physical Options** – 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)**. Follow the procedure in *Defining Default Physical Options* on page 89, to specify options and set values for them.

Physical options can vary widely by DBMS. For example, in Oracle, you specify the tablespace where the table is stored with the `Tablespace` keyword, while in SAP® Sybase®

SQL Anywhere®, you use `In`. When you change DBMS, the physical options selected are preserved as far as possible. If a specific physical option was selected, the default value is preserved for the option in the new DBMS. Unselected physical options are reset with the new DBMS default values.

For detailed information about the syntax of physical options and how they are specified, see *Customizing and Extending PowerDesigner > DBMS Definition Files > Physical Options*.

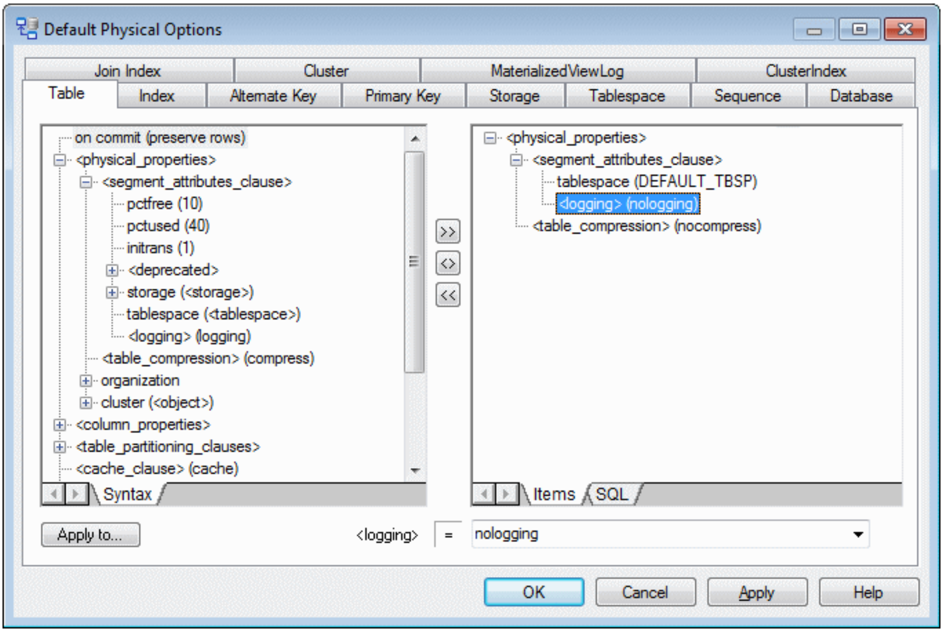
Note: In Oracle, the `storage` composite physical option is used as a template to define all the storage values in a storage entry to avoid having to set values independently each time you need to re-use them same values in a storage clause. For this reason, the Oracle physical option does not include the storage name (%s).

Defining Default Physical Options

You can define default physical options for all the objects of a particular type in the model.

1. Select **Database > Default Physical Options** to open the Default Physical Options dialog. There is a tab for each kind of object that supports physical options.

   The **Table** tab opens by default. The **Syntax** sub-tab in the left pane lists the physical options available in the DBMS, and the **Items** sub-tab in the right pane lists the physical options that have been selected for the object.



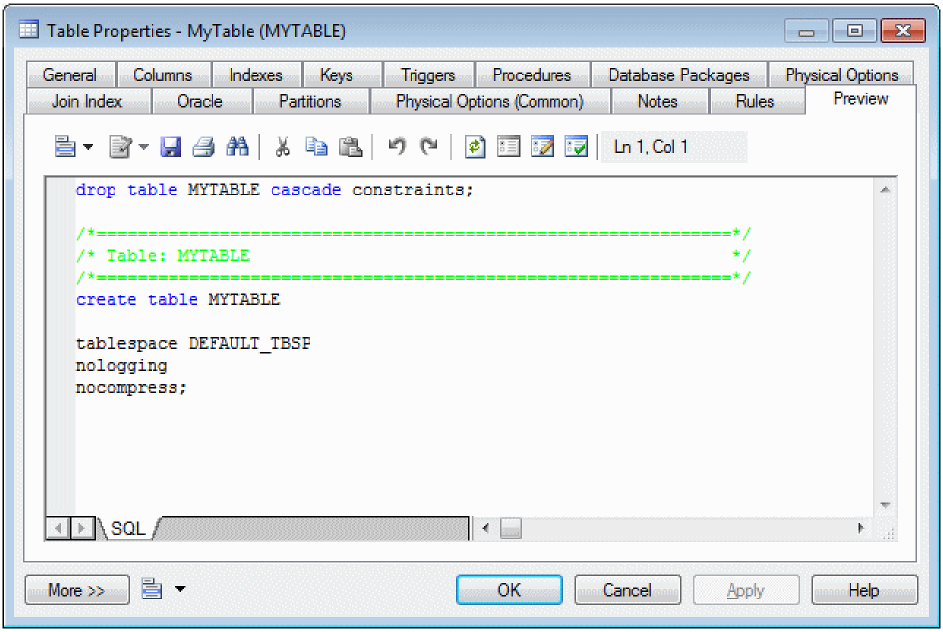
   The following tools are available for adding and removing physical options to an object:

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

2. To add a default option for the object, select it in the **Syntax** pane and click the **Add** tool 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.
3. To set a default value for a physical option parameter, select it in the **Items** pane and enter or select the appropriate value in the field below the pane. The entered value will then be displayed against the parameter in the Items list.
4. Repeat the above steps as many times as necessary to specify all your required physical options. By default, these options will be applied to all tables created subsequently in the model. To apply them to existing tables, click the **Apply to** button to select the tables to which you want to apply the options, and then click **OK**.
5. Select the other tabs to specify physical options for other object types. (Note that the **Apply to** button is not available on the **Database** tab).
6. Click **OK** to close the dialog and return to your model.

   To override the default physical options for a particular object, set the appropriate values on the the object's **Physical Options (Common)** or **Physical Options** tab

   You can view the physical options set for an object in its **Preview** tab.



Note: The default physical options are stored in your model file.

Columns (PDM)

A column is a set of values of a single type in a table. Each row of the table contains one instance of each column. Each table must have at least one column, which must have a name and code and to which you can assign a data type, either directly, or via a domain.

Creating a Column

You can create a column from the property sheet of, or in the Browser under, a table.

- Open the **Columns** tab in the property sheet of a table, and click the **Add a Row** or **Insert a Row** tool
- Right-click a table in the Browser, and select **New > Column**

For general information about creating objects, see *Core Features Guide > Modeling with PowerDesigner > Objects*.

Column Properties

To view or edit a column's properties, double-click its Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

Property	Description
Name/Code/ Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
Table	Specifies the table which contains the column.
Data type/ Length/ Precision	Specifies the form of data to be stored, such as numeric, alphanumeric, or Boolean, and, where appropriate, the maximum number of characters or numerals that can be stored, and the maximum number of places after the decimal point. Click the ellipsis button to choose from the list of standard data types (see *PowerDesigner Standard Data Types* on page 165). To review the data types permitted by your DBMS, select **Database > Edit Current DBMS** and navigate to **Script** > **DataType** > **PhysDataType**). The following variables specify length and precision requirements: • `%n` - length • `%s` - length with precision • `%p` - decimal precision For example, the data type `char(%n)` , requires you to specify a length.
Domain	Specifies the domain associated with the object (see *Domains (CDM/LDM/PDM)* on page 162). Use the tools to the right of this field to create or browse to a domain, or to open the property sheet of the selected domain.
Primary key	Specifies that the values in the column uniquely identify table rows (see *Creating Primary Keys* on page 105).
Foreign key	Specifies that the column depends on and migrates from a primary key column in another table (see *Creating Foreign Keys* on page 107).
Sequence	[if supported by your DBMS] Specifies the sequence associated with the column (see *Sequences (PDM)* on page 169).
Displayed	Specifies that the column can be displayed in the table symbol.
With default	[if supported by your DBMS] Specifies that the column must be assigned a value that is not null.

Property	Description
Mandatory	[if supported by your DBMS] Specifies that a non-null value must be assigned.
Identity	[if supported by your DBMS] Specifies that the column is populated with values generated by the database. Identity columns are often used as primary keys.
Computed	[if supported by your DBMS] Specifies that the column is computed from an expression using values from other columns in the table (see *Creating a Computed Column* on page 102).
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

Detail Tab

The Detail tab contain the following properties:

Property	Description
Column fill parameters	The fields in this groupbox model the size and distinctness of data values that you expect to store in the column and are used in conjunction with test data profiles (see *Populating Columns with Test Data* on page 98). You can specify: • **Null values** - [Default: 0%] Specifies the percentage of values to leave empty. • **Distinct values** - [Default: 100%] Specifies the percentage of values that must be unique • **Average Length** - [read only] Used for estimating the size of the database (see *Estimating Database Size* on page 306). The default value is the maximum length for the data type defined for the column. You can enter values by hand or obtain them from your database by selecting the **Statistics** option in the Reverse Engineering dialog (see *Reverse Engineering from a Live Database* on page 314). To refresh the values in these fields for all a table's columns at any time, right-click the table symbol or its entry in the Browser and select **Update Statistics**. To update the column statistics for all the tables in a model, select **Tools > Update Statistics** (see *Reverse Engineering Database Statistics* on page 321).
Profile	Specifies a test data profile to use to generate test data (see *Populating Columns with Test Data* on page 98). Use the tools to the right of this field to create or browse to a profile, or to open the property sheet of the selected profile.
Computed Expression	Specifies an expression used to compute data for the column (see *Creating a Computed Column* on page 102).

The following tabs are also available:

- **Standard Checks** - Specifies constraints to control the range and format of permitted data (see *Setting Data Profiling Constraints* on page 94)
- **Additional Checks** - Displays an editable SQL statement, initialized with the standard checks, which can be used to generate more complex constraints (see *Specifying Advanced Constraints* on page 97).
- **Rules** - Lists the business rules associated with the object (see *Business Rules (CDM/LDM/PDM)* on page 184).

Setting Data Profiling Constraints

PowerDesigner allows you to define data profiling constraints to control the range and format of data allowed in your database. You can specify constraints on the **Standard Checks** and **Additional Checks** tabs of table columns in your PDM, entity attributes in your CDM or LDM, and domains. You can also specify data quality rules on the **Rules** tab of PDM tables and columns, CDM/LDM entities and attributes, and domains.

The following constraints are available on the **Standard Checks** tab of PDM columns, CDM/LDM entity attributes, and CDM/LDM/PDM domains:

Property	Description
Values	Specifies the range of acceptable values. You can set a: • Minimum - The lowest acceptable numeric value • Maximum - The highest acceptable numeric value • Default - The value assigned in the absence of an expressly entered value. For the PDM, you can directly enter a default value or select a keyword (defined in the `Script\Sql\Keywords\ReservedDefault` entry of the DBMS definition file) from the list. Default objects (see *Defaults (PDM)* on page 160) are also available for selection if your DBMS supports them.
Characteristics	These properties are for documentation purposes only, and will not be generated. You can choose a: • Format - A number of standard formats are available in the list. You can enter a new format directly in the field or use the tools to the right of the field to create a data format for reuse elsewhere. • Unit - A standard measure. • No space - Space characters are not allowed. • Cannot modify - The value cannot be updated after initialization.

Property	Description
Character case	Specifies the acceptable case for the data. You can choose between: • Mixed case [default] • Uppercase • Lowercase • Sentence case • Title case
List of values	Specifies the various values that are acceptable. When specifying strings in the list of values, single or double quotation marks (depending on the DBMS) will be added around the values in the generated script unless: • You surround the value by the appropriate quotation marks. • You surround the value by tilde characters. • The value is a keyword (such as NULL) defined in the DBMS. • PowerDesigner does not recognize your data type as a string. The following examples show how string values are generated for a DBMS that uses single quotation marks: • `Active` - generates as `'Active'` • `'Active'` - generates as `'Active'` • `"Active"` - generates as `'"Active"'` • `~Active~` - generates as `Active` • `NULL` - generates as `NULL` If you have specified a non-automatic test data profile, you can use the values defined in the profile to populate the list by clicking the **Update from Test Data Profile** tool. Select the **Complete** check box beneath the list to exclude all other values not appearing in the list.

Specifying Constraints Through Business Rules

In addition to the constraints specified on the **Standard Checks** tab, you can specify business rules of type `Validation` or `Constraint` to control your data. Both types of rule contain SQL code to validate your data, and you can attach them to tables and table columns in your PDM, entities and entity attributes in your CDM or LDM, and domains.

You can use the following PowerDesigner variables when writing your rule expression:

Variable	Value
%COLUMN%	Code of the column to which the business rule applies

Variable	Value
%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

To attach a business rule (see *Business Rules (CDM/LDM/PDM)* on page 184) to a table, column, entity, attribute, or domain, open the object's property sheet, select the **Rules** tab, and click the **Add Objects** tool.

At generation time, business rules of type `validation` are concateneated together into a single constraint, while rules of type `Constraint` will be generated as separate constraints if your DBMS supports them.

Creating Data Formats For Reuse

You can create data formats to reuse in constraints for multiple objects by clicking the **New** button to the right of the **Format** field on the **Standard Checks** tab. Data formats are informational only, and are not generated as constraints.

Note: To create multiple data formats, use the List of Data Formats, available by selecting **Model** > **Data Formats**.

Data Format Properties

To view or edit a data format's properties, double-click its Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

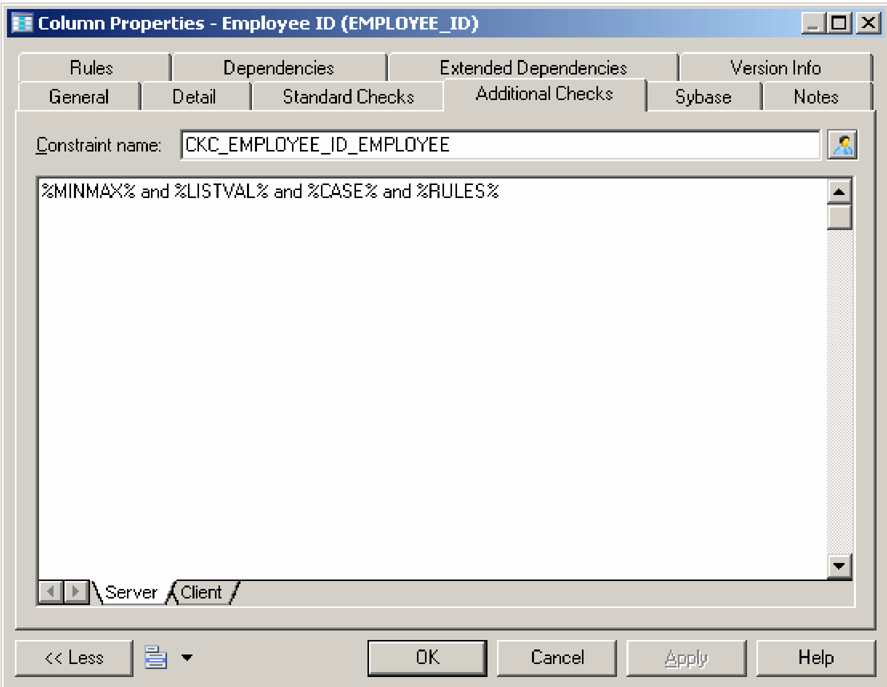
Property	Description
Name/Code/ Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.

Property	Description
Type	Specifies the type of the format. You can choose between: • Date/Time • String • Regular Expression
Expression	Specifies the form of the data to be stored in the column; For example, `9999.99` would represent a four digit number with two decimal places.
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

Specifying Advanced Constraints

The **Additional Checks** tab is initialized with PowerDesigner variables to generate the data profiling constraints specified on the **Standard Checks** tab and the validation rules specified on the **Rules** tab. You can edit the code on this tab by entering an appropriate SQL expression to supplement, modify, or replace these constraints.

For columns, you can override the default **Constraint name**. To revert to the default name, click to reset the **User-Defined** button to the right of the field:



The following variables are inserted by default:

- %MINMAX% - Minimum and maximum values specified on the **Standard Checks** tab
- %LISTVAL% - List of values specified on the **Standard Checks** tab
- %CASE% - Character case specified on the **Standard Checks** tab
- %RULES% - Constraint and validation rules specified on the **Rules** tab

Populating Columns with Test Data

You can use test data to quickly fill your database with large amounts of data in order to test its performance and estimate its size. You can also use test data as the basis for data profiling. PowerDesigner allows you to create test data profiles, which generate or provide lists of data items and are assigned to columns or domains. You can create test data profiles that contain number, character, or date/time data.

For example, you could create a test data profile called Address that specifies character data appropriate to represent addresses, and then associate that profile with the columns Employee Location, Store Location, and Client Address.

If you associate a test data profile with a domain, its data will be generated to all columns that are attached to the domain. If you specify a data profile as the default for its type, its data will be generated to all columns that are not associated with another profile.

To generate test data with or without test data profiles, see *Generating Test Data to a Database* on page 304

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

- Select **Model > Test Data Profiles** to access the List of Test Data Profiles, and click the **Add a Row** tool
- Right-click the model (or a package) in the Browser, and select **New > Test Data Profile**

Note: You can import and export test data profiles to reuse them across multiple models by using the commands under the: **Tools > Test Data Profile** menu. The `*.xpf` file format can contain one or more test data profiles.

Test Data Profile Properties

To view or edit a test data profile's properties, double-click its Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

Property	Description
Name/ Code/Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
Class	Specifies the kind of data to be generated from the profile. You can choose between: • Number - to populate numerical columns • Character - to populate text columns • Date & Time - to populate date columns
Generation source	Specifies from where PowerDesigner will draw the data to populate the columns associated with the profile. You can choose between: • Automatic - PowerDesigner generates the data based on the parameters you set on the Detail tab. • List - PowerDesigner draws the data from the list you define on the Details tab. • Database - PowerDesigner draws the data using a query from a live database connection that you specify on the Details tab. • File - PowerDesigner draws the data from the CSV file that you specify on the Details tab.
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

Detail Tab (Automatic Number Data)

If you have selected to automatically generate number data on the General tab, you must define the following properties on the Detail tab:

Property	Description
Type	Specifies whether the data is to be generated randomly or sequentially.
Range	Specifies the range of numbers to generate from and, if a sequential type is specified, the step value to use when traversing the range.
Decimal numbers	Specifies that the numbers to be generated are decimal, and the number of digits after the decimal point to generate.

Detail Tab (Automatic Character Data)

If you have selected to automatically generate character data on the General tab, you must define the following properties on the Detail tab:

Property	Description
Valid characters	Specifies the characters that can be generated (by default, all alphanumeric characters and spaces), separated by commas. You can specify: • Single characters or strings of characters - surrounded by double quotes. For example, `"a", "bcd", "e"`. • Character intervals - in which the boundary characters are surrounded by single quotes and separated by a dash. For example, `'a'-'z', 'A'-'Z'` To allow any character, select the All checkbox.
Invalid characters	Specifies the characters that cannot be generated, using the same syntax as for the valid characters. To disallow accented characters, select the No accents checkbox.
Mask	Specifies the mask characters used to tell users what kind of character they must enter in a given context. By default the test data profile uses the following mask characters: • A - Letter • 9 - Number • ? - Any character
Case	Specifies the case in which to generate the data. If you select Lower or Mixed case, select the First Uppercase checkbox to require that each word begin with a capital letter.
Length	Specifies the length of character strings to generate. You can specify either an exact required length or a range.

Detail Tab (Automatic Date & Time Data)

If you have selected to automatically generate date and time data on the General tab, you must define the following properties on the Detail tab:

Property	Description
Date range	Specifies the upper and lower limits of the date range within which data can be generated.
Time range	Specifies the upper and lower limits of the time range within which data can be generated.
Step	Specifies step values for use when traversing the date and time ranges, if sequential values are generated.

Property	Description
Values	Specifies whether the values are to be generated randomly or sequentially.

Note: The format in which date and time data is generated can be controlled by DBMS items in the `Script/Sql/Format` category (see *Customizing and Extending PowerDesigner > DBMS Definition Files > Script/Sql Category*.

Detail Tab (List Data)

If you have selected to provide list data on the General tab, enter as many value-label pairs as necessary on the Detail tab.

Detail Tab (Database Data)

If you have selected to provide data from a database on the General tab, you must define the following properties on the Detail tab:

Property	Description
Data Source	Specifies the data source from which to draw data for the profile. Click the Select a Data Source tool to the right of this field to open a separate dialog on which you can specify your connection parameters.
Login and Password	Specifies the login and password to use when connecting to the data source.
Table, Column, and Query	Specifies the table and column from which the data will be drawn. By default, a query selecting distinct values from the column is used.

Detail Tab (File Data)

If you have selected to provide data from a file on the General tab, you must define the following properties on the Detail tab:

Property	Description
File	Specifies the file from which to draw data for the profile.
Type	Specifies whether the values are to be drawn randomly or sequentially.

Assigning Test Data Profiles to Columns

You can associate a test data profile directly to a column or to a domain.

Note: To assign a test data profile to a domain (see *Domains (CDM/LDM/PDM)* on page 162, open the domain property sheet and select the appropriate test data profile in the **Profile** list on the **General** tab. A test data profile assigned to a domain will generate test data for all the columns attached to the domain.

1. Open the property sheet of a column and click the **Detail** tab.
2. Select the appropriate test data profile.
3. [optional] Adjust the following properties in the **Column fill parameters** group box as appropriate:
    - **Null values** - [Default: 0%] Specifies the percentage of values to leave empty.
    - **Distinct values** - [Default: 100%] Specifies the percentage of values that must be unique. For example, if you set this field to 100 % for one column and to 80% for a second column, and then generate the table with 10 rows, all 10 rows in the first column will have different values, while 2 values in the second column will be repeated. This is a maximum value, and can change depending on the referential integrity parameters of primary key columns. Alternately, you can enter a specific value without a percentage sign, to indicate the exact number of column rows that should contain unique entries.

      Note: If you use 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** - [read only] Used for estimating the size of the database (see *Estimating Database Size* on page 306). The default value is the maximum length for the data type defined for the column.

    Note: These properties on the column property sheet **General** may override values entered in the **Column fill parameters** groupbox:
    - **Mandatory** (M) - Specifies that the column must contain a value and sets **Null values** to 0%.
    - **Unique** (U) - Specifies the column must contain a unique value and sets **Null value** to 0% and **Distinct values** to 100%.
    - **Foreign** (F) - The column is a foreign key column and takes the values of the corresponding primary key column in the parent table.
4. Click **OK** to close the column property sheet and return to the model.

    Note: To quickly assign test data profiles to multiple columns, use the List of Columns or the Columns tab of a table property sheet. If the Test Data Profile column is not visible in your list, use the **Customize Columns and Filter** tool to display it.
5. [optional] Generate your test data (see *Generating Test Data to a Database* on page 304).

Creating a Computed Column

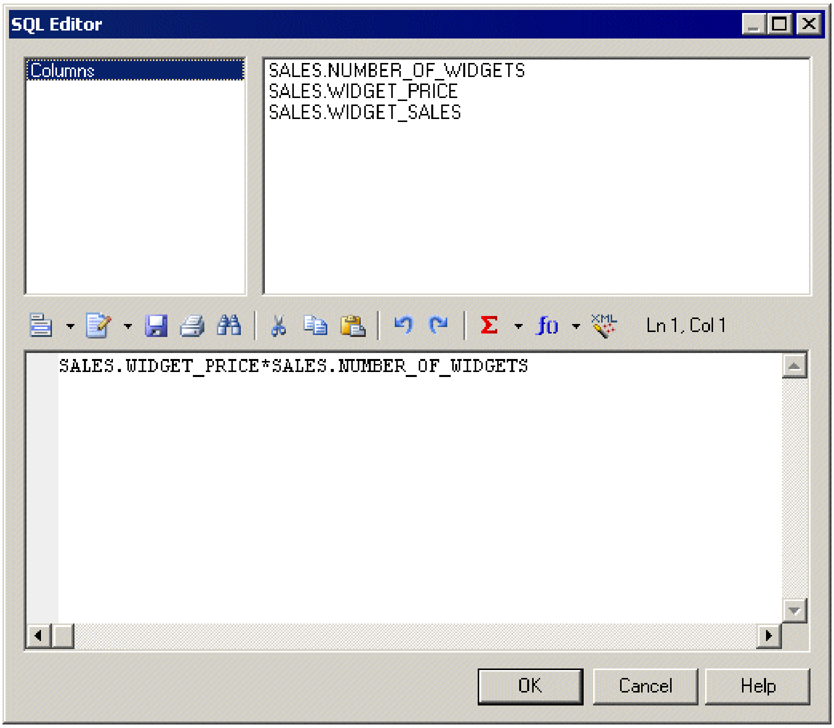
Computed columns are columns whose content is computed from values in other columns in the table. Computed columns are not supported by all DBMSs.

1. Open the table 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.

   Simple computed expressions can be entered directly in the **Computed expression** field. For more complex expressions, click the Edit tool to the right of the field to access the SQL Editor (see *Writing SQL Code in PowerDesigner* on page 281).

   In the following example a column must be filled with the total sales of widgets computed by multiplying the number of widgets by the widget price:



4. Click **OK** to return to the column property sheet.

   The expression is displayed in the **Computed Expression** pane.

Attaching a Column to a Domain

You can attach a column to a domain, and have the domain specify the data type, check parameters, and business rules for the column. Domains can help with data consistency across columns storing similar types of data.

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

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

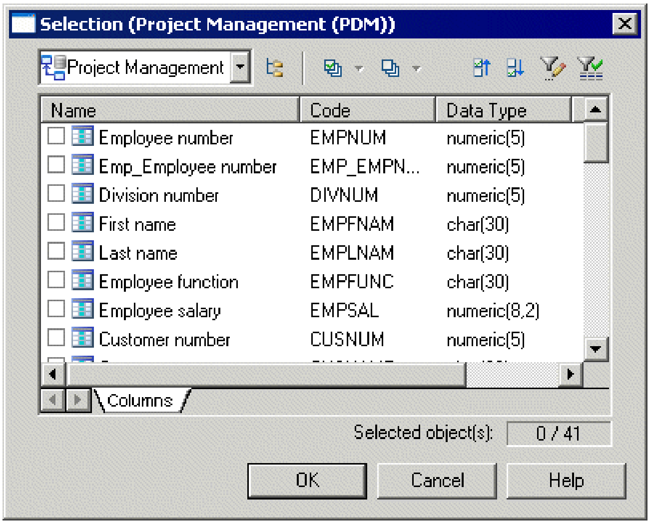
   For detailed information about working with domains, see *Domains (CDM/LDM/PDM)* on page 162.

Copying or Replicating a Column from Another Table

You can reuse existing columns from other tables by copying or replicating them using the tools on the table property sheet **Columns** tab or by drag and drop. If your table already contains a column with the same name or code as the copied column, the copied column is renamed.

Copying a column creates a simple copy that you can modify as you wish. Replicating a column creates a synchronized copy which remains synchronized with any changes made to the original column (see *Core Features Guide > Linking and Synchronizing Models > Shortcuts and Replicas*).

1. Open the property sheet of the table you want to copy or replicate the columns to, and click the **Columns** tab.
2. Click the **Add Columns** or **Replicate Columns** to open a selection box listing the columns attached to all other tables in the model.



3. Select one or more columns in the list and then click **OK** to copy or replicate them to the table.
4. Click **OK** to close the table property sheet and return to your model.

Note: To copy or replicate a column from one table to another in the diagram or browser, select the column in the table symbol or its Browser entry, and then right-click and hold while dragging the column to over the second table symbol or its Browser entry. Release and select **Copy Here** or **Replicate Here**.

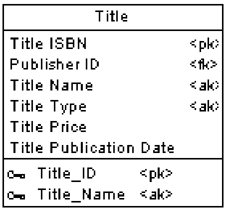
Primary, Alternate, and Foreign Keys (PDM)

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.

You can create the following types of keys:

- Primary keys - Contain one or more columns whose combined values uniquely identify every row in a table. Each table can have only one primary key.
- Alternate keys - Contain one or more columns whose combined values uniquely identify every row in a table.
- Foreign keys - Contain one or more columns whose values match a primary or alternate key in some other table.

In the following example, the `TITLE` table has a primary, alternate and foreign key:



- The primary key, `TITLE_ID` contains the column `TITLE ISBN`, and uniquely identifies each book in the table.
- The alternate key, `TITLE_NAME`, contains the columns `TITLE NAME` and `TITLE TYPE`, and enforces a constraint that no two titles of the same type can have the same name.
- The foreign key contains the column `PUBLISHER ID` and references the primary key column in the `Publisher` table.

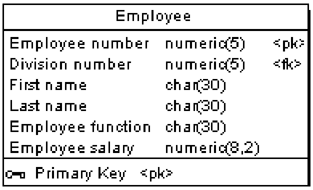
Creating Primary Keys

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

1. Open the property sheet of the table and click the **Columns** tab, which lists all the columns defined for the table (see *Columns (PDM)* on page 91).
2. Select the check box in the **P** column for one or more columns in the list to associate them with the primary key.
3. [optional] Click the **Keys** tab and rename the key or select it and click the **Properties** tool to open its property sheet.

4. Click **OK** to close the property sheet and return to the diagram.

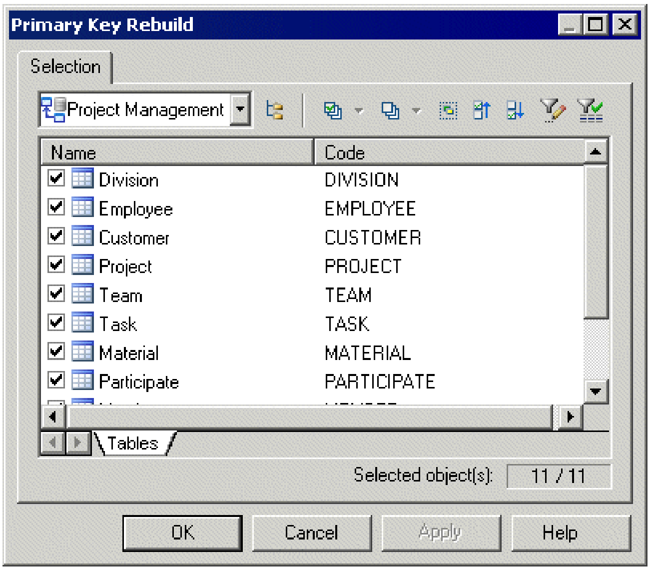
   In the following example, `Employee number` is the primary key for the table `Employee`, and each employee must have a unique employee number:



Rebuilding Primary Keys

Rebuilding primary keys in a physical diagram updates primary keys for tables by creating primary keys for tables that have no key and a single unique index. Rebuilding primary keys is useful when not all of the primary keys could be reverse engineered from a database, or if you did not select the rebuild option for primary keys during reverse engineering.

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.



Note: 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, click the **Include Sub-Packages** tool, and then select a sub-package from the dropdown list.

2. Select the tables containing the primary keys that you want to rebuild and then click **OK**.

Creating Alternate Keys

An alternate key is a key associated with one or more columns whose values uniquely identify every row in the table, but which is not the primary key. For example, where the primary key for a table may be the employee id, the alternate key might combine the first, middle, and last names of the employee. Each alternate key can generate a unique index or a unique constraint in a target database.

1. Open the property sheet of a table and select the **Columns** tab.
2. Select the column or columns to associate with the alternate key and click the **Create Key** tool.

   The new key property sheet opens.
3. Enter a name for the key. Alternate keys are conventionally named `AK`*`x`*`_`*`ColumnCodes`* (for example `AK1_CUSNAME`).
4. [optional] Modify the default **Constraint Name**.
5. Click **OK** to complete the creation of your alternate key and return to the table property sheet.

   Note: You can also create an alternative key using the **Add a Row** tool on the table property sheet **Keys** tab, click the **Properties** tool to open its property sheet, and select the **Columns** tab to manually associate columns with the key.

Creating Foreign Keys

A foreign key is a primary or alternate key migrates from another table. Foreign keys are generally migrated automatically when you draw a reference from a child to a parent table.

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 (see *References (PDM)* on page 174). For information about auto-migration of foreign keys, see *Automatic Reuse and Migration of Columns* on page 178.

Key Properties

To view or edit a key's properties, double-click its Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

Property	Description
Name/Code/Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
Table	Specifies the name of the table where the key is defined.
Constraint name	Specifies the name of the key constraint. A primary key constraint is a named check that enforces the uniqueness and the presence of values in a primary key column. PowerDesigner automatically creates a default constraint name for a key, which you can modify. To return to the default click to release the **User-Defined** button. You can use the following variables: • %AK% and %AKNAME% - Code and name of the alternate key. • %TABLE%, %PARENT%, %CHILD% - Code of the table, the parent table, and the child table. • %REFRCODE% and %REFRNAME% - Code and name of the reference. For a complete list of PDM variables, see *Customizing and Extending PowerDesigner > DBMS Definition Files > PDM Variables and Macros*
Primary key	Specifies that the key is the primary key of the table. There can be only one primary key in a table, so selecting this key as the primary key will deselect any existing primary key.
Cluster	Specifies that the key constraint is a clustered constraint (for those DBMSs that support clustered indexes).
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

The following tabs are also available:

- **Columns** - Lists the columns associated with the key. Use the **Add Columns** tool to associate additional columns with the key.

Indexes (PDM)

An index is a data structure associated with one or more columns ordered by the column values. Indexes are typically created for columns that you are frequently searched on to improve response times. Most types of index are more effective when applied to columns with high cardinality.

For example, in an `Author` table, you might create an index for the primary key `ID` and another for the `LastName` column, as it is regularly searched on, but you will probably not create an index for the `BirthCity` column, as it is not often searched on.

Note: PowerDesigner supports the creation of indexes for materialized views, if your DBMS allows them (see *Materialized Views* on page 117).

Creating Standard, Key, or Function-Based Indexes

You can create indexes by selecting columns on a table property sheet **Columns** tab and clicking the **Create Index** tool.

1. Open the property sheet of a table and select the **Columns** tab.
2. Select the column or columns on which to base the index and click the **Create Index** tool.

   The index is created and its property sheet opens.
3. Enter a name for the index and then click the **Columns** tab.
4. PowerDesigner supports the creation of the following types of index:
   - *Standard indexes* are associated with one or more columns containing high-cardinality values that are frequently searched on. Use the arrow buttons at the bottom of the list to reorder the columns in order of descending cardinality.
   - *Key indexes* are associated with a primary, foreign, or alternate key and based on the same columns as the key. Select the appropriate key from the **Columns definition** field above the list to empty the list and replace it with the columns associated with the key.

     Note: Key indexes are conventionally named after the table with a `_PK`, `_FK`, or `AK` suffix (for example, `Project_AK`).
   - *Function-based indexes* [if supported by the DBMS] are populated with values derived from a function or expression based on one or more columns, and provide an efficient mechanism for evaluating statements that contain functions in their WHERE clauses. Click the **Add a Row** tool, then click in the **Expression** column and click the ellipsis button to open the SQL Editor to specify an expression.
5. Select an ascending or descending sort order for each column using the list's **Sort** column.
6. Click **OK** to complete the creation of your index and return to the table property sheet.

   Note: You can alternatively create an index using the **Add a Row** tool on the table property sheet **Indexes** tab, click the **Properties** tool to open its property sheet, and select the **Columns** tab to manually associate columns with the index.

Index Properties

To view or edit an index'sproperties, double-click its Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

Property	Description
Name/Code/ Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
Owner	[if supported by the DBMS] Specifies the user who is the owner of the object. This is usually its creator. Use the tools to the right of the list to create, browse for, or view the properties of the currently selected user.
Table	Specifies the table to which the index belongs.
Type	Specifies the type of index (if supported by your DBMS). For information about SAP® Sybase® IQ index types, see *Indexes (IQ)* on page 531.
Unique	Specifies that the index cannot contain duplicate values.
Cluster	Specifies that the index is a clustered index. A table cannot have more than one clustered index. **Note:** Clusters in Oracle 11 and higher are modeled as extended objects with a `<<Cluster>>` stereotype (see *Clusters (Oracle)* on page 474).
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

The following tabs are also available:

- **Columns** - Lists the columns with which the index is associated (see *Creating Standard, Key, or Function-Based Indexes* on page 109). Use the following tools to specify columns:

Tool	Description
N/A	**Columns definition** - Select the appropriate key to empty the list and replace it with the columns associated with the key.

Tool	Description
	Add Columns - Choose columns from the table to add to the list.
	Add / Insert a Row [if your DBMS supports function-based indexes] - Create a new row, then click in the **Expression** column and click the ellipsis button to open the SQL Editor to specify an expression. For example, to define an index to convert all names to lowercase to simplify searching, you could enter an expression such as: `lower(SURNAME)`

Rebuilding Indexes

You can rebuild indexes at any time to reflect any changes that you have made to primary keys, foreign keys, or alternate keys in your model.

1. Select **Tools > Rebuild Objects > Rebuild Indexes**, and enter the appropriate options:

Option	Description
Primary key	Rebuilds primary key indexes. The field displays the naming convention for primary key indexes, which is by default `%TABLE%_PK`. You can use the following variables: • `%TABLE%` - Generated code of the table. The generated code of a variable is the code defined in the object property sheet, but may be truncated if it contains characters not supported by the DBMS. • `%TNAME%`, `%TCODE%`, `%TLABL%` - Table name, code, and comment.
Other keys	Rebuilds alternate key indexes. The field displays the naming convention for alternate key indexes, which is by default `%AKEY%_AK`.
Foreign key indexes	Rebuilds foreign key indexes. The field displays the naming convention for foreign key indexes, which is by default `%REFR%_FK`. You can use the following variables: • `%REFR%`, `%PARENT%`, `%CHILD%` - Generated code of the reference, parent, and child table. • `%PNAME%`, `%PCODE%`, `%PQUALIFIER%` - Parent table name, code, and qualifier. • `%CNAME%`, `%CCODE%`, `%CQUALIFIER%` - Child table name or code, and qualifier. • `%REFRNAME%`, `%REFRCODE%` - Reference name or code.
Foreign key threshold	Specifies the minimum number of estimated records in a table (specified in the **Number** field in the table property sheet) that are necessary before a foreign key index can be created. If the **Number** field is empty, foreign key indexes are generated.

Option	Description
Mode	Specifies the type of rebuild. You can select: • Delete and Rebuild – Delete and rebuild all indexes attached to primary, alternate, and foreign keys. • Add missing indexes – Preserve existing key indexes and add any missing key indexes.

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

Views (PDM)

A view is a query that provides access to all or a subset of the data in a table or multiple tables connected by joins. Views do not copy the data from their underlying tables and are updated when data in those tables changes. Views can reference other views, can order or filter data as necessary, and may be indistinguishable from tables for users accessing them.

Creating a View

You can create a view populated with columns from selected tables and other views via the **Tools** menu. Alternately, you can create an empty view from the Toolbox, Browser, or **Model** 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. Select the appropriate objects and then click **OK**.

   A view symbol is created in the diagram, displaying 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.

3. [optional] Edit the view's query to remove unwanted columns or otherwise modify the view (see *View Queries* on page 115).

Alternatively, you can create an empty view, which you should complete by specifying a query (see *View Queries* on page 115) in the following ways:

- Use the **View** tool in the Toolbox.
- Select **Model > Views** to access the List of Views, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New** > **View**.

View Properties

To view or edit a view's properties, double-click its diagram symbol or Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

Property	Description
Name/Code/ Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
Type	Specifies the type of the view, where supported by your DBMS (see *Materialized Views* on page 117 and *Creating an XML Table or View* on page 79).
Owner	Specifies the user who is the owner of the object. This is usually its creator. Use the tools to the right of the list to create, browse for, or view the properties of the currently selected user.

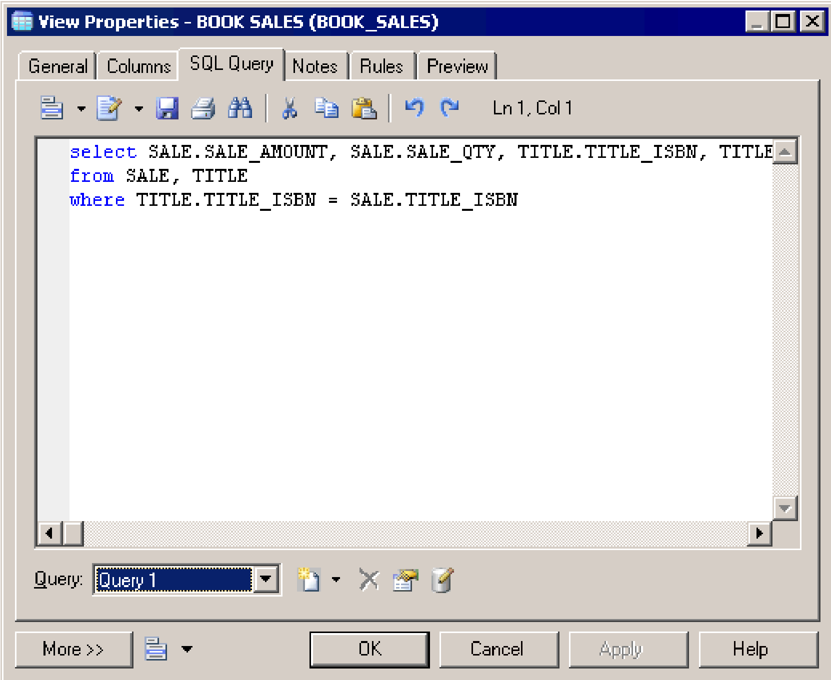
Property	Description
Usage	Specifies how the view will be used. You can choose between: • Query only - Consultation only. The view cannot update tables. • Updatable - Consultation and update of underlying tables. • With Check options - Implements controls on view insertions.
Dimensional type	Specifies the type of the view for purposes of creating star or snowflake schemas containing fact tables and dimensions. You can choose between: • Fact - see *Facts (PDM)* on page 219 • Dimension - see *Dimensions (PDM)* on page 222 • Exclude - PowerDesigner will not consider the view when identifying or generating multidimensional objects. You can instruct PowerDesigner to complete this field for you (see *Identifying Fact and Dimension Tables* on page 217). PowerDesigner's support for the generation of BusinessObjects universes (see *Generating a BusinessObjects Universe* on page 301) and of facts and dimensions in a multidimensional diagram (see *Generating Cubes* on page 217) depends on the value of this field.
Generate	Selects the view for generation to the database.
User-defined	By default, the view query is updated to reflect changes to model objects on which it is based. Selecting this option freezes the view and protects your manual changes.
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

The following tabs are also available:

- **Columns** - Lists the columns in the view based on the SELECT orders from the queries. You can modify column properties in this list but to add or remove columns, you must modify the appropriate view query. View column properties are initialized from the properties of their source columns. The read-only **Expression** column specifies the qualified name of the view column.
- **Indexes** - [materialized views] Lists the indexes defined on the materialized view (see *Indexes (PDM)* on page 108).
- **SQL Query** - 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 (see *View Queries* on page 115).
- **Triggers** - [if supported by your DBMSs] Lists the triggers associated with the view (see *Triggers (PDM)* on page 119. You can define a trigger to fire when one or more attributes of a table view column are modified.
- **Preview** - Displays the SQL code to be generated for the view (see *Previewing SQL Statements* on page 285).

View Queries

You can edit queries associated with a view from the SQL Query tab of the view property sheet.



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.

You can edit the code shown in the SQL Query tab:

- Directly in the tab.
- Click the **Edit with SQL Editor** tool to edit the code in the PowerDesigner SQL Editor (see *Writing SQL Code in PowerDesigner* on page 281).
- Click the **Edit with** tool (CTRL+E) to open the code in your favorite editor.

Any edits you make in this tab will propagate to the property sheets of the associated individual queries, which are available from the Query list at the bottom of the tab. Use the tools to the right of this list to create a new query (with the appropriate linking construct), delete the selected query, or open the property sheet of the selected query.

The following SQL constructs are available (if supported by your DBMS) for linking queries:

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

The following tabs are available:

- **SQL** tab - displays the SQL code for the query. You can edit the query directly in this tab or in PowerDesigner's built-in SQL Editor (see *Writing SQL Code in PowerDesigner* on page 281) by clicking the **Edit with SQL Editor** tool or in an external editor by clicking the **Edit with** tool (CTRL+E). Any edits you make in this tab will propagate to the query's other tabs and the SQL Query tab of the parent view, as changes made in other tabs will propagate here and to the parent view.
- **Tables** tab - lists the tables in the FROM clause.You can add or delete tables in the list, and reorder the list using the arrows at the bottom of the tab. For each line, select a table or click the elipsis button to enter a more complex expression in the SQL Editor and, optionally, enter an alias in the Alias column. For the second and subsequent lines in the list you can specify an appropriate join condition keyword, and then specify the join condition.
- **Columns** tab - lists the columns in the SELECT clause. You can add or delete columns in the list, specify aliases for them, and reorder the list using the arrows at the bottom of the tab.
- **Where** tab - lists the expressions in the WHERE clause.You can add or delete expressions in the list, and reorder the list using the arrows at the bottom of the tab. For each line, select a column in each of the two Expression columns (or click the ellipsis button to specify a more complex expression), and select the appropriate operator between them. You can optionally enter a prefix and suffix.
- **Group By** tab - lists the columns in the GROUP BY clause. You can add or delete columns in the list, and reorder the list using the arrows at the bottom of the tab.
- **Having** tab - lists the expressions in the HAVING clause. You can add or delete expressions in the list, and reorder the list using the arrows at the bottom of the tab. For each line, select a column in each of the two Expression columns (or click the ellipsis

button to specify a more complex expression), and select the appropriate operator between them. You can optionally enter a prefix and suffix.

- **Order By** tab - lists the columns in the ORDER BY clause. You can add or delete columns in the list, and reorder the list using the arrows at the bottom of the tab. For each line, select a column (or click the ellipsis button to specify a more complex expression), and select ASC or DESC for the sort direction.

Materialized Views

A materialized view is a table containing the results of a query. PowerDesigner supports materialized views for the DB2, HP Neoview, Netezza, Oracle, and Sybase SQL Anywhere DBMS families.

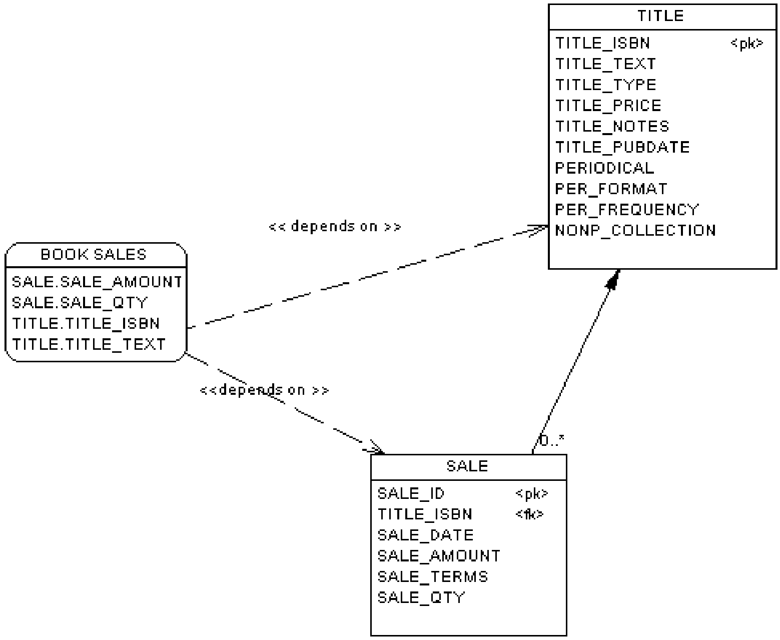
Materialized views are supported in the following ways:

- DB2 - Select `materialized query table` (or for earlier versions, `summary table`) in the **Type** list on the **General** tab of a view property sheet.
- HP Neoview - Use the List of Materialized Views (available from **Model** > **Materialized Views**).
- Netezza - Use the List of Materialized Views (available from **Model** > **Materialized Views**).
- Oracle - Use the List of Materialized Views (available from **Model** > **Materialized Views**).
- SQL Anywhere - Select `Materialized View` in the **Type** list on the **General** tab of a view property sheet to display the **DB space** field, and specify the dbspace in which to create the materialized view. The default is the current dbspace.

Showing View Dependencies using Traceability Links

You can use traceability links to make the relationships between views and tables clearer. These links are not interpreted and checked by PowerDesigner.

In the following example, the `Book Sales` view is shown as depending on the `Title` and `Sale` tables via two traceability links with their type set to `depends on`:



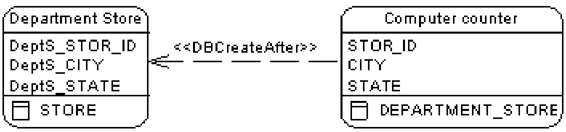
For detailed information about traceability links, see *Core Features Guide > Linking and Synchronizing Models > Getting Started with Linking and Syncing > Creating Traceability Links*.

Defining a Generation Order for Views

You can define the order of the generation of views by using traceability links with a type of `DBCreateAfter`. The view from which you start the traceability link is dependent on the view you link it to, and this influent view will be generated before the dependent view.

For example you create the view `DEPARTMENT STORE` from the table `STORE`, and then another view called `COMPUTER COUNTER` from the view `DEPARTMENT STORE` to show only part of the department store offer.

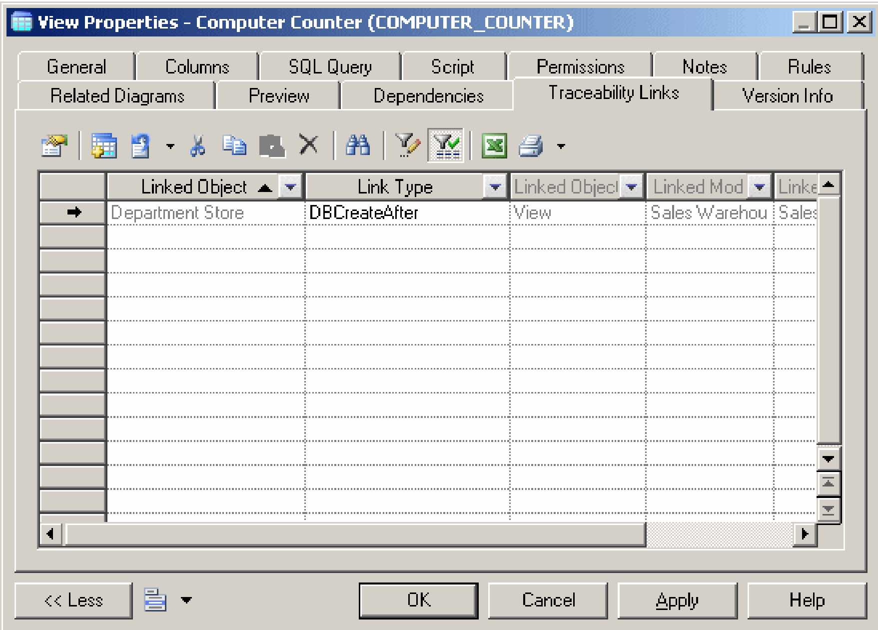
By default, views are generated in alphabetical order, so the generation of `COMPUTER COUNTER` will fail since the view `DEPARTMENT STORE` on which it depends is not yet generated. To bypass this problem, you should create a traceability link of type `<<DBCreateAfter>>` from `COMPUTER COUNTER` to `DEPARTMENT STORE` to ensure that `DEPARTMENT STORE` is generated before `COMPUTER COUNTER`:



Note: There is a model check to warn you if you create a reflexive or circular set of traceability links of type `DBCreateAfter`. If you generate without correcting this error, views will be generated in alphabetical order, without taking into account the generation order.

1. Select the **Traceability Links** tool in the toolbox.
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 traceability link to open the property sheet of the dependent object at the **Traceability Links** tab.

   The influent view is displayed in the **Linked Object** column.
4. Click in the **Link Type** column, click the down arrow and select `DBCreateAfter`.



5. Click **OK** to close the property sheet and return to your model.

   Note: You can, alternatively, create `DBCreateAfter` traceability links directly on the **Traceability Links** tab (see *Defining a Generation Order for Stored Procedures* on page 137). For detailed information about traceability links, see *Core Features Guide > Linking and Synchronizing Models > Getting Started with Linking and Syncing > Creating Traceability Links*.

Triggers (PDM)

A trigger is a segment of SQL code associated with a table or a view, which is invoked automatically whenever there is an attempt to modify data in the associated table or view with an insert, delete, or update command. A DBMS trigger is not associated with any table or view, and fires on modifications to the database structure itself, such as the creation or dropping of a table or events like startup, shutdown, login etc. You can use triggers to enforce referential integrity (where declarative constraints are not sufficient) and to implement sequences for columns.

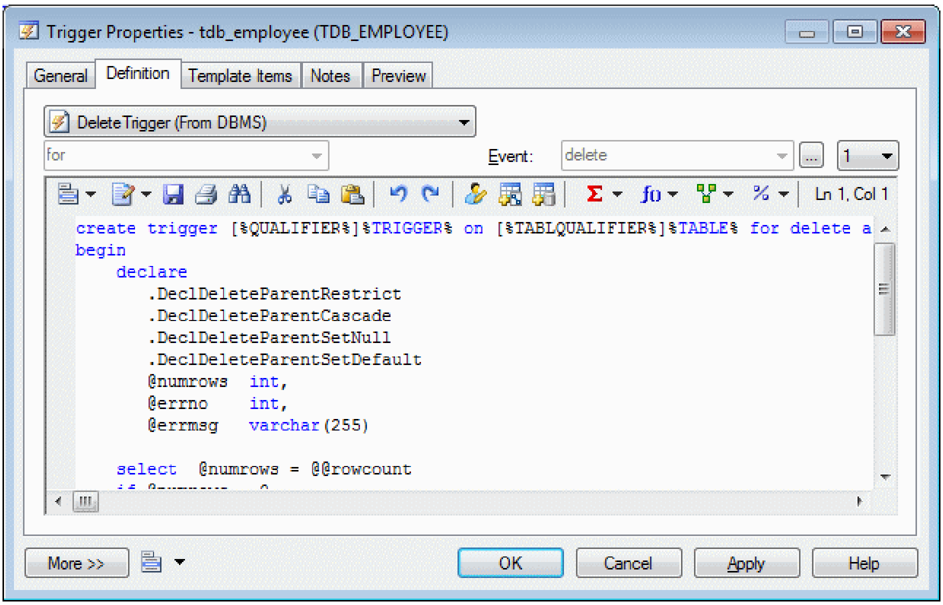
In the PowerDesigner interface, table and view triggers are called simply *triggers*, while DDL or database triggers are called *DBMS triggers*. View and DBMS triggers are not supported by all DBMSs.

PowerDesigner provides trigger templates to generate triggers (see *Trigger Templates* on page 127) and template items, which are reusable blocks of SQL script that can be inserted into triggers or trigger templates (see *Trigger Template Items* on page 128) and you can modify these templates and items and create your own.

Creating a Table or View Trigger

You can create a trigger for a table from its property sheet and base it on a PowerDesigner template, or on a template of your own, or write it from scratch.

1. Open the table or view property sheet, and then click the **Triggers** tab.
2. Click the **Add a Row** tool to create a new trigger, enter a name and code, and then click the **Properties** tool to open its property sheet.
3. Click the **Definition** tab, and select a trigger template (see *Trigger Templates* on page 127) from the Template list. The time and event fields will be set and the template code copied into the definition editor.



Note: You can create a trigger by entering code by hand, but we recommend that you use a template as this will simplify reuse of your code and make your triggers more portable.

4. [optional] Modify the trigger definition code. You can insert trigger template items (see *Trigger Template Items* on page 128), use PDM variables and macros and various other tools available from the toolbar (see *Writing SQL Code in PowerDesigner* on page 281).

   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 125).
5. Click **OK** to return to your model.

Creating Triggers from References

You can create triggers to enforce referential integrity individually or instruct PowerDesigner to create them by default.

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 (see *Reference Properties* on page 175), and then click **OK** to return to the diagram.
4. If you have set the **Automatically rebuild triggers** model option (see *Reference Model Options* on page 15,) then triggers will have been created automatically in the parent and child tables. To verify this open the table property sheet and click the **Triggers** tab. If the triggers are not present, you will need to rebuild your triggers manually (see *Rebuilding Triggers* on page 125).

   Note: To instruct PowerDesigner to implement referential integrity between tables using triggers by default whenever you create a reference, select **Tools** > **Model Options**, click **Model Settings** > **Reference** in the Category list, select `Trigger` in the **Default implementation** list.

Creating a DBMS Trigger

DBMS triggers are not associated with any table or view. You create them directly under the model.

You can create a DBMS trigger in any of the following ways:

- Select **Model** > **Triggers** > **DBMS Triggers** to access the List of DBMS Triggers, and click the **Add a Row** tool
- Right-click the model (or a package) in the Browser, and select **New** > **DBMS Trigger**

For general information about creating objects, see *Core Features Guide > Modeling with PowerDesigner > Objects*.

Trigger and DBMS Trigger Properties

To view or edit a trigger's properties, double-click its Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

Property	Description
Name/Code/ Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
Owner	Specifies the user who is the owner of the object. This is usually its creator. Use the tools to the right of the list to create, browse for, or view the properties of the currently selected user.
Table	[Table or view triggers only] Specifies the table to which the trigger belongs.
Scope	[DBMS triggers only] Specifies the scope of the DBMS trigger. You can choose either Schema or Database, and this choice will control the types of events that you can select in the DBMS trigger definition.
Generate	Specifies to generate the trigger.
User-defined	[Read-only] Specifies that the trigger definition has been modified. You modify a trigger definition when you change the trigger template script in the Definition tab of the trigger
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

Definition Tab

This tab allows you to enter code for the trigger. For information about the tools available on the toolbar, see *Writing SQL Code in PowerDesigner* on page 281. The following properties are available:

Property	Description
Template	Specifies the template on which the trigger is based (see *Trigger Templates* on page 127). The **User-defined** button is automatically depressed when you modify the definition of a trigger. Click the button to release it and restore the template trigger definition.
Time	Specifies when the trigger will fire in relation to the associated event. The content of the list depends on the values defined in the trigger template and in the Time entry in the Trigger category of the DBMS.
Event	Specifies the event that will cause the trigger to fire. Click the ellipsis tool to the right of this field to select multiple events (see *Defining Triggers with Multiple Events* on page 125) For table and view triggers, this field is a list, the content of which depends on the values defined in the trigger template and in the Event entry in the Trigger category of the DBMS. You can add your own events to this entry and they will appear in this list. For DBMS triggers, this field allows you to enter any text.
Order	[table and view triggers only] Specifies the firing order of trigger.

The following tabs are also available:

- **Template Items** - lists the trigger template items available for use in the trigger definition (see *Trigger Template Items* on page 128).
- **Preview** - displays the SQL code that will be generated for the trigger (see *Previewing SQL Statements* on page 285).

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.

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;
/
```

Indicating Trigger Order for Multiple Triggers

Some DBMSs allow you to have multiple triggers for the same insert, update, or delete event at the same time. You can indicate the order in which each trigger within the group fires.

In the following example, a company is considering candidates for various positions, and must ensure that new employees are offered a salary that is within the range of others working in the same field, and less than their prospective manager.

The `EMPLOYEE` table contains two `BeforeInsert triggers` to perform these tests:

```
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
```

1. Open the trigger property sheet and click the **Definition** tab.
2. Select a number from the **Order** list to indicate the position in which the trigger fires.
3. Click **OK** to return to your model.

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.

Rebuilding Triggers

PowerDesigner can rebuild triggers to ensure that they are attached to all tables joined by references to ensure referential integrity. You can instruct PowerDesigner to automatically rebuild triggers whenever a relevant change is made and you can manually rebuild triggers at any time.

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.

To instruct PowerDesigner to automatically rebuild triggers, select **Tools > Model Options**, click **Model Settings > Trigger**, select **Automatically rebuild triggers**, and click **OK**. PowerDesigner rebuilds all triggers and will, from now on, rebuild triggers whenever you make a relevant change in the model.

To rebuild triggers manually:

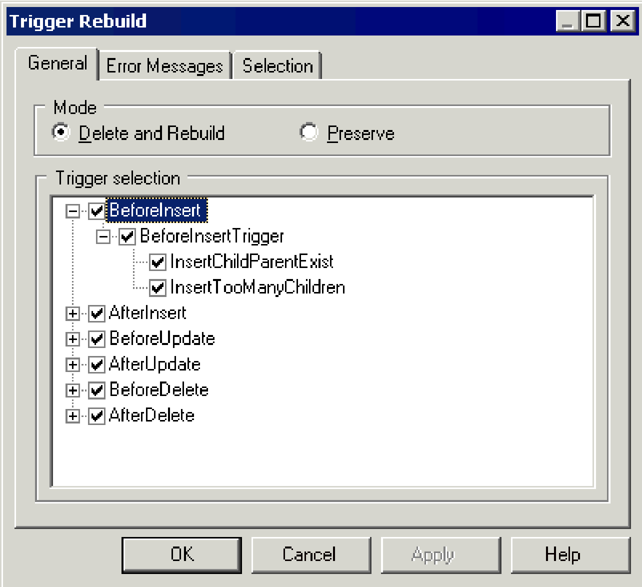
1. Select **Tools** > **Rebuild Objects** > **Rebuild Triggers**
2. Specify a rebuild mode. You can choose between:
   - Delete and Rebuild – all triggers attached to templates are deleted and rebuilt, including those to which you have made modifications

- 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. Expand the tree and select the types to rebuild. 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`:



4. [optional] Click the **Error Messages** tab to define the types of error messages to generate (see *Generating a User-Defined Error Message* on page 139).
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.

Note: If you change the target DBMS family, for example from Sybase to Oracle or IBM DB2, triggers are automatically rebuilt.

For information about rebuilding dependencies between triggers and other objects, see *Tracing Trigger and Procedure Dependencies* on page 142.

Trigger Templates

PowerDesigner trigger templates allow you to write trigger code in a modular reusable fashion. We provide basic templates for `before`, `after`, and `with insert`, `update`, and `delete` events and for other types of triggers where supported by the DBMS. You can modify the code specified in these templates or create your own templates in the DBMS definition file or in your model.

To apply a trigger template to your trigger definition, select the template from the list on the trigger property sheet **Definition** tab (see *Trigger and DBMS Trigger Properties* on page 122).

To review or modify the provided trigger templates, select **Database > Edit Current DBMS**, and then click the **Trigger Templates** tab. You cannot delete or rename these templates.

Warning! The resource files provided with PowerDesigner inside the `Program Files` folder cannot be modified directly. To create a copy for editing, use the **New** tool on the resource file list, and save it in another location. To include resource files from different locations for use in your models, use the **Path** tool on the resource file list.

To create a new template, click the **Create from Trigger Template** tool (to copy the code of an existing template to your new template) or the **Add a Row** tool (to start from scratch).

Note: You can, alternatively, create trigger templates in your model by selecting **Model > Triggers > Trigger Templates**, but these templates will not be accessible from other models.

Trigger Template Properties

The **General** tab contains the following properties:

Property	Description
Name/Code/Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
DBMS	Specifies the parent DBMS.
Applies to	[for DBMSs supporting multiple types of triggers] Specifies whether the template can be applied to table, view, or DBMS triggers.
Trigger time	Specifies when triggers based on the template will fire in relation to their associated event.
Trigger event	Specifies the event that will cause the firing of triggers based on the template.

Property	Description
Trigger name	Specifies the conventions for naming triggers based on the template.
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

The following tabs are also available:

- **Definition** - Contains a field for entering the trigger code for the template.You can use trigger template items, PDM variables and macros and other tools available from the toolbar (see *Writing SQL Code in PowerDesigner* on page 281).
- **Template Items** - Lists the template items (see *Trigger Template Items* on page 128) that are defined in the trigger template and that will be generated when a trigger is generated from the template.

  Note: If you delete a template item from this list, it is not deleted from the template definition, but is excluded from generation when rebuilding triggers. PowerDesigner-provided template items listed on this tab are generated in a trigger if they match the trigger implemented referential integrity defined for a reference attached to the table. User-created template items are always generated regardless of trigger referential integrity constraints.

Trigger Template Items

Trigger template items are named reusable blocks of script that can be inserted into triggers or trigger templates. 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.

To insert a trigger template item into your trigger or template definition, click the **Add Trigger Item from Model** or **Add Trigger Item from DBMS** tool, select the items from the list and click **OK**. The item is inserted with a dot followed by its name, and is also added to the list on the **Template Items** tab. 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
/
```

Note: Certain DBMSs require that a cursor and variables are declared for each template item before the template item name is used in the script. You can use the following format to declare a template item:

```
.Decltemplate item name
```

For example, the trigger definition for Oracle 8 declares and then inserts the `.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;
/
```

To review or modify the provided trigger template items, select **Database > Edit Current DBMS**, and then click the **Trigger Template Items** tab. You cannot delete or rename these items.

Warning! The resource files provided with PowerDesigner inside the `Program Files` folder cannot be modified directly. To create a copy for editing, use the **New** tool on the resource file list, and save it in another location. To include resource files from different locations for use in your models, use the **Path** tool on the resource file list.

To create a new template item, click the **Create from DBMS Trigger Item** tool (to copy the code of an existing item to your new item) or the **Add a Row** tool (to start from scratch).

Note: You can, alternatively, create trigger template items in your model by selecting **Model > Triggers > Trigger Template Items**, but these templates will not be accessible from other models.

Trigger Template Item Properties

The **General** tab contains the following properties:

Property	Description
Name/Code/Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
DBMS	Specifies the parent DBMS.
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

The following tabs are also available:

- **Definition** - Contains a field for entering the trigger code for the item.You can use PDM variables and macros and other tools available from the toolbar (see *Writing SQL Code in PowerDesigner* on page 281).
- **Declaration** - Contains a field for entering the declaration for the item in trigger scripts.

PowerDesigner Pre-Defined Trigger Template Items

PowerDesigner provides pre-defined template items for the trigger templates defined in each 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. The item is available for generation if it is present on the **Template Items** tab of a trigger property sheet and will be generated if it is present on the **Template Items** tab of a trigger template property sheet.

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

Template item	Integrity constraint	Description
`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
`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

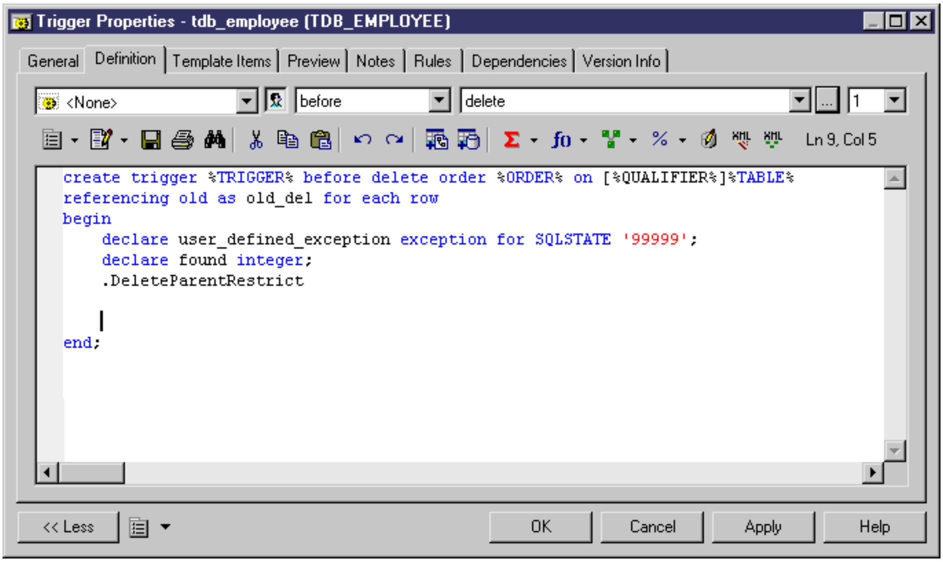
You can insert the following template items in any trigger template. They generate error messages that indicate the violation of an integrity constraint.

Template item	Description
UseErrorMsgText	Error handling without a message table
UseErrorMsgTable	Error handling with a message table

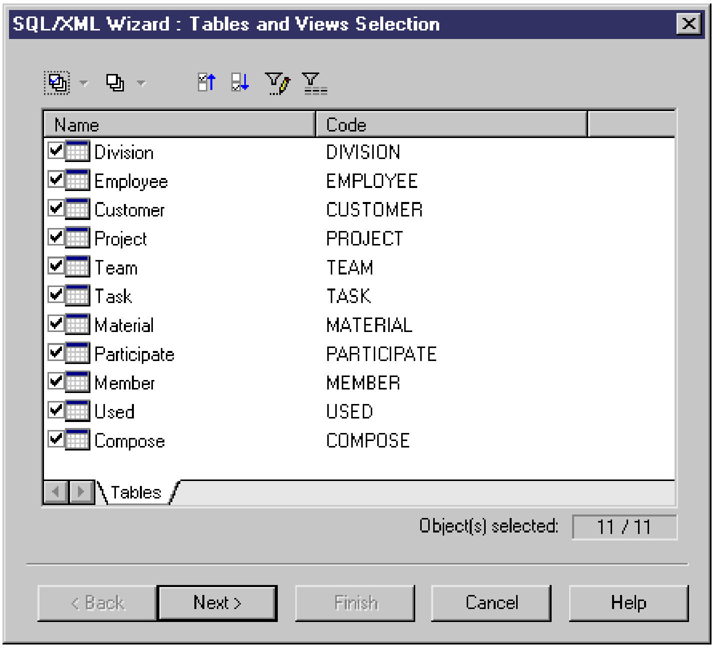
Creating SQL/XML Queries with the Wizard

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.

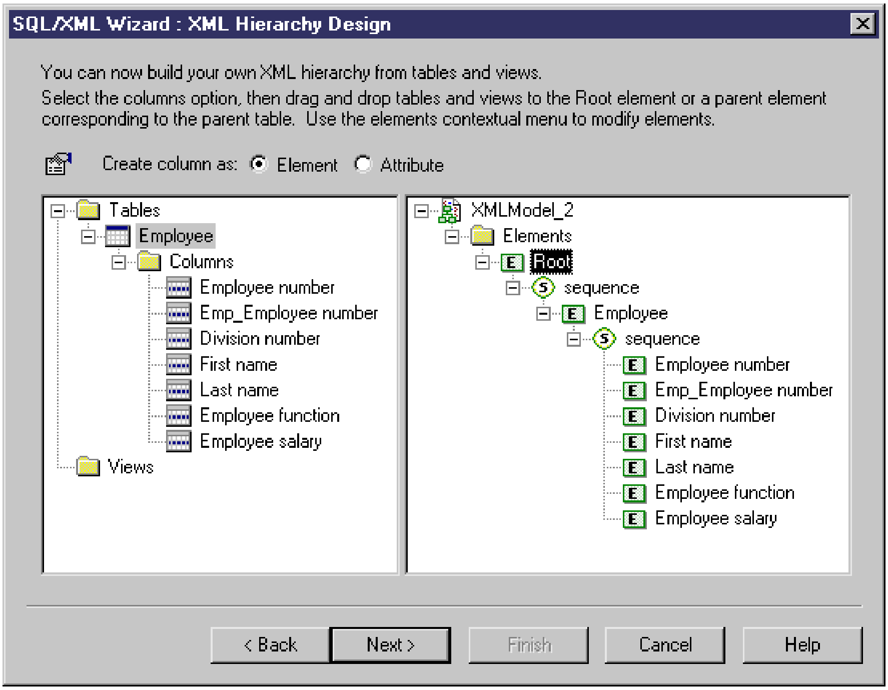
1. Open the trigger property sheet, click the *Definition* tab and position the cursor in the trigger definition where you want to insert the SQL/XML query:



2. Click the *SQL/XML Wizard* tool to launch the wizard at the Tables and Views Selection page:



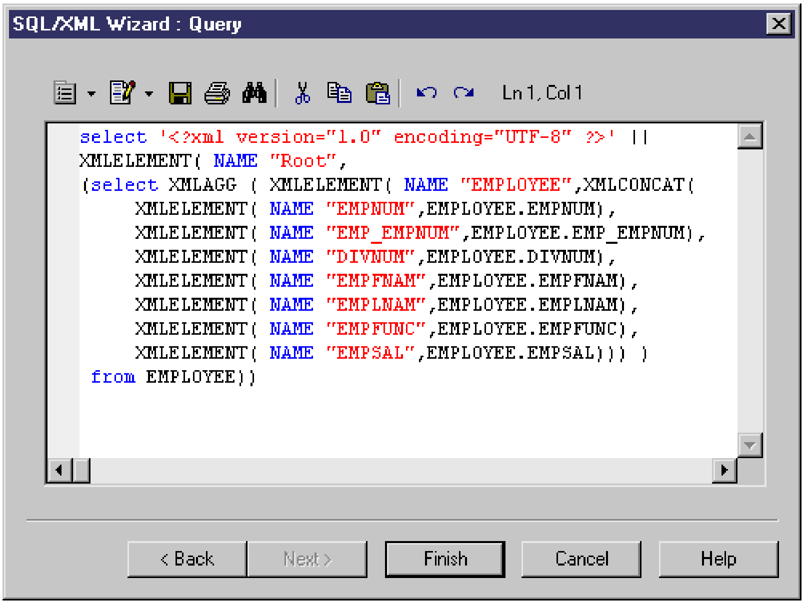
3. Select the tables and views that you want to include in your query and click Next to go to the XML Hierarchy Design page:



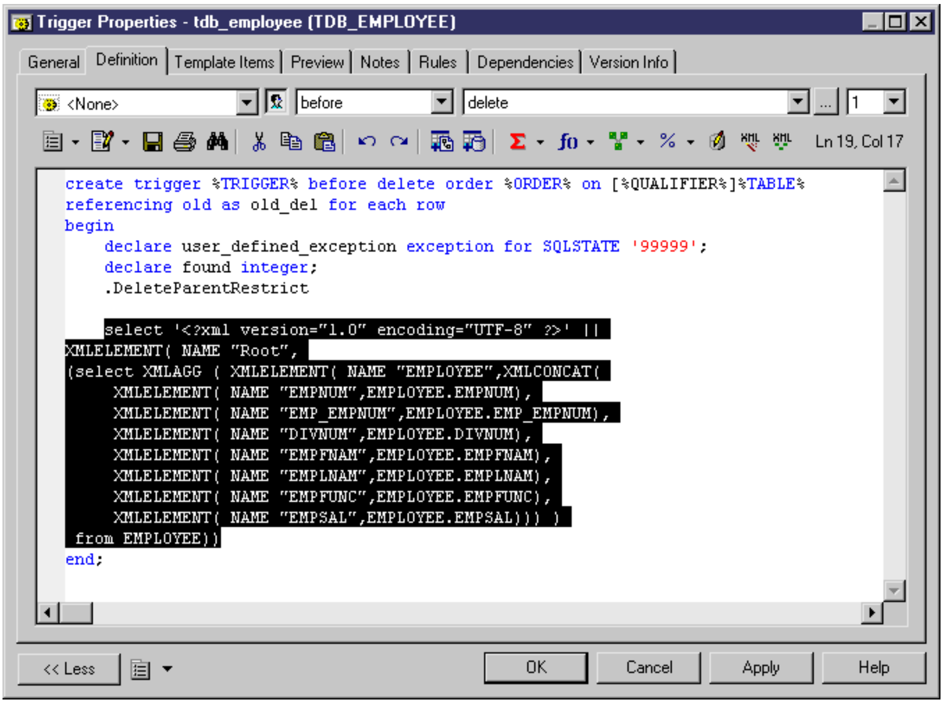
On this tab, you construct the XML hierarchy that you want to generate:

- 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.

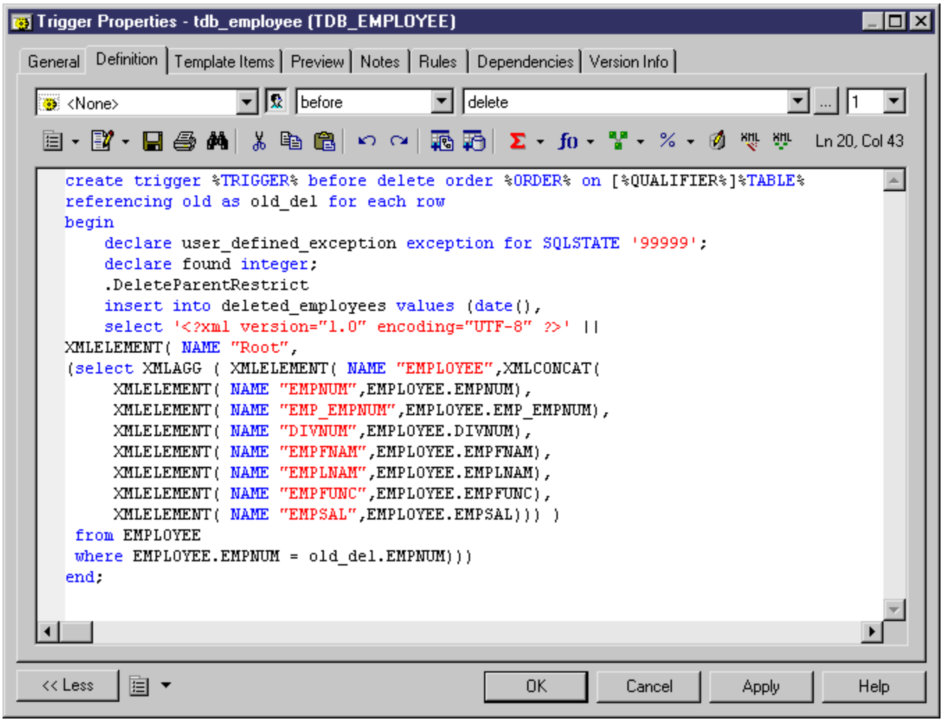
4. You can build your XML hierarchy using the following techniques:
   - Specify whether columns will be generated as elements or attributes by using the radio buttons above the panes.
   - Drag and drop a table, view, or column onto a node in the XML hierarchy. You must respect the PDM hierarchy: You 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:



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



7. [optional] Add code to complete the SQL/XML query:



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.

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* on page 290.

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* on page 297.

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.

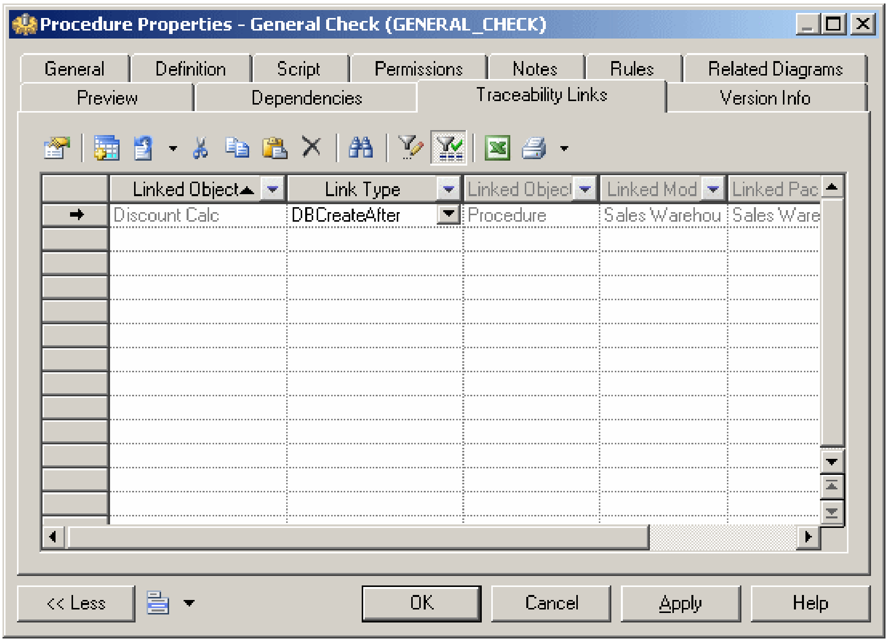
Defining a Generation Order for Stored Procedures

You can define the order of the generation of stored procedures by using traceability links with a type of *DBCreateAfter*. The procedure from which you start the traceability link is dependent on the procedure you link it to, and this influent procedure will be generated before the dependent procedure.

For example, a publisher may decide to sell certain books at a reduced rate (15%) when a customer's order is above 10 000$. The `GENERAL CHECK` stored procedure verifies orders globally by checking availability, the order amount, if a discount rate is required, and so on. This procedure calls the `DISCOUNT CALC` procedure to calculate the 15% discount rate. Consequently, `DISCOUNT CALC` must be generated before `GENERAL CHECK`, and you can enforce this by creating a traceability link of type `DBCreateAfter` from `GENERAL CHECK` to `DISCOUNT CALC`.

Note: There is a model check to warn you if you create a reflexive or circular set of traceability links of type `DBCreateAfter`. If generate without correcting this error, procedures will be generated in alphabetical order, without taking into account the generation order.

1. Open the property sheet of the dependent stored procedure and click the **Traceability Links** tab.
2. Click the **Add Objects** tool, click the **Procedure** sub-tab in the Add Object selection dialog, select the influent stored procedure, and click **OK**.
3. Click in the **Link Type** column, click the down arrow and select `DBCreateAfter`.



4. Click **OK** to close the property sheet and return to your model.

Note: You can also create `DBCreateAfter` traceability links using the **Traceability Links** tool (see *Defining a Generation Order for Views* on page 118). For detailed information about traceability links, see *Core Features Guide > Linking and Synchronizing Models > Getting Started with Linking and Syncing > Creating Traceability Links*.

Creating User-Defined Error Messages

You can create a message table in your database to store user-defined error messages. When you select trigger generation parameters, you can choose to generate an error message from this 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**.
4. Select a data source, fill in connection parameters, and click **Connect**.

   An SQL query editor box is displayed.

5. Enter an SQL statement to insert a message number and text in the appropriate columns. 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')
```

6. Click **Execute**.

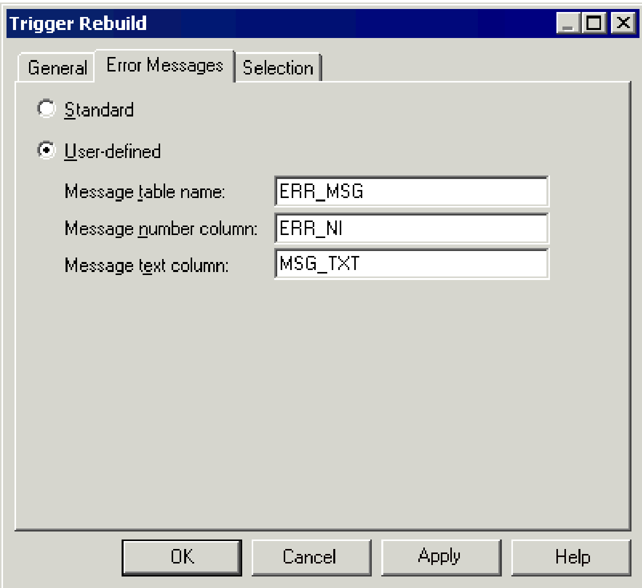
   A message box tells you that the command has been successfully executed.

7. Click **OK** to return to the SQL query dialog.
8. Click **Close**.

Generating a User-Defined Error Message

You can choose to generate a user-defined error message from the trigger generation parameters box.

1. Select **Tools** > **Rebuild Objects** > **Rebuild Triggers**.
2. Click the **Error Messages** tab, and select the **User-defined** radio button.
3. Enter the name of the table that contains the error message, the name of the column that contains the error number, and the name of the column that contains the error message text.



4. Click the **General** tab and select the mode and triggers to create.
5. Click the **Selection** tab and select the tables for which you want to create triggers.

   For more information on rebuilding triggers, see *Rebuilding Triggers* on page 125.

6. Click **OK**.

   The trigger rebuilding process is shown in the Output window.
7. Select **Database > Generate Database**, select generation parameters as required (see *Generating Triggers and Procedures* on page 136), and click **OK**.

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

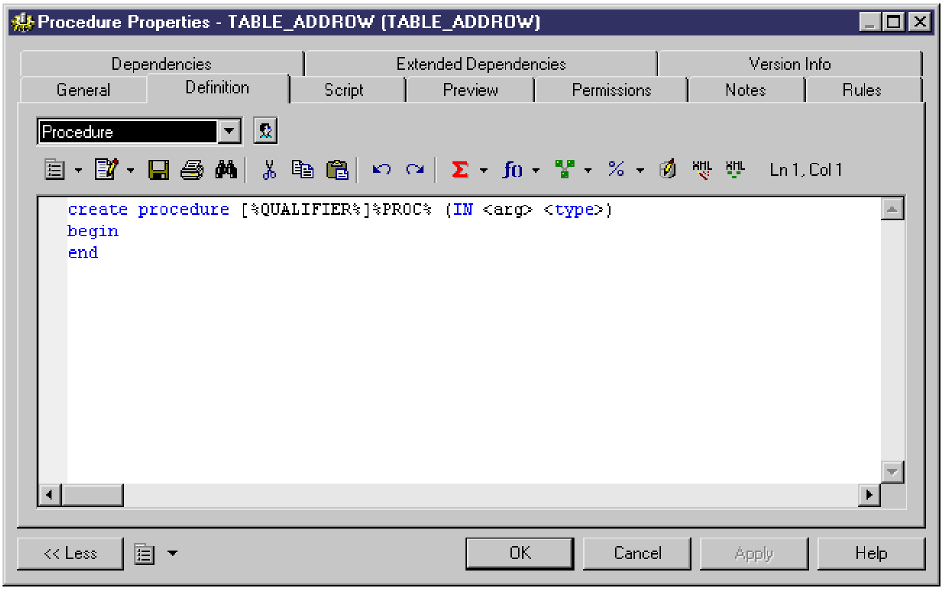
You can create a stored procedure or function from a table property sheet or from the Toolbox, Browser, or **Model** menu.

- Use the Procedure tool in the diagram Toolbox
- 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 *Core Features Guide > Modeling with PowerDesigner > Objects*.

You can create a procedure based on one of the PowerDesigner templates or on a template of your own.

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:



5. [optional] Select a procedure template from the Template list (see *Procedure Templates (PDM)* on page 148).
6. Modify the procedure definition code. You can use PDM variables and macros and various other tools available from the toolbar (see *SQL Editor Tools* on page 281).
7. You can also modify the procedure's other properties. For a full list of the properties available, see *Procedure Properties* on page 141.
8. Click OK in each of the dialog boxes.

Note: 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

To view or edit a procedure's properties, double-click its diagram symbol or Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

Property	Description
Name/Code/ Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
Owner	Specifies the name of the procedure owner.
Table	Specifies the table to which the procedure is attached. Use the tools to the right of the list to create, browse for, or view the properties of the currently selected table.
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

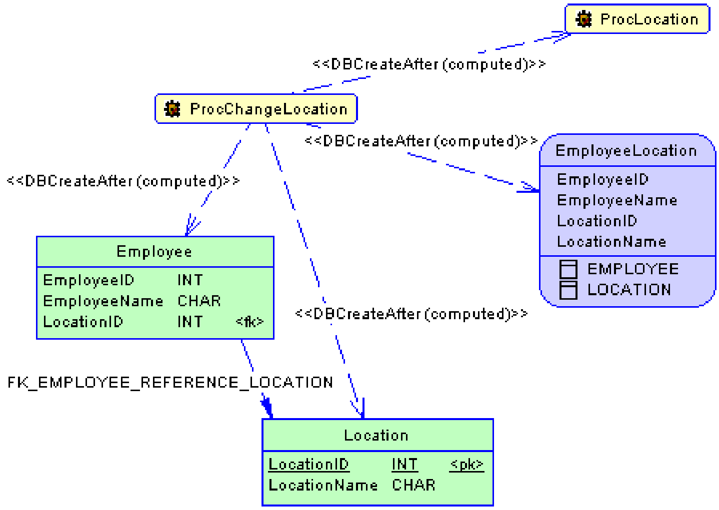
The following tabs are also available:

- Definition - allows you to enter the SQL code for the procedure. For information about the tools available, see *SQL Editor Tools* on page 281.

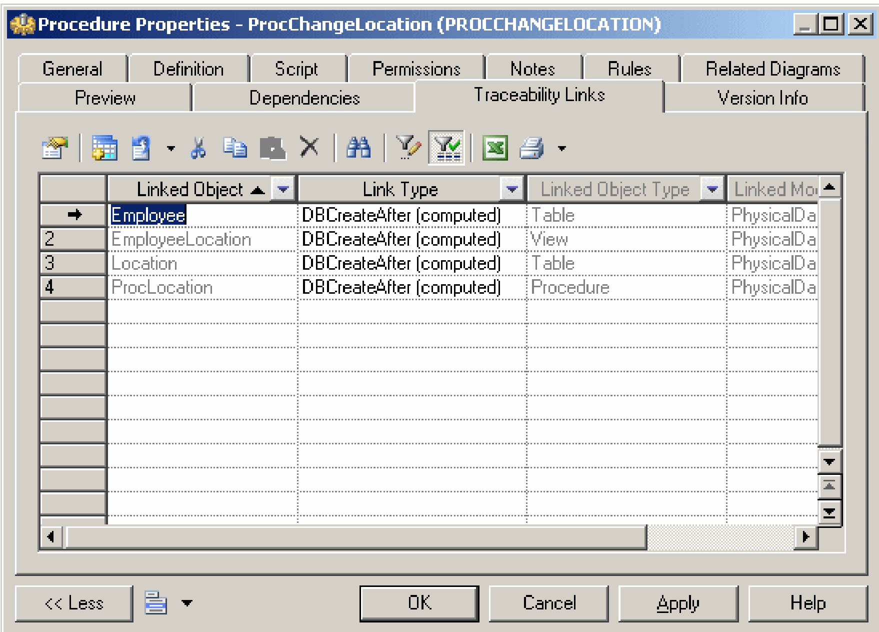
Tracing Trigger and Procedure Dependencies

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.

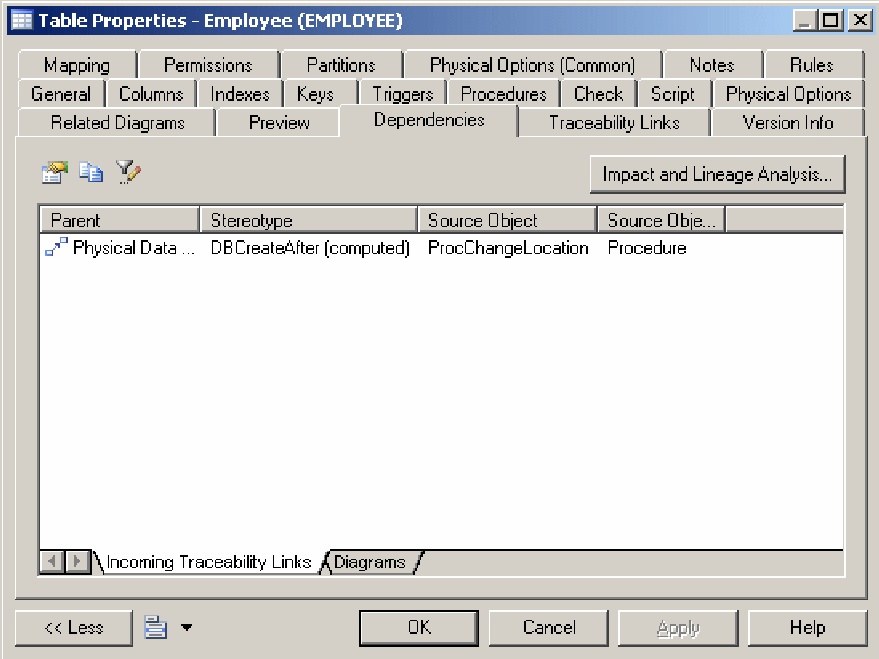
The diagram below shows a procedure, `ProcChangeLocation`, which is dependent on a number of other objects:



Its **Traceability Links** tab lists the objects upon which it depends, and the link type of `DBCreateAfter (computed)` shows that PowerDesigner has determined that it can only be created after these objects:



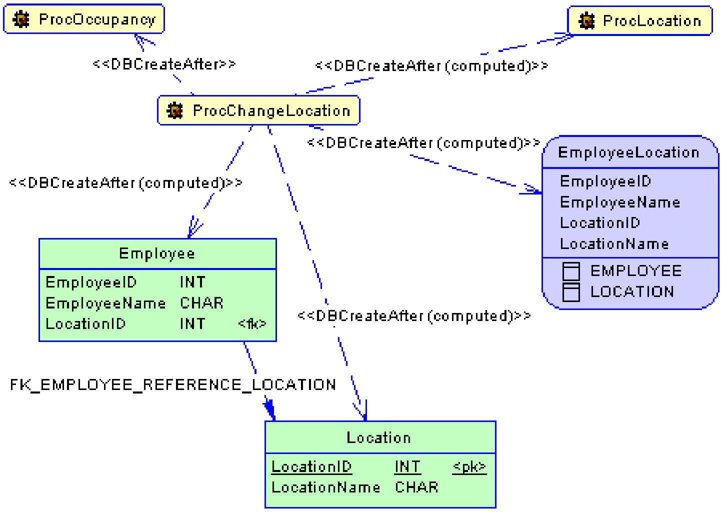
The `Employee` table **Dependencies** tab shows that `ProcChangeLocation` is dependent upon it, and if you were to perform an impact analysis prior to deleting the `Employee` table, you would be warned of the procedure's dependency on it.



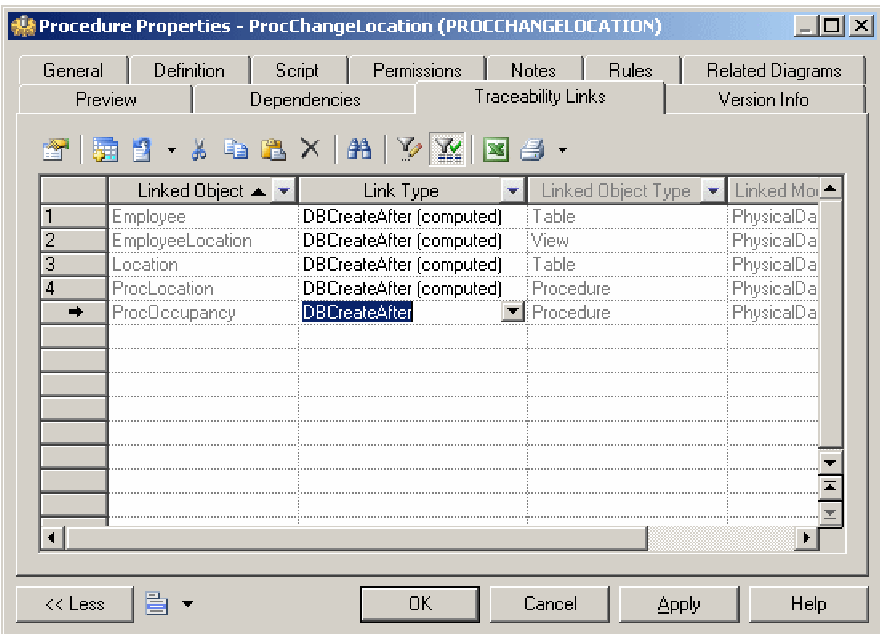
Creating Procedure Dependencies Manually

Since procedures have diagram symbols, you can manually add dependencies for them using the Traceability Links tool in the toolbox.

In the diagram below, `ProcChangeLocation` has a dependency on a new procedure, `ProcOccupancy`:



Since `ProcOccupancy` is not directly referenced in `ProcChangeLocation`, you must manually set the type of the link to `DBCreateAfter` on the **Traceability Links** tab of the `ProcChangeLocation` property sheet:



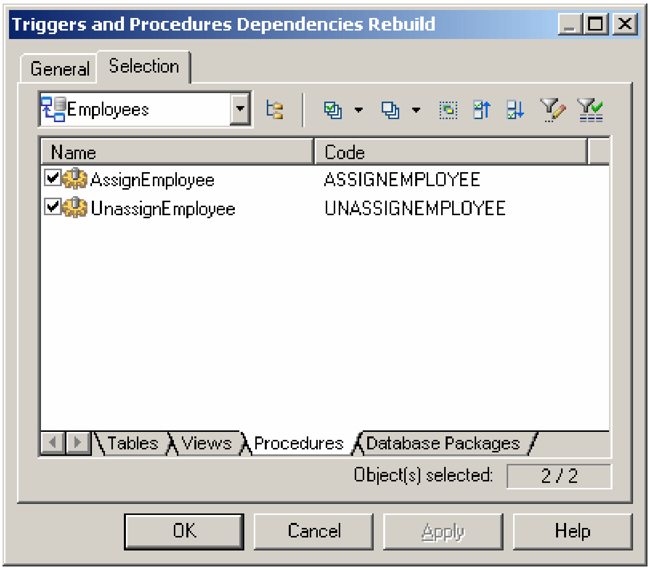
Rebuilding Trigger and Procedure Dependencies

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.

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.



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.

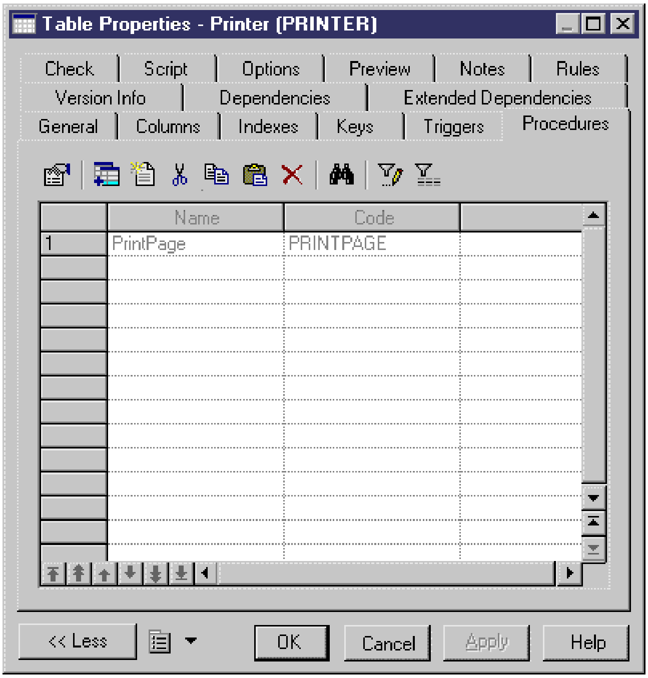
When you generate an OOM from a PDM, the procedures attached to a table become operations with the <<procedure>> stereotype in the generated class. By attaching procedures to tables, you are able to define class operations in the generated OOM.

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.

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.

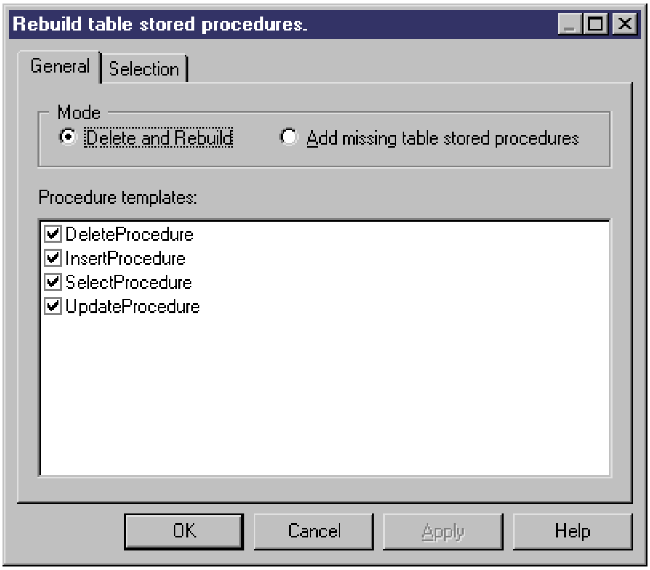


3. Click OK.

Rebuilding Procedures Attached to Tables

You can rebuild procedures attached to tables at any time.

1. Select **Tools > Rebuild Objects > Rebuild Table Stored Procedures** to open the Rebuild Table Stored Procedures window.



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 write table procedures in a modular reusable fashion. We provide basic templates for `insert`, `select`, `update`, and `delete` procedures. You can modify the code specified in these templates or create your own templates in the DBMS definition file.

To apply a procedure template to your procedure definition, select the template from the list on the procedure property sheet **Definition** tab (see *Procedure Properties* on page 141).

To review or modify the provided procedure templates, select **Database > Edit Current DBMS**, and then click the **Procedure Templates** tab. You cannot delete or rename these templates.

Warning! The resource files provided with PowerDesigner inside the `Program Files` folder cannot be modified directly. To create a copy for editing, use the **New** tool on the resource file list, and save it in another location. To include resource files from different locations for use in your models, use the **Path** tool on the resource file list.

To create a new template, click the **Add a Row** tool.

Procedure Template Properties

The **General** tab contains the following properties:

Property	Description
Name/Code/Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
DBMS	Specifies the parent DBMS.
Function	Specifies whether the template defines procedures or functions.
Procedure Name	Specifies the conventions for naming procedures based on the template.
Linked to table	Specifies whether the resulting procedure will be linked to a table.
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

The following tabs are also available:

- **Definition** - Contains a field for entering the procedure code for the template.You can use PDM variables and macros and other tools available from the toolbar (see *Writing SQL Code in PowerDesigner* on page 281).

Users, Groups, and Roles (PDM)

A user is a database object that identifies a person who can login or connect to the database. Groups and roles are used to simplify the granting of rights to users, as privileges and permissions granted to a group or role are inherited by users who belong to that group or incarnate that role.

Not all DBMSs support each of the concepts of user, role, and group.

Note: For many DBMSs, users can have an implicit schema, and PowerDesigner can reverse-engineer create statements contained within a schema. For SQL Server 2005 and higher, where users can have multiple schemas, PowerDesigner reverse-engineers schemas as separate objects (see *Schemas (SQL Server)* on page 432).

Creating a User, Group, or Role

You can create a user, group, or role from the Browser or **Model** menu. You can also create a user from the **Owner** field of various objects.

- Select **Model > Users and Roles > *Type*** to access the appropriate model object list, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New > *Type***.
- [users] Click the **Create** tool to the right of the **Owner** field on the **General** tab of a table (see *Table Properties* on page 76) or other object that allows you to specify an owner.

For general information about creating objects, see *Core Features Guide > Modeling with PowerDesigner > Objects*.

User, Group, and Role Properties

To view or edit a user, group, or role's properties, double-click its Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

Property	Description
Name/Code/ Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
Password	[users and groups] Password used for database connection.
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

The following tabs are also available:

- Privileges - lists the system privileges granted to the user (see *Granting System Privileges* on page 151).
- Permissions - lists the operations that the user is permitted to perform on various database objects (see *Granting Object Permissions* on page 154).
- Users - [groups and roles] Lists the users belonging to the group or role.
- Groups - [groups and roles] Lists the groups belonging to the group or role.
- Roles - [roles] Lists the roles belonging to the role.

Assigning an Owner to an Object

The database user who creates an object (table, view, stored procedure, etc) is the owner of the object and is automatically granted all permissions on it. In a PDM, you can specify the owner of an object by attaching a user to it. Each object can have only one owner. Where other users must access the object, you can restrict object modifications to the owner and grant `Select` or other permissions for the other users.

Note: To automatically assign a default owner for any type of object that supports the concept of ownership, select **Tools > Model Options**, choose the appropriate object type in the left-hand pane, and select the appropriate user in the **Default owner** field (see *Other Object Model Options* on page 16).

1. Open the property sheet of the object to the **General** tab.
2. Select a user in the **Owner** field. To create a new user, click the **Create** tool to the right of this field.
3. Click **OK** to return to your model.

   Note: When generating to your database (see *Generating a Database from a PDM* on page 290), you can restrict the tables and other objects generated to only those belonging to a particular owner, by selecting the owner on the Database Generation dialog **Selection** tab.

Granting System Privileges

System privileges are granted to users, groups, and roles to give them the right to perform particular types of action in the database. By default, a user belonging to a group or having a role inherits the group or role privileges and these inherited privileges are identifies as such in the **Privileges** tab of the user property sheet. A user with an administrative profile is also allowed to *revoke* a privilege.

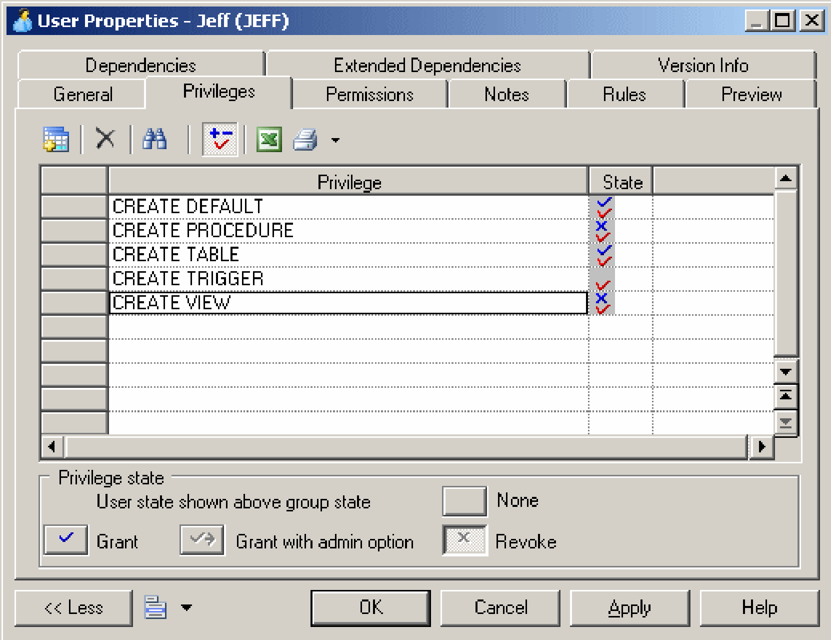
System privileges are used in association with object permissions (see *Granting Object Permissions* on page 154) to evaluate the rights of a user, group, or role. For example, even if a user has the `Modify` privilege, he cannot modify an object on which he has no `Update` permission.

Note: In some DBMSs, system privileges are called permissions. In PowerDesigner, the term privilege is reserved for any right granted to a user, a group, or a role. Permissions are defined for objects.

1. Open the property sheet of a user, role, or group, and click the **Privileges** tab.
2. [optional] Click the **Show/Hide All Inherited Privileges** tool to show privileges that have been inherited from a group. Inherited privileges are red, while privileges directly granted to the user are blue.
3. Click the **Add Objects** tool to choose one or more of the privileges available in the DBMS, and click **OK** to grant them to the user, role, or group:

   System privileges are defined in the DBMS definition file. To review and edit the list of available privileges, select **Database > Edit Current DBMS**, select the item **Script > Objects > Privilege > System**, and edit the list as appropriate. The **Privilege** category also contains entries that define the syntax for the necessary SQL statements for granting and revoking privileges. For more information, see *Customizing and Extending PowerDesigner > DBMS Definition Files > Script/Objects Category*.
4. [optional] To change the state of a privilege (whether granted directly, or inherited from a group), click in the **State** column to cycle through the available states, or click on the appropriate tools in the **Privilege state** group box at the bottom of the tab:

Privilege	Description
	**Grant** – [default] Assigns the privilege to the user.
	**Inherited/None** - Reverts the cell to the inherited state.
	**Revoke** – Revokes the privilege inherited from a group or role for the current user or group.
	**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.



5. When the privileges are correct, click **OK** to return to the model.

Generating Privileges

You can generate privileges to a script or to a live database connection.

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* on page 290.

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* on page 297.

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 granted to users, groups, and roles to give them the right to perform operations on particular database objects. PowerDesigner allows you to define permissions on tables, views, columns, procedures, packages, and other objects depending on your DBMS.

System privileges are used in association with object permissions (see *Granting System Privileges* on page 151) to evaluate the rights of a user, group, or role.

Note: The owner of an object (see *Assigning an Owner to an Object* on page 151) 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.

1. Open the property sheet of a user, role, or group, and click the **Permissions** tab. A sub-tab is displayed for each type of object supporting permissions. The columns in the list on each tab show the permissions available for a given type of object in the current DBMS (for example, Select, Insert, Alter, Delete, Update, etc).

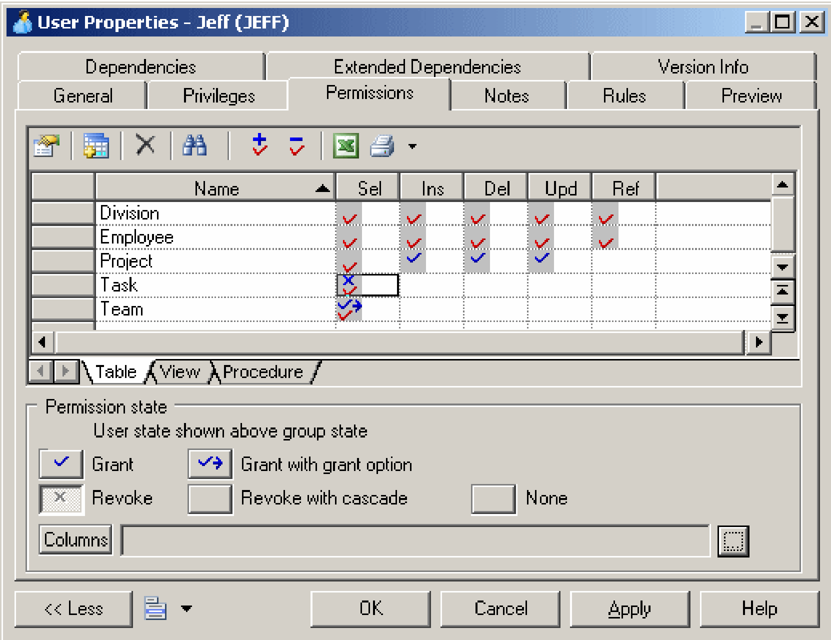
   The permissions available for each type of object are defined in the DBMS definition file. To review and edit the list of available permissions, select **Database > Edit Current DBMS**, select the item **Script > Objects > *object_type* > Permission**, and edit the list as appropriate. The syntax for inserting permissions in your scripts is defined in the **Script > Objects > Permission** category. For more information, see *Customizing and Extending PowerDesigner > DBMS Definition Files > Script/Objects Category*.

   Note: You can assign permissions for multiple users, groups, and roles to an object on the **Permissions** tab of its property sheet.

2. Click the **Add Objects** tool to choose one or more objects of the present type, and click **OK** to add them to the list to assign permissions. If the user belongs to a group with permissions on the added objects, these permissions appear in red in the list.
3. [optional] Click the **Show All Inherited Permissions** or **Hide Inherited Permissions** tool to show or hide permissions that have been inherited from a group. Inherited permissions are red, while permissions directly granted to the user are blue.
4. [optional] To change the state of a permission (whether granted directly, or inherited from a group), click in the appropriate column to cycle through the available states, or click on the appropriate tools in the **Permission state** group box at the bottom of the tab:

Permission	Description
	**Grant** – Assigns the permission to the user.
	**Inherited/None** - Reverts the cell to the inherited state.
	**Revoke** – Revokes the permission inherited from a group or role for the current user or group.

Permission	Description
	**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 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.

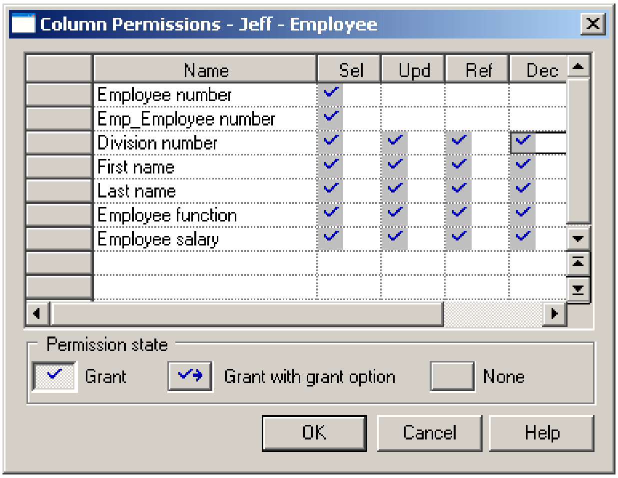


5. [optional] For tables, you can specify permissions on individual columns (see *Defining Column Permissions* on page 155).
6. When the permissions are correct, click **OK** to return to the model.

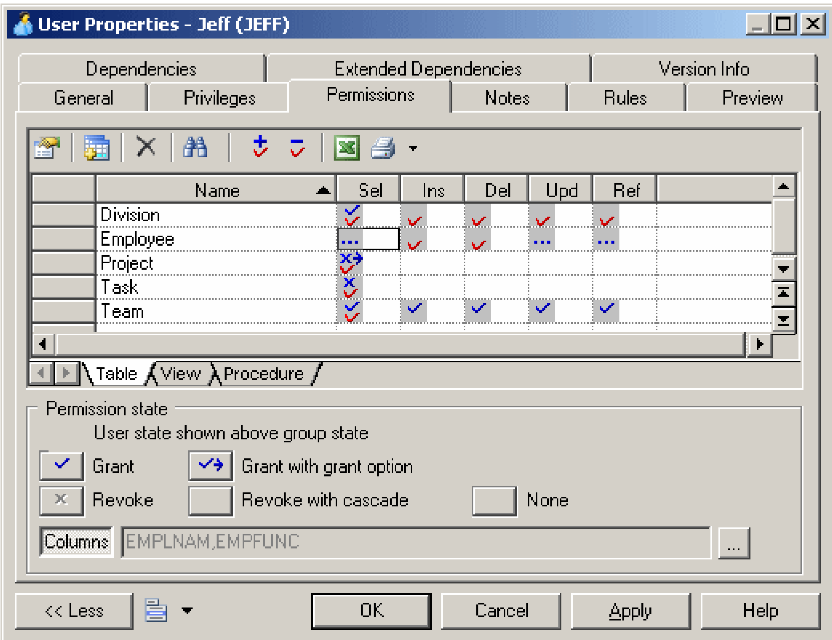
Defining Column Permissions

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.

1. Open the property sheet of a table, user, role, or group, and click the **Permissions** tab. For a table, select a user, group or role in the list to whom you want to grant column permissions. For a user, group or role, select a table in the list for which you want to specify permissions.
2. Click the ellipsis button to the right of the **Columns** field to open the Column Permissions dialog. The columns in the list show the permissions available for each of the table's columns.



3. To change the state of a permission (whether granted directly, or inherited from a group), click in the appropriate column to cycle through the available states, or click on the appropriate tools in the **Permission state** group box at the bottom of the tab.
4. Click **OK** to close the dialog and return to the property sheet. The cells for which specific permissions have been set for columns now contain ellipsis symbols. Click on one of these symbols to display the associated column permissions information in the **Columns** field:

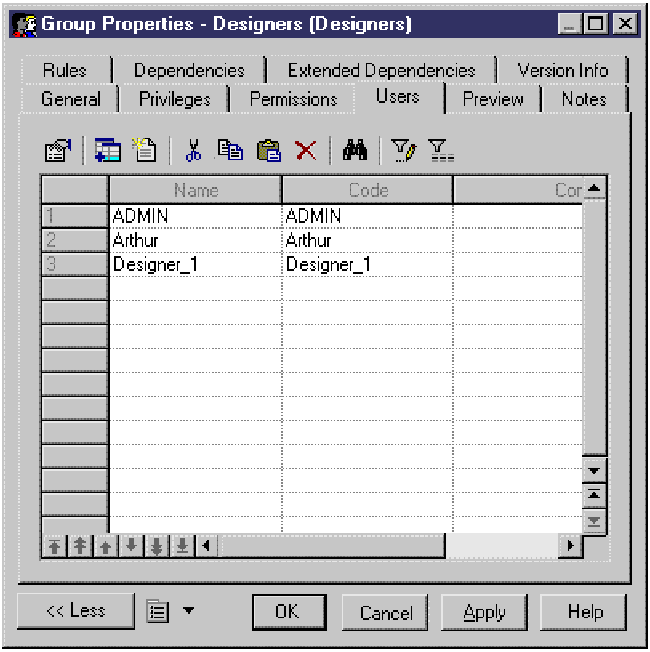


5. Click **OK** to close the property sheet and return to the model.

Assigning a User to a Group or Role

Once you have created a group or role, you can assign users to it.

1. Select **Model** > **Users and Roles** > **Groups** or **Roles** to open the appropriate list.
2. Select a group or 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 insert them into the group.



5. Click **OK** to return to the model.

Synonyms (PDM)

Synonyms are alternative names for various types of database object, which can be used to mask the name and owner of the object, provide location transparency for remote objects of a distributed database, and simplify SQL statements for database users.

For example, if the table `SALES_DATA` is owned by the user `JWARD`, you could define a synonym `Sales` for it to hide the ownership and simplify the required SQL select statement:

Standard Statement	Statement with Synonym
`SELECT * FROM jward.sales_data`	`SELECT * FROM sales`

You can create multiple synonyms for a base object (table, view, etc.), but each synonym can have only one base object. You can view the synonyms defined for a particular base object on the **Dependencies** tab of its property sheet. If you delete the base object of a synonym, the synonym is deleted as well.

Note: PowerDesigner supports the generation and 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, but then you should define a base object for it in your model.

Creating a Synonym

You can create synonyms from the **Model** menu.

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, select one or more objects, and click **OK** to create synonyms for them in the list.

   Note: By default, synonyms are created with the same name as their base objects.
3. Click in the **Name** column and enter a new name for the synonym. Alternatively, click the **Properties** tool to open the property sheet of the synonym and edit its name and other properties there.
4. Click **OK** to return to your model.

   For general information about creating objects, see *Core Features Guide > Modeling with PowerDesigner > Objects*.

Synonym Properties

To view or edit a synonym's properties, double-click its Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

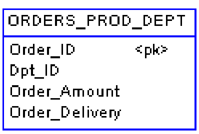
Property	Description
Name/Code/ Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
Owner	Specifies the user who is the owner of the object. This is usually its creator. Use the tools to the right of the list to create, browse for, or view the properties of the currently selected user.
Base Object	Specifies the name of the object that is aliased by the synonym. Click the **Select** tool to the right of the field to select an object from among the PDMs of the same DBMS family open in the workspace.
Visibility	Specifies whether the synonym is public (accessible to all users) or private (available only to its owner).
Type	[if your DBMS supports synonyms and aliases] Specifies whether to create a synonym or an alias, both of which are modeled in the same way in PowerDesigner.
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

Creating a View from a Synonym

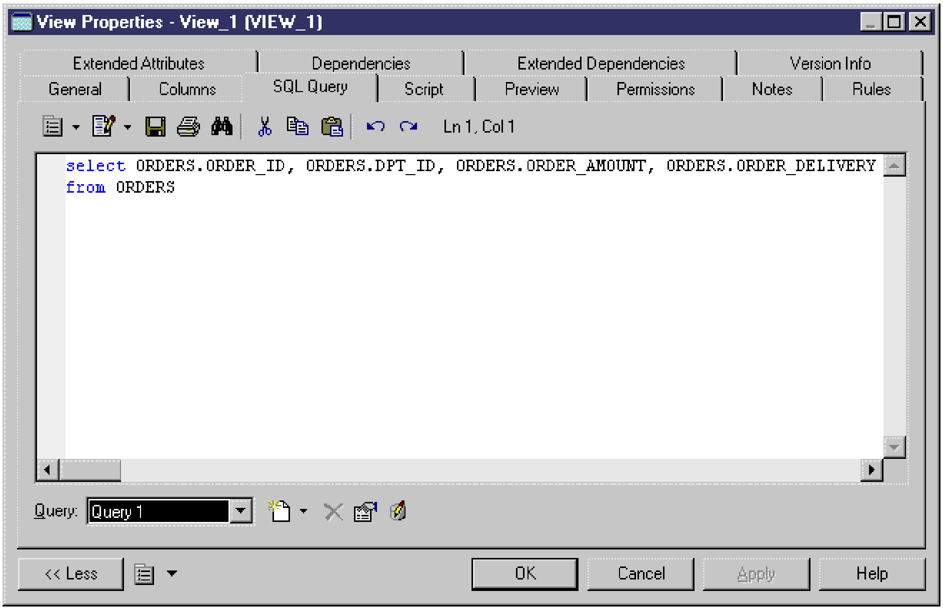
You can create views from synonyms in the same way as for tables. The view query displays the content of the object aliased by the synonym.

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 example, the `ORDERS_PROD_DEPT` table has a synonym `ORDERS`:

   If you create a view for the `ORDERS` synonym, the view query displays the select order of the table content:



For more information about views, see *Views (PDM)* on page 112.

Defaults (PDM)

Default objects are named values that can be assigned to columns or domains. Defaults are available for selection from the **Default** list on the **Check Parameters** tab of column and domain property sheets. Defaults are not supported by all DBMSs.

For example, if you must set a default value for all columns of type `city`, you can create a default object `citydflt` to assign the value `London` to it. To review how the default will be generated to your database, click the **Preview** tab:

```
create default CITYDFLT
  as 'London'
```

Creating a Default

You can create a default from the Browser or **Model** menu.

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

Note: You can also convert default values assigned to column and domains into default objects for reuse through rebuilding (see *Rebuilding Defaults* on page 162).

For general information about creating objects, see *Core Features Guide > Modeling with PowerDesigner > Objects*.

Default Properties

To view or edit a default's properties, double-click its Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

Property	Description
Name/Code/ Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
Owner	Specifies the user who is the owner of the object. This is usually its creator. Use the tools to the right of the list to create, browse for, or view the properties of the currently selected user.
Value	Specifies the value that will be generated for the default object.
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

The following tabs are also available:

- **Preview** - Displays the SQL code associated with the default (see *Previewing SQL Statements* on page 285).

Assigning a Default to a Column or a Domain

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

1. Open the property sheet of a column or a domain, and click the **Standard Checks** tab (see *Setting Data Profiling Constraints* on page 94).
2. Select a default in the **Default** list in the **Value** groupbox.

   You can, alternatively, enter a default value in the listbox. The value entered is assigned as a default value for the column or domain, but a default object is not created in the model, and the default cannot be reused elsewhere. If you enter a name that is already in the list, the relevant default object is attached to the column or domain.

Note: You can create default objects for reuse from default values, through the **Rebuild Default** command (see *Rebuilding Defaults* on page 162).

3. Click **OK** to return to your model.

Rebuilding Defaults

You can generate default objects from default values entered into the **Default** list on the **Check Parameters** tab of a column or domain property sheet. The new default objects replace the previously entered values, and can be reused with other columns and domains.

Note: If your model's DBMS does not support default objects and you have assigned default values to domains then, if you change to a DBMS that does support default objects, an object will be created for each value. Default values assigned to columns will not be converted into objects. When changing from a DBMS that supports default objects to one that does not, default objects are converted into default values.

1. Select **Tools** > **Rebuild Objects** > **Rebuild Defaults**, and enter the appropriate options:

Option	Description
Domains / Columns	Specifies the naming conventions for defaults applied to domains and columns respectively, which are both, by default, `D_%.U:VALUE%`. You can specify different names for each type of default, and use the following variables: • `%DOMAIN%`, `%COLUMN%`, `%TABLE%` - Code of the domain, column, or table using the default.
Mode	Specifies the type of rebuild. You can select either or both of: • Reuse default with identical value – Creates a single default for each value, even if the value is found in multiple columns and domains. If you deselect this option, multiple defaults may be created with the same value. • Delete and rebuild – Delete and rebuild all existing default objects.

2. [optional] Click the **Selection** tab to specify which domains and tables to search for defaults to rebuild.
3. 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 defaults.

Domains (CDM/LDM/PDM)

Domains allow you to group together a data type, length, precision, mandatoriness, check parameters, and business rules to standardize their application to a set of columns and entity attributes. You can define domains for columns of type ID, name, address, or any other kind of data whose use you want to standarize across multiples columns or attributes in your model.

Creating a Domain

You can create a domain from the Browser or **Model** menu.

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

For general information about creating objects, see *Core Features Guide > Modeling with PowerDesigner > Objects*.

Domain Properties

To view or edit a domain's properties, double-click its Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

Property	Description
Name/Code/ Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
Owner	[PDM] Specifies the user who is the owner of the object. This is usually its creator. Use the tools to the right of the list to create, browse for, or view the properties of the currently selected user.
Data type/ Length/ Precision	Specifies the form of data to be stored, such as numeric, alphanumeric, or Boolean, and, where appropriate, the maximum number of characters or numerals that can be stored, and the maximum number of places after the decimal point. Click the ellipsis button to choose from the list of standard data types (see *PowerDesigner Standard Data Types* on page 165). To review the data types permitted by your DBMS, select **Database > Edit Current DBMS** and navigate to **Script > DataType > PhysDataType**). The following variables specify length and precision requirements: • `%n` - length • `%s` - length with precision • `%p` - decimal precision For example, the data type `char(%n)` , requires you to specify a length.

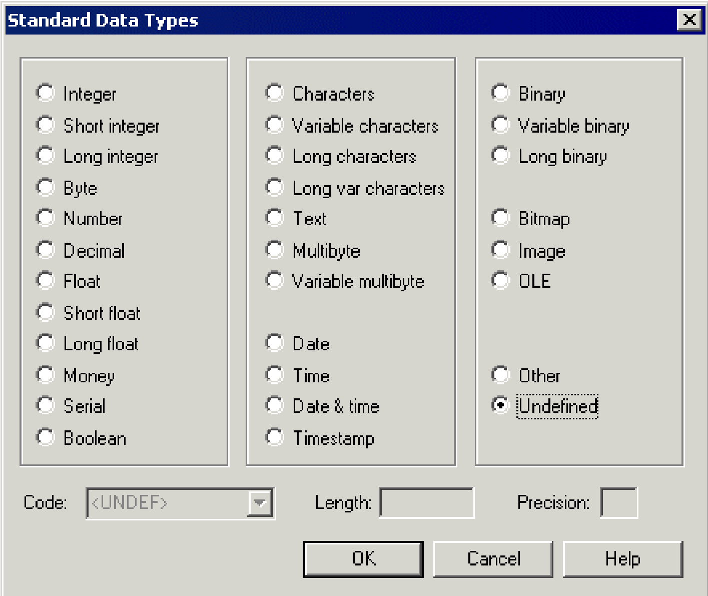
Property	Description
Mandatory	[if supported by your DBMS] Specifies that a non-null value must be assigned.
Identity	[if supported by your DBMS] Specifies that the column is populated with values generated by the database. Identity columns are often used as primary keys.
With default	[PDM] [if supported by your DBMS] Specifies that the column is populated with values generated by the database. Identity columns are often used as primary keys.
Profile	[PDM] Specifies a test data profile to use to generate test data (see *Populating Columns with Test Data* on page 98). Use the tools to the right of this field to create or browse to a profile, or to open the property sheet of the selected profile.
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

The following tabs are also available:

- **Standard Checks** - Specifies constraints to control the range and format of permitted data (see *Setting Data Profiling Constraints* on page 94)
- **Additional Checks** - Displays an editable SQL statement, initialized with the standard checks, which can be used to generate more complex constraints (see *Specifying Advanced Constraints* on page 97).
- **Rules** - Lists the business rules associated with the object (see *Business Rules (CDM/LDM/PDM)* on page 184).

PowerDesigner Standard Data Types

To open the list of Standard Data Types, click the ellipsis button to the right of the **Data Types** field on the **General** tab of a column, entity attribute, data item, or domain property sheet.



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 / SMALLINT	16-bit integer	—
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

Standard data type	DBMS-specific physical data type	Content	Length
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	—

Standard data type	DBMS-specific physical data type	Content	Length
Date & Time	datetime / DATE	Date and time	—
Timestamp	timestamp / TIMESTAMP	System date and time	—

Other Data Types

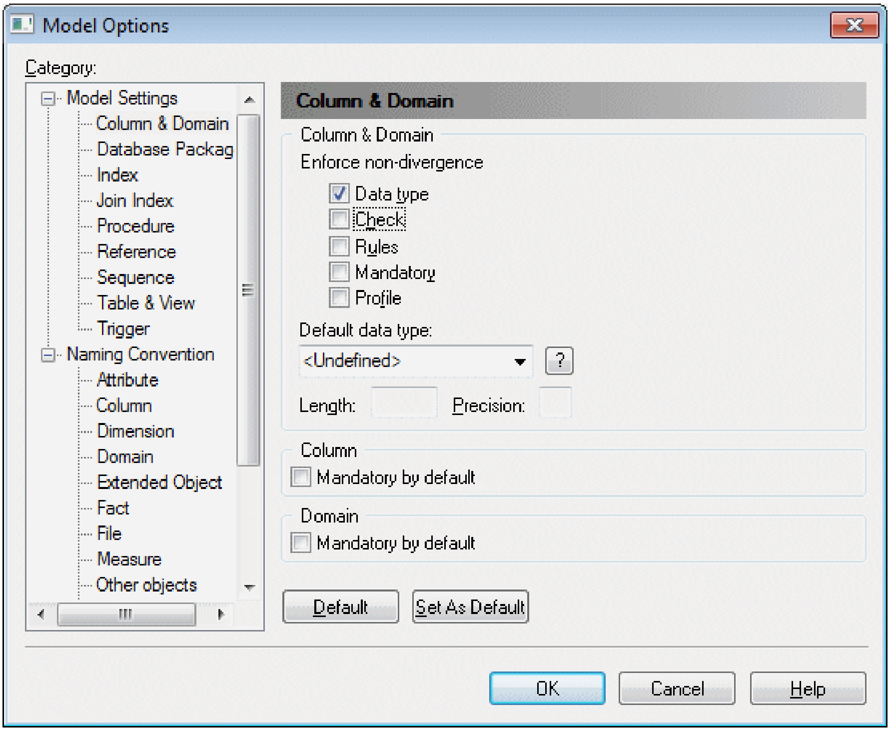
The following other data types are available:

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.	—

Controlling Non-Divergence from a Domain

You can specify which of the properties of your domains must be applied to the columns or entity attributes associated with the domain, and which propeties are permitted to diverge.

1. Select **Tools > Model Options** to open the Model Options dialog box. In a PDM, click the **Column and Domain** sub-category in the left-hand Category pan:



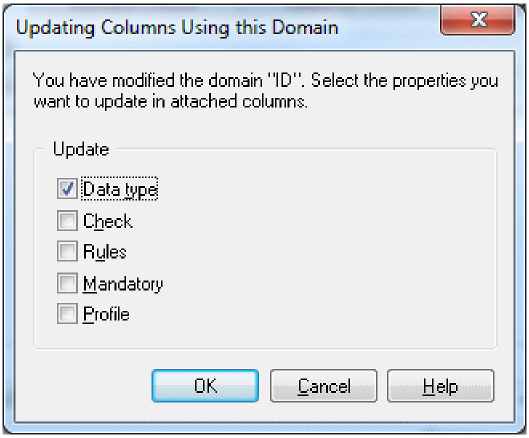
2. Select the checkboxes of the column or entity attribute properties that you want to prevent from diverging from those defined in the domain:

   - **Data type** - Data type, length, and precision.
   - **Check** (see *Setting Data Profiling Constraints* on page 94).
   - **Rules** (see *Business Rules (CDM/LDM/PDM)* on page 184).
   - **Mandatory** – Mandatory property of the column or attribute.
   - [PDM] **Profile** (see *Populating Columns with Test Data* on page 98).

3. Click **OK** to close the dialog and return to your model.

   You are prompted to apply domain properties to columns or attributes currently attached to the domain. If you click **OK**, the properties of these objects are modified in order to be consistent with the properties of their domain.

   If you subsequently modify properties of the domain that are not selected for enforcement, you will be prompted to apply your changes to the columns or attributes attached to the domain. To choose not to apply your changes, deselect the appropriate checkbox. Properties that are enforced may not be deselected and if you only modify enforced properties, then this dialog will not be displayed.



Note: Properties specified as non-divergent are read-only in lists and property sheets for associated columns and attributes. If you want to modify a non-divergent column or attribute property, you must detach the column or attribute from its domain.

Sequences (PDM)

Sequences are auto-incremented columns that allow you to define complex incrementations. Sequences are available for selection from the **Sequence** list on the **General** tab of column property sheets. Sequences are not supported by all DBMSs.

Note: If you generate a CDM or OOM from your PDM, then the data types of table columns attached to sequences are converted to serial numerical data types for entity properties or class attributes with the format `NO%`*n*, where *n* indicates the length of the data type.

Creating a Sequence

You can create a sequence from the Browser or **Model** menu.

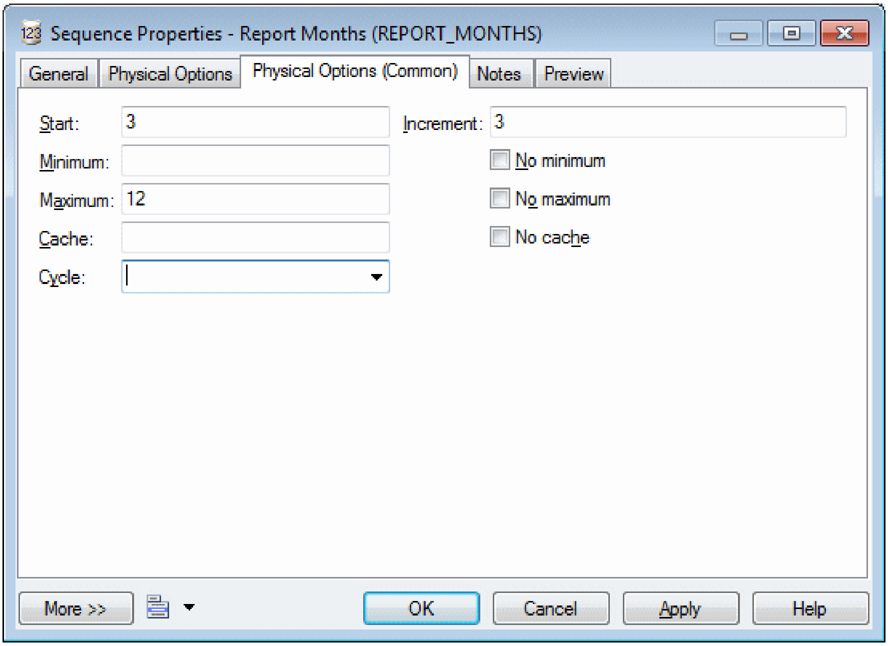
Note: If your model's DBMS does not support sequences and contains auto-incremented columns then, if you change to a DBMS that does support sequences, one will be created for each auto-incremented column. When changing from a DBMS that supports sequences to one that does not, sequences are converted into auto-incremented columns.

1. Select **Model > Sequences** to open the List of Sequences, and click the **Add a Row** tool. Then click the **Properties** tool to open the property sheet of the new sequence.

   Alternatively, you can create a sequence by right-clicking the model (or a package), and selecting **New > Sequence**.

2. Enter an appropriate name for the sequence and then click the **Physical Options** or **Physical Options (Common)** tab and enter any DBMS-specific options.

The following example shows a sequence created in Sybase SQL Anywhere to represent the months in a year when quarterly reports are published.



For information about working with physical options, see *Physical Options (PDM)* on page 88.

3. Click **OK** to save the sequence and return to your model.

Assigning a Sequence to a Column

You can select a sequence from the **Sequence** list on the **General** tab of a column property sheet. You must enable sequences with the **Rebuild Triggers** command.

1. Open the property sheet of a column with a numeric data type, and select a sequence in the **Sequence** list on the **General** tab.
2. Click **OK** to save the change and return to your model.
3. Select **Tools > Rebuild Objects > Rebuild Triggers** to open the Rebuild Triggers dialog (see *Rebuilding Triggers* on page 125).
4. Click the **Selection** tab and select the tables containing the column to which you have assigned the sequence.
5. Click **OK** to rebuild the triggers and enable the sequence on the column.

Sequence Properties

To view or edit a sequence's properties, double-click its Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

Property	Description
Name/Code/ Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
Owner	Specifies the user who is the owner of the object. This is usually its creator. Use the tools to the right of the list to create, browse for, or view the properties of the currently selected user.
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

The following tabs are also available:

- **Physical Options / Physical Options (Common)** - lists the physical options (see *Physical Options (PDM)* on page 88) that control the incrementation of the sequence. For information about these options, see your DBMS documentation.

Abstract Data Types (PDM)

An *abstract data type (ADT)* is a user-defined data type which can encapsulate a data set and functions that can be performed on the data. Abstract data types are not supported by all DBMSs.

For example you could create an abstract data type for the Gregorian calendar to read and write roman numerals and convert dates between the Julian and Gregorian calendars.

If your model contains abstract data types of type `JAVA`, you can link them to Java classes in an OOM to model and review the Java class properties (see *Linking an Abstract Data Type to a Java Class* on page 174).

Creating an Abstract Data Type

You can create an abstract data type from the Browser or **Model** menu.

1. Select **Model > Abstract Data Types** to open the List of Abstract Data Types, and click the **Add a Row** tool. Then click the **Properties** tool to open the property sheet of the new type.

   Alternatively, you can create an abstract data type by right-clicking the model (or a package), and selecting **New > Abstract Data Type**.
2. Select the type for the ADT in the **Type** list on the **General** tab. Depending on your DBMS, you can choose from:
   - Array - Fixed length collection of elements. For example, `VARRAY` (Oracle 8 and higher).
   - List - Open collection of objects. For example, `TABLE` (Oracle 8 and higher).
   - Java - Java class. For example, `JAVA` (Sybase SQL Anywhere and SAP® Sybase® Adaptive Server® Enterprise).
   - Object - Contains lists of attributes and procedures. For example, `OBJECT` or `SQLJ OBJECT` (Oracle 8 and higher).
   - Structured - Contains a list of attributes. For example, `NAMED ROW TYPE` (Informix 9.x and higher).
3. [for object and structured types] Click the **Attributes** tab and create any appropriate attributes.
4. [for object types] Click the **Procedures** tab and create any appropriate procedures.
5. Click **OK** to return to your model.

   For general information about creating objects, see *Core Features Guide > Modeling with PowerDesigner > Objects*.

Abstract Data Type Properties

To view or edit an abstract data type's properties, double-click its Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

Property	Description
Name/Code/ Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
Type	Specifies the kind of the abstract data type (see *Creating an Abstract Data Type* on page 172), which will change the other properties that are available.
Owner	Specifies the user who is the owner of the object. This is usually its creator. Use the tools to the right of the list to create, browse for, or view the properties of the currently selected user.
Authorization	[objects] Specifies the Invoker Right attribute used for DDL generation.
Supertype	[objects] Specifies the parent type from which the type is derived, and from which it can inherit the procedures.
Final/Ab-stract	[objects] Mutually exclusive. If **Final**, the abstract data type cannot be used as supertype by another abstract data type. If **Abstract**, the abstract data type cannot be instantiated.
Data type/ Length/Preci-sion	[tables, varrays] Specify the data type of the abstract data type.
Size	[arrays] Specifies the size of the abstract data type array.
Java class/ Java data	[SQLJ objects] Specify the name of an external Java class to which the SQLJ object points (see *Linking an Abstract Data Type to a Java Class* on page 174) and the mapping interface (CustomDatum, OraData or SQLData).
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple key-words, separate them with commas.

The following tabs are also available:

- **Attributes** - [object and structured types] Use the **Add a Row** tool to create appropriate attributes, specifying a **Name**, **Code**, **Data Type**, and, if appropriate, select the **Mandatory (M)** check box.

- **Procedures** - [object types] Use the **Add a Row** tool to create appropriate procedures, specifying a **Name** and **Code**, and, if appropriate, selecting the **Final (F)**, **Static (S)** and/or **Abstract (A)** check boxes.

  Note: An object abstract data type with a supertype can inherit non-final procedures. Use the **Inherit Procedure** tool to select a non-final procedure from a parent abstract data type.

Linking an Abstract Data Type to a Java Class

You can link an abstract data type in a PDM to a Java class in an OOM open in the Workspace to access the properties of the Java class within the PDM.

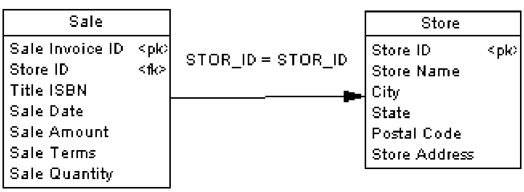
Note: If you reverse engineer Java classes from a database into an OOM (see *Object-Oriented Modeling*) before reverse-engineering the tables and other database objects into a PDM, then the Java classes that are reverse engineered into the PDM are created as abstract data types of type JAVA and linked to the appropriate classes in the OOM (if it remains open in the Workspace).

1. Create an abstract data type and select `Java` from the **Type** list.
2. Click the **Select** tool to the right of the **Class** field to open a selection dialog listing all the Java classes that are available for linking.
3. Select a Java class and click **OK** to link it to the abstract data type. The class name is displayed in the abstract data type property sheet **Class** field. Click the **Properties** tool to the right of this field to open the Java class property sheet.

References (PDM)

A *reference* is a link between a parent table and a child table, which defines a referential integrity constraint between column pairs for a primary or alternate key and a foreign key, or between user-specified columns. Each column pair is linked by a *join*, and each reference can contain one or more joins. Each value in the child table column is equal to the value in the parent table column.

In the following example, the `STORE` parent table is linked to the `SALE` child table by a reference containing 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.

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

- Use the **Reference** tool in the Toolbox.
- Select **Model > References** to access the List of References, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New > Reference**.

Note: You can control whether the creation of a reference automatically creates a join between a primary key in the parent table to a foreign key in the parent table (default) or whether the join columns are left undefined with the **Default link on creation** model option (see *Automatic Reuse and Migration of Columns* on page 178).

For general information about creating objects, see *Core Features Guide > Modeling with PowerDesigner > Objects*.

Reference Properties

To view or edit a reference's properties, double-click its diagram symbol or Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

Property	Description
Name/Code/ Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
Parent table/ Child table	Specify the parent table (which contains the primary or alternate key or a user-selected column) and the child table (which contains the foreign key or a user-selected column) linked by the reference. Use the tools to the right of the Parent table field to create, browse for, or view the properties of the currently selected table.

Property	Description
Parent role/ Child role	Specify the roles of the parent and child tables in the reference (for example `Contains` and `Is contained by`.
Generate	Specifies to generate the reference in the database.
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

Joins Tab

The **Joins** tab lists the joins defined between parent and child table columns. Joins can link primary or alternate and foreign keys, or any user-specified columns.

Note: You can control the default joins created using the **Default link on creation** and **Auto-migrate columns** model options (see *Automatic Reuse and Migration of Columns* on page 178).

On this tab, you can either:

- Select a key from the parent table in the **Parent key** field on which to base the join to autopopulate the list with its associated parent and child columns. If necessary, you can modify the specified child columns.
- Specify `<None>` in the **Parent key** field and specify your own column pairs on which to base the join using the following tools:

Tool	Description
	Reuse Columns - Create a join by matching parent and child columns that share the same code and data type.
	Migrate Columns - First specify columns in the **Parent Table Column** column and then click this tool to migrate them to foreign key columns in the child table. If the columns do not exist in the child table, they are created.
	Cancel Migration - Remove any columns migrated to the child table.
 	Insert/Add a Row - Inserts a row before the selected row in the list or at the end of the list to specify another column to join on.

Note: Select the **Auto arrange join order** check box to sort the list by the key column order or deselect it to re-arrange the columns using the arrow buttons. If this option is not available, to enable it, add the `EnableChangeJoinOrder` item to the Reference category in the DBMS definition file and set the value to YES (see *Customizing and Extending PowerDesigner > DBMS Definition Files*).

Integrity Tab

Referential integrity governs data consistency between primary or alternate keys and foreign keys by dictating what happens when you update or delete a value or delete a row in the parent table. The Integrity tab contains the following properties:

Property	Description
Constraint name	Specifies the name of the referential integrity constraint. Maximum length is 254 characters. If you edit this name, the **User-defined** button will be depressed. To return to the default name, click to release this button.
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.
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. • 0..1 - A parent can have zero or one children. • 1..* - A parent can have one or more children. • 1..1 – A parent must have exactly one child Alternately, you can enter your own integer values in one of the following formats (using `*` or `n` to represent no limit): • `x..y` - A parent can have between x and y children. For example: `2..n` – There must be at least 2 children. • `x` - A parent can have exactly x children. For example: `10` - There must be exactly 10 children. • `x..y, a..b` - A parent can have between x and y or between a and b children. For example: `1..2, 4..n` – There must be one, two, four or more children.

Property	Description
Update/Delete constraint	Specifies how updating a key value in the parent table will affect the foreign key value in the child table. Depending on the implementation and DBMS, you can choose between: • None - No effect on the child table. • Restrict - Values in the parent table cannot be updated or deleted if one or more matching child values exists. • Cascade - Updates or deletions of parent table values are cascaded to matching values in the child table. • Set null - Updates or deletions of parent table values set matching values in the child table to NULL. • Set default - Updates or deletions of parent table values set matching values in the child table to the default value.
Mandatory parent	Specifies that 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	[SQL Anywhere® only] Verifies referential integrity only on commit, instead of after row insertion. You can use this feature to control circular dependencies.
Cluster	Specifies that the reference constraint is a clustered constraint (for those DBMSs that support clustered indexes).

Automatic Reuse and Migration of Columns

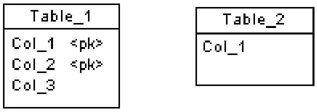
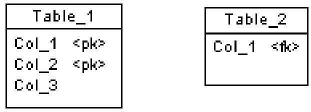
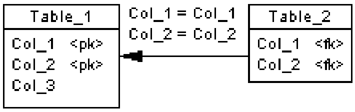
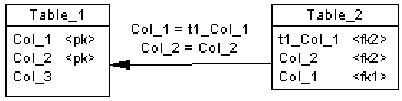
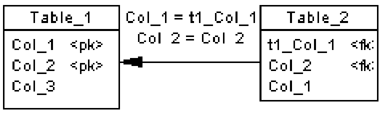
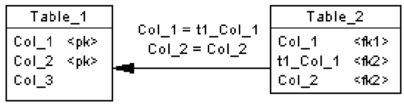
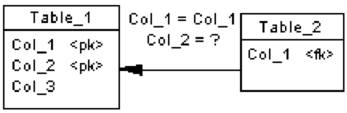
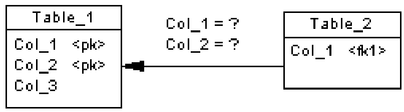
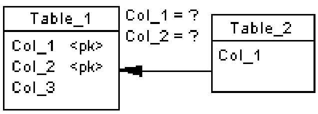
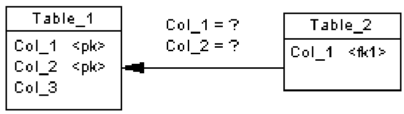
When you create a reference, PowerDesigner can automatically reuse an appropriate existing column in the child table as the foreign key column and migrate the primary key column in the parent table to create a foreign key column in the child table.

1. Select **Tools > Model Options** to open the Model Options dialog box and select the **Reference** sub-category in the left-hand **Category** pane.
2. Select the following options as appropriate:

Option	Function
Auto-reuse columns	Enables the reuse of columns in a child table as foreign key columns when creating references if the following conditions are satisfied: • The child column has the same code as the migrating primary key column. • The child column is not already a foreign key column. 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. • Data types are compatible.
Auto-migrate columns	Enables the automatic migration of primary key columns from the parent table as foreign key columns to the child table when creating references. Select the following column property checkboxes as appropriate to specify parent column properties to migrate: • **Domains** (see *Domains (CDM/LDM/PDM)* on page 162) • **Check** (see *Setting Data Profiling Constraints* on page 94). • **Rules** (see *Business Rules (CDM/LDM/PDM)* on page 184). • **Last position** - Adds migrated columns at the end of the table column list. If this 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:** During intermodel generation, whether or not this option is selected, any selected column property is migrated from the PK to the FK.
Default link on creation	Specifies whether reference joins are automatically created: • **Primary key** – Automatically create joins between the parent table primary key and a child table foreign key. If the **Auto-migrate columns** option is not selected then you must manually specify foreign key columns on the reference **Joins** tab. • **User-defined** – Does not create joins. You must manually select columns on the reference **Joins** tab.

3. Click **OK** to close the dialog and return to your model.

The following table shows the results of migrating and reusing primary key columns to a child table that contains a matching child table column, and where that child table column is already a foreign key column for another table:

Options Selected	Matching Child Table Column Exists	Matching Child Table Column Is Already a FK Column
[Original tables before migration]	The child table contains a matching column for one of the primary key columns: 	The child table contains a matching column that is already a foreign key column for another table: 
[default] Auto-reuse and Auto-migrate	Col_1 is reused and Col_2 is created: 	T1_Col_1 and Col_2 are created: 
Auto-migrate only	T1_Col_1 and Col_2 are created: 	T1_Col_1 and Col_2 are created: 
Auto-reuse only	Col_1 is reused but Col_2 is not created: 	No columns are reused or created: 
Neither	No column is reused or created 	No columns are reused or created: 

Note:

- 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** model option is selected (see *Controlling Non-Divergence from a Domain* on page 167).
- If you have selected the **Auto-migrate columns** model option and you modify a reference attach point then you will migrate primary keys in the parent table to foreign keys in the child table, delete unused foreign key columns, and modify the reference join. If you delete the parent primary key column then you will delete the corresponding foreign key and reference join.

For more information about other reference model options, see *Reference Model Options* on page 15.

Rebuilding References

You can rebuild references at any time to create default references between PK columns in one table and columns with identical code and data type in another table. Rebuilding is not possible between two tables with PK columns. Rebuilding references can be useful following the reverse engineering of a database in which not all the references could be reverse engineered.

1. Select **Tools > Rebuild Objects > Rebuild References**, and specify 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.
2. [optional] Click the **Selection** tab to specify which tables you want to rebuild references for. By default, all tables are selected.

   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** tool next to the list, and then select a sub-package from the dropdown list.
3. 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.

Displaying Information on Reference Symbols

You can display the cardinality, referential integrity, join, table roles and other properties on the source and destination ends and in the center of a reference. To set display preferences for references, select **Tools > Display Preferences**, and select the Reference sub-category in the left-hand Category pane.

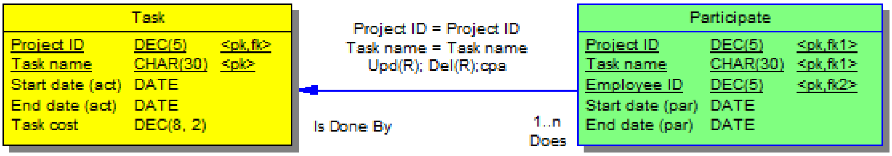
The notation for referential integrity and constraints on reference symbols is as follows:

Referential integrity	Constraint Types
• `upd(`*`constraint`*`)` - Update • `del(`*`constraint`*`)` - Delete • `cpa` - Change Parent Allowed	• `( )` - None • `(R)` - Restrict • `(C)` - Cascade • `(N)` - Set null • `(D)` - Set default

The Cardinality attribute displays the minimum and maximum number of instances in a child table that can appear for each corresponding instance in the parent table as follows:

```
min..max
```

In this example, the source of the reference symbol shows a cardinality of `1..n` (one or more children is acceptable), and the child table role (`Does`) and the destination of the reference shows the parent table role (`Is Done By`). The center of the symbol shows the two primary keys that form the join, as well as the referential integrity (updates and deletions are restricted and change parent is allowed:



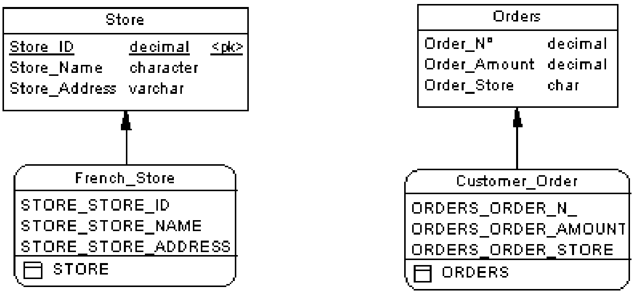
For information about changing the notation of references, see *Setting PDM Model Options* on page 13. For detailed information about working with display preferences, see *Core Features Guide > Modeling with PowerDesigner > Diagrams, Matrices, and Symbols > Display Preferences*.

View References (PDM)

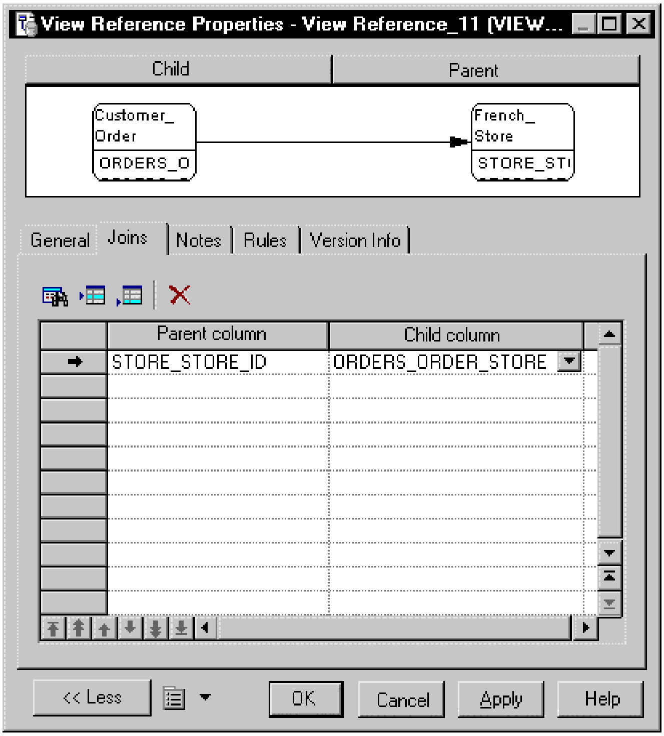
A *view reference* is a link between a parent table or view and a child table or view, which defines the joins between the parent and child columns. View references are not generated to the database

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.

In this example, `French_Store` is a view of the `Store` table with a view reference defining a join between Store_ID in the table and `STORE_STORE_ID` in the view. `Customer_Orders` is a view of the `Orders` table with a view reference defining a join between Order_No in the table and `ORDER_ORDER_N` in the view:



You can create a view reference between the two views to define a join between `Customer_Order.ORDER_ORDER_STORE` and `French_Store.STORE_STORE_ID`:



If you were then to create a view from the `French_Store` and `Customer_Order` views, the `SELECT` order of the new view will take into account the join defined between the views to retrieve only those orders sent to French stores.

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 Toolbox.
- Select **Model > View References** to access the List of View References, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New > View Reference**.

For general information about creating objects, see *Core Features Guide > Modeling with PowerDesigner > Objects*.

View Reference Properties

To view or edit a view reference's properties, double-click its diagram symbol or Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

Property	Description
Name/Code/ Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
Parent/ Child	Specify the parent and child tables or views linked by the reference. Use the tools to the right of the list to create, browse for, or view the properties of the currently selected table or view.
Parent role	Specify the roles of the parent and child tables or views in the reference.
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

Joins Tab

The **Joins** tab lists the joins defined between parent and child table or views columns. You can specify column pairs on which to base the join using the following tools:

Tool	Description
	Reuse Columns - Create a join by matching parent and child columns that share the same code and data type.
	Insert/Add a Row - Inserts a row before the selected row in the list or at the end of the list to specify another column to join on.

Business Rules (CDM/LDM/PDM)

A business rule can represent a government-imposed law, a customer requirement, or an internal guideline. They may start as simple observations, such as "customers call toll-free numbers to place orders", and develop into more detailed expressions during the design process such as what information a customer supplies when placing an order or how much a customer can spend based on a credit limit.

Business rules complement your diagrams with information that is not easily represented graphically, and can help guide the creation of a model. For example, the rule "an employee belongs to only one division" can help you define the link between an employee and a division. Business rules are generated as part of intermodel generation and can be further specified in the generated model.

There are three ways to use business rules in a data model:

- Apply a business rule to a model object as part of its definition (see *Attaching a Business Rule to a Model Object* on page 187).
- [PDM only] Create a server expression that can be generated to a database (see *Creating and Attaching a Constraint Rule* on page 187).
- [PDM only] Insert a business rule expression in a trigger or stored procedure using the `.CLIENTEXPRESSION` or `.SERVEREXPRESSION` macros (see *Customizing and Extending PowerDesigner > DBMS Definition Files > PDM Variables and Macros*).

When creating business rules, you may find it helpful to ask the following kinds of question:

- Do any mandatory regulations impact my system?
- How can I clearly and concisely define the specifications for my project?
- Do any constraints limit my options?
- Is this rule a definition, fact, formula, or avalidation rule?

Creating a Business Rule

You can create a business rule from the Browser or **Model** menu, or from the **Rules** tab of an object property sheet.

- Select **Model > Business Rules** to access the List of Business Rules, and click the **Add a Row** tool.
- Right-click the model (or a 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 *Core Features Guide > Modeling with PowerDesigner > Objects*.

Business Rule Properties

To view or edit a business rule's properties, double-click its Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

Property	Description
Name/Code/ Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
Type	Specifies the nature of the business rule. You can choose between: • Constraint – a check constraint on a value. For example, "The start date should be inferior to the end date of a project." In a PDM, constraint rules attached to tables or columns are generated. If the DBMS supports multiple constraints, constraint rules are generated as separate constraint statements with the name of the rule. • 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". In a PDM, validation rules attached to tables or columns are generated as part of the primary constraint for the table or column.
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

The following tabs are also available:

- **Expression** - As you develop your model and analyze your business problem, you can complete the rule by adding a technical expression. The syntax of expressions depends on the target database, and each 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

Attaching a Business Rule to a Model Object

You can attach a business rule to a model object from the object's property sheet.

1. Open the object's property sheet and click the **Rules** tab.
2. Click the **Add Objects** tool to open a list of available business rules.
3. Select one or more business rules and click **OK**.

   The business rules are attached to the object and appear on the list of business rules for the object.

   Note: When you attach a business rule to an object, it is marked as used in the model. You can review which rules are used by opening the List of Business Rules and consulting the **U** (Used) column.
4. Click **OK** to return to the model diagram.

Creating and Attaching a Constraint Rule

Validate and constraint business rules have their expressions generated as constaints for DBMSs where this is supported. Validate rules can be reused by multiple objects, but constraint rules can only be used once, and will be generated as a separate constraint for DBMSs that support multiple constraints.

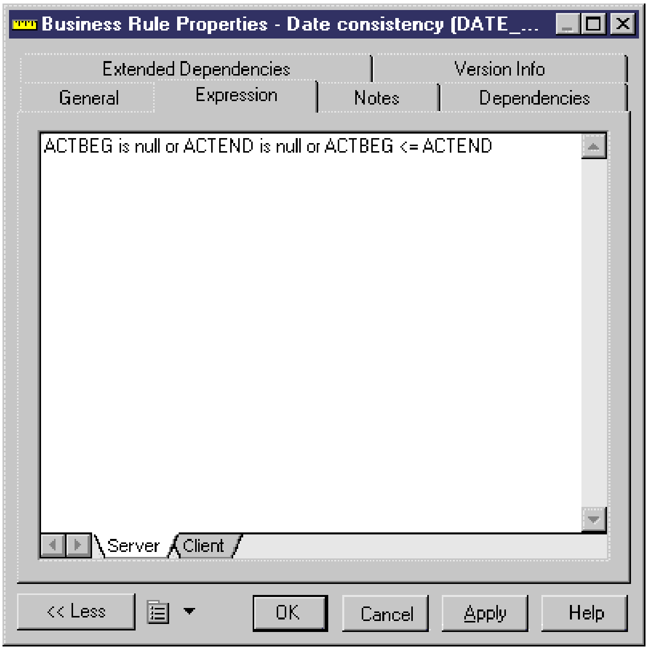
Support for the generation of constraint rules to your database is controlled by the following items in the **General** category of your DBMS definition file:

- **EnableCheck** - Permits the generation of constraints to the database.
- **EnableMultiCheck** - Permits the generation of check parameters (see *Setting Data Profiling Constraints* on page 94) and validation business rules as a single constraint, followed by the generation of each constraint business rule as a separate constraint in the order in which they are attached to the table. If this option is not enabled, then check parameters and all constraint and validation rules are concatenated into a single constraint expression.
- **UniqueConstName** - Requires that all validate and constraint rules have unique codes.

You can preview the constraints that will be generated on the **Preview** tab of the table property sheet.

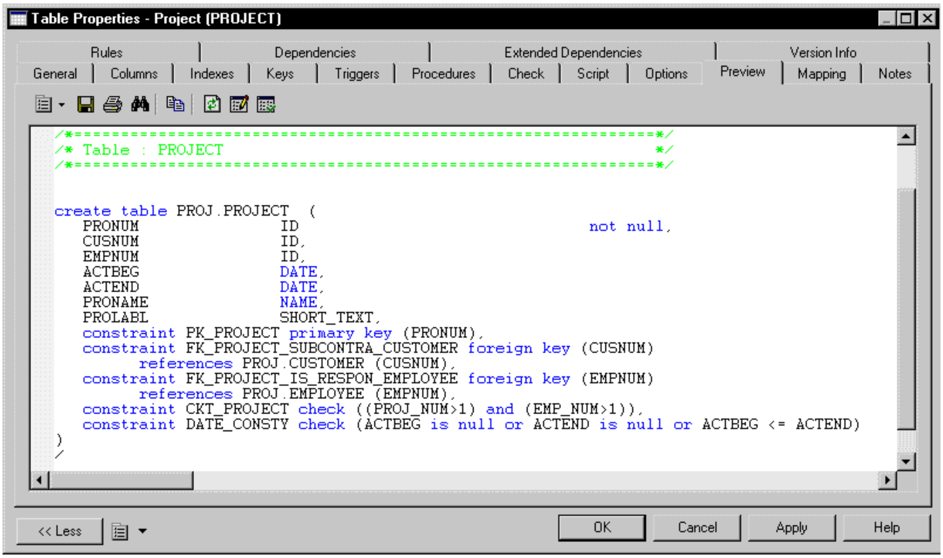
When reverse engineering, the constraint order is respected, with the first constraint retrieved to the **Check** tab of the table property sheet, and each subsequent constraint retrieved as a constraint business rule attached to the table

1. Create a business rule, enter a name and code, select **Constraint** in the **Type** list, and then click the **Expression** tab
2. Enter an expression on the **Server** sub-tab:



3. Click **OK** to save your changes and return to the model.
4. Open a table or column property sheet and click the **Rules** tab.
5. Click the **Add Objects** tool to open a list of available business rules, select your constraint business rule from the selection list and click **OK** to attach it.
6. [optional] Click **Apply** to confirm the attachment of the rule and then click the table property sheet **Preview** tab to verify that the constraint has been created in the script.

   In the following example, multiple constraints are defined on the Project table:
   - Check parameter (in the **Check** tab of the table) - Verifies that the customer number is different from the employee number.
   - Validation business rules - `PROJ_NUM` to check that the column project number is not null and `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.



Lifecycles (PDM)

A lifecycle allows you to model the movement of data from expensive, rapid storage, through various forms of cheaper slower storage as the data ages and access requirements diminish. The period during which data remain in each kind of storage are modeled as phases, which are associated with tablespaces.

Note: Data lifecycle modeling is supported for Sybase IQ v15.0 and higher.

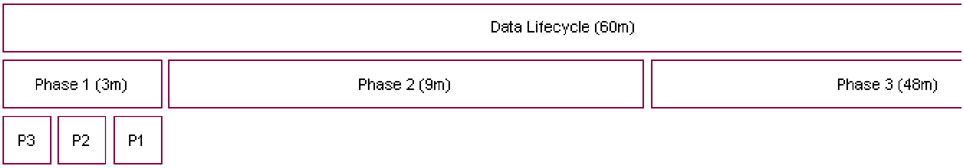
You can attach any number of tables to a lifecycle, and create multiple lifecycles to provide different speeds and/or methods for data aging. Each table can only be associated with one lifecycle. A lifecycle can be:

- **Age-based** - Data moves through the lifecycle in named partitions, remaining in each phase only for the specified retention period. The partitions move through the lifecycle in a predictable fashion and will become candidates for purging at the end of the lifecycle's total retention period.
- **Access-based** - Tables (and any associated indexes) move through the lifecycle based on the permitted idle time for each phase, which specifies how long a table can remain in the phase without being accessed. Tables must remain in the lifecycle for (as a minimum) the total retention period, and their movement to the end of the lifecycle can be delayed indefinitely if the data they contain continue to be accessed.

The following diagram illustrates an age-based lifecycle covering a period of five years, which is divided into three phases:

- Phase 1 (3 months) - high performance (tier-1) storage for new data that is frequently accessed.
- Phase 2 (9 months) - nearline (tier-2) storage for data from the last year.
- Phase 3 (48 months) - historical (tier-3) storage for data that is infrequently accessed but which must be retained.

The data is packaged in partitions (P1, P2, and P3), which each contain one month of data:



PowerDesigner can generate all the necessary scripts to automate all this data movement. In the example above, scripts will be generated for every month of the lifecycle. At the point illustrated in the picture, the scripts will:

- Move partition P1 from the tablespace associated with Phase 1 to the tablespace associated with Phase 2.
- Create a new partition, P4, to begin collecting new table rows in the tablespace associated with Phase 1.

As the data ages, scripts will additionally treat the movement of data aged more than one year from the tablespace associated with Phase 2 to the tablespace associated with Phase 3.

Once a lifecycle is put in place, you can generate scripts to perform data movement indefinitely. Additional scripts are generated to regularly purge data that arrive at the end of their lifecycle.

Modeling a Lifecycle

To correctly model a lifecycle you must define the lifecycle and its phases, and then associate your tables to it.

1. Create a lifecycle in any of the following ways:
   - Select **Model > Lifecycles** (or **Database > Information Lifecycle Management > List of Lifecycles**) to access the List of Lifecycles, and click the **Add a Row** tool.
   - Right-click the model in the Browser, and select **New > Lifecycle**. Note that lifecycles can only be created at the model level and not within packages.
2. Click the **Properties** tool to open the lifecycle property sheet and specify a name for the lifecycle.
3. Click the **Definition** tab, and select the policy type:

- **Age-based** - Data moves through the lifecycle in named partitions, based on the time since the data was created. Specify a **Start date** and the **Total retention** period (the length of time covered by the lifecycle).
- **Access-based** - Tables move through the lifecycle based on the time since the table was last accessed. Specify a **Total retention** period, which is treated as the minimum total period of time that a table's data must remain in the lifecycle.

4. Click the **Create Phase** tool to create as many phases as you need. Lifecycles often contain three phases to manage the movement of data from high performance, through nearline, to historical storage.

   Note: Your phase will display a yellow warning overlay until it is completely defined.

5. Click on each phase in turn to open its property sheet (see *Phases (PDM)* on page 195). Specify a name, retention period (or, for access-based lifecycles, idle period) and tablespace to represent the physical storage in which the data is stored during this phase.

   For age-based lifecycles, you can assign data from an external database to the first phase of your lifecycle and have that data loaded to your warehouse database for the second phase (see *Archiving Data From External Databases* on page 198).
6. Open the property sheet for each of your tablespaces (see *Tablespace and Storage Properties* on page 200) and enter any appropriate properties, including a value for the cost per GB to be used when calculating cost savings.

   When you have completed the definition of your phases and tablespaces, return to the lifecycle property sheet and verify that the warning overlays on the phase buttons are no longer present.
7. [age-based lifecycles] Enter a partition range to specify the length of time covered by each table partition governed by the lifecycle. For example, a partition range of one month means that each partition will contain one month's data.
8. In the **Managed tables** groupbox, select the tables you want to associate with the lifecycle. For each table, specify the start date on which you want it to become subject to the lifecycle, and enter an estimate for the initial number of rows and a percentage growth rate to permit the calculation of cost savings.
9. [age-based lifecycles] You must, for each table, specify a column with a date datatype as the partition key used to determine to which partition a row must be assigned. The partition key can alternately be assigned on the **Sybase IQ** tab of the table property sheets.
10. [optional] Select the **Cost savings analysis** checkbox and then click the **Refresh Cost Savings Analysis** tool to display a summary of the cost savings to be obtained by managing your data with the lifecycle.

    You can also view the detail of the cost savings by year for a single table on the **Lifecyle** tab of the table property sheet (see *Table Properties* on page 76).

    Note: If you intend to model multiple lifecycles, and/or want to confirm that all of your tables are associated with a lifecycle, you may find it useful to visualize these associations in the form of a dependency matrix. To view the Lifecycle/Table Matrix, select **Database > Information Lifecycle Management > View Lifecycle/Table Matrix**.

Generating Data Archiving Scripts to Implement your Lifecycle

Once you have modeled your lifecycles, you can instruct PowerDesigner to generate scripts to automate the creation, movement, and purging of data through your lifecycle phases.

Before you generate your data movement scripts, ensure that you have completed all the steps listed in *Modeling a Lifecycle* on page 190.

1. Select **Database > Information Lifecycle Management > Generate Data Archiving Scripts** to open the Generate dialog.
2. Specify a directory in which to generate the scripts, and, optionally, select to check your model before generation.
3. Click the **Selection** tab, and select the tables for which you want to generate data archiving scripts.
4. [for age-based lifecycles] Click the **Options** tab, specify the start and end date for the period for which you want to generate scripts. You can generate scripts for all or part of the period covered by your lifecycle, and also to cleanup data created before the start date of your lifecycle.

   Note: For age-based lifecycles used to archive data from an external database, if you specify a generation start date before the start date of a table associated with the lifecycle, additional scripts will be generated to advance immediately older data created between the generation start date and the table lifecycle start date to the appropriate stages of the lifecycle.

5. [for age-based lifecycles] On the **Options** tab, specify the method for creating partitions. You can choose between creating partitions:
   - Individually, when the previous partition ends
   - All at the beginning (default)
6. Click **OK** to begin the generation.

   The scripts are generated in the specified directory and listed in the **Results** pane.

   The following scripts are generated for age-based lifecyles, and should be run on the date specified in the order specified by their numerical prefix. You can run the scripts manually or use Sybase Control Center to automate this process:
   - `IQ.CreateRemoteServerAndLogin.`*`date`*`.sql` - if you are achiving data stored in an external database.
   - One or more folders named *`yyyymmdd`* for each date on which scripts must be run containing one or more of the following scripts:
     - `01.IQ.CreateAndMovePartition.`*`date`*`.sql` - one script per date on which a data movement action is required between the start and end dates you specify. For example, if you specify a start date of 01/01/2009 and an end date of 12/31/2009, a partition range of one month, and to create the partitions

individually, then twelve scripts will be generated. The scripts should be run on the dates included in their filenames.
    - `02.IQ.PurgePartition.`*`date`*`.sql` - one script per date on which a data purge action is required for partitions arriving at the end of the lifecycle.
    - `03.DB.DeleteSourceData.`*`date`*`.sql` - if there is data to be purged in an external database.
- `OldData` - if you have specified a generation start date earlier than your table start dates, this folder will be created and will contain dated subfolders containing scripts to create, move, and purge older data.

The following scripts are generated for access-based lifecyles:

- `CreateProcedures.sql` - creates procedures to test the idle time during which tables have not been accessed and to move and/or delete them on demand. This script should be run immediately to prepare the database for data movements called for by an access-based lifecycle
- `MoveData.sql` - calls the procedures to test for and implement data movement based upon the specified idle times using the current date on the IQ server. This script should be scheduled to run regularly.
- `DeleteData.sql` - calls the procedure to test for and implement data purging based upon the specified idle times and the specified minimum retention period using the current date on the IQ server. You can schedule this script to run regularly or run it by hand as needed.

Lifecycle Properties

To view or edit a lifecycle's properties, double-click its Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

Property	Description
Name/Code/ Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

Definition Tab

This tab contains all the properties necessary to define your lifecycle. The **Policy** group box contains the following properties:

Property	Description
Policy type	Specifies the criteria used to advance data through the lifecycle. You can choose between: • Age-based - where data are moved from phase to phase in named partitions depending on the time since their creation. • Access-base - where tables are moved from phase to phase depending on the time since the data in the tables were last accessed.
Start date	[age-based lifecycles only] Specifies the date from which you want the lifecycle to manage data movement.
Total retention	Specifies the total length of time during which data is controlled by the lifecycle. For example, if you specify a total retention of 5 years, the lifecycle will manage the movement of each record from the moment of its creation until it has existed for 5 years. For age-based lifecycles, the total retention time must be equal to the sum of all the retention times of all the phases contained within the lifecycle. For access-based lifecycles, the total retention time is used as the minimum total time that the data must remain in the lifecycle.
Phases	Lists the phases (see *Phases (PDM)* on page 195) associated with the lifecycle. You can create phases using the **Create a New Phase** tool. Click on a phase to open its property sheet. **Note:** Your phase will display a yellow warning overlay until it is completely defined.
Partition range	[age-based lifecycles only] Specifies the period of data to be contained in partitions for tables governed by the lifecycle. For example, a partition range of one month means that each partition will contain one month's data.

The **Managed Tables** group box lists the tables whose data are managed by the lifecycle. Use the **Add Objects** and **Create an Object** tools to populate the list. If the lifecycle is used to archive data in an external database, the choice of tables to attach is limited to the tables in the external database, and the selected tables are generated to the warehouse PDM if they were not already present.

The following properties must be completed for each table in order to correctly generate data archiving scripts:

- **Name** and **Code** - to identify the table.

- **Start Date** - [optional] Specifies the start date from which to generate the first partition.
- **Initial Rows** and **Growth Rate** - Specifies the number of rows that the table will start with, and the percentage growth per year
- **Partition Key** - [age-based lifecycles] Specifies the column to use to determine to which partition a row is assigned.

Click the **Generate Data Archiving Script** button to generate scripts to implement your lifecycle (see *Generating Data Archiving Scripts to Implement your Lifecycle* on page 192).

Select the **Cost Saving Analysis** checkbox and then click the **Refresh Cost Savings Analysis** tool to display a list of the cost savings to be obtained by managing data with the lifecycle. Use the tools above the list to export the cost savings data to Excel or to print it.

Phases (PDM)

A phase defines the period of time that data governed by a lifecycle will be retained by a particular tablespace.

Creating a Phase

You create phases on the **Definition** tab of a lifecycle using the **Create Phase** tool.

Phase Properties

To view or edit a phase's properties, double-click its Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator. The **General** tab contains the following properties:

Property	Description
Name/ Code/Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
Retention/ Time unit	[age-based] Specifies the length of time that data will be retained in this phase.
Idle period/ Time unit	[access-based] Specifies the minimum length of time that the table must remain unaccessed before it is moved to the next phase.

Property	Description
Source	Specifies where the data to populate the phase is located. The default is the current (warehouse) database. For the first phase only in an age-based lifecycle, you can specify instead an external database (see *Archiving Data From External Databases* on page 196), in which case you must also specify a data source to link to the PDM that models the external database.
Tablespace	[Current database only] Specifies the tablespace with which the phase is associated. Select a tablespace from the list or click the tools to the right of this field to create a new tablespace or open the property sheet of the currently selected one.
Data Source	[External database only] Specifies the data source used to connect to the external database. Click the **Create** tool to the right of this field to launch the Data Source Wizard (see *Linking an External Database via the Data Source Wizard* on page 198) to create a data source and apply the appropriate tables to the lifecycle.
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

Archiving Data From External Databases

When developing an age-based lifecycle policy, you can assign data from an external database modeled in another PDM to the first phase. At the end of the first phase the data will be loaded from the external database to your warehouse.

In order to model external database data archiving, you must:

1. Create a PDM to model the external database.
2. Create a PDM to model the data warehouse.
3. Link the second PDM to the first through a data source.
4. Specify access parameters for the warehouse database and the external database on the **Database Connection** and **Data Movement (Lifecycle)** tabs of the data source.
5. Create mappings between the external tables that contain the data to be archived and the warehouse tables to which this data will be loaded.
6. Create a lifecycle in the warehouse PDM and create the first phase.
7. Set the **Source** of the first phase to `External Database` and specify the data source through which you have connected the external database PDM.
8. Select the tables to attach to the lifecycle.

PowerDesigner provides various tools to help you create parts of this archiving environment:

- PDM-PDM model generation - can create the data warehouse PDM, the data source and mappings (see *Linking an External Database by Generation* on page 197)

- The Mapping Editor - can help you create (or modify) the mappings between the external database and warehouse PDM tables (see *Linking an External Database through the Mapping Editor* on page 197)
- The Data Source Wizard - can create the data source and table mapping, set the lifecycle source for the first phase and attach tables to the lifecycle (see *Linking an External Database via the Data Source Wizard* on page 198)

Linking an External Database by Generation

You can use the model generation mechanism to generate tables from your external database to your warehouse PDM and create the required data source and mappings in your warehouse PDM.

1. Create a PDM to model the external database containing the tables to be archived by the lifecycle.
2. Select **Tools > Generate Physical Data Model** to open the PDM Generation Options dialog.
3. On the **General** tab, choose whether you will create a new PDM to represent your warehouse database or add the tables to be generated to an existing warehouse PDM.
4. On the **Detail** tab, ensure that the **Generate mappings** option is selected.

   These mappings are used in the subsequent generation of the lifecycle to route the data to be archived in the warehouse.
5. On the **Selection** tab, select the tables that contain the data you want to archive via the lifecycle.
6. Click OK to begin the generation.

   If you are adding the tables to an existing warehouse PDM, the Merge Models dialog will open, allowing you to review the changes that will be made to it before clicking OK to continue with the generation.

   The selected tables are generated to the warehouse PDM, along with a data source object and the appropriate mappings.

   Note: For detailed information about model generation, see *Chapter 7, Generating Other Models from a Data Model* on page 323. For information about using the Merge Models dialog, see *Core Features Guide > Linking and Synchronizing Models > Generating Models and Model Objects*.

Linking an External Database through the Mapping Editor

You can use the Mapping Editor to manually create (or modify) mappings between the external database and warehouse tables that will be used to archive the data governed by the lifecycle. This method can be useful when you have PDMs to represent your external and warehouse databases and will be using non-standard mappings to load your data.

To open the Mapping Editor from your warehouse PDM, select **Tools > Mapping Editor**. If you have no data sources defined in the model, the Data Source Wizard will open, and you should use it to define a data source pointing to the external database PDM, which will then be opened in the Mapping Editor.

Note: For detailed information about using the Mapping Editor (and the Data Source Wizard) see *Core Features Guide > Linking and Synchronizing Models > Object Mappings*.

Linking an External Database via the Data Source Wizard

The Data Source Wizard guides you through creating an external database data source in your model, and to attach it and the tables to be managed to the first phase of your lifecycle

1. Create an age-based lifecycle policy (see *Creating a Lifecycle* on page 190), add a first phase to it, and open the property sheet for this phase.
2. Set the retention period for the phase and set the **Location** property to **External database**.
3. Click the **Create** tool to the right of the data source field to open the Data Source Creation Wizard.
4. On the first page, select the PDM that represents your external database and then click **Next**.
5. On the second page, select the tables that you want to associate with the lifecycle.
6. Click Finish to associate the selected tables with the lifecycle.

   The wizard creates a data source in the warehouse PDM and associates it with the first phase of the lifecycle. The selected tables are generated to the warehouse PDM if they were not already present, and appropriate mappings are created between the tables in the external database and those in the warehouse PDM.

Tablespaces and Storages (PDM)

Tablespaces and storages are generic objects used to represent physical locations (in named partitions) 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 use concepts that are represented by tablespaces and storages in PowerDesigner:

DBMS	Tablespace represents...	Storage represents...
ADABAS	NA	NA
IBM DB2 UDB Common Server	tablespace `create tablespace`	buffer pool `create bufferpool`

DBMS	Tablespace represents...	Storage represents...
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
Teradata	NA	NA

Note: When tablespace or storage options are not applicable for a DBMS, the corresponding model menu item is not available.

Creating a Tablespace or Storage

You can create a tablespace or storage from the Browser or **Model** menu.

- Select **Model > Tablespaces** (or **Storages**) to access the appropriate list, and click the **Add a Row** tool
- Right-click the model (or a package) in the Browser, and select **New > Tablespace** (or **Storage**)

For general information about creating objects, see *Core Features Guide > Modeling with PowerDesigner > Objects*.

Tablespace and Storage Properties

To view or edit a tablespace or storage's properties, double-click its Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

Property	Description
Name/Code/ Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

Cost tab

The Cost tab is available if data lifecycle modeling (see *Lifecycles (PDM)* on page 189) is supported by your DBMS.

Property	Description
Cost (per GB)	Specifies the cost per GB of the storage represented by the tablespace
Currency	Displays the currency to use for the cost per GB of storage. You can change the currency by selecting **Tools > Model Options** and choosing a currency from the list on the Model Settings page.

Other tabs

The following tabs are also available:

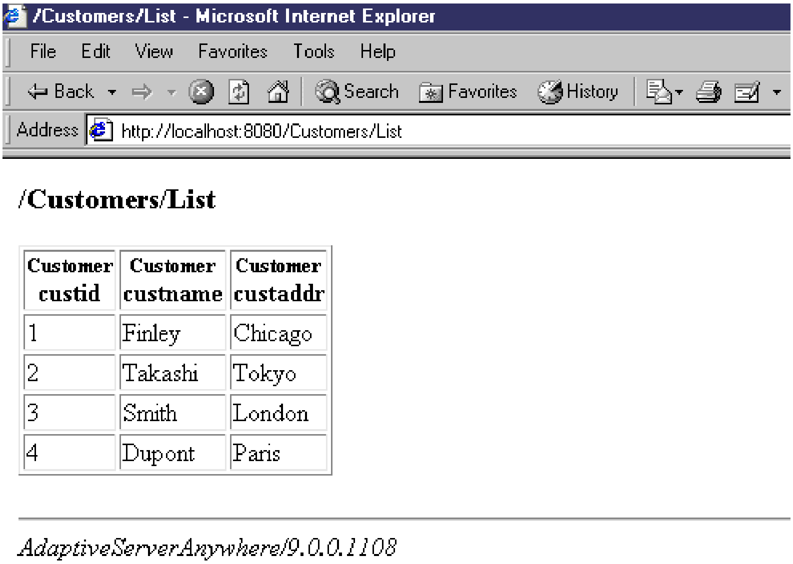
- Physical Options - lists all the physical options that can be applied to the tablespace or storage (see *Physical Options* on page 88).
- Physical Options (Common) - lists the most commonly used physical options that can be applied to the tablespace or storage.

Note: For detailed information about tablespace and storage options for a particular DBMS, see its reference manual.

Web Services (PDM)

Web services are applications stored on web servers that are accessed through standard web protocols (HTTP, SOAP) and data formats (HTML, XML...), whatever the systems and programming languages. PowerDesigner supports modeling for both the SOAP protocol, in which queries are encapsulated into services, and HTTP, where operations are invoked directly.

If you use web services to query databases, you no longer need database drivers. The following example shows the result of an HTTP request for a database web service:



Web services comprise a set of operations, each of which contains a SQL query for retrieving data from a database. Web parameters are the parameters which appear in the SQL statements, and result columns display the results. These objects have no symbols, and appear only in the Browser. Web services can be modeled for the following DBMSs:

- 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 - Document Access Definition Extension (DADX) files specify Web services 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 (see *Generating Web Services for IBM DB2* on page 209 and *XML Modeling > Working with XML and Databases > Generating a DAD File for IBM DB2*.

You can test a Web service of type DISH or SOAP from within your model by right-clicking its Browser entry and selecting **Show WSDL**. You can test a web service operation belonging to a

Web service of another type by right-clicking the operation and selecting **Test Web Service Operation**. Review the generated URL and then click **OK** to display the WSDL file (for SOAP) or results (for RAW) in your Web browser.

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 (see *Business Process Modeling > Service Oriented Architecture (SOA) > Service Providers (BPM)*).

Creating a Web Service

You can create a web service from the Browser or **Model** menu.

- Select **Model > Web Services** to access the List of Web Services, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New > Web Service**.

For general information about creating objects, see *Core Features Guide > Modeling with PowerDesigner > Objects*.

Web Service Properties

To view or edit a web service's properties, double-click its Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

Property	Description
Name/Code/Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field. In URIs, the name of the web service is used to access the web service, and should not start with a slash nor contain two consecutive slashes.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.

Property	Description
Local path	Specifies the name prefixing the web service, which is by default, the name of the web service. When several web services concern the same table, their local path 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). PowerDesigner treats HTTP web operations which share a local path as belonging to the web service with that local path name. If you enter a path, the **User-Defined** tool is depressed. Click the tool to release it and recover the original path.
Service type	[ASA, ASE, and IQ only] Specifies the type of web service. A web service invoked via an HTTP request can have a RAW, HTML or XML type. A web service invoked in a SOAP request can have a SOAP or a DISH type: • *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 Name prefix on the **Sybase** tab (see *Chapter 19, SAP Sybase SQL Anywhere* on page 541) 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 204) 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* file. • *XML* - the result of the SQL statement or procedure is sent in XML. By default, the result is converted into XML RAW format.
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

Security Tab

This tab is available for ASA/SQL Anywhere and IQ only, and 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

Property	Description
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

The following tabs are also available:

- Operations - Lists the Web operations associated with the Web service (see *Web Operations (PDM)* on page 204).
- Sybase - [ASA/SQL Anywhere, ASE, and IQ] Includes Sybase-specific properties (see *Chapter 19, SAP Sybase SQL Anywhere* on page 541)
- Namespaces - [IBM DB2] Lists the namespaces associated with the Web service, including their prefix, URI and a 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 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 in the following ways:

- Open the **Operations** tab in the property sheet of a web service, and click the **Add a Row** tool.
- Right-click a web service in the Browser, and select **New > Web Operation**.

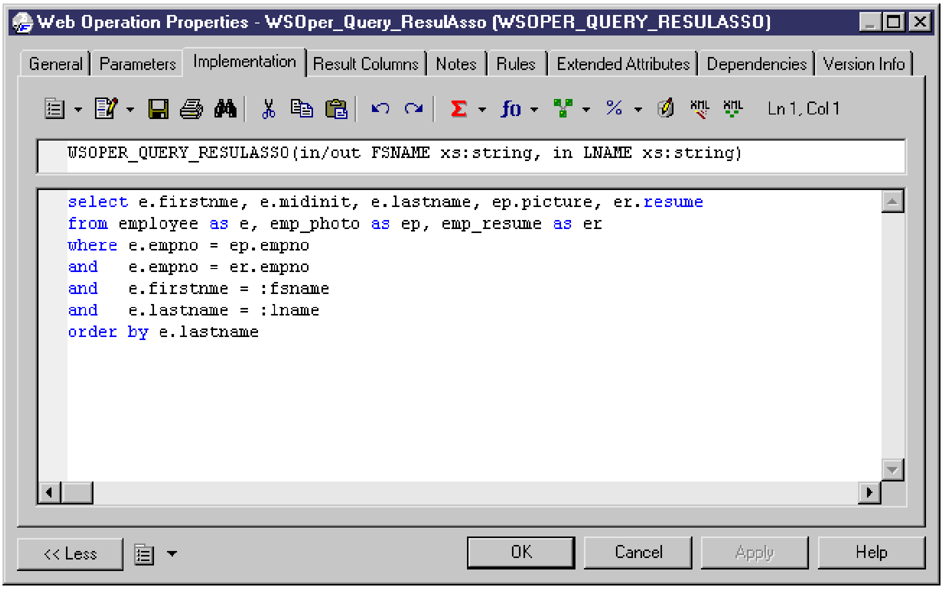
Web Operation Properties

To view or edit a web operation's properties, double-click its Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator. The **General** tab contains the following properties:

Property	Description
Name/Code/ Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field. In URIs, the name of the web operation comes after the name of the web service followed by a slash, and should not start with a slash nor contain two consecutive slashes.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
Web Service	Code of the web service containing the web operation. Click the Properties tool to open the web service property sheet
Owner	[ASE 15 only] Specifies the owner of the operation.
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
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

The following tabs are also available:

- **Implementation** - Contains the SQL statement of the Web operation to select which data you want to retrieve from the database. For DISH web services, SQL statements are defined in the SOAP web services bearing their prefix name. For information about the tools on this tab, see *Writing SQL Code in PowerDesigner* on page 281.



- **Security** - [SQL Anywhere/IQ] Displays the following properties:

Property	Description
Secured connection	Requires an HTTPS connection.
Required authorization	Requires users to provide a name and a password.
Connection User	When authorization is required, you can select the default `<None>` (all users are granted access), or a list of user names. When authorization is not required, you must select a user name.

- **Parameters** - Lists the parameters associated with the Web operation, which are part of the SQL statement defined on the Implementation tab (see *Web Parameters (PDM)* on page 206).
- **Result Columns** - Lists the result columns associated with the Web operation (see *Web Result Columns (PDM)* on page 208).
- **Sybase** - [ASE] Displays Sybase-specific options (see *Chapter 15, SAP Sybase Adaptive Server Enterprise* on page 479).

Web Parameters (PDM)

Web parameters are part of the SQL statement defined in the **Implementation** tab of a web operation property sheet, and are listed on its **Parameters** tab.

Creating a Web Parameter

You can create a Web parameter in the following ways:

- Open the **Parameters** tab in the property sheet of a Web operation, and click the **Add a Row** tool. Alternatively, use the **Add Parameters from SQL Implementation** tool (ASA, ASE, and IQ only) to display the parameters resulting from the reverse engineering of the web service.
- Right-click a web operation in the Browser, and select **New** > **Web Parameter**.

Web Parameter Properties

To view or edit a web parameter's properties, double-click its Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator. The **General** tab contains the following properties:

Property	Description
Name/Code/ Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
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.
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.
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

Web Result Columns (PDM)

Result columns are part of the SQL statement defined in the **Implementation** tab of a web operation property sheet, and are listed on its **Result Columns** tab. They belong to a table in the target database.

Creating a Web Result Column

You can create a Web result column in the following ways:

- Open the **Result Columns** tab in the property sheet of a Web operation, and click the **Add a Row** tool. Alternatively, use the **Add Result Columns from Executing SQL Statement** tool to display the result columns resulting from the execution of the SQL statement in the database.
- Right-click a web operation in the Browser, and select **New** > **Result Column**.

Web Result Column Properties

To view or edit a result column's properties, double-click its Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator. The **General** tab contains the following properties:

Property	Description
Name/Code/ Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Data Type	[IBM DB2] Select an XML schema data type from the list, or click the **Select Object** tool to select a global element in an XML model open in the workspace.
Is element	[IBM DB2] Checked and greyed when a global element is attached to a result column.
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

Generating Web Services for Sybase ASA, ASE, and IQ

You can generate database web services to a script or to a live database connection.

1. Select **Database** > **Generate Database** to open the Database Generation dialog, and specify the standard options, including whether you want to generate to a script or to a live database connection (see *Generating a Database from a PDM* on page 290).

2. [optional] Click the **Options** tab and click **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.

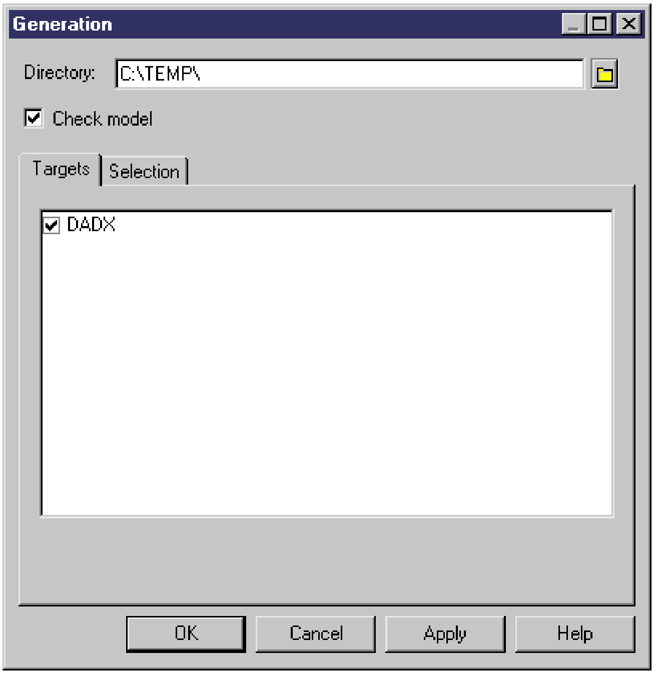
Note: For web services generated to a live database connection, you may have to refresh the Web Services folder before they appear.

Generating Web Services for IBM DB2

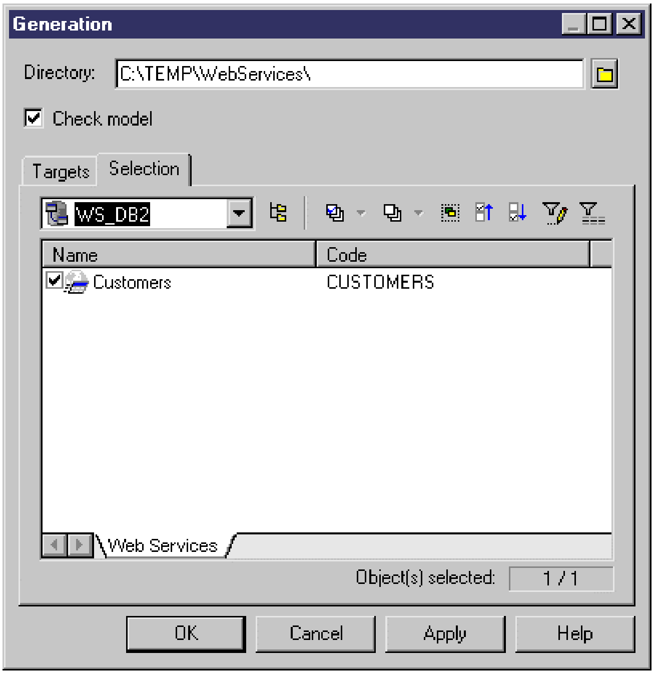
For IBM DB2 Web services, PowerDesigner can generate Document Access Definition Extension (DADX) files.

To enable the DADX generation extensions in your model, select **Model > Extensions**, click the **Attach an Extension** tool, select the `DADX` file (on the **General Purpose** tab), and click **OK** to attach it.

1. Select **Tools > Extended Generation** to open the Generation dialog with DADX selected in the **Targets** tab.

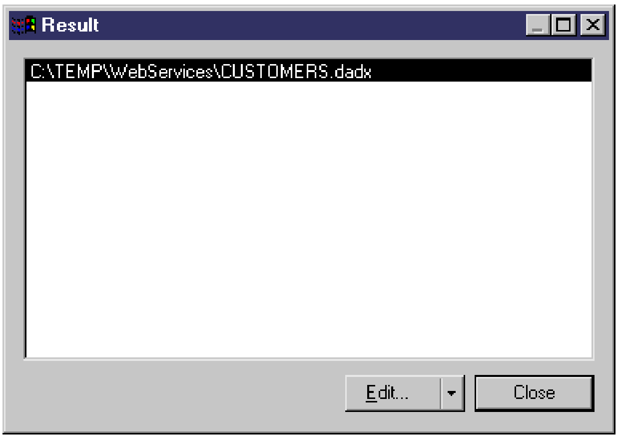


2. Click the **Select a Path** tool to the right of the **Directory** field, and specify a path for the DADX files.
3. Click the **Selection** tab, and select the web services for which you want to generate a DADX file.



4. Click **OK** to begin generation.

   When generation is complete, the Result dialog displays the paths of the DADX files.



5. [optional] Select the path of a DADX file and click **Edit** to display the DADX file 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>
```

6. 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 can reverse engineer Web services from a Sybase ASA, ASE, and IQ database to a PDM. 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 general information about database reverse engineering, see *Reverse Engineering a Database into a PDM* on page 312. 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 Dish-Prefix 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

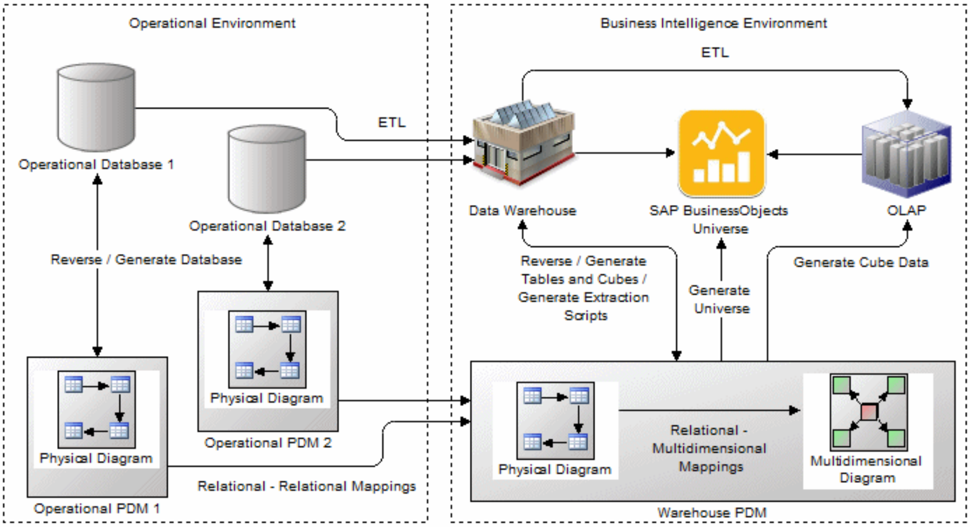
- Database DISH web services with or without a prefix name are considered identically in PowerDesigner:

Software	Web service name	Type	Web operation name
Database	Customers	DISH	—

Software	Web service name	Type	Web operation name
PowerDesigner	Customers (with or without DishPrefix as prefix)	DISH	—

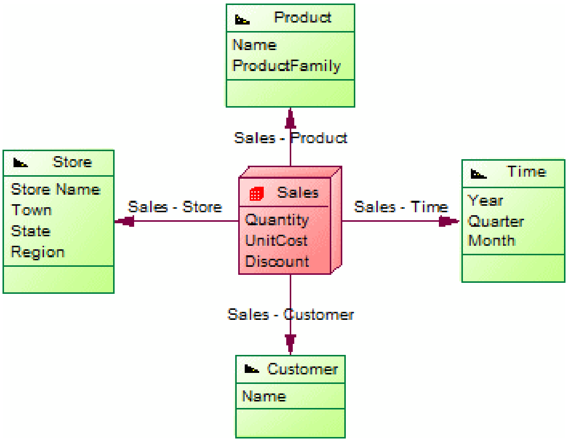
CHAPTER 4 Multidimensional Diagrams

A *multidimensional data diagram* provides a graphical view of your datamart or data warehouse database, and helps you identify the facts and dimensions that will be used to build its cubes.



Note: Multi-dimensional diagrams are generally generated from a physical diagram (see *Generating Cubes* on page 217). To manually create a multidimensional diagram in an existing PDM, right-click the model in the Browser and select **New** > **Multidimensional Diagram**. To create a new model, select **File** > **New Model**, choose Physical Data Model as the model type and **Multidimensional Diagram** as the first diagram, and then click **OK**.

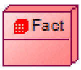
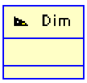
Numeric values or measures such as sales total, budget, and cost, are the facts of the business, while the areas covered by the business, in terms of geography, time, people and products, are the dimensions of the business. A multidimensional diagram shows the facts, surrounded by their dimensions, which will be used to populate cubes for enterprise information management, query and analysis tool and enterprise reporting. In the following example, the Sales fact is surrounded by the Product, Time, Customer, and Store dimensions to allow sales data to be analyzed by any of these criteria:



PowerDesigner maps facts and dimensions to their original operational database tables to enable population of the cubes (see *Operational to Warehouse Data Mappings* on page 227).

Multidimensional Diagram Objects

PowerDesigner supports all the objects necessary to build multidimensional diagrams.

Object	Tool	Symbol	Description
Fact			Group of measures related to aspects of the business and used to carry out a decision support investigation. See *Facts (PDM)* on page 219.
Dimension			Axis of investigation of a cube (time, product, geography). See *Dimensions (PDM)* on page 222.
Attribute	[none]	[none]	Used to qualify a dimension. For example, attribute Year qualifies the Date dimension. See *Fact and Dimension Attributes (PDM)* on page 223.
Measure	[none]	[none]	Variable linked to a fact, used as the focus of a decision support investigation. See *Measures (PDM)* on page 220.
Hierarchy	[none]	[none]	Organizational structure that describes a traversal pattern though a dimension. See *Hierarchies (PDM)* on page 225.
Association			Association that relates a fact to a dimension. See *Associations (PDM)* on page 226.

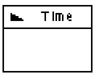
Identifying Fact and Dimension Tables

When designing a data warehouse, you will need to identify which of your tables and views represent facts (containing numerical values such as sales, revenue, or budget figures), and which dimensions (providing ways of aggregating these figures, such as by region, date, customer, or product). PowerDesigner can retrieve the multidimensional type of a table by analyzing the references attached to it, where child tables or views are identified as candidate facts and parent tables or views are identified as candidate dimensions.

1. Select **Tools > Multidimension > RetrieveMultidimensional Objects** to open the Multidimensional Objects Retrieval Wizard.
2. Specify the objects to be retrieved. By default both Facts and Dimensions will be retrieved.

   Note: 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 *Join Indexes (IQ/Oracle)* on page 534.
3. [optional] Click the **Selection** tab to specify which tables to consider as candidates for fact or dimension tables. By default, all tables except those that have their **Dimensional type** set to `Exclude` are selected (see *Table Properties* on page 76).
4. Click **OK** to retrieve the multidimensional objects.

   The selected tables are assigned a multidimensional type, and a type icon is displayed in the upper left corner of each table's symbol:

Fact table	Dimension table
	

5. [optional] Review the types identified by PowerDesigner and, if necessary, modify them by changing the value of the **Dimensional type** field on the **General** tab of the table or view property sheet.

Generating Cubes

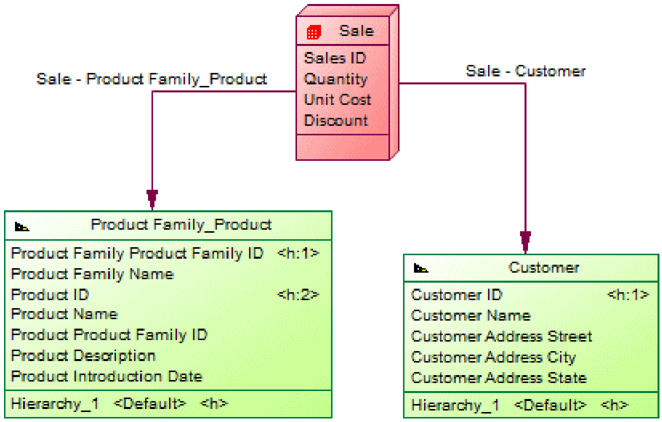
PowerDesigner can generate facts and dimensions from your operational tables to create a multidimensional diagram representing a cube. The generation will create mappings between your operational and warehouse objects as the basis for extraction scripts or in preparation for generating a BusinessObjects universe.

You can prepare and preview the multidimensional types of your operational tables and views before launching this wizard either manually by setting the **Dimensional type** value (see *Table Properties* on page 76) or have PowerDesigner retrieve them (see *Identifying Fact and*

Dimension Tables on page 217). You can generate a BusinessObjects universe at any time (see *Generating a BusinessObjects Universe* on page 301).

1. Select **Tools** > **Multidimensional Objects** > **Generate Cube** to open the wizard.
2. Select the package where you want to create the multidimensional diagram, and then click **Next**. For DBMSs, such as SAP HANA®, which require that you create your multidimensional objects in a package, PowerDesigner will force the creation of a new package if none exist.
3. Select the operational tables from which to build your facts and dimensions, and then click **Next**. By default, PowerDesigner selects all the tables in your model.
4. Select the operational tables from which to build your facts, and then click **Next**. By default, PowerDesigner selects tables with only outgoing references as facts.
5. Select the operational tables from which to build dimensions around each of your facts, and then click **Next**. By default, PowerDesigner selects all the tables with direct or indirect references from your fact tables and will merge second and higher order references into the dimensions created from first order references.
6. Select fact table columns as measures or attributes of your facts, and then click **Next**. By default, PowerDesigner selects non-key numeric columns as measures and all other columns as attributes. You can drag and drop columns between the Candidates, Measures, and Attributes trees as necessary
7. Review the list of facts that will be generated, and click **Finish** to begin the generation.

   The Generate Cubes Wizard creates a multidimensional object containing facts and dimensions to represent your cubes:



Modifying Cubes

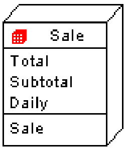
PowerDesigner can update your facts and dimensions in a multidimensional diagram representing a cube to reflect changes made to your operational tables or simply to add or remove dimensions, measures, or attributes.

1. Select the cube fact in the multidimensional diagram you want to update, and then select **Tools > Modify Cube** to open the wizard.
2. Select the operational tables from which to build dimensions around your facts, and then click **Next**. By default, PowerDesigner selects only those tables that you have previously selected as dimensions.
3. Select fact table columns as measures or attributes of your facts, and then click **Next**. By default, PowerDesigner reproduces your previous choices and you can drag and drop columns between the Candidates, Measures, and Attributes trees as necessary.
4. Review the objects that will be generated, and click **Finish** to begin the generation.

   The wizard updates your multidimensional diagram to reflect your new choices.

Facts (PDM)

Facts define the focus of the data to be analyzed and how it is calculated. Examples of facts are sales, costs, employee hours, revenue, budget. Facts contain a list of measures, which represent the actual numerical data, and are surrounded by dimensions, which control how that data will be analyzed.



Creating a Fact

Facts are generally generated from operational database tables or views. You can also manually create facts from the Toolbox, Browser, or **Model** menu.

- Use the **Fact** tool in the Toolbox.
- Select **Model > Facts** to access the List of Facts, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New > Fact**.

For general information about creating objects, see *Core Features Guide > Modeling with PowerDesigner > Objects*.

Fact Properties

To view or edit a fact's properties, double-click its diagram symbol or Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

Property	Description
Name/Code/ Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
Mapped to	Specifies the operational database table or view to which the fact is mapped. Click the **Properties** tool to open the source table property sheet. To map a manually-created fact to its source, open the Mapping Editor and drag and drop the table or view from the **Source** pane onto the fact in the **Target** pane (see *Operational to Warehouse Data Mappings* on page 227).
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

The following tabs are also available:

- **Attributes** - specifies attributes that are used by the fact for joins to dimensions or as the basis of calculated measures (see *Fact and Dimension Attributes (PDM)* on page 223).
- **Measures** - lists the measures manipulated by the fact (see *Measures (PDM)* on page 220).
- **Measures** - lists the dimensions linked to the fact in the cube (see *Dimensions (PDM)* on page 222).
- **Mapping** - specifies the mapping between the fact and the source operational database table or view (see *Operational to Warehouse Data Mappings* on page 227).

Measures (PDM)

Measures are mapped to numerical columns in fact tables and aggregate the values in the columns along the selected dimensions. For example, when a user chooses to view the sales in Texas in 2012 Q1, the calculation is performed via the Sales measure using a Sum aggregation. Measures can also be based on operations or calculations or derived from other measures.

Creating a Measure

Measures are generally generated from numerical columns in operational database tables. You can also manually create measures from the property sheet of, or in the Browser under, a fact.

- 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**.

Measure Properties

To view or edit a measure's properties, double-click its Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

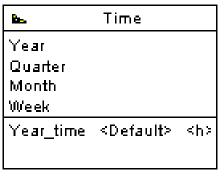
Property	Description
Parent	Specifies the parent fact of the measure. Click the **Properties** tool to open the fact property sheet.
Name/Code/ Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
Type	Specifies how the value of the measure is determined. In each case, specify the aggregation function to be applied to values and then choose from: • `Standard` - the measure is mapped to the operational table column specified in the **Mapped to** field. To map a manually-created measure to its source, open the Mapping Editor and drag and drop the column from the **Source** pane onto the measure in the **Target** pane (see *Operational to Warehouse Data Mappings* on page 227). • `Calculated` - the measure is calculated from an expression specified in the **Formula expression** field. Enter the expression directly or click the **Edit with SQL Editor** tool (see *Writing SQL Code in PowerDesigner* on page 281). • `Restricted` - the measure is derived from the measure specified in the **Base measure** field, and constrained by the values specified for each of the fact or dimension attributes added to the list.

Property	Description
Hidden	Specifies that the measure will not be visible to business users consulting the cube.
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

Dimensions (PDM)

A dimension is an axis of analysis in a multidimensional structure. Typical dimensions for a sales database include time, region, department, and product.

A dimension is made of an ordered list of attributes that share a common semantic meaning in the domain being modeled. For example a Time dimension often contains attributes that allow you to analyze data by year, quarter, month, and week:



A dimension may have one or more hierarchies representing different ways of traversing the list of attributes.

Creating a Dimension

Dimensions are generally generated from operational database tables or views. You can also manually create a dimension from the Toolbox, Browser, or **Model** menu.

- Use the **Dimension** tool in the Toolbox.
- Select **Model > Dimensions** to access the List of Dimensions, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New > Dimension**.

For general information about creating objects, see *Core Features Guide > Modeling with PowerDesigner > Objects*.

Dimension Properties

To view or edit a dimension's properties, double-click its diagram symbol or Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

Property	Description
Name/Code/Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
Mapped to	Specifies the operational database table or view to which the dimension is mapped. Click the **Properties** tool to open the source table property sheet. To map a manually-created dimension to its source, open the Mapping Editor and drag and drop the table or view from the **Source** pane onto the dimension in the **Target** pane (see *Operational to Warehouse Data Mappings* on page 227).
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
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

The following tabs are also available:

- Attributes - lists the attributes that qualify the dimension (see *Fact and Dimension Attributes (PDM)* on page 223).
- Hierarchies - lists the hierarchies used to organize the dimension attributes (see *Hierarchies (PDM)* on page 225).
- Mapping - defines the mapping between the current dimension and a table or a view in a data source.

Fact and Dimension Attributes (PDM)

Fact attributes are used by the fact for joins to dimensions or as the basis of calculated measures. Dimension attributes provide data points around which the data in a fact can be interrogated.

Creating an Attribute

Fact and dimension attributes are generally generated from operational database table columns. You can also manually create attributes as follows:

- Open the **Attributes** tab in the property sheet of a fact or dimension, and click the **Add a Row** or **Insert a Row** tool. The **Add Attributes** tool allows you to reuse an attribute from another fact or dimension.
- Right-click a fact or dimension in the Browser, and select **New > Attribute**.

Attribute Properties

To view or edit an attribute's properties, double-click its Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

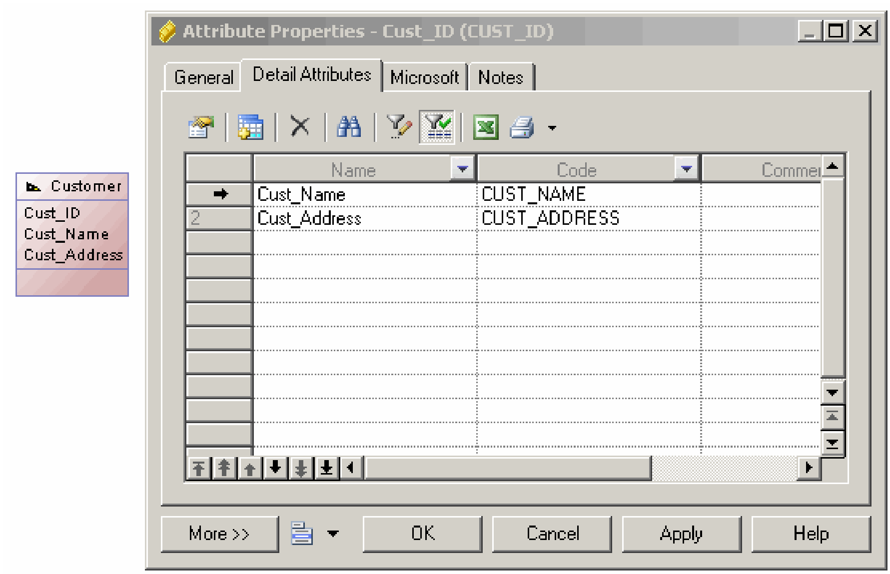
The **General** tab contains the following properties:

Property	Description
Parent	Specifies the parent fact or dimension of the attribute.
Name/Code/Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
Type	Specifies how the value of the attribute is determined: • `Standard` - the attribute is mapped to the operational table column specified in the **Mapped to** field. To map a manually-created attribute to its source, open the Mapping Editor and drag and drop the column from the **Source** pane onto the attribute in the **Target** pane (see *Operational to Warehouse Data Mappings* on page 227). • `Calculated` - the attribute is calculated from an expression specified in the **Formula expression** field. Enter the expression directly or click the **Edit with SQL Editor** tool (see *Writing SQL Code in PowerDesigner* on page 281).
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

Dimension attributes include the following tab:

- **Detail Attributes** - Lists other dimension attributes that are use to further define the attribute. Click the **Add Detail Attributes** tool to select attributes defined on the current dimension to further define the attribute.

  In the following example, attributes `Cust_Name` and `Cust_Address` are used as detail attributes for `Cust_ID`:



Hierarchies (PDM)

A *hierarchy* defines a path for navigating through the attributes in a dimension when drilling down or rolling up through the data. For example, a time dimension with the attributes Year, Quarter, Month, Week, Day may have a default hierarchy listing all these periods in order and a second hierarchy which includes only Year, Month, and Week.

Creating a Hierarchy

You can create a hierarchy from the property sheet of, or in the Browser under, a dimension.

- Open the **Attributes** tab in the property sheet of a dimension, select the attributes you want to include in your dimension and then click the **Create Hierarchy** tool.
- Open the **Hierarchies** tab in the property sheet of a dimension, click the **Add a Row** tool, then click the **Properties** tool and add your attributes manually.
- Right-click a dimension in the Browser, and select **New** > **Hierarchy**.

Hierarchy Properties

To view or edit a hierarchy's properties, double-click its Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

Property	Description
Name/Code/Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
Dimension	Specifies the parent dimension of the hierarchy.
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

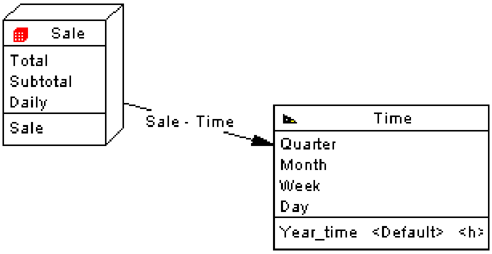
The following tabs are also available:

- **Attributes** - lists the attributes associated with the hierarchy in ascending order of specificity (see *Fact and Dimension Attributes (PDM)* on page 223).

Associations (PDM)

An association connects a fact to the dimension that defines it.

For example, the `Sale` fact 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 fact and a dimension.

Creating an Association

Associations are generally generated from operational database references. You can manually create associations from the Toolbox, Browser, or **Model** menu.

- Use the **Association** tool in the Toolbox.

- Select **Model > Associations** to access the List of Associations, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New** > **Association**.

Association Properties

To view or edit an association's properties, double-click its diagram symbol or Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

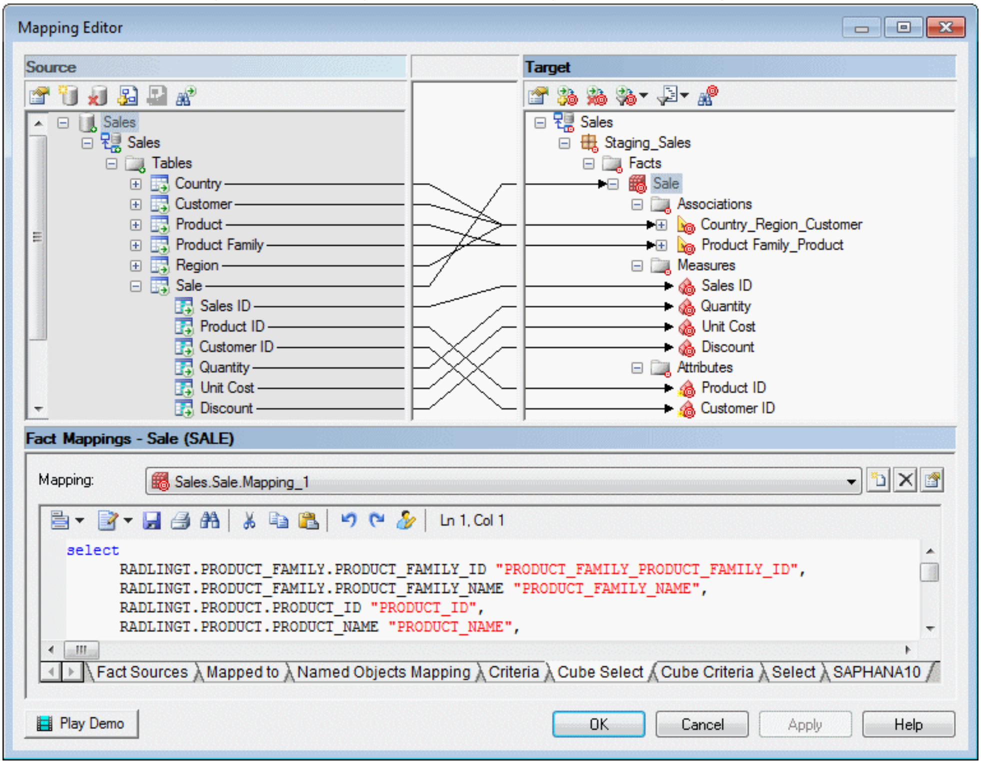
Property	Description
Fact	Specifies the fact at the origin of the association. Use the tools to the right of the list to create, browse for, or view the properties of the currently selected fact.
Dimension	Specifies the destination dimension of the association. Use the tools to the right of the list to create, browse for, or view the properties of the currently selected dimension.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
Reference	Specifies the reference upon which the association is based. Click the **Properties** tool to view the properties of the selected reference.
Hierarchy	Specifies the default hierarchy used by the cube for the consolidation calculation. Click the **Properties** tool to view the properties of the selected hierarchy.
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

Operational to Warehouse Data Mappings

Data warehousing requires the extraction, transformation, and loading of data from operational systems to a data warehouse database. You can create mappings between operational and data warehouse data and from the data warehouse data and OLAP cubes. To review or edit these mappings, open your multidimensional diagram, and then select **Tools > Mapping Editor**.

You can model operational and data warehouse data structures in PDMs, and specify mappings between the operational data sources and the data warehouse to generate extraction scripts to populate the data warehouse with operational data. In this kind of relational-to-relational mapping, operational tables are mapped to data warehouse tables with a type of fact or dimension, and operational columns are mapped to warehouse columns.

The Generate Cube wizard automatically creates mappings between source tables and facts and dimensions and you can modify these or manually create mappings between these objects:



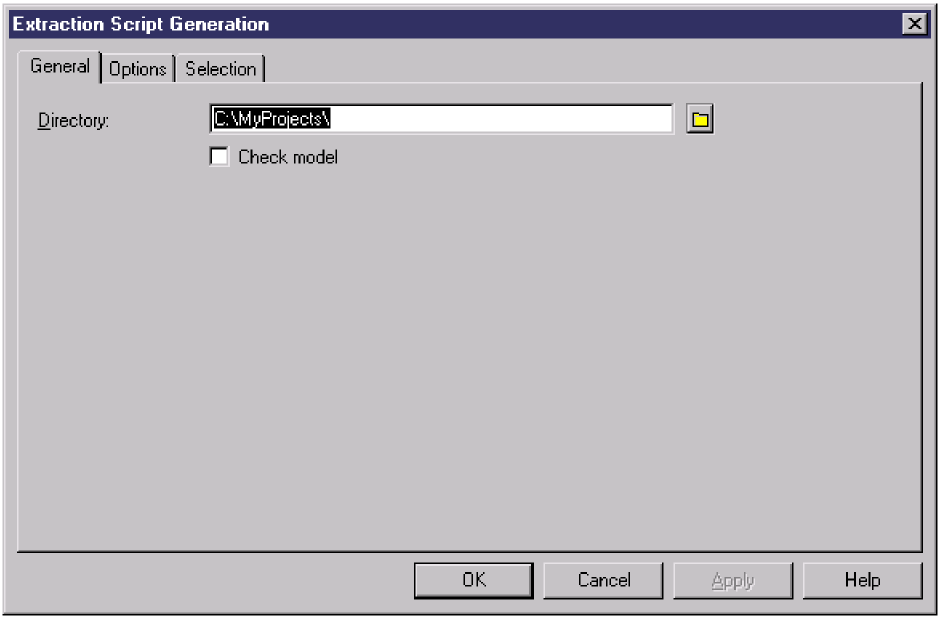
The **Select** sub-tab displays the SQL statement used to select data in the data source. The Generate Cube Data wizard uses this SQL statement to fill the text files used to populate cubes in an OLAP database.

Generating Data Warehouse Extraction Scripts

You can model operational and data warehouse data structures in PDMs, and specify mappings between the operational data sources and the data warehouse to generate extraction scripts to populate the data warehouse with operational data.

In this kind of relational-to-relational mapping, operational tables are mapped to data warehouse tables with a type of fact or dimension, and operational columns are mapped to warehouse columns.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.

1. In the Physical Diagram, select **Database** > **Generate Extraction Scripts**:



2. Specify a destination directory for the generated file, and select the **Check Model** option if you want to verify the PDM syntax before generation.
3. [optional] Click the **Options** tab and specify any appropriate options:

Option	Description
Title	Specifies to insert the database header and the name of the tables before each select query.
Encoding	Specifies the encoding format. You should select the format that supports the language used in your model and the database encoding format.
Character Case	Specifies the case to use in the generated file.
No Accent	Specifies to remove any accents from generated characters.

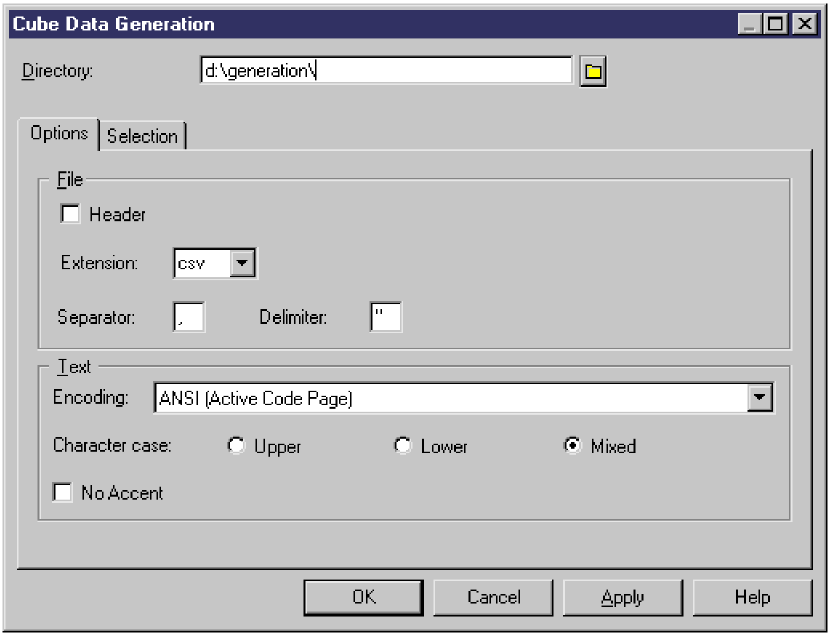
4. [optional] Click the **Selection** tab, and select the tables for which you want to generate extraction scripts.
5. Click **OK** to generate the script files in the specified directory. The name of the script is identical to the name of the data source.

Generating Cube Data

You can map physical tables (including those of type dimension or fact) to cube dimensions or cube measures in OLAP databases, and use these mappings to generate cube data in text files to be loaded by OLAP engines. When you use the Rebuild Cubes command to create cubes and dimensions from fact and dimension tables, mappings between source tables and OLAP objects are automatically created.

In a PDM multidimensional diagram, each fact is associated with a query. There is one fact per mapping and per data source. The query defined on a fact 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.

1. In the multidimensional diagram, select **Tools** > **Generate Cube Data**.



2. Specify a destination directory for the generated file, and select any appropriate options in the **Options** tab:

Option	Description
Header	Specifies to include the name of the attribute at the beginning of the generated text file
Extension	Specifies the extension of the generated text file.You can choose either `.txt` and `.csv`.
Separator	Specifies the separator to use between columns. The default is `,` (comma).
Delimiter	Specifies the character to delimit string values. The default is `"` (double-quote).
Encoding	Specifies the encoding format. You should select the format that supports the language used in your model and the database encoding format.

Option	Description
Character Case	Specifies the case to use in the generated file.
No Accent	Specifies to remove any accents from generated characters.

3. Select the facts and data sources for which you want to generate a file from the sub- tabs in the Selection tab.
4. Click OK.

   The generated files are stored in the destination directory you have defined. PowerDesigner produces one file for each selected fact and each selected data source, named by concatenating the names of the fact and the data source, and containing 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

CHAPTER 5 Checking a Data Model

The data model is a very flexible tool, which allows you quickly to develop your model without constraints. 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

Note: 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 your 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 dialog opens, allowing you to specify the kinds of checks to perform, and the objects to apply them to. 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 and for detailed information about using the Check Model Parameters dialog, see *Core Features Guide > Modeling with PowerDesigner > Objects > Checking Models*.

Abstract Data Type Checks (PDM)

PowerDesigner provides default model checks to verify the validity of abstract data types.

Check	Description and Correction
Name/Code contains terms not in glossary	[if glossary enabled] Names and codes must contain only approved terms drawn from the glossary. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - None.

Check	Description and Correction
Name/Code contains synonyms of glossary terms	[if glossary enabled] Names and codes must not contain synonyms of glossary terms. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - Replaces synonyms with their associated glossary terms.
Name/Code uniqueness	Object names must be unique in the namespace. • Manual correction - Modify the duplicate name or code. • Automatic correction - Appends a number to the duplicate name or 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)

PowerDesigner provides default model checks to verify the validity of abstract data type procedures.

Check	Description and Correction
Name/Code contains terms not in glossary	[if glossary enabled] Names and codes must contain only approved terms drawn from the glossary. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - None.

Check	Description and Correction
Name/Code contains synonyms of glossary terms	[if glossary enabled] Names and codes must not contain synonyms of glossary terms. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - Replaces synonyms with their associated glossary terms.
Name/Code uniqueness	Object names must be unique in the namespace. • Manual correction - Modify the duplicate name or code. • Automatic correction - Appends a number to the duplicate name or code.
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)

PowerDesigner provides default model checks to verify the validity of associations.

Check	Description and Correction
Name/Code contains terms not in glossary	[if glossary enabled] Names and codes must contain only approved terms drawn from the glossary. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - None.

Check	Description and Correction
Name/Code contains synonyms of glossary terms	[if glossary enabled] Names and codes must not contain synonyms of glossary terms. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - Replaces synonyms with their associated glossary terms.
Name/Code uniqueness	Object names must be unique in the namespace. • Manual correction - Modify the duplicate name or code. • Automatic correction - Appends a number to the duplicate name or 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.
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.

Check	Description and Correction
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.
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)

PowerDesigner provides default model checks to verify the validity of associations.

Check	Description and Correction
Existence of hierarchy	An association must have a hierarchy specified in order to perform the consolidation calculation. • Manual correction: Select a hierarchy in the Hierarchy list in the association property sheet • Automatic correction: None

Column Checks (PDM)

PowerDesigner provides default model checks to verify the validity of columns.

Check	Description and Correction
Name/Code contains terms not in glossary	[if glossary enabled] Names and codes must contain only approved terms drawn from the glossary. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - None.
Name/Code contains synonyms of glossary terms	[if glossary enabled] Names and codes must not contain synonyms of glossary terms. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - Replaces synonyms with their associated glossary terms.
Name/Code uniqueness	Object names must be unique in the namespace. • Manual correction - Modify the duplicate name or code. • Automatic correction - Appends a number to the duplicate name or code.
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

Check	Description and Correction
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
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

Check	Description and Correction
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
Invalid mapping from source column	A column in a table managed by a lifecycle policy in which the first phase is associated with an external database must not be mapped to more than one column in the corresponding table in the external database. • Manual correction: Remove the additional mappings. • Automatic correction: None
Data type compatibility of mapped columns	A column in a table managed by a lifecycle policy in which the first phase is associated with an external database must be mapped to a column with the same data type in the corresponding table in the external database. • Manual correction: Harmonize the data types in the source and target columns. • Automatic correction: None
Existence of mapping for mandatory columns	A mandatory column in a table managed by a lifecycle policy in which the first phase is associated with an external database must be mapped to a column in the corresponding table in the external database. • Manual correction: Map the mandatory column to a column in the external database. • Automatic correction: None

Cube Checks (PDM)

PowerDesigner provides default model checks to verify the validity of cubes.

Check	Description and Correction
Name/Code contains terms not in glossary	[if glossary enabled] Names and codes must contain only approved terms drawn from the glossary. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - None.

Check	Description and Correction
Name/Code contains synonyms of glossary terms	[if glossary enabled] Names and codes must not contain synonyms of glossary terms. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - Replaces synonyms with their associated glossary terms.
Name/Code uniqueness	Object names must be unique in the namespace. • Manual correction - Modify the duplicate name or code. • Automatic correction - Appends a number to the duplicate name or 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)

PowerDesigner provides default model checks to verify the validity of databases.

Check	Description and Correction
Name/Code contains terms not in glossary	[if glossary enabled] Names and codes must contain only approved terms drawn from the glossary. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - None.
Name/Code contains synonyms of glossary terms	[if glossary enabled] Names and codes must not contain synonyms of glossary terms. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - Replaces synonyms with their associated glossary terms.

Check	Description and Correction
Name/Code uniqueness	Object names must be unique in the namespace. • Manual correction - Modify the duplicate name or code. • Automatic correction - Appends a number to the duplicate name or 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
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)

PowerDesigner provides default model checks to verify the validity of database packages.

Check	Description and Correction
Name/Code contains terms not in glossary	[if glossary enabled] Names and codes must contain only approved terms drawn from the glossary. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - None.
Name/Code contains synonyms of glossary terms	[if glossary enabled] Names and codes must not contain synonyms of glossary terms. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - Replaces synonyms with their associated glossary terms.
Name/Code uniqueness	Object names must be unique in the namespace. • Manual correction - Modify the duplicate name or code. • Automatic correction - Appends a number to the duplicate name or code.

Check	Description and Correction
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 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 (at **Objects > DB Package > MaxLen**)

Existence of package *sub-object*	A database package must have a number of sub-objects defined in order to be correctly modeled.
	
	• Manual correction: Create one or more of the relevant object on the appropriate tab of the database package property sheet:
	  • Procedures (or use existing stored procedures and duplicate them in the database package)
	  • Cursors
	  • Variables
	  • Types
	  • Exceptions
	• Automatic correction: None

Database Package Sub-Object Checks (PDM)

PowerDesigner provides default model checks to verify the validity of database package cursors, exceptions, procedures, types, and variables.

Check	Description and Correction
Name/Code contains terms not in glossary	[if glossary enabled] Names and codes must contain only approved terms drawn from the glossary. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - None.
Name/Code contains synonyms of glossary terms	[if glossary enabled] Names and codes must not contain synonyms of glossary terms. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - Replaces synonyms with their associated glossary terms.

Check	Description and Correction
Name/Code uniqueness	Object names must be unique in the namespace. • Manual correction - Modify the duplicate name or code. • Automatic correction - Appends a number to the duplicate name or code.
Package *sub-object* definition empty	[cursors, procedures, types] These sub-objects must have a definition. • Manual correction: Create the definition in the **Definition** tab of the sub-object's property sheet • Automatic correction: None
Check for undefined return types	[cursors, procedures] These sub-objects must have a return data type. • Manual correction: Select a return data type in the subobject's property sheet • Automatic correction: None
Existence of parameter	[cursors, procedures] These sub-objects must contain parameters for input values. • Manual correction: Create one or several parameters in the **Parameters** tab of the sub-object's property sheet • Automatic correction: None
Undefined data type	[variables] Variables must have a data type. • Manual correction: Select a data type in the variable property sheet • Automatic correction: None

Data Format Checks (CDM/LDM/PDM)

PowerDesigner provides default model checks to verify the validity of data formats.

Check	Description and Correction
Empty expression	Data formats must have a value entered in the **Expression** field. • Manual correction: Specify an expression for the data format. • Automatic correction: None

Data Item Checks (CDM)

PowerDesigner provides default model checks to verify the validity of data items.

Check	Description and Correction
Name/Code contains terms not in glossary	[if glossary enabled] Names and codes must contain only approved terms drawn from the glossary. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - None.
Name/Code contains synonyms of glossary terms	[if glossary enabled] Names and codes must not contain synonyms of glossary terms. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - Replaces synonyms with their associated glossary terms.
Name/Code uniqueness	Object names must be unique in the namespace. • Manual correction - Modify the duplicate name or code. • Automatic correction - Appends a number to the duplicate name or 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).

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 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)

PowerDesigner provides default model checks to verify the validity of data sources.

Check	Description and Correction
Name/Code contains terms not in glossary	[if glossary enabled] Names and codes must contain only approved terms drawn from the glossary. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - None.

Check	Description and Correction
Name/Code contains synonyms of glossary terms	[if glossary enabled] Names and codes must not contain synonyms of glossary terms. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - Replaces synonyms with their associated glossary terms.
Name/Code uniqueness	Object names must be unique in the namespace. • Manual correction - Modify the duplicate name or code. • Automatic correction - Appends a number to the duplicate name or 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.
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
Unsupported source models	Each lifecycle policy can only manage one external database, so any data sources defined (and the models they reference) must all point to the same database. • Manual correction: Remove any data sources pointing to other databases. • Automatic correction: None

Default Checks (PDM)

PowerDesigner provides default model checks to verify the validity of defaults.

Check	Description and Correction
Name/Code contains terms not in glossary	[if glossary enabled] Names and codes must contain only approved terms drawn from the glossary. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - None.

Check	Description and Correction
Name/Code contains synonyms of glossary terms	[if glossary enabled] Names and codes must not contain synonyms of glossary terms. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - Replaces synonyms with their associated glossary terms.
Name/Code uniqueness	Object names must be unique in the namespace. • Manual correction - Modify the duplicate name or code. • Automatic correction - Appends a number to the duplicate name or 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)

PowerDesigner provides default model checks to verify the validity of dimensions.

Check	Description and Correction
Name/Code contains terms not in glossary	[if glossary enabled] Names and codes must contain only approved terms drawn from the glossary. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - None.

Check	Description and Correction
Name/Code contains synonyms of glossary terms	[if glossary enabled] Names and codes must not contain synonyms of glossary terms. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - Replaces synonyms with their associated glossary terms.
Name/Code uniqueness	Object names must be unique in the namespace. • Manual correction - Modify the duplicate name or code. • Automatic correction - Appends a number to the duplicate name or 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

Domain Checks (CDM/LDM/PDM)

PowerDesigner provides default model checks to verify the validity of domains.

Check	Description and Correction
Name/Code contains terms not in glossary	[if glossary enabled] Names and codes must contain only approved terms drawn from the glossary. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - None.
Name/Code contains synonyms of glossary terms	[if glossary enabled] Names and codes must not contain synonyms of glossary terms. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - Replaces synonyms with their associated glossary terms.
Name/Code uniqueness	Object names must be unique in the namespace. • Manual correction - Modify the duplicate name or code. • Automatic correction - Appends a number to the duplicate name or code.
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

Check	Description and Correction
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.

Entity Attribute Checks (CDM/LDM)

PowerDesigner provides default model checks to verify the validity of entity attributes.

Check	Description and Correction
Name/Code contains terms not in glossary	[if glossary enabled] Names and codes must contain only approved terms drawn from the glossary. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - None.
Name/Code contains synonyms of glossary terms	[if glossary enabled] Names and codes must not contain synonyms of glossary terms. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - Replaces synonyms with their associated glossary terms.

Check	Description and Correction
Name/Code uniqueness	Object names must be unique in the namespace. • Manual correction - Modify the duplicate name or code. • Automatic correction - Appends a number to the duplicate name or code.
Detect differences between attribute and associated domain	[LDM only] 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).
Detect inconsistencies between check parameters	[LDM only] 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	[LDM only] 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	[LDM only] 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	[LDM only] 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)

PowerDesigner provides default model checks to verify the validity of entity identifiers.

Check	Description and Correction
Name/Code contains terms not in glossary	[if glossary enabled] Names and codes must contain only approved terms drawn from the glossary. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - None.
Name/Code contains synonyms of glossary terms	[if glossary enabled] Names and codes must not contain synonyms of glossary terms. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - Replaces synonyms with their associated glossary terms.
Name/Code uniqueness	Object names must be unique in the namespace. • Manual correction - Modify the duplicate name or code. • Automatic correction - Appends a number to the duplicate name or 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.
Primary identifier in child entity	[Barker notation] Primary identifiers are no permitted in child entities • Manual correction: Move the primary identifier to the parent entity. • Automatic correction: None

Entity Checks (CDM/LDM)

PowerDesigner provides default model checks to verify the validity of entities.

Check	Description and Correction
Name/Code contains terms not in glossary	[if glossary enabled] Names and codes must contain only approved terms drawn from the glossary. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - None.
Name/Code contains synonyms of glossary terms	[if glossary enabled] Names and codes must not contain synonyms of glossary terms. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - Replaces synonyms with their associated glossary terms.
Name/Code uniqueness	Object names must be unique in the namespace. • Manual correction - Modify the duplicate name or code. • Automatic correction - Appends a number to the duplicate name or 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.
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.

Check	Description and Correction
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)

PowerDesigner provides default model checks to verify the validity of facts.

Check	Description and Correction
Name/Code contains terms not in glossary	[if glossary enabled] Names and codes must contain only approved terms drawn from the glossary. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - None.

Check	Description and Correction
Name/Code contains synonyms of glossary terms	[if glossary enabled] Names and codes must not contain synonyms of glossary terms. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - Replaces synonyms with their associated glossary terms.
Name/Code uniqueness	Object names must be unique in the namespace. • Manual correction - Modify the duplicate name or code. • Automatic correction - Appends a number to the duplicate name or 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 and Dimension Hierarchy and Attribute Checks (PDM)

PowerDesigner provides default model checks to verify the validity of fact measures and dimension hierarchies and attributes.

Check	Description and Correction
Name/Code contains terms not in glossary	[if glossary enabled] Names and codes must contain only approved terms drawn from the glossary. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - None.
Name/Code contains synonyms of glossary terms	[if glossary enabled] Names and codes must not contain synonyms of glossary terms. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - Replaces synonyms with their associated glossary terms.
Name/Code uniqueness	Object names must be unique in the namespace. • Manual correction - Modify the duplicate name or code. • Automatic correction - Appends a number to the duplicate name or code.
Existence of attribute	[hierarchies only] 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

Horizontal and Vertical Partitioning and Table Collapsing Checks (PDM)

PowerDesigner provides default model checks to verify the validity of horizontal and vertical partitioning and table collapsing objects.

Check	Description and Correction
Name/Code contains terms not in glossary	[if glossary enabled] Names and codes must contain only approved terms drawn from the glossary. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - None.

Check	Description and Correction
Name/Code contains synonyms of glossary terms	[if glossary enabled] Names and codes must not contain synonyms of glossary terms. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - Replaces synonyms with their associated glossary terms.
Name/Code uniqueness	Object names must be unique in the namespace. • Manual correction - Modify the duplicate name or code. • Automatic correction - Appends a number to the duplicate name or code.
Existence of partition	[horizontal and vertical partitionings] A partitioning object cannot be empty, it must contain at least one partition. • Manual correction: Delete the partitioning object or create at least one partition in its property sheet • Automatic correction: Deletes empty horizontal partitioning object
Existence of target table	[collapsings] 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	A partition or collapsing object requires a table to act upon. • Manual correction: Delete the partitioning or collapsing with no corresponding table • Automatic correction: Deletes the partitioning or collapsing with no corresponding table

Index and View Index Checks (PDM)

PowerDesigner provides default model checks to verify the validity of indexes and view indexes.

Check	Description and Correction
Name/Code contains terms not in glossary	[if glossary enabled] Names and codes must contain only approved terms drawn from the glossary. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - None.

Check	Description and Correction
Name/Code contains synonyms of glossary terms	[if glossary enabled] Names and codes must not contain synonyms of glossary terms. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - Replaces synonyms with their associated glossary terms.
Name/Code uniqueness	Object names must be unique in the namespace. • Manual correction - Modify the duplicate name or code. • Automatic correction - Appends a number to the duplicate name or code.
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	[indexes] 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	[indexes] 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)

PowerDesigner provides default model checks to verify the validity of inheritances.

Check	Description and Correction
Name/Code contains terms not in glossary	[if glossary enabled] Names and codes must contain only approved terms drawn from the glossary. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - None.
Name/Code contains synonyms of glossary terms	[if glossary enabled] Names and codes must not contain synonyms of glossary terms. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - Replaces synonyms with their associated glossary terms.
Name/Code uniqueness	Object names must be unique in the namespace. • Manual correction - Modify the duplicate name or code. • Automatic correction - Appends a number to the duplicate name or 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)

PowerDesigner provides default model checks to verify the validity of join indexes and bitmap join indexes.

Check	Description and Correction
Name/Code contains terms not in glossary	[if glossary enabled] Names and codes must contain only approved terms drawn from the glossary. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - None.
Name/Code contains synonyms of glossary terms	[if glossary enabled] Names and codes must not contain synonyms of glossary terms. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - Replaces synonyms with their associated glossary terms.
Name/Code uniqueness	Object names must be unique in the namespace. • Manual correction - Modify the duplicate name or code. • Automatic correction - Appends a number to the duplicate name or 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
Reference without parent key	Each reference associated with a join index must have a parent key. • Manual correction: Set the parent key on the Joins tab of the reference 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
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

Check	Description and Correction
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)

PowerDesigner provides default model checks to verify the validity of keys.

Check	Description and Correction
Name/Code contains terms not in glossary	[if glossary enabled] Names and codes must contain only approved terms drawn from the glossary. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - None.
Name/Code contains synonyms of glossary terms	[if glossary enabled] Names and codes must not contain synonyms of glossary terms. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - Replaces synonyms with their associated glossary terms.
Name/Code uniqueness	Object names must be unique in the namespace. • Manual correction - Modify the duplicate name or code. • Automatic correction - Appends a number to the duplicate name or 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

Lifecycle and Lifecycle Phase Checks (PDM)

PowerDesigner provides default model checks to verify the validity of lifecycles and phases.

Check	Description and Correction
Name/Code contains terms not in glossary	[if glossary enabled] Names and codes must contain only approved terms drawn from the glossary. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - None.
Name/Code contains synonyms of glossary terms	[if glossary enabled] Names and codes must not contain synonyms of glossary terms. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - Replaces synonyms with their associated glossary terms.
Name/Code uniqueness	Object names must be unique in the namespace. • Manual correction - Modify the duplicate name or code. • Automatic correction - Appends a number to the duplicate name or code.
Existence of phases	[lifecycle] A lifecycle must contain phases. • Manual correction: Add phases to the lifecycle (on the Phases tab) • Automatic correction: None
Incorrect total retention setting	[lifecycle] The total retention for the lifecycle must equal the retentions of all the phases. • Manual correction: Adjust the total retention or the retentions of individual phases as appropriate. • Automatic correction: Adjust the total retention to equal the retentions of all the phases.

Check	Description and Correction
Invalid partition range setting	[lifecycle] The partition range must be no longer than the shortest phase retention. • Manual correction: Reduce the partition range so that it is equal to the shortest phase retention. • Automatic correction: Reduces the partition range so that it is equal to the shortest phase retention.
Existence of tablespace	[phase] Specified tablespace does not exist. • Manual correction: Specify another tablespace. • Automatic correction: None
Invalid tablespace setting	[phase] The tablespace cannot be a catalog store. • Manual correction: Deselect the catalog store property on the tablespace property sheet. • Automatic correction: Deselects the catalog store property.
Phase tablespace uniqueness	[phase] Each phase must be associated with a different tablespace. • Manual correction: Move one or more phases to another tablespace. • Automatic correction: None
Consistency of cost currency setting	[phase] The same currency must be used for all tablespaces. • Manual correction: Harmonize the currency settings. • Automatic correction: Applies the currency specified in the model options to all tablespaces.
Invalid retention setting	[phase] Age-based lifecyle phases must have a retention period greater than 0. • Manual correction: Set the retention period to greater than 0. • Automatic correction: Sets the retention period to 1.
Invalid idle period setting	[phase] Access-based lifecyle phases must have an idle period greater than 0. • Manual correction: Set the idle period to greater than 0. • Automatic correction: Sets the idle period to 1.
Existence of data source	[phase] A lifecycle phase associated with an external database must have a data source specified. • Manual correction: Specify a data source for the phase. • Automatic correction: None

Check	Description and Correction
Invalid lifecycle management scope	[phase] Only the first phase in a lifecycle can have an external source. Subsequent phases must have the source set to the current database. • Manual correction: Set the phase source to the current database. • Automatic correction: None

Package Checks (CDM/LDM/PDM)

PowerDesigner provides default model checks to verify the validity of packages.

Check	Description and Correction
Name/Code contains terms not in glossary	[if glossary enabled] Names and codes must contain only approved terms drawn from the glossary. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - None.
Name/Code contains synonyms of glossary terms	[if glossary enabled] Names and codes must not contain synonyms of glossary terms. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - Replaces synonyms with their associated glossary terms.
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

Check	Description and Correction
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
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] Traceability links of type `<<DBCreateAfter>>` can be used to define a generation order for stored procedures and views. These links should not introduce circular dependencies in the model. • Manual correction: Remove the link. • 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.

Check	Description and Correction
Shortcut code uniqueness	Shortcuts codes must be unique in a namespace. • Manual correction: Change the code of one of the shortcuts • Automatic correction: None
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)

PowerDesigner provides default model checks to verify the validity of procedures.

Check	Description and Correction
Name/Code contains terms not in glossary	[if glossary enabled] Names and codes must contain only approved terms drawn from the glossary. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - None.
Name/Code contains synonyms of glossary terms	[if glossary enabled] Names and codes must not contain synonyms of glossary terms. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - Replaces synonyms with their associated glossary terms.
Name/Code uniqueness	Object names must be unique in the namespace. • Manual correction - Modify the duplicate name or code. • Automatic correction - Appends a number to the duplicate name or 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

Check	Description and Correction
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 and View Reference Checks (PDM)

PowerDesigner provides default model checks to verify the validity of references and view references.

Check	Description and Correction
Name/Code contains terms not in glossary	[if glossary enabled] Names and codes must contain only approved terms drawn from the glossary. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - None.
Name/Code contains synonyms of glossary terms	[if glossary enabled] Names and codes must not contain synonyms of glossary terms. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - Replaces synonyms with their associated glossary terms.
Name/Code uniqueness	Object names must be unique in the namespace. • Manual correction - Modify the duplicate name or code. • Automatic correction - Appends a number to the duplicate name or code.
Reflexive and mandatory reference	[references only] 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

Check	Description and Correction
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	[references only] 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	[references only] Joins must be complete. • Manual correction: Select a foreign key column or activate the primary key column migration • Automatic correction: None
Join order	[references only] 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

Relationship Checks (CDM/LDM)

PowerDesigner provides default model checks to verify the validity of relationships.

Check	Description and Correction
Name/Code contains terms not in glossary	[if glossary enabled] Names and codes must contain only approved terms drawn from the glossary. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - None.

Check	Description and Correction
Name/Code contains synonyms of glossary terms	[if glossary enabled] Names and codes must not contain synonyms of glossary terms. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - Replaces synonyms with their associated glossary terms.
Name/Code uniqueness	Object names must be unique in the namespace. • Manual correction - Modify the duplicate name or code. • Automatic correction - Appends a number to the duplicate name or 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.
Name uniqueness constraint between 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.

Check	Description and Correction
Identifier link from child entity	[Barker notation CDM only] A child entity may not be dependant on any entity other than its parents. • Manual correction: Remove the dependant relationship with the non-parent. • 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.

Sequence Checks (PDM)

PowerDesigner provides default model checks to verify the validity of sequences.

Check	Description and Correction
Name/Code contains terms not in glossary	[if glossary enabled] Names and codes must contain only approved terms drawn from the glossary. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - None.
Name/Code contains synonyms of glossary terms	[if glossary enabled] Names and codes must not contain synonyms of glossary terms. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - Replaces synonyms with their associated glossary terms.
Name/Code uniqueness	Object names must be unique in the namespace. • Manual correction - Modify the duplicate name or code. • Automatic correction - Appends a number to the duplicate name or 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

Synonym Checks (PDM)

PowerDesigner provides default model checks to verify the validity of synonyms.

Check	Description and Correction
Name/Code contains terms not in glossary	[if glossary enabled] Names and codes must contain only approved terms drawn from the glossary. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - None.
Name/Code contains synonyms of glossary terms	[if glossary enabled] Names and codes must not contain synonyms of glossary terms. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - Replaces synonyms with their associated glossary terms.
Name/Code uniqueness	Object names must be unique in the namespace. • Manual correction - Modify the duplicate name or code. • Automatic correction - Appends a number to the duplicate name or 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
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 and View Checks (PDM)

PowerDesigner provides default model checks to verify the validity of tables and views.

Check	Description and Correction
Name/Code contains terms not in glossary	[if glossary enabled] Names and codes must contain only approved terms drawn from the glossary. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - None.
Name/Code contains synonyms of glossary terms	[if glossary enabled] Names and codes must not contain synonyms of glossary terms. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - Replaces synonyms with their associated glossary terms.
Name/Code uniqueness	Object names must be unique in the namespace. • Manual correction - Modify the duplicate name or code. • Automatic correction - Appends a number to the duplicate name or code.
Name and code length	The table and view name and code length is limited by the maximum length specified in the DBMS definition (MaxLen entry, in the **Objects > Table** and **View** categories) 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 conflicts with index name	[tables only] 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	[tables only] 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
Number of auto-incremented columns	[tables only] 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

Check	Description and Correction
Table index definition uniqueness	[tables only] 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	[tables only] 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	[tables only] 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 or view for a particular user, group or role. • Manual correction: Define permissions on the table or view for users, groups and roles • Automatic correction: None
Existence of partition key	[tables only] A table managed by an age-based lifecycle policy must have a column specified as its partition key. • Manual correction: Specify a column as the partition key. • Automatic correction: None
Invalid start date setting	[tables only] A table managed by an age-based lifecycle policy must not have a start date earlier than the start date of the lifecycle. • Manual correction: Change one or other date so that the table start date is equal to or later than the lifecycle start date. • Automatic correction: Changes the table start date to the lifecycle start date.

Check	Description and Correction
Missing lifecycle policy	[tables only] A table managed by a lifecycle must not reference tables not managed by a lifecycle. • Manual correction: Add the referenced tables to the lifecycle. • Automatic correction: None
Invalid mapping from source table	[tables only] In a lifecycle where the first phase references an external database, each archive table must be mapped to exactly one external table. • Manual correction: Remove the additional mappings. • Automatic correction: None
Partial column mapping of source table	[tables only] In a lifecycle where the first phase references an external database, all columns in each source table must be mapped to columns in the same archive table. • Manual correction: Create the missing mappings. • Automatic correction: None
Existence of partition key mapping	[tables only] In a lifecycle where the first phase references an external database, the partition key column in the archive table must be mapped to a column in the source table. • Manual correction: Create the missing mapping. • Automatic correction: None
Tablespace outside lifecycle	[tables only] A table managed by a lifecycle must be assigned to a tablespace associated with the lifecycle. • Manual correction: Assign the table to a tablespace associated with the lifecycle. • Automatic correction: If the table is not assigned to any tablespace it will be assigned to the tablespace associated with the first phase of the lifecycle.

Tablespace and Storage Checks (PDM)

PowerDesigner provides default model checks to verify the validity of tablespaces and storages.

Check	Description and Correction
Name/Code contains terms not in glossary	[if glossary enabled] Names and codes must contain only approved terms drawn from the glossary. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - None.

Check	Description and Correction
Name/Code contains synonyms of glossary terms	[if glossary enabled] Names and codes must not contain synonyms of glossary terms. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - Replaces synonyms with their associated glossary terms.
Name/Code uniqueness	Object names must be unique in the namespace. • Manual correction - Modify the duplicate name or code. • Automatic correction - Appends a number to the duplicate name or code.
Code maximum length	The code of the tablespace or 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
Not used	The tablespace or storage you have created is not used in the model. • Manual correction: Delete the tablespace or storage or apply it 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 and DBMS Trigger Checks (PDM)

PowerDesigner provides default model checks to verify the validity of triggers and DBMS triggers.

Check	Description and Correction
Name/Code uniqueness	Object names must be unique in the namespace. • Manual correction - Modify the duplicate name or code. • Automatic correction - Appends a number to the duplicate name or code.
Trigger code maximum length	The trigger code length is limited by the maximum length specified in the DBMS definition (MaxLen). • 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

Check	Description and Correction
Invalid event	The event specified in the DBMS trigger definition must be available in its chosen scope. • Manual correction: Modify the trigger code to reference an event in the chosen scope. • Automatic correction: None

User, Group, and Role Checks (PDM)

PowerDesigner provides default model checks to verify the validity of users, groups, and roles.

Check	Description and Correction
Name/Code contains terms not in glossary	[if glossary enabled] Names and codes must contain only approved terms drawn from the glossary. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - None.
Name/Code contains synonyms of glossary terms	[if glossary enabled] Names and codes must not contain synonyms of glossary terms. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - Replaces synonyms with their associated glossary terms.
Name/Code uniqueness	Object names must be unique in the namespace. • Manual correction - Modify the duplicate name or code. • Automatic correction - Appends a number to the duplicate name or code.
Code maximum length	The code length is limited by the maximum length specified in the DBMS definition (MaxLen entry, in the **Objects > User** and **Group** categories). • Manual correction: Modify the code length to meet this requirement • Automatic correction: Truncates the code length to the maximum length specified in the DBMS definition
Existence of user	[groups, roles] 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

Check	Description and Correction
Password empty	[users, groups] Users and groups must have a password to be able to connect to the database. • Manual correction: Define a password for the user or group • Automatic correction: None

View Checks (PDM)

PowerDesigner provides default model checks to verify the validity of views.

Check	Description and Correction
Name/Code contains terms not in glossary	[if glossary enabled] Names and codes must contain only approved terms drawn from the glossary. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - None.
Name/Code contains synonyms of glossary terms	[if glossary enabled] Names and codes must not contain synonyms of glossary terms. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - Replaces synonyms with their associated glossary terms.
Name/Code uniqueness	Object names must be unique in the namespace. • Manual correction - Modify the duplicate name or code. • Automatic correction - Appends a number to the duplicate name or 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

Web Service and Web Operation Checks (PDM)

PowerDesigner provides default model checks to verify the validity of Web services and Web operations.

Check	Description and Correction
Name/Code contains terms not in glossary	[if glossary enabled] Names and codes must contain only approved terms drawn from the glossary. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - None.
Name/Code contains synonyms of glossary terms	[if glossary enabled] Names and codes must not contain synonyms of glossary terms. • Manual correction - Modify the name or code to contain only glossary terms. • Automatic correction - Replaces synonyms with their associated glossary terms.
Name/Code uniqueness	Object names must be unique in the namespace. • Manual correction - Modify the duplicate name or code. • Automatic correction - Appends a number to the duplicate name or code.
Code maximum length	Web service and Web operation code lengths are limited by the maximum length specified in the DBMS definition (Maxlen entry, in the **Objects** > **Web Service** and **Web Operation** categories). • Manual correction: Modify the code length to meet this requirement • Automatic correction: Truncates the code length to the maximum length specified in the DBMS definition

CHAPTER 6

Generating and Reverse-Engineering Databases

PowerDesigner provides full support for round trip generation and reverse-engineering between a PDM and a database.

Writing SQL Code in PowerDesigner

The objects that you create in your model display the SQL code that will be generated for them on the **Preview** tab of their property sheets. Certain objects provide editors on other tabs to allow you to modify the SQL statements.

For example, you may need to write SQL code in order to:

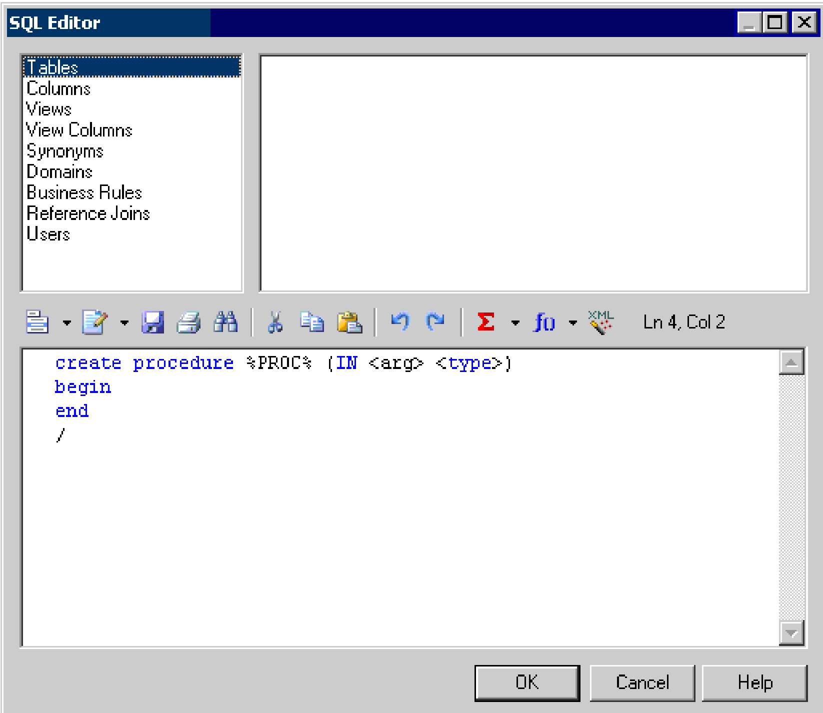
- Specify a view query (see *View Queries* on page 115).
- Write a procedure or trigger (see *Triggers (PDM)* on page 119).
- Define a computed column (see *Creating a Computed Column* on page 102).
- Insert scripts at the beginning and/or end of database or table creation (see *Customizing Scripts* on page 298).

The following tools are available in the PowerDesigner SQL editors:

Tool	Description
	**Add Trigger Item From Model / DBMS** - [triggers and trigger templates only] Opens a dialog box to list trigger template items defined in the model or in the DBMS definition file for insertion in the trigger definition (see *Trigger Template Items* on page 128).
	**Operators / Functions** - List logical operators and group, number, string, date, conversion and other functions for insertion in the SQL code. Operators and functions are DBMS-specific and these lists are populated from entries in the `Script\Sql\Keywords` category (see *Customizing and Extending PowerDesigner > DBMS Definition Files > Script/Sql Category*).
	**Macros / Variables** - List PDM macros and variables for insertion in the SQL code (see *Customizing and Extending PowerDesigner > DBMS Definition Files > PDM Variables and Macros*). You can also use formatting variables to force values to lower-case or upper-case or to truncate the length of values characters.
	**Edit with SQL Editor** - Opens the full SQL Editor dialog which gives access to model objects for insertion in the SQL code.

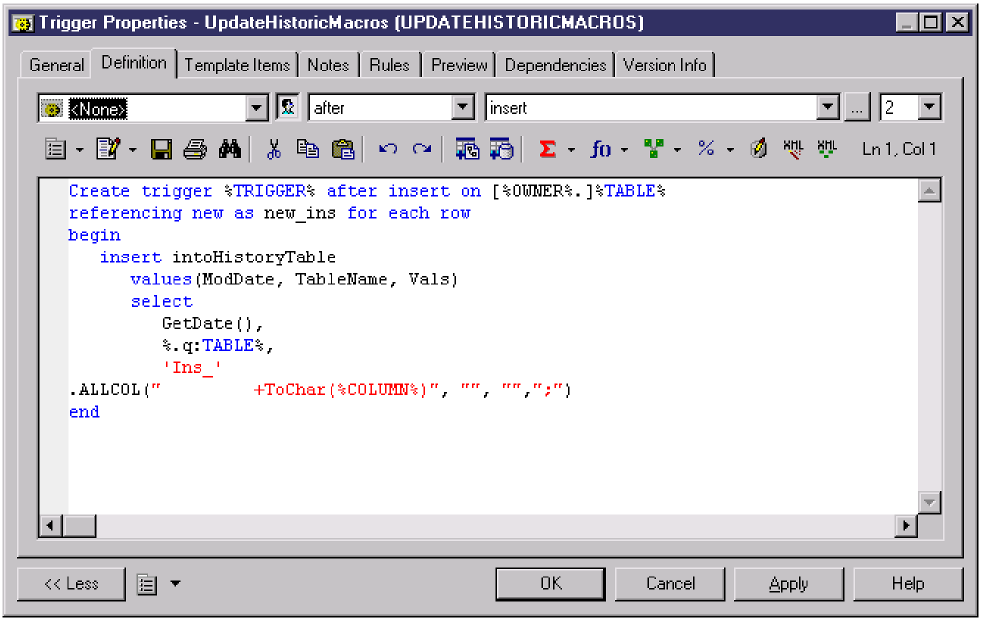
Tool	Description
	SQL/XML Wizard - Opens the SQL/XML Wizard to build a SQL/XML query from a table or a view for insertion in the SQL code (see *Creating SQL/XML Queries with the Wizard* on page 132).
	Insert SQL/XML Macro - Opens a dialog box to select a global element from an XML model open in the workspace (and which must have the SQL/XML extension file attached) for insertion in the SQL code.

In addition to these tools, the pop-out SQL Editor lists PDM object types in the upper left pane and the available objects of the selected type in the upper right pane. Double-click an object to insert it into your code in the lower pane:

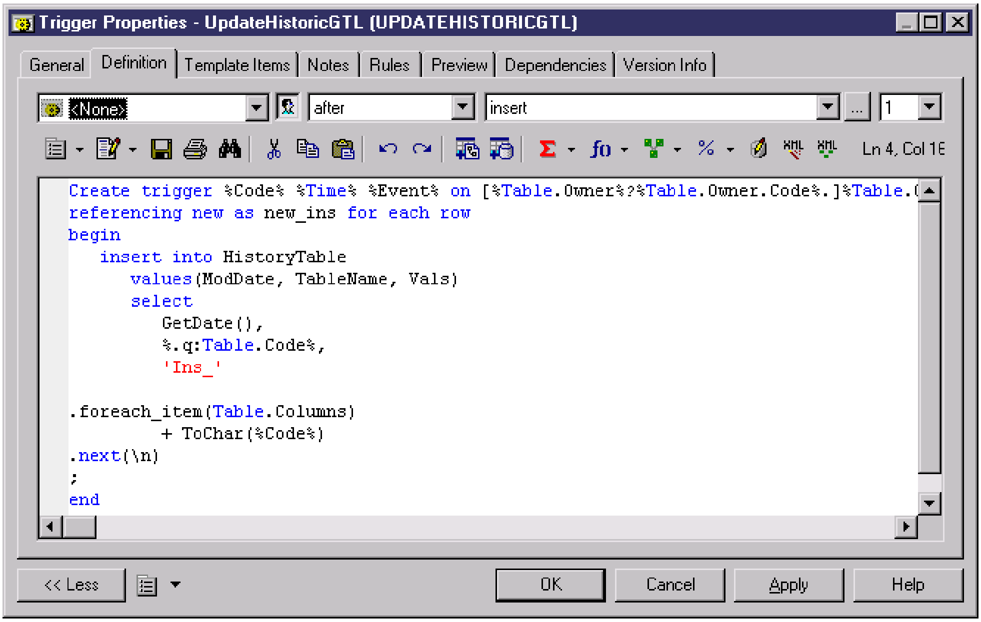


You can use the PowerDesigner Generation Template Language (GTL) and PDM variables and macros to reference objects and object properties and iterate over collections when writing SQL statements. While you can perform many tasks using the PDM variables and macros, GTL is more powerful, as it allows you to access any information about any object in the model.

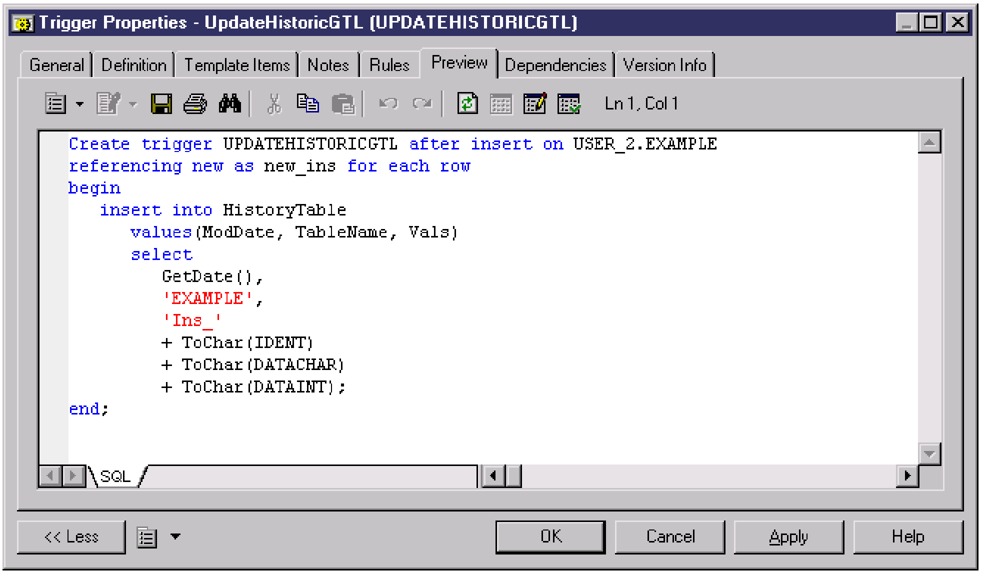
In the following example, a trigger is written using the PDM variables and macros and attached to the Example table, to write the contents of any insertion to HistoryTable.



The same trigger can be written using GTL:



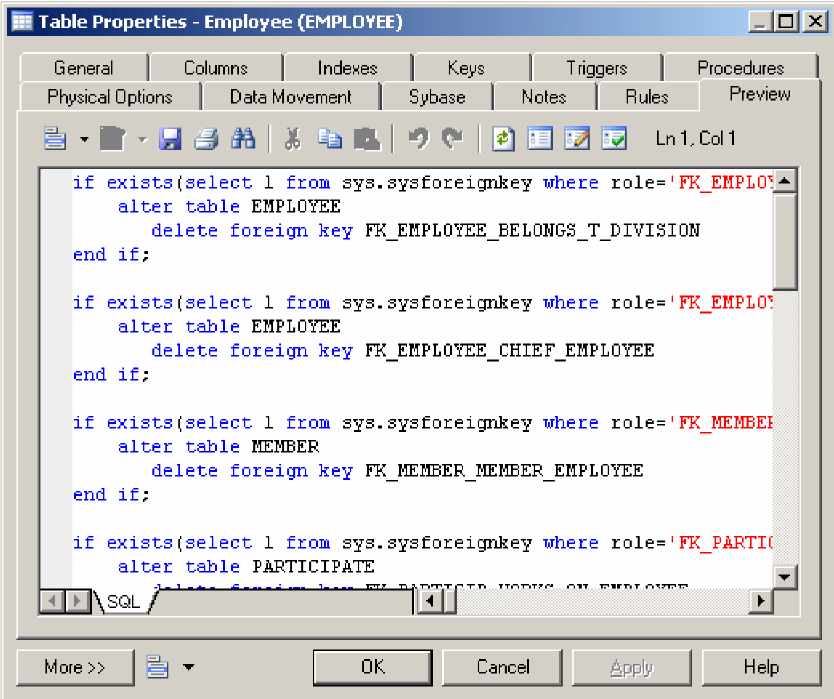
In each case, the trigger code to be generated is the same, and can be viewed by clicking the **Preview** tab:



For detailed information about working with GTL, see *Customizing and Extending PowerDesigner > Customizing Generation with GTL* . For lists of the available variables and macros, see *Customizing and Extending PowerDesigner > DBMS Definition Files > PDM Variables and Macros*.

Previewing SQL Statements

Click the **Preview** tab in the property sheet of the model, packages, tables, and various other model objects in order to view the code that will be generated for it.



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

The following tools are available on the **Preview** tab toolbar:

Tools	Description
	Editor Menu [Shift+F11] - Contains the following commands: • **New [Ctrl+N]** - Reinitializes the field by removing all the existing content. • **Open... [Ctrl+O]** - Replaces the content of the field with the content of the selected file. • **Insert... [Ctrl+I]** - Inserts the content of the selected file at the cursor. • **Save [Ctrl+S]** - Saves the content of the field to the specified file. • **Save As...** - Saves the content of the field to a new file. • **Select All [Ctrl+A]** - Selects all the content of the field. • **Find... [Ctrl+F]** - Opens a dialog to search for text in the field. • **Find Next... [F3]** - Finds the next occurence of the searched for text. • **Find Previous... [Shift+F3]** - Finds the previous occurence of the searched for text. • **Replace... [Ctrl+H]** - Opens a dialog to replace text in the field. • **Go To Line... [Ctrl+G]** - Opens a dialog to go to the specified line. • **Toggle Bookmark [Ctrl+F2]** Inserts or removes a bookmark (a blue box) at the cursor position. Note that bookmarks are not printable and are lost if you refresh the tab, or use the **Show Generation Options** tool • **Next Bookmark [F2]** - Jumps to the next bookmark. • **Previous Bookmark [Shift+F2]** - Jumps to the previous bookmark.
	Edit With [Ctrl+E] - Opens the previewed code in an external editor. Click the down arrow to select a particular editor or **Choose Program** to specify a new editor. Editors specified here are added to the list of editors available at **Tools > General Options > Editors**.
	Save [Ctrl+S] - Saves the content of the field to the specified file.
	Print [Ctrl+P] - Prints the content of the field.
	Find [Ctrl+F] - Opens a dialog to search for text.
	Cut [Ctrl+X], **Copy [Ctrl+C]**, and **Paste [Ctrl+V]** - Perform the standard clipboard actions.
	Undo [Ctrl+Z] and **Redo [Ctrl+Y]** - Move backward or forward through edits.
	Refresh [F5] - Refreshes the Preview tab. You can debug the GTL templates that generate the code shown in the Preview tab. To do so, open the target or extension resource file, select the **Enable Trace Mode** option, and click **OK** to return to your model. You may need to click the **Refresh** tool to display the templates.

Tools	Description
	**Select Generation Targets [Ctrl+F6]** - Lets you select additional generation targets (defined in extensions), and adds a sub-tab for each selected target. For information about generation targets, see *Customizing and Extending PowerDesigner > Extension Files > Generated Files (Profile) > Generating Your Files in a Standard or Extended Generation*.
	**Show Generation Options [Ctrl+W]** - Opens the Generation Options dialog, allowing you to modify the generation options and to see the impact on the code.
	**Ignore Generation Options [Ctrl+D]** - Ignores changes to the generation options made with the **Show Generation Options** tool.

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

Connecting to a Database

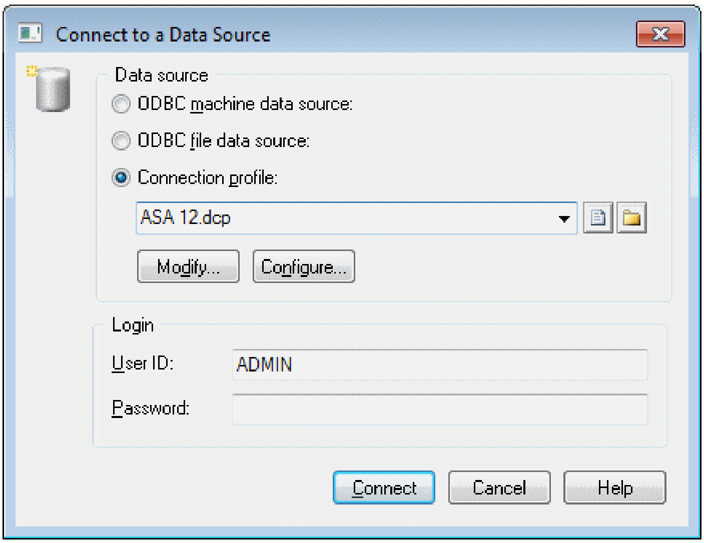
PowerDesigner provides various methods for connecting to your database.

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

For detailed information about creating, configuring, and using connection profiles, see *Core Features Guide > Modeling with PowerDesigner > Getting Started with PowerDesigner > Connecting to a Database*.

1. Select **Database** > **Connect** to open the Connect to a Data Source window:



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.

   You stay connected until you disconnect or terminate the shell session.

   You can display information about your connection at any time by selecting **Database > Connection Information**. The amount of information available depends on your DBMS and your connection profile.

   To disconnect from a database, select **Database > Disconnect**.

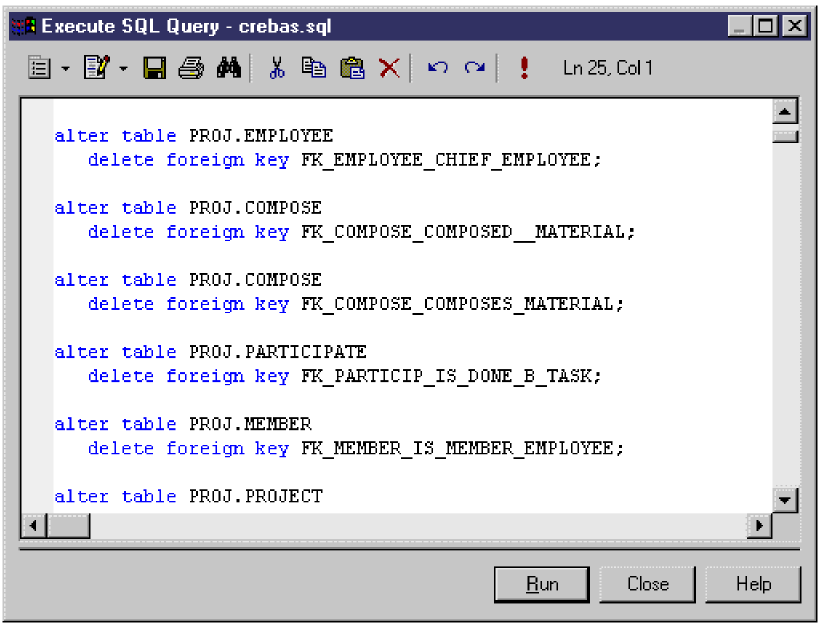
Executing SQL Queries

You can send SQL queries to a database and display the results.

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.



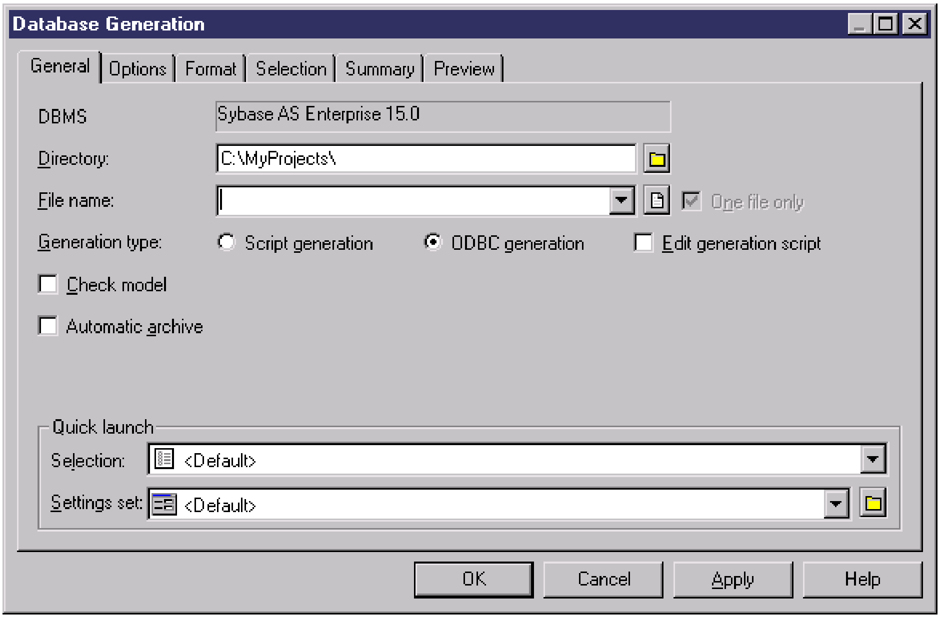
The query results are displayed in the Results window.

Generating a Database from a PDM

PowerDesigner can generate sophisticated SQL scripts as files or for direct execution via a live database connection.

Note: To generate to a SAP HANA® database, use the HANA wizard (see *Exporting Objects to the HANA Repository* on page 504).

1. Select **Database** > **Generate Database** to open the Database Generation dialog box.



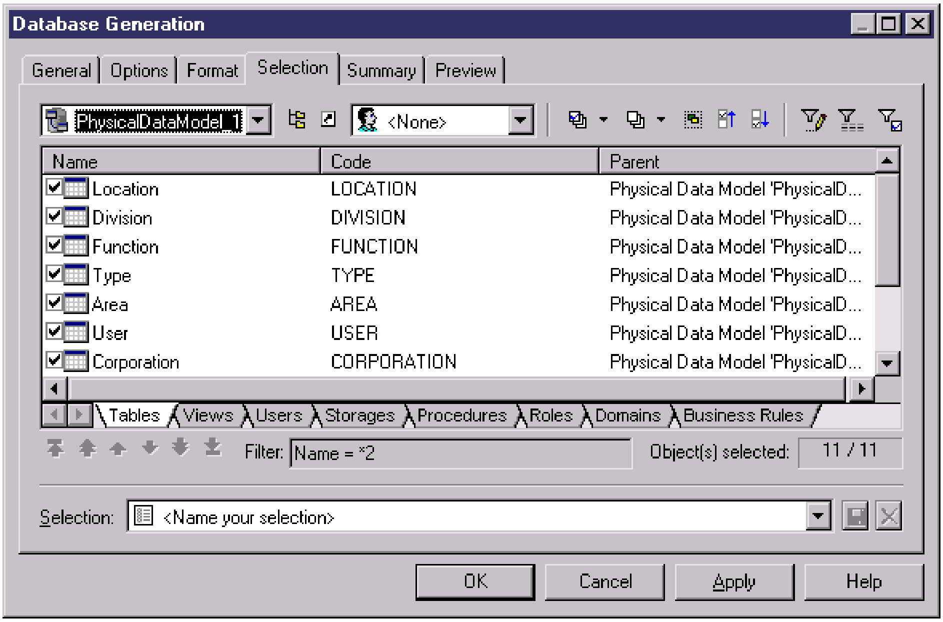
Note: To load a pre-configured selection or settings set (see *Quick Launch Selection and Settings Sets* on page 297), select it in the appropriate list in the **Quick launch** group box.

2. Enter a destination **Directory** and **File Name** for the script file.
3. Specify the type of generation (script or live database connection) to perform:
   - Script generation - generate a script to be executed on a DBMS at a later time. Optionally select **One file only** to create the generation script as a single file. By default, a separate script file is created for each table.
   - Direct generation – generate a script and execute it on a live database connection. Optionally select **Edit generation script** to open the script in an editor for review or editing before execution.

4. [optional] Select the following options as appropriate:

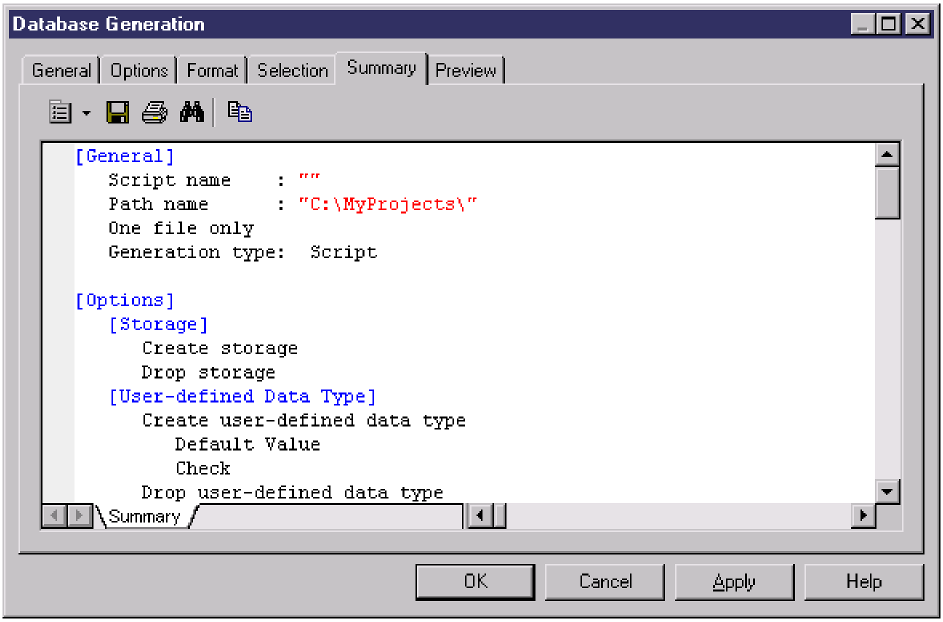
Option	Description
Check model	Specifies that a model check is performed before script generation.
Automatic archive	Creates an archive version of the PDM after generation to use to determine changes during your next database modification (see *Archive PDMs* on page 322).

5. [optional] To change the default generation options, click the **Options** tab (see *Database Generation Dialog Options Tab* on page 293).
6. [optional] To change the format of your script, click the **Format** tab (see *Database Generation Dialog Format Tab* on page 296).
7. [optional] To control which database objects will be generated, click the **Selection** tab:

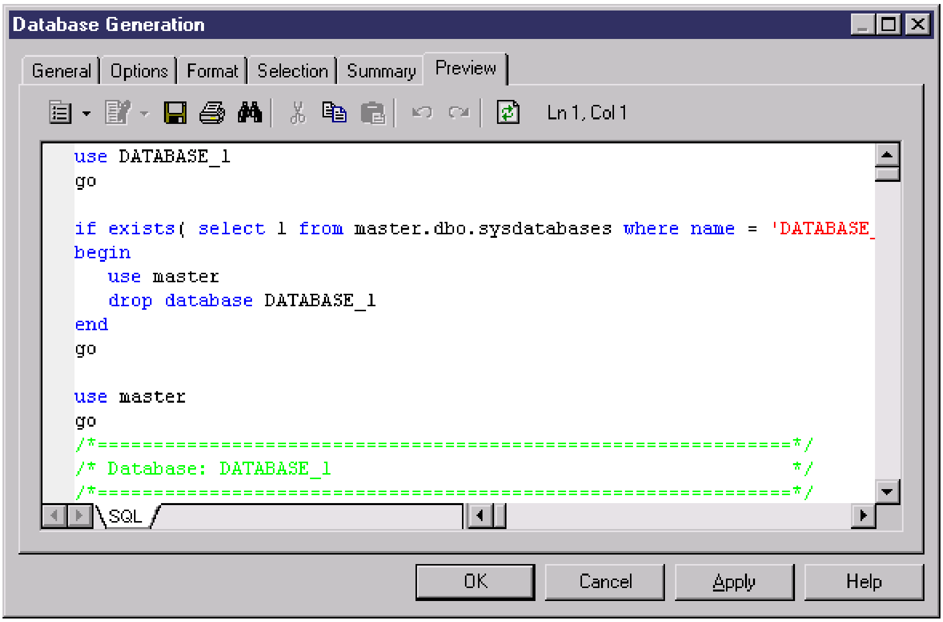


You can save your selection via the Selection bar at the bottom of the tab (see *Quick Launch Selection and Settings Sets* on page 297).

8. [optional] Click the **Summary** tab to view the summary of your settings and selections. The summary is not editable, but you can search, save, print, and copy its contents.



9. [optional] Click the **Preview** tab to preview the SQL script to be generated. The script is not editable, but you can search, save, print, and copy its contents.



10. Click **OK** to begin the generation.

If you are generating a database script, the **Output** window shows the progress of the generation process, and gives instructions for running the script. When generation is complete, the Generated Files dialog opens listing the paths to the generated script files. Click **Edit** to open the script in a text editor or **Close** to close the Result box.

Note: For information about the additional steps required to generate for MS Access, see *Generating a Microsoft Access Database* on page 570).

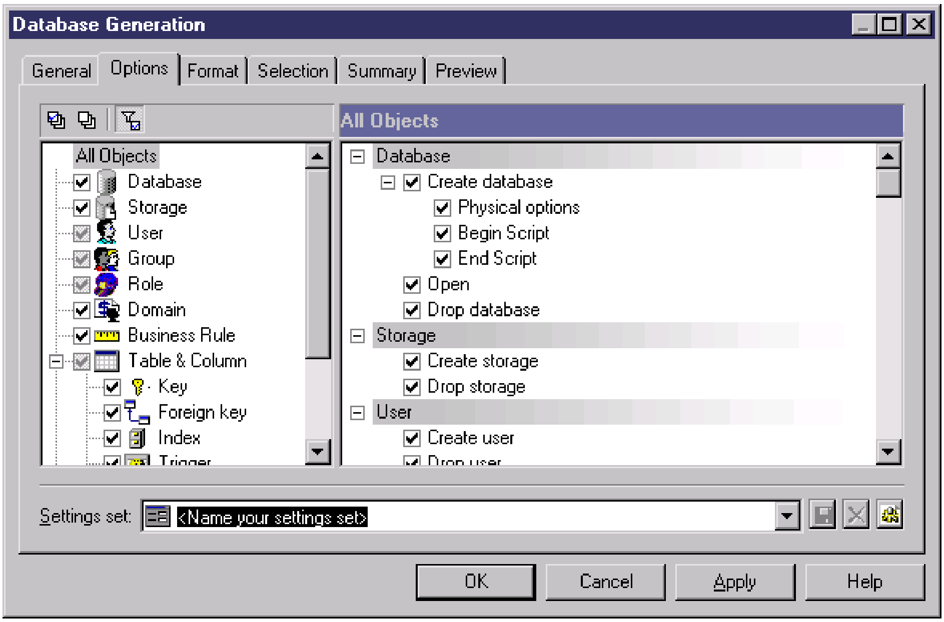
If you are generating a database directly, and are not currently connected to a database, a dialog box asks you to identify a data source and connection parameters (see *Connecting to a Database* on page 288).

Note: Advanced users can further customize database generation by, for example, changing the order in which objects are generated, adding scripts to run before or after generation, and generating additional objects. For information about these and other advanced topics, see *Customizing and Extending PowerDesigner > DBMS Definition Files*.

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.



Depending on the objects present in your model, some or all of the following options will be available.

Parameter	Description
Create <object>	Generates the object. When generating primary, alternate, or foreign keys or indexes, you can choose between: • Inside Table – keys or indexes are generated during table creation • Outside - keys or indexes are generated with a separate SQL command, generally using an ALTER command after the creation of the table The generation of keys or indexes outside the table is possible only if the Create entry exists in the Pkey, Key, Reference, and/or Index categories of your DBMS.
Drop <object>	Deletes an existing object, before recreating it. Note that when generating defaults, 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 *Customizing and Extending PowerDesigner > DBMS Definition Files*.
Begin script	Inserts a customized script before creation of the object.
End script	Inserts a customized script after creation of the object.
Physical options	Generates physical options for the object.
Comment	Generates a comment for the object.
Privilege	[users, groups, and roles] Generates privileges for the user, group, or role.
Permission	Generates the permission statement for a given user during creation of the object.
Check	[domains, tables, and columns] Generates check parameters and validation rules for domains, tables, and columns. For table and columns, if this option is 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 only if the AddTableCheck entry exists in the Table category of your DBMS.
Open database	[databases] Opens the database.
Close database	[databases] Closes the database.
Default value	[domains and columns] Specifies a default value for the domain or column.
Install JAVA class	[abstract data types] Installs a Java class, which is stored on a server.
Remove JAVA class	[abstract data types] Deletes an existing Java class, before installing a new Java class.

Parameter	Description
User-defined type	[columns] Generates a user-defined data type for the column.
Decl. Integrity	[foreign keys] Generates declarative referential integrity for references specified to be Declarative in their property sheets. 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
Index Filter	[indexes] 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 Filter	[triggers] You can specify the creation of triggers: • For insert • For update • For delete
Synonym Filter	[synonyms] 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

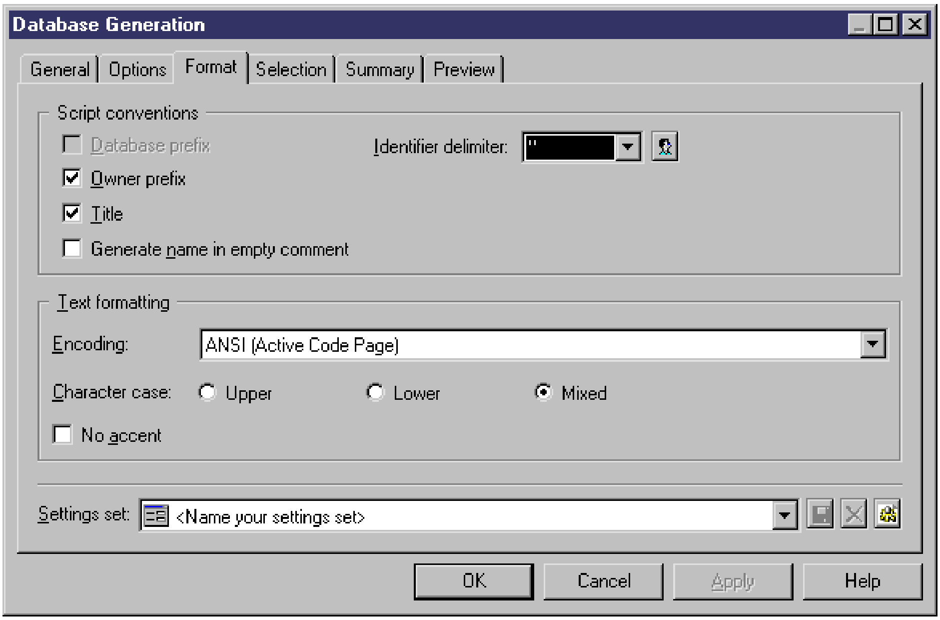
Parameter	Description

Force column list	[views] 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`

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 297.

Database Generation Dialog Format Tab

The options on the Format tab allow you to control the format of database generation scripts.



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 297.

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`).
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

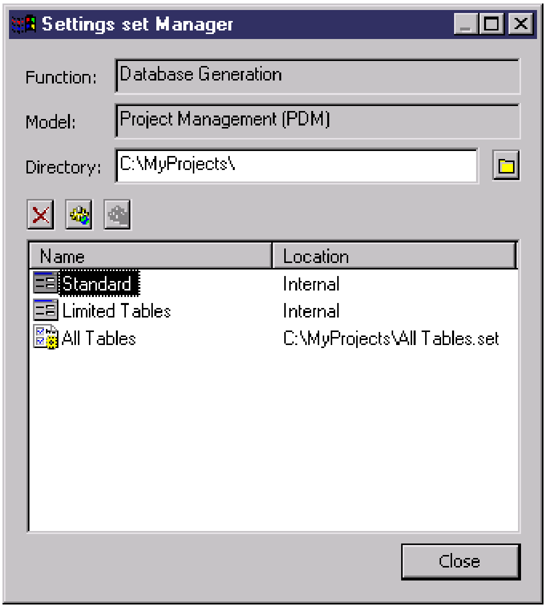
Quick Launch Selection and Settings Sets

The Quick Launch groupbox at the bottom of the Database Generation dialog **General** tab allows you to load pre-configured selections and settings sets for use when generating the database.

- Selection - the ensemble of selections of database objects made on the **Selection** tab. To save a selection, enter a name in the Selection bar at the bottom of the **Selection** tab and then click the **Save** tool. The selection is saved as part of the model file.
- Settings Set - the ensemble of generation options (see *Database Generation Dialog Options Tab* on page 293) and format options (see *Database Generation Dialog Format Tab* on page 296)

  To save a settings set, enter a name in the Settings set bar at the bottom of the **Options** or **Format** tab and then click the **Save** tool, specify whether you want to save the settings set inside the model or as an external file, and click **OK**.

  To review your settings sets, click the **Settings Set Manager** tool to the right of the field on the **Options** or **Format** tab:



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: 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.

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.

The **Script** tab provides tools to help edit scripts:

Tool	Description	Keyboard shortcut
	Editor context menu	**Shift+F11**
	Edit With. Opens the script in your preferred editor (see *Core Features Guide > Modeling with PowerDesigner > Customizing Your Modeling Environment > General Options > Text Editors*)	**Ctrl+E**

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.

Inserting Begin and End Scripts for Database Creation

In a database creation script, you can insert a Begin script before the command that creates the database and an End script after the last command in the database creation script.

You can use the following variables in these scripts:

1. Select **Model > Model Properties** or right-click the diagram background and select **Properties**.
2. Click the **Create** tool to the right of the **Database** field and click **Yes** in the confirmation dialog to open the database property sheet.
3. Enter a name and code for the database and then click the **Script** tab.
4. Enter a Begin and/or End script as necessary on the appropriate subtab. 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

Variable	Description
%AUTHOR%	Author of the current model

For a complete list of the variables available and how to format them, see *Customizing and Extending PowerDesigner > DBMS Definition Files > PDM Variables and Macros*.

5. Click **OK** to close the database property sheet and return to your model.

Inserting Begin and End Scripts for Table and Tablespace Creation

For each table and tablespace, you can insert a Begin script after the table title and an End script after the table or tablespace creation command.

These scripts can appear in database creation scripts and database modification scripts.

1. Open the property sheet of the tablespace and click the **Script** tab.
2. Enter a Begin and/or End script as necessary on the appropriate subtab. 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
%COLNLIST%	*Column list*
%DBMSNAME%	Code of the DBMS for the target database
%OWNER%	Table owner
%OWNERPREFIX%	Owner prefix of table owner
%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

For a complete list of the variables available and how to format them, see *Customizing and Extending PowerDesigner > DBMS Definition Files > PDM Variables and Macros*.

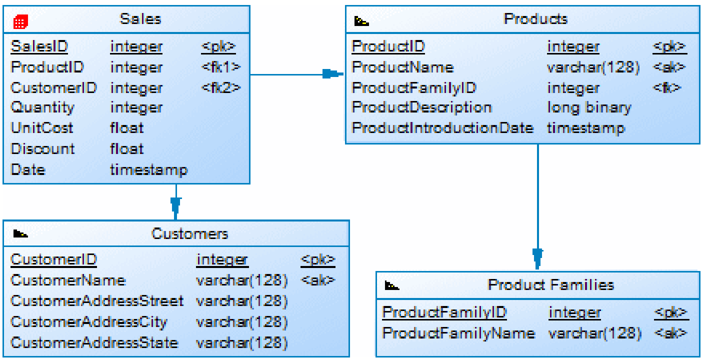
3. Click **OK** to close the database property sheet and return to your model.

Generating a BusinessObjects Universe

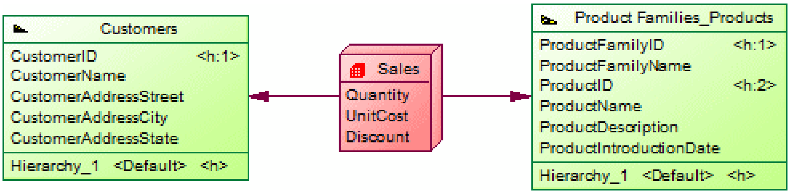
PowerDesigner can generate a SAP® BusinessObjects™ universe from your PDM for editing in the BusinessObjects Universe Design or Information Design tools, or for direct consumption by the Web Intelligence rich client. Generating a universe from your PDM gives you access to table, view, and column names and comments and more reliable cardinality information than if you create a universe directly from your database.

Note: To use this feature, you must have SAP® BusinessObjects™ SBOP BI Platform Clients 4.0 SP04 Patch 3 (v14.0.4.819) or higher installed on your workstation. On Windows Vista or Windows 7 machines, if PowerDesigner fails to recognize a valid BusinessObjects installation, it may be necessary separately to launch the Universe Design tool one time with administrator privileges to enable the BusinessObjects SDK.

1. [optional] Optimize your PDM for generation of a universe in the following ways:
   - Specify auto-incrementing primary keys (see *Creating Primary Keys* on page 105) together with one or more human-readable alternative keys (see *Creating Alternate Keys* on page 107) to uniquely identify dimension rows.
   - Identify fact and dimension tables either manually or by retrieval (see *Identifying Fact and Dimension Tables* on page 217) and review the choices that PowerDesigner has made:



   - [optional] To completely control the format of your multidimensional objects, retrieve facts and dimensions in a multidimensional diagram (see *Generating Cubes* on page 217), and edit them as necessary:



2. Select **Tools** > **SAP BusinessObjects** > **Generate BusinessObjects Universe**.
3. [optional] Click the **Connect** button to connect to the BusinessObjects CMS.
4. Select a data connection to allow BusinessObjects to connect to your database. If you have not connected to the CMS, you can use an existing local connection from the BusinessObjects connection list; otherwise choose from the list of secured connections. You can, alternatively, click the **Create** button to create a new connection with the BusinessObjects New Connection wizard.

   Note: The user that you specify in this connection must have sufficient privileges and permissions to read all of the database objects contained in the PDM you are creating your universe from.

5. Click **Next** to select the objects to generate from your model. PowerDesigner will propose objects to generate as follows:
   - If facts and dimensions are present in your model, the facts are proposed for generation.
   - If no facts are present, but one or more tables have been specified as fact tables, then these will be proposed for generation.
   - If no facts or fact tables are present, then PowerDesigner will evaluate all the tables in the model and propose those which could serve as fact tables for generation.

   Note: By default, tables that have no links to other tables are excluded from the list. Select the **Include isolated tables** option to add them for selection.

6. [when facts are not present] Click **Next** to select any appropriate generation options:

Option	Description
Expand fact date columns as time dimensions	[selected by default] Creates a time dimension with the standard Year, Quarter, and Month attributes for each date column in each fact table.
Add Large Object dimension details	[deselected by default] Specifies that dimension attributes are created for columns of type blob (which commonly contain images, audio, or other binary data). If this option is deselected, these columns will still appear in the data foundation, but will not be visible in the business layer.

Option	Description

Use primary keys as dimension identifiers	Specifies whether dimension identifiers can or must be generated from the primary keys of their source tables. You can choose from the following settings:
	• Force - Dimension identifiers must be generated from the primary keys of their source tables.
	• Allow - [default] PowerDesigner chooses the first available columns in the following list to use as dimension identifiers:
	  • The first alternative key (all associated columns concatenated).
	  • The first unique index not identified as a primary key.
	  • The first column with a string data type, including primary keys with a string data type.
	  • The first non-key column.
	  • The first key column.
	• Disallow - Same as allow, but dimension identifiers cannot be generated from primary keys even if they have a string data type (for example a primary key containing a GUID).

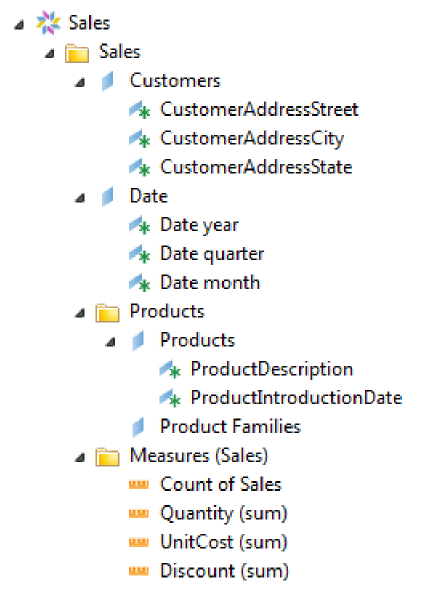
7. Click **Next** to review your choices and then click **Finish** to begin the universe generation.

When the universe is generated, you can:

- Open it in the Universe Design tool or import it into the Information Design tool (select **File > Convert UNV Universe**) for further editing.

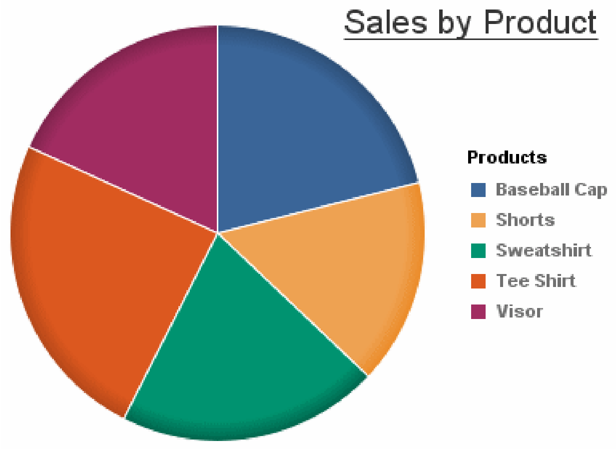
PowerDesigner generates a universe comprising a connection, data foundation, and business layer. The business layer has one folder for each fact containing:

- A dimension for each dimension associated with the fact in PowerDesigner. Dimension series, such as the Product dimension in our example are grouped within their own subfolder. Dimensions with more than one attribute list each attribute beneath them.
- A measure for every numeric column in the fact.



After the import is complete, open the data foundation view and select **Actions > Refresh Structure** to obtain access to the richer selection of data types available in the Information Design tool.

- [if you are connected to the CMS] Import it into the CMS for editing or consumption.
- Consume it directly in the Web Intelligence rich client:



Generating Test Data to a Database

PowerDesigner can generate sample data to your database tables to verify performance or to help in estimating the amount of memory that the database will require. You can generate test data for some or all of the tables in your PDM to an empty or existing database.

Note: 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

1. [optional] Specify one or more test data profiles to define the range of data to be generated or to draw data from a file or other database (see *Populating Columns with Test Data* on page 98). If you do not define profiles, PowerDesigner will generate random data that is appropriate to each data type.

   Note: The format in which date and time data is generated with or without profiles can be controlled by DBMS items in the `Script/Sql/Format` category (see *Customizing and Extending PowerDesigner > DBMS Definition Files > Script/Sql Category*.

2. Select **Database > Generate Test Data** to open the Test Data Generation dialog.
3. On the **General** tab, enter or select the appropriate parameters:

Option	Description
Directory	Specifies the directory in which the file will be saved.
File name	Specifies the name of the test data file to generate. Select the **One file only** checkbox to specify that a single file should be generated.
Generation type	Specifies how the test data will be generated: • Script generation • Direct generation – to a live database connection. **Note:** As triggers are not needed in this context and can block insertions and considerably increase the time required to generate the database, we recommend that you do not implement triggers or remove them from your test database. • Data file – as a set of values in a file.
Commit mode	Specifies when the data will be committed: • Auto - during script generation • At end - after script generation • By packet - at defined intervals during script generation
Data file format	Specifies the format when generating a data file: • 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 to generate for tables. This number can be overrridden for individual tables on the **Number of Rows** tab.
Default number/ character/ date profile	Specifies the default test data profiles (see *Populating Columns with Test Data* on page 98) to use to generate data. We recommend that you create test data profiles to accurately model your data and associate them with each of your columns and domains as appropriate, but if you have not done so, then these default profiles are used.

4. [optional] Click the **Number of Rows** tab to change the number of rows to be generated for each table.

   By default, PowerDesigner generates the number of rows that is specified in the **Number** property in the table property sheet (see *Table Properties* on page 76) or, if no number is specified, the default number specified on the **General** tab of this Test Data Generation dialog.

5. [optional] Click the **Format** tab and modify the script formatting options as appropriate:

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.

6. [optional] Click the **Selection** tab and select which tables you want to generate test data for. By default all tables are selected.
7. Click **OK** to start the generation.

   If you are generating test data to a live database connection, then the Connect to a Data Source dialog box opens. Select a data source, and then click **Connect**. 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.

   A message in the Output window indicates that the test data generation is completed.

Estimating Database Size

You can estimate the size of a database for all or some of the tables and other objects in your model. You can estimate the initial size of the database or project its growth over a number of years.

The estimate is based on the following elements:

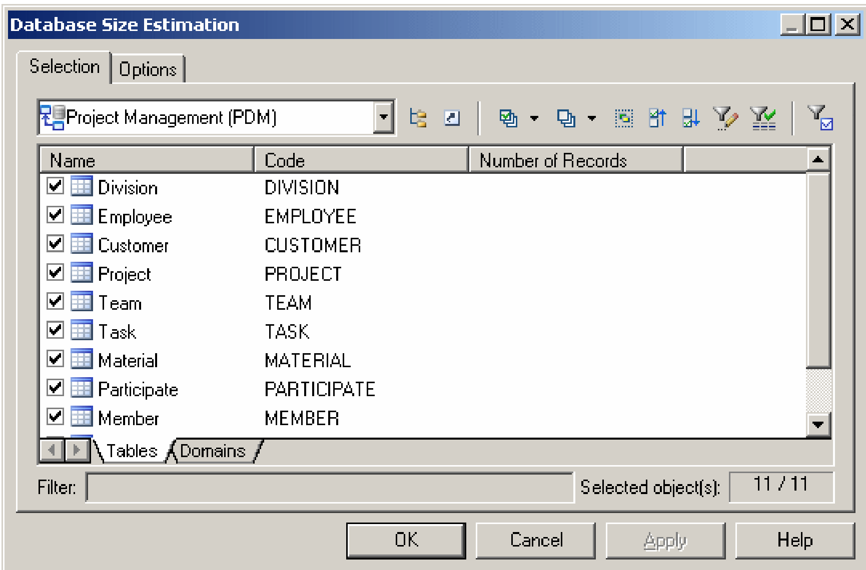
- Estimated number of records in tables - Specify the number of rows (and their annual projected growth rate) in a table in the **Number** and **Row growth rate** fields on the **General** tab of its property sheet (see *Table Properties* on page 76).
- Table columns and their sizes - Specify the average size for variable length columns in the **Average length** field on the **Detail** tab of its property sheet (see *Column Properties* on page 91). If you do not specify an average length for variable length columns, then the maximum length is used. It is particularly important to specify an average length for strings or long binary data types, as a Binary Long OBject (BLOB), such as a picture, can represent the largest portion of the space actually taken by a table.

Note: To specify values for multiple tables or columns, select **Model > Tables** or **Model > Columns**. If you do not see the appropriate property column, then add it using the **Customize Columns and Filter** tool.

- Indexes in the model - including primary, alternate, and foreign key indexes (if supported) and database-specific indexes such as IQ join indexes.
- Tablespaces in the model - the size of a tablespace is estimated as a total of all the tables and all the indexes in the tablespace.
- DBMS and its storage options.

Note: The default estimation algorithms can be overridden in the DBMS definition file (see *Customizing and Extending PowerDesigner > DBMS Definition Files > Profile Category > Modifying the Estimate Database Size Mechanism*).

1. Select **Database > Estimate Database Size** to open the Database Size Estimation dialog.
2. Select the tables for which you want to estimate the size.



3. [optional] Click the **Options** tab and specify the number of years of growth that you want to include in your estimate. By default, only the initial size of the database is calculated, without allowing for any growth.
4. Click **OK** to begin the estimation.

   Size estimates are generated to both the Result List and Output windows. The **Database Size** tab of the Result List provides a list of objects which can be double-clicked to open their property sheets, while the **Database Size** tab of the Output window prints a textual list of objects with sizes and a total for the database:

```
Estimate of the size of the Database "Project Management"...
```

```
   Number      Estimated size   Object
----------   ------------------   ----------------------------
 1,000,000          136,224 KB   Table 'Customer'
                                   Index 'Primary' (4,880 KB)
     1,000               48 KB   Table 'Division'
    10,000              696 KB   Table 'Employee'
                                   Index 'Primary' (48 KB)
     5,000              312 KB   Table 'Material'
    10,000               96 KB   Table 'Member'
    10,000              392 KB   Table 'Participate'
    10,000              640 KB   Table 'Project'
                                   Index 'Primary' (48 KB)
    10,000              464 KB   Table 'Task'
     1,000               80 KB   Table 'Team'
    10,000               96 KB   Table 'Used'
----------   ------------------   ----------------------------
                    139,048 KB   Total estimated space

Database size estimation completed.
The number of records was not defined for 1 table(s).
```

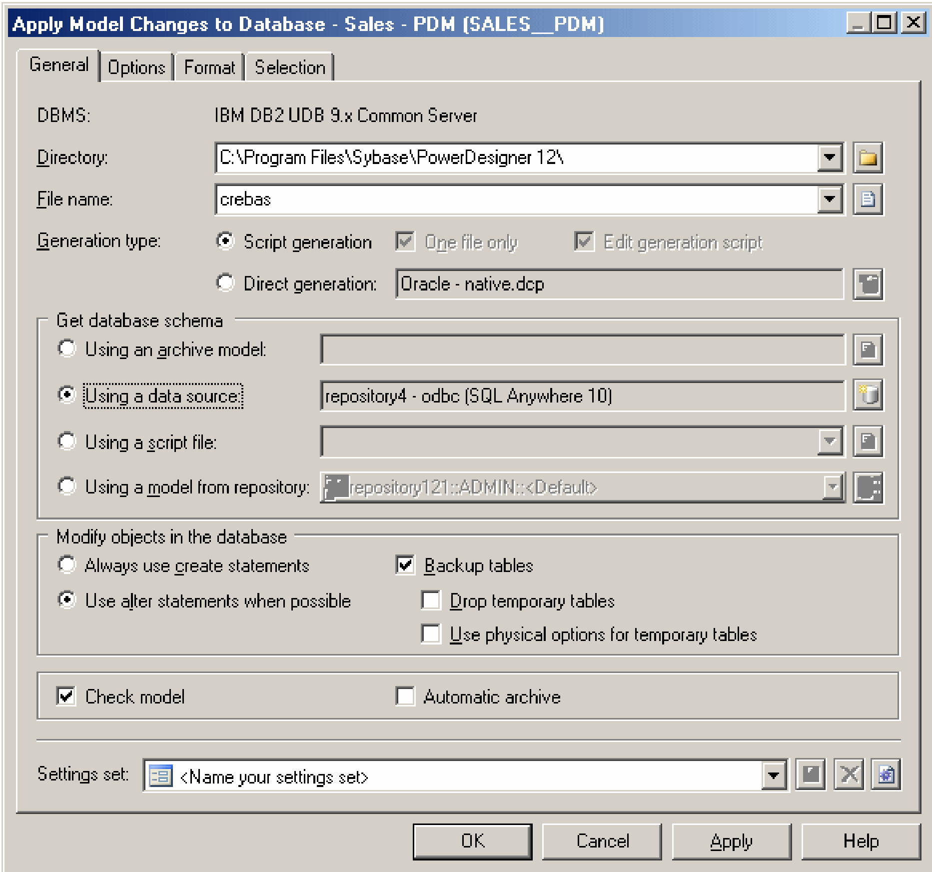
A warning is given if any tables in the model do not have a number of a records defined.

Modifying a Database

You can modify an existing database schema by to reflect changes in your model. 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.

Note: To update a HANA database, use the HANA wizard (see *Exporting Objects to the HANA Repository* on page 504).

1. Select **Database > Apply Model Changes to Database**



Note: To load a pre-configured settings set (see *Quick Launch Selection and Settings Sets* on page 297), select it in the list at the bottom of the dialog.

2. Enter a destination **Directory** and **File Name** for the script file.
3. Specify the type of generation (script or live database connection) to perform:
   - Script generation - generate a script to be executed on a DBMS at a later time. Optionally select **One file only** to create the generation script as a single file. By default, a separate script file is created for each table.
   - Direct generation – generate a script and execute it on a live database connection. Optionally select **Edit generation script** to open the script in an editor for review or editing before execution.
4. Specify how PowerDesigner will determine the changes to apply. You can choose to compare your model against:
   - **Archive model** – Click the button to the right to browse to the archived model (see *Archive PDMs* on page 322).

- **Data source** – Click the button to the right to connect to your data source.
- **Script file** – Select a script from the list or click the button to the right to browse to the script.
- **Model from repository** – Select a model from the list and optionally click the button to the right to browse to a version of it.

5. [optional] Select the following options as appropriate:

Option	Description
Always use create statements/ Use alter statements when possible	Specify whether create statements should always be used to modify database tables, or whether alter statements should be used where possible.
Backup tables	Specifies that any existing table will be copied to a temporary backup during the modification, and then restored to the updated tables. If this option is not selected, then all existing data will be erased. If you select this option then you can also specify to : • **Drop temporary tables** - Specifies that the temporary backup tables are removed after script execution. • **Use physical options for temporary tables** - 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 to use to determine changes during your next database modification (see *Archive PDMs* on page 322).

6. [optional] To change the default generation options, click the **Options** tab (see *Database Generation Dialog Options Tab* on page 293).

7. [optional] To change the format of your script, click the **Format** tab (see *Database Generation Dialog Format Tab* on page 296).

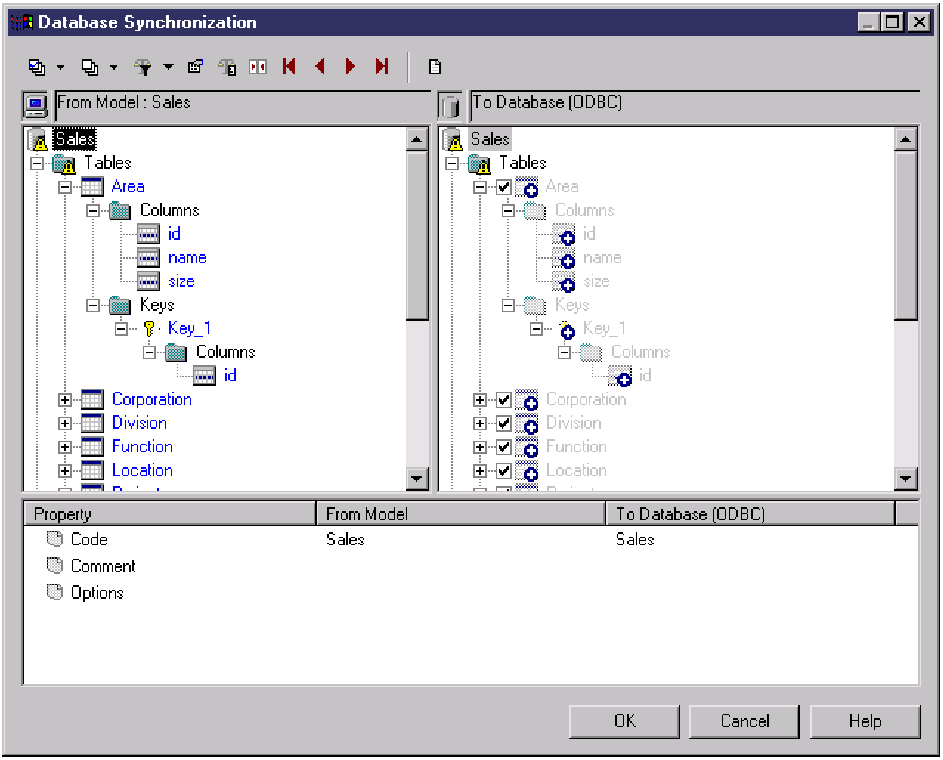
8. [optional] To control which database objects will be modified, click the **Selection** tab.

You can save your selection via the Selection bar at the bottom of the tab (see *Quick Launch Selection and Settings Sets* on page 297).

9. Click **OK** to begin the update. 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, and then click **OK** to continue.

For more information about comparing and merging models, see *Core Features Guide > Modeling with PowerDesigner > Comparing and Merging Models*.



- If you are generating a script, a result box opens listing the file path of the generated file. To open the script in a text editor, select the file in the result box and click the **Edit** 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.

Displaying Data from a Database

You can connect to a database and display the data that corresponds to a PDM table, view, or reference.

1. Right-click a table, view, or reference and select **View Data**.

   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.

2. A Query Results windows list all the database records corresponding to the selected table, view, or reference.

Reverse Engineering a Database into a PDM

Reverse engineering is the process of generating a PDM (or certain PDM objects) from an existing database schema. You can reverse engineer into a new PDM or an existing PDM from one or more script files or from a live database. The database user that PowerDesigner uses to connect must have public access to the database.

Note: To reverse-engineer from a HANA database, use the HANA wizard (see *Importing Objects from the HANA Repository* on page 506).

Reverse Engineering from Scripts

PowerDesigner can reverse engineer a PDM for one or more SQL script files. The script will normally be the script used to generate the database but can also include other scripts.

Warning! In general, only statements that create objects are reverse-engineered and `alter` statements, except for those that add columns to a table, are not supported.

1. To reverse engineer a script into an existing PDM, select **Database > Update Model from Database**.

   or

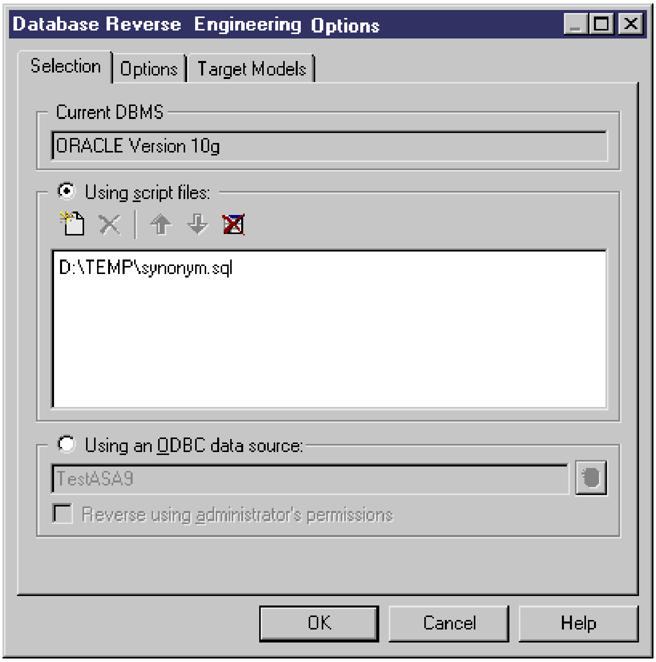
   To reverse engineer a script and create a new PDM, select **File > Reverse Engineer > Database** to open the New Physical Data Model dialog. Specify a model name, choose a DBMS from the list, and then click **OK**.

   or

   When working with the PowerDesigner Eclipse plug-in, select any SQL file in the Navigator, right-click it and select **Reverse Engineer from SQL File**. You are given the option to reverse into an existing or new PDM.

   Note: To reverse-engineer an MS Access database, you must first prepare a `.dat` file (see *Reverse Engineering a Microsoft Access Database* on page 570).

2. When the Database Reverse Engineering Options dialog opens, select **Using script files**:



The following tools are provided to help with script selection:

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.

Note: You can add as many script files as necessary to the list. If you are reversing more than one script file, the order in which the files are reversed must respect any 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.

3. [optional] Click the **Options** tab to specify any reverse engineering options (see *Reverse Engineering Options Tab* on page 316).

Note: References and primary keys are not rebuilt by default. To enable rebuilding, select the appropriate options on the **Options** tab.

4. [optional] Click the **Target Models** tab to specify any external shortcuts (see *Reverse Engineering Target Models Tab* on page 319).
5. Click **OK** to begin reverse engineering.

   If you are reverse engineering to an existing PDM, then the Merge Models dialog box opens to allow you to control the merging of the new objects into your PDM (see *Core Features Guide > Modeling with PowerDesigner > Comparing and Merging Models*). When the process is complete, a confirmation message is given in the Output window.

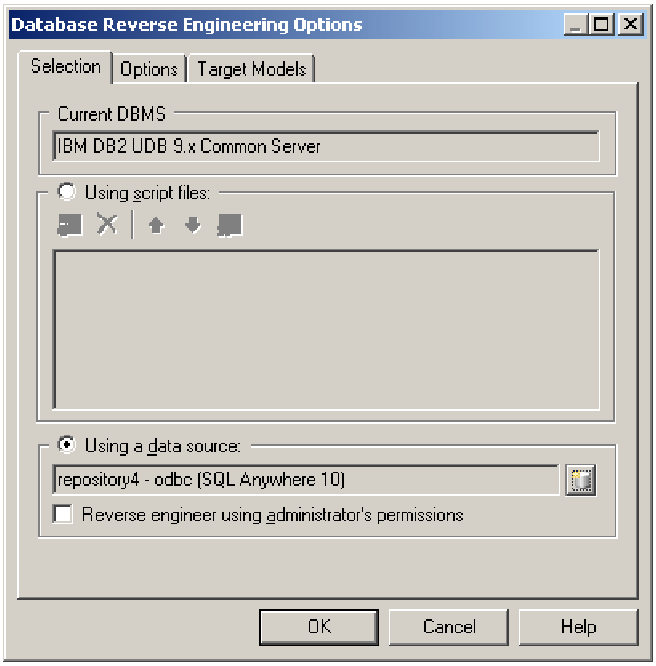
Reverse Engineering from a Live Database

PowerDesigner can reverse engineer a PDM from a live database connection. You must specify a 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 administrator.

1. To reverse engineer from a live database connection into an existing PDM, select **Database > Update Model from Database**.

   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. Specify a model name, choose a DBMS from the list, and then click **OK**.
2. In the Database Reverse Engineering Options dialog, select **Using a data source**:

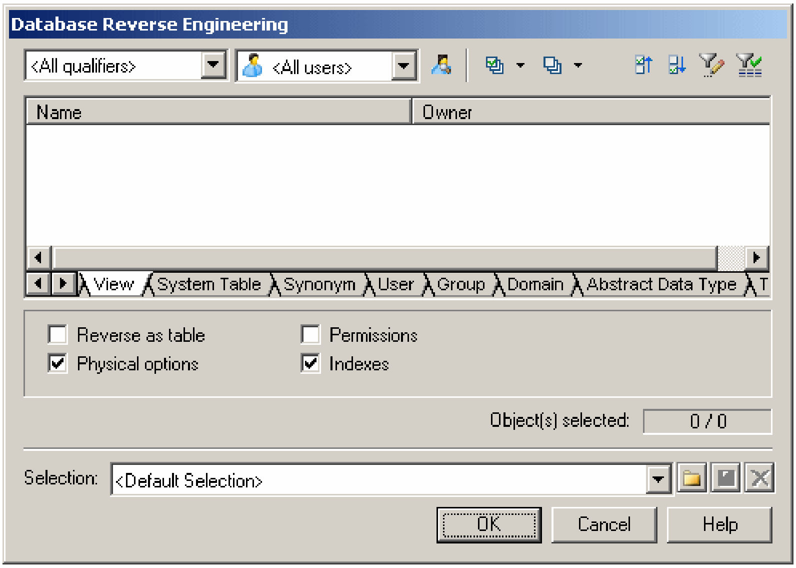


Note: A data source might be predefined, or you can enter the name of an existing data source. In both cases, if you need to specify additional connection parameters, a database connection dialog box opens when you click **OK**. Enter the necessary parameters and click **Connect** to open the Database Reverse Engineering dialog.

3. Select your data source. You can either accept the selected data source (if one is present) or click the **Connect to a Data Source** tool to select or define one. For detailed information about working with data sources, see *Core Features Guide > Modeling with PowerDesigner > Getting Started with PowerDesigner > Connecting to a Database*.
4. [optional] To reverse engineer tables reserved to the database administrator, select **Reverse using administrator's permissions**.
5. [optional] Click the **Options** tab to specify any reverse engineering options (see *Reverse Engineering Options Tab* on page 316).

Note: References and primary keys are not rebuilt by default. To enable rebuilding, select the appropriate options on the **Options** tab.

6. [optional] Click the **Target Models** tab to specify any external shortcuts (see *Reverse Engineering Target Models Tab* on page 319).
7. Click **OK** to open the Database Reverse Engineering dialog, which allows you to specify the objects to reverse engineer (see *Database Reverse Engineering Selection Window* on page 318). Only tables and triggers are selected by default.



8. Click **OK** to begin reverse engineering.

   If you are reverse engineering to an existing PDM, then the Merge Models dialog box opens to allow you to control the merging of the new objects into your PDM (see *Core*

Features Guide > Modeling with PowerDesigner > Comparing and Merging Models). When the process is complete, a confirmation message is given in the Output window.

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:

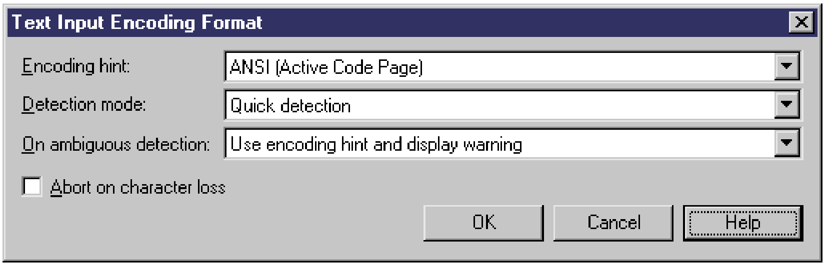
Option	Description
Automatically rebuild references when no reference is reversed	Rebuilds references (see *Rebuilding References* on page 181) when no references are reverse engineered. A reference is created between each column belonging to a primary key and a column, with identical name and data type, 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 (see *Rebuilding Primary Keys* on page 106) 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.
Create symbols	Creates a symbol for each reversed object in the diagram. If this option is not selected, reversed objects are visible only in the browser. Where there are a large number of objects with complex interactions, PowerDesigner may create synonyms of objects to improve diagram readability. For example, if a table has a large number of references, PowerDesigner may create a synonym of the table in another location in the diagram to reduce the length required for references.
Apply code to name conversion to reversed objects	Applies the code to name conversion script specified in the model options (see *Core Features Guide > Modeling with PowerDesigner > Objects > Naming Conventions*).
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 317).
Block/ Command terminator	Specify the end of block and end of command characters for the reversed script. By default, these value are defined in the DBMS definition file at `Script\SQL\Syntax`, and modifications made here are saved in the Registry for reuse in other models. To restore the DBMS value, click the **Restore from DBMS** tool.

Option	Description
Case sensitive database	Specifies that the database is case sensitive and enables the case sensitive option in the model.

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.

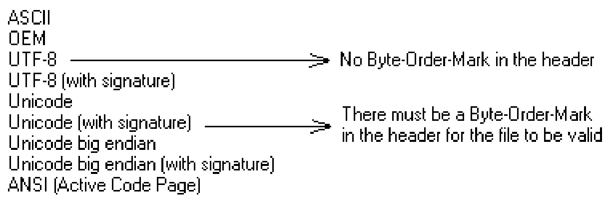


The Text Input Encoding Format dialog box includes the following options:

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

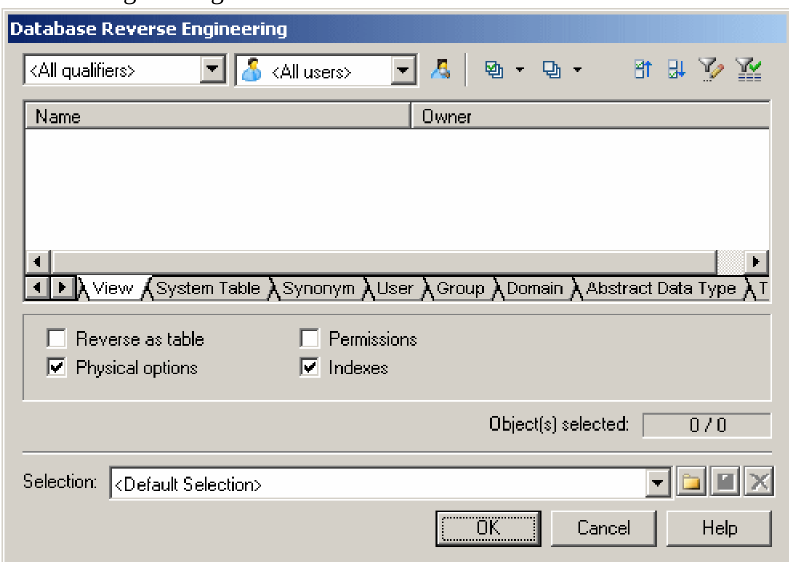
Option	Description
On ambiguous detection	Specifies what action should be taken in case of ambiguity. The following settings are available: • Use encoding hint and display warning - the encoding hint format is used and a warning message is displayed. • Use encoding hint - the encoding hint format is 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:



Database Reverse Engineering Selection Window

When you reverse engineer a database from a live database connection, you can choose to populate your PDM with a subset of the available objects by selecting them in the Database Reverse Engineering Selection window.

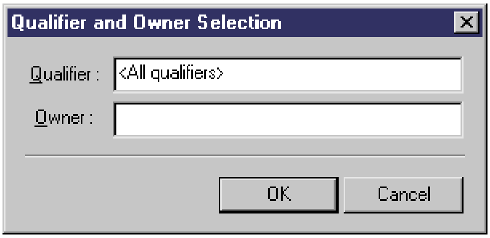


Click the subtabs to view the different types of objects. Certain object types have attributes, or options, that appear below the object lists. Options that are not available for the selected object type or DBMS are grayed. When you select tables containing triggers on the **Table** tab, the triggers are selected on the **Trigger** tab.

You can restrict database objects to reverse engineer in the top area of the window by selecting to filter by:

- Qualifier - such as a database or a partition that contains one or more tables. 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. To reverse engineer objects from multiple owners, select *All users*. Only users that have creation rights are reverse engineered.

Note: If the selected qualifier contains a large number of table owners, it may be faster to click the **Select Qualifier and Owner** tool and enter a qualifier and/or owner in the dialog box, as opening the Owner list may take a very long time.



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.

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. The Target Models tab displays the list of detected target models containing target objects for shortcuts in the current model to reverse.

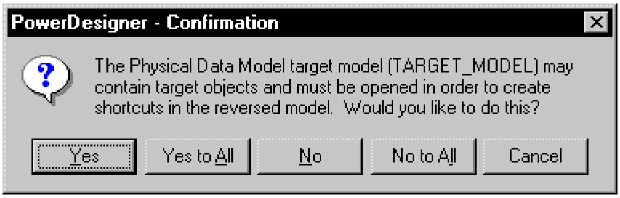
This tab is always visible, even if the model does not contain shortcuts, so that you can add target models and create shortcuts instead of duplicating objects.

The following tools are available on this tab:

Tool	Description
	Change Target Model - Displays a standard Open dialog box to let you select another file as target model

Tool	Description
	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:



If you are reverse engineering from a:

- 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.
- Live data source - External shortcuts are created for all selected objects that already exist in another target model. These existing objects are deselected by default in the **Selection** tab of the Reverse Engineering dialog box, except for target objects corresponding to shortcuts already existing in the reversed model.

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`, and if the total number of listed items is:

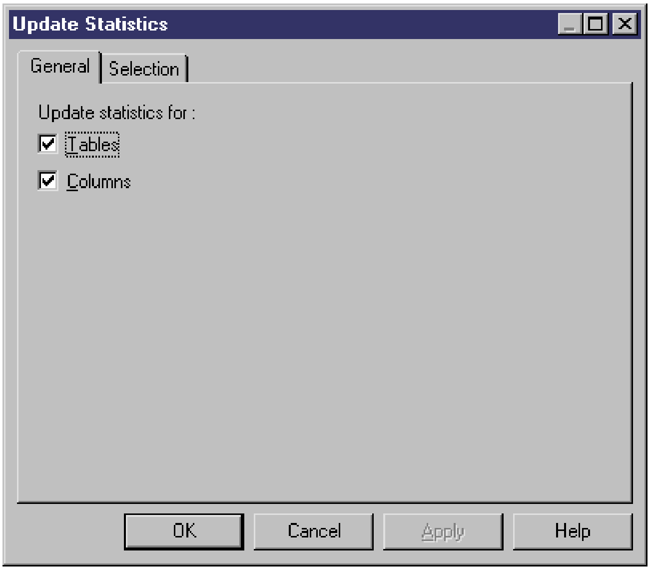
- lower than `RevOdbcMinCount` - then a global reverse query is executed.
- higher than `RevOdbcMinCount` - then the process uses key `RevOdbcMinPerct`, and if the percentage of reversed items is :
  - lower than `RevOdbcMinPerct` - then the same query is executed for each object.
  - higher than `RevOdbcMinPerct` - then a global query is executed.

Reverse Engineering Database Statistics

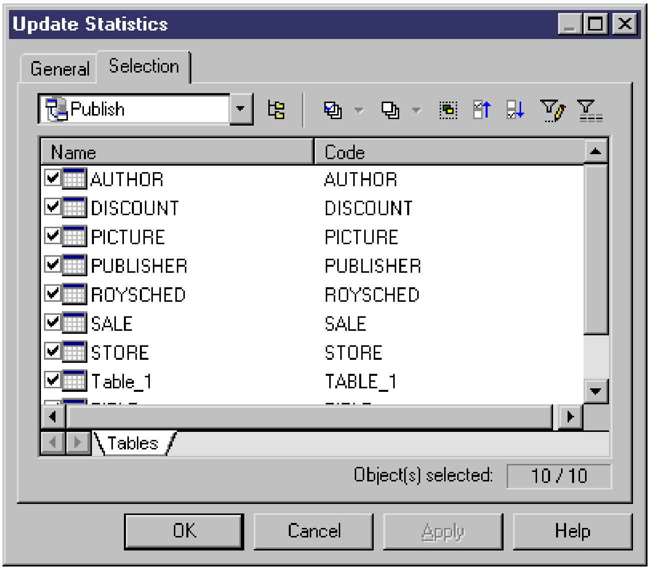
You can reverse engineer statistics for an existing database, such as the number of distinct or null values in a column or the average length of a character field. These can provide helpful information when optimizing a design.

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 314), or update them at any other time, using the dedicated Update Statistics window.

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):



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:



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.

Archive PDMs

Archive PDMs provide a snapshot of the structure of your database at a point in time to allow you to determine model changes since that time when updating your database. When comparing your model directly with a database or script (and not with an archive PDM), some differences (particularly around renamed objects) can be lost, leading to more drop/creates in place of alter statements.

Archives are created by default when you generate or update your database (using the **Automatic Archive** option), and can be created manually at any time by clicking **File > Save As**, and selecting Archived PDM (bin) or Archived PDM (xml) in the Save As Type list.

CHAPTER 7 Generating Other Models from a Data Model

You can generate various types of PowerDesigner 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

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.

   Note: For detailed information about the options available on the various tabs of the Generation window, see *Core Features Guide > Linking and Synchronizing Models > Generating Models and Model Objects*.
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 objects to generate.
7. Click **OK** to begin generation.

Generating Other Models from a CDM

You can generate CDM objects to other model objects.

CDM	OOM	PDM
Entity	Class - All entities with the **Generate** property selected are generated as persistent classes with the `Generate table` persistence mode. If an entity's **Generate** property is not selected, the generated class has the `Migrate columns` persistence mode.	Table - If the entity is involved in an inheritance, the inheritance **Generation Mode** setting (see *Inheritance Properties* on page 69) affects whether parents and children are generated.
Entity attribute	Attribute	Table column **Note:** 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 to the first three letters of the original entity name followed by the code of the attribute.
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

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.

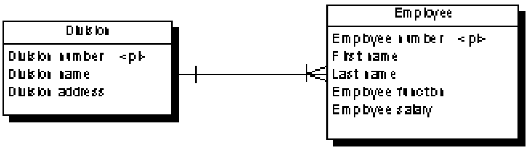
- A *foreign key* is a column or columns that depend on and migrate from a primary key column in another table.
- An *alternate key* is a column or columns whose values uniquely identify a row in a table, and is not a primary key.

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:

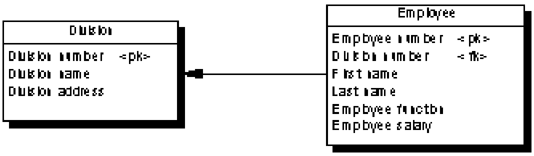
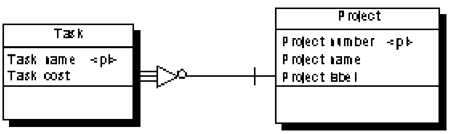


Table	Primary key	Foreign key
Division	Division number	—
Employee	Employee number	Division number

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:

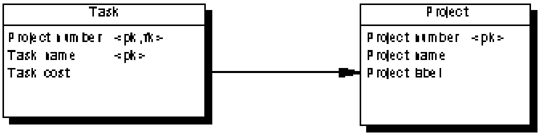
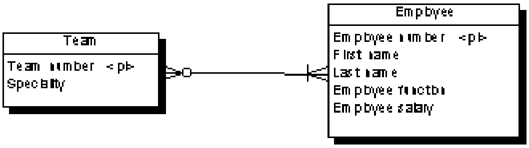


Table	Primary key	Foreign key
Project	Project number	—
Task	Project number/Task number	Project number

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:

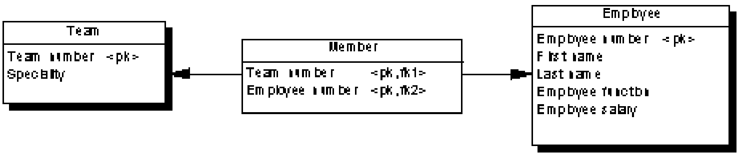


Table	Primary key	Foreign key
Team	Team number	—
Employee	Employee number	—
Member	Team number/Employee number	Team number/Employee number

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 Other Models from an LDM

You can generate LDM objects to other model objects.

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

You can generate PDM objects to other model objects.

PDM	CDM	LDM	OOM	XSM
Domain	Domain	Domain	Domain	Simple Type
Table	Entity	Entity	Class	Element
Table column	Entity attribute	Entity attribute	Attribute	Attribute or element
Primary key	Primary identifier	Primary identifier	Primary identifier	-
Alternate key	Identifier	Identifier	Identifier	-
Foreign key	-	-	-	Keyref constraint
Stored-Procedures	-	-	Operation	-
View	-	-	-	Element
View column	-	-	-	Attribute
Index	-	-	-	Unique
Abstract data type	-	-	-	Complex type

PDM	CDM	LDM	OOM	XSM
Reference	Relationship	Relationship	Association	-

Note: 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 *Core Features Guide > Modeling with PowerDesigner > Objects > Naming Conventions*.

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 the table column attached to a sequence is translated to a serial data type in the CDM with the format `NO%n`, where `%n` is the length of the data type (see *Sequences (PDM)* on page 169).

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

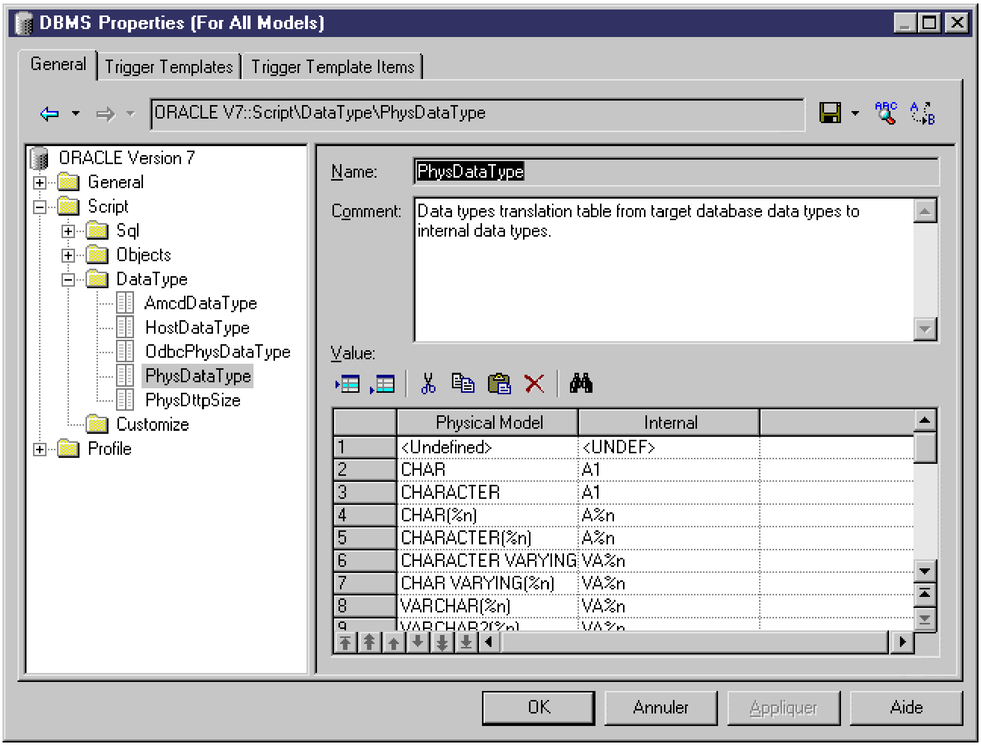
Note: If the code of the generated OOM objects does not correspond to the target language naming conventions, you can define a code naming convention script to convert object names into codes. For more information, see *Core Features Guide > Modeling with PowerDesigner > Objects > Naming Conventions*.

Customizing Data Type Mappings

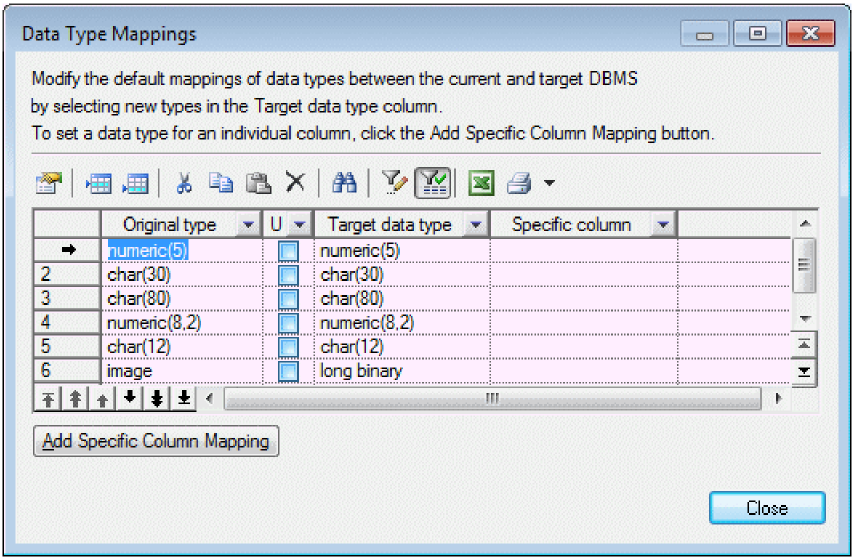
When generating another PDM from your PDM, PowerDesigner maps the existing column datatypes to appropriate data types in the new model. If the standard mappings are not sufficient for you, you can use the `Enhance Data Type Mapping` extension to specify alternative mappings, including on a column-by-column basis.

To review the conversions that PowerDesigner makes by default between the data types of a database or other modeling target and its standard conceptual types (which are also used in the

CDM), select **Tools > Resources > *Type***, select the appropriate file in the list and click the **Properties** tool. Expand the **Script > DataType** (for DBMSs) or **Settings > DataType** (for other resource files), and review each of the entries (which are described in their **Comment** field):



1. Select **Tools > Generate Physical Data Model**, enter the appropriate generation options (see *Core Features Guide > Linking and Synchronizing Models > Generating Models and Model Objects*).
2. On the **Detail** tab, click the **Enable Transformations** button to display the **Extensions** tab, and select the `Enhance Data Type Mapping` extension.
3. Click **OK** to start the generation. The Data Type Mappings dialog appears, with the existing data types present in the model listed in the **Original type** column, and those that PowerDesigner proposes in the new DBMS in the **Target data type** column:



4. You can change data type mappings in two ways:
   - To change the mapping for all columns of a certain data type, select the desired new data type from the list in the **Target data type** column.
   - To change the mapping for one column only, click the **Add Specific Column Mapping** button, select the column from the tree, click **OK**, choose the new data type for the column, and click **OK** to add this mapping to the list.
5. When you have modified all the necessary data types, click **Close** and the generation will continue, using your custom mappings where appropriate.

Note: You can also customize data type mappings when changing the DBMS of your model with the **Database > Change Current DBMS** command. To do so, you must first attach the `Enhance Data Type Mapping` extension, by selecting **Model > Extensions**, clicking the **Attach an Extension** tool, select the extension, and clicking **OK** to attach it to your model.

For more information about data types, see *Customizing and Extending PowerDesigner > DBMS Definition Files > Script/Data Type Category* and *Customizing and Extending PowerDesigner > Object, Process, and XML Language Definition Files > Settings Category: Object Language*.

Customizing XSM Generation for Individual Objects

When generating an XSM from a PDM or OOM, you can specify global generation options to generate tables/classes as elements with or without complex types and columns/attributes as elements or attributes. You can override these options for individual objects by attaching the `PDM XML Generation` or `OOM XML Generation` extension to your source model and selecting from their XML generation options.

Note: The extension provides new property sheet tabs for setting generation options for individual objects, but you can also set these options with or without the extension by selecting **Model > *objects*** to open the appropriate object list, clicking the **Customize Columns and Filter** tool, and selecting to display the `XML Generation Mode` column.

For example, if you want to generate the majority of your table columns to an XSM as XML attributes, but want to generate certain columns as elements, you should:

- Modify the XML generation options for those columns that you want to generate as elements.
- Select to generate columns as attributes on the Model Generation Options **Detail** tab.

1. Select **Model > Extensions** to open the List of Extensions, and click the **Attach an Extension** tool.
2. On the **General Purpose** tab, select `PDM XML Generation` or `OOM XML Generation` and click **OK** to attach the extension to your model and **OK** to close the List of Extensions.

   These extension files enable the display of the **XML** tab in all table and column or class and attribute property sheets.
3. Open the property sheet of the table, column, class, or attribute whose generation you want to customize, and click the **XML** tab.
4. Use the radio buttons to specify how you want to generate the object in an XSM.
   - For tables and classes, you can specify to generate them as:
     - Elements - the table/class is generated as an untyped element directly linked to its columns/attributes generated as attributes or sub-elements.
     - Elements with complex types - the table/class is generated as an element typed by a complex type, generated in parallel, to contain the columns/attributes.
     - Default - generation of the table/class is controlled by the option selected in the **XML Generation** group box on the Model Generation Options **Detail** tab.
   - For tables, you can additionally specify to generate keys as:
     - Key - [default] The primary
       key columns are generated and also KEY and KEYREF wherever the table is ref erenced.
     - ID attribute - The primary key columns are not generated and an ID attribute, `id`, is generated to replace them.
       Wherever the table is referenced, an IDREF attribute is generated to reference th e appropriate element. If the reference role name is assigned, this attribute is given this
       name. Otherwise, the referenced table name is used and the standard renaming m echanism is enforced.
     - Key and ID attribute - In many cases the primary
       key columns have significant data and you may want to generate them, as well as an ID attribute.

In this case an ID attribute is generated for the element and IDREF is used syste matically for any reference to the table:

The following rules apply to the generation of keys:

- If a Table generates an ID, all its child tables will generate an ID attribute.
- If a Table generates Key columns, all its child tables will generate Key columns.
- If a child table is flagged to generate PK only, ID Attribute will be automatically generated.
- If a table generates ID attribute, No Key nor KeyRef will be generated, and ALL references will generate IDREF attribute.. (Even if the table generates also Key Columns)
- If a table generates ID attribute ONLY, All Foreign Key Columns referencing its Key columns will be systematically removed and replaced by an IDREF attribute

- For columns and attributes, you can specify to generate them as:
  - Elements - [default] the column/attribute is generated as an sub-element of its table/class element or complex type.
  - Attributes - the column/attribute is generated as an attribute of its table/class element or complex type.
  - Default - generation of the column/attribute is controlled by the option selected in the **XML Generation** group box on the Model Generation Options **Detail** tab.

5. Modify the XML generation options for any other objects that you want to generate in a different manner.
6. Select **Tools > Generate XML Model**, ensure that the appropriate options are set in the **XML Generation** group box on the Model Generation Options **Detail** tab, and start your generation.

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.

CHAPTER 8 Migrating from ERwin to PowerDesigner

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.

PowerDesigner supports the import of the following ERwin v3.x and higher model files, though v4.x or higher files are recommended, as they contain more metadata:

- 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 **Only save minimum amount of information** in the ERwin Save as XML File dialog box.

Note: Before importing, we recommend that you 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.

An ERwin logical model can be imported into either a PowerDesigner conceptual or logical model (CDM or LDM), while an ERwin Physical Model is imported into a PowerDesigner physical data model (PDM).

PowerDesigner cannot import the following ERwin objects:

- ERwin triggers and stored procedures (not directly possible, but see the process in *Post-Import* on page 338)
- ERwin reports
- ER1 files
- ERwin data sources
- ERwin target clients

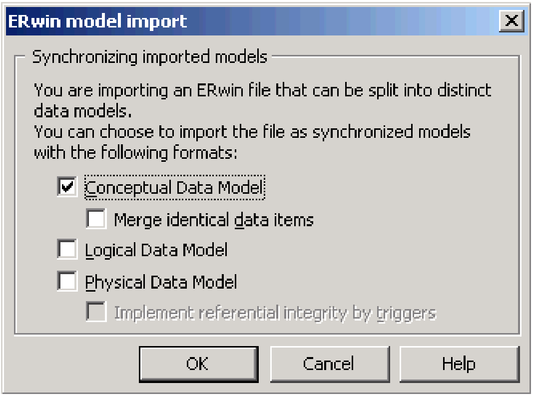
While PowerDesigner can import all your object display preferences and will retain color and font information, it does not support multiple colors for columns in a single table. The default column color will be used during the import.

Importing Individual ERwin Files

PowerDesigner provides a wizard to help you import individual ERwin files.

1. Select **File > Import > ERwin File**.
2. 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:

   The options available depend on the type of ERwin model that you are importing. PowerDesigner supports data modeling at the conceptual, logical, and physical levels. 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.
   - 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 model 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.

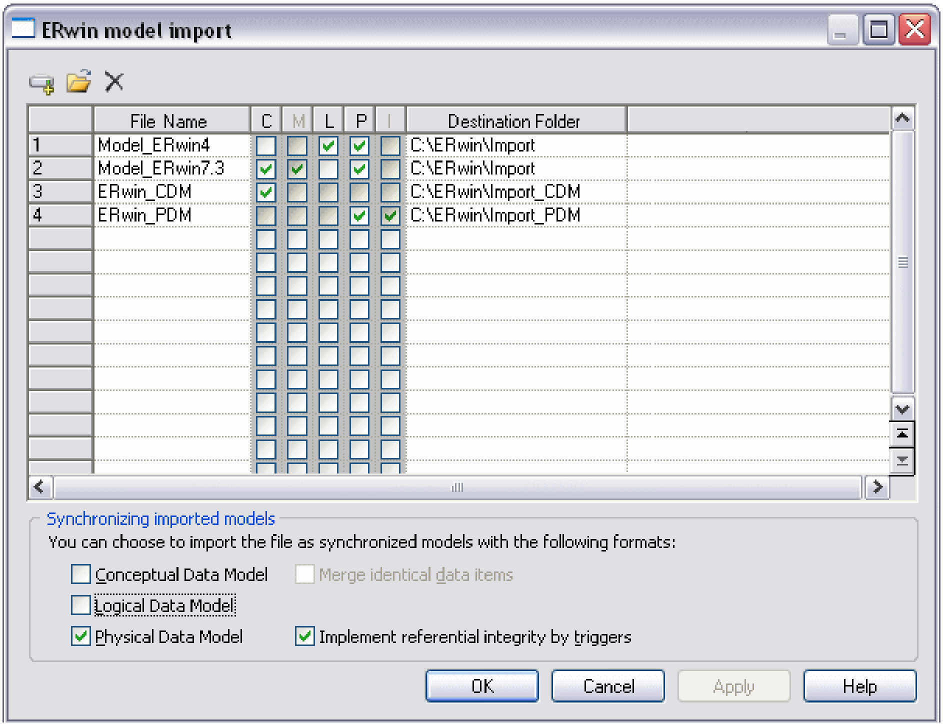
For example, your ERwin logical model may contain multiple entities that contain an attribute "address". By default, PowerDesigner will create a separate data item for each of these entity attributes. However if you select the **Merge identical data items** checkbox, then a single data item will be created, 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 by triggers**.
6. Click **OK** to begin the import. When the process is complete, the imported models will appear in the Browser.

Importing Multiple ERwin Files

PowerDesigner provides a wizard to help you import multiple ERwin files.

1. Select **File > Import > Multiple ERwin Files** to open the ERwin model import dialog:



2. Use the **Add Directory** or **Open Files** tools to add `.xml` or `.erx` files to import to the list.
3. Use the following checkbox columns (or the equivalent options at the bottom of the dialog) to specify import options for the files.

- **[C]onceptual Data Model** - import the file as a CDM
- **[M]erge identical data items** - [CDMs only] create a single data item for all entity attributes with the same name (eg "address")
- **[L]ogical Data Model** - import the file as an LDM
- **[P]hysical Data Model** - import the file as a PDM
- **[I]mplement referential integrity by triggers** - [PDMs only]

You can select to import a single ERwin file as multiple model types. To select multiple files and set the same options for them, click and hold while dragging your cursor over the far-left numbered column.

4. Specify a **Destination Folder** in which to create the PowerDesigner models.
5. Click **OK** to begin the import.

PowerDesigner will import each model and add it to your workspace. Note that to avoid problems of memory allocation when importing many models, the PowerDesigner models are closed by default. To open a model, simply double-click it.

Post-Import

You should perform a certain number of checks after import, and also be prepared for certain differences in your models.

We recommend that you perform the following post-import checks:

- *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* - 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* - 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 object naming conventions* - Select **Tools > Model Options**, expand the Naming Convention category and select the object entry (see *Core Features Guide > Modeling with PowerDesigner > Objects > Naming Conventions*).
- *Select other model options* - Select **Tools > Model Options**, and select the Model Settings category or one of its children (see *Setting CDM/LDM Model Options* on page 10 and *Setting PDM Model Options* on page 13)

The following are some differences that are commonly encountered when working with a newly imported ERwin model:

- *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".

PowerDesigner vs ERwin Terminology

PowerDesigner and ERwin use different terms to describe certain model objects.

The import process converts general model 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 in an extension file called `Imported Attributes` and embedded in the model. For information about working with extension files, see *Customizing and Extending PowerDesigner > Extension Files*.

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

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

Getting Started Using PowerDesigner for Former ERwin Users

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**.
- *To find the browser entry for an object symbol*: Right-click the symbol in the diagram and select **Find in Browser**.
- *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 (see *Core Features Guide > Linking and Synchronizing Models > Shortcuts and Replicas*).

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]** .
- *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 *Core Features Guide > Modeling with PowerDesigner > The Browser > Packages*.

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 *Core Features Guide > Storing, Sharing and Reporting on Models > Reports*

Databases

How do I create or update a model from a database? Select **File > Reverse Engineer > Database** and complete the dialog. When updating a model, a Merge dialog will open to allow you to verify the changes to be made before committing them. For more information, see *Reverse Engineering a Database into a PDM* on page 312.

How do I generate a database from my model? Select **Database > Generate Database** and complete the dialog. For more information, see *Generating a Database from a PDM* on page 290.

How do I update a database from my model? Select **Database > Apply Model Changes to Database** and complete the dialog. A Database Synchronization window will open to allow you to verify the changes to be made before committing them. For more information, see *Modifying a Database* on page 308.

Models

How do I compare or merge models? Select **Tools > Compare Models** or **Tools > Merge Model**. For more information, see *Core Features Guide > Modeling with PowerDesigner > Comparing and Merging Models*.

PART II

DBMS Definition Reference

The chapters in this part provide information specific to the DBMSs supported by PowerDesigner.

CHAPTER 9 HP Neoview

To create a PDM with support for features specific to the HP Neoview DBMS family, select the appropriate version in the DBMS field of the New Model dialog. To view these extensions to the PowerDesigner metamodel in the Resource Editor, select **Database > Edit Current DBMS** and expand the **Profile** node.

The following sections list the extensions provided for HP Neoview.

Note: We do not provide documentation for the properties on the **Physical Options** and certain other tabs, though minimal information is available for them in the Resource Editor. For information about these properties, consult your DBMS reference documentation.

Tables

The following extensions are available on the General tab:

Name	Description
Set	Specifies that the table is a SET table, and thus discards duplicate rows. Scripting name: Set
Volatile	Specifies that indexes associated with the table have lifespans limited to the SQL session in which the index is created and are dropped when the session ends. Scripting name: Volatile

Columns

The following extensions are available on the Neoview tab:

Name	Description
Identity	Specifies that the column is an identity column. Scripting name: Identity
Type	Specifies the type of identity column. You can choose between: • by default - allows both user-supplied and system-generated column values for the identity column • always - provides system-generated unique values and does not allow user-supplied identity column values. Scripting name: IdentityType

Name	Description
Start with	Specifies the start value of the cycle range for the identity column. Scripting name: StartWith
Increment	Specifies the value by which each value is incremented to obtain the next value. Scripting name: Increment
Minimum	Specifies the minimum value of the data type of the identity column starting the cycle range. Scripting name: MinValue
Maximum	Specifies the maximum value of the data type of the identity column starting the cycle range. Scripting name: MaxValue
Cycle	Specifies that when the maximum value is reached for the identity column, the values are restarted from the minimum. If this option is not selected, and error will be raised. Scripting name: Cycle
Unsigned	Specify that the column is unsigned. By default, columns are signed. Scripting name: Unsigned
Character set	[character columns] Specifies the character set to use. Scripting name: Charset
Upshift	[character columns] Specifies that the contents are stored as uppercase. Scripting name: Upshift
Mandatory	Specifies that the column must not contain a null value. Scripting name: Mandatory
Constraint name	Specifies the name of the not null column constraint. Scripting name: MandConstName

Indexes

The following extensions are available on the General tab:

Name	Description
Volatile	Specifies that the index has a lifespan limited to the SQL session in which it is created and is dropped when the session ends. Scripting name: Volatile
Unique	Specifies that the index is a unique index. Scripting name: Unique
No populate	Specifies that the index is not to be populated when it is created. The indexes are created, but no data is written to the index, and it is marked offline. Scripting name: NoPopulate
Partition	Specifies the partitioning columns. If you do not specify the partitioning columns, the default is the same partitioning column or columns as the base table for a non-unique index, and all the columns in the index for a unique index. Scripting name: HashPartitionColumns

References

The following extensions are available on the General tab:

Name	Description
Enforced	Specifies that the reference is checked. Scripting name: Enforced

Materialized Views

The following extensions are available on the Neoview tab:

Name	Description
Refresh type	Specifies the method that will be used to update the materialized view. Scripting name: RefreshType
Ignore	[on request only] Instructs the refresh operation of a materialized view over several base tables to ignore the changes to the listed base tables. Scripting name: IgnoreChangesOn
Initialize	Specifies when the materialized view gets its initial content, either upon creation or at the time of its first refresh. Scripting name: Initialize

Name	Description
Clustering columns	Specifies the order of rows within the physical file that holds the table, determines the physical organization of the table, and the ways you can partition the table. Scripting name: Clustering
Partition	Specifies hash partitioning, which is the only partitioning scheme supported for materialized views. Scripting name: HashPartition
Partitioning keys	Specifies the the partitioning keys of the materialized view. Scripting name: PartitionColumnList
Commit each	Specifies the number of rows that refresh processes from the log before committing a transaction and starting another one. Scripting name: MVAttribute
Text	Provides a textual view of the materialized view options. This field auto-updates as you select options, and you edits you make here are reflected in the options. Scripting name: ViewOption

Materialized View Groups (Neoview)

Materialized view groups allow you to collect together materialized views (views with the **Type** property set to `Materialized view`) that should be refreshed together. PowerDesigner models materialized view groups as extended objects with a stereotype of <<MVGroup>>.

Creating a Materialized View Group

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

- Select **Model > Materialized View Groups** to access the List of Materialized View Groups, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New > Materialized View Group**.

Materialized View Group Properties

You can modify an object's properties from its property sheet. To open a materialized view group property sheet, double-click its diagram symbol or its Browser entry in the Materialized View Groups folder.

The following extended attributes are available on the **Neoview** tab:

Name	Description
Owner	Specifies the group's owner. Scripting name: Owner

The following tabs are also available:

- Materialized Views - lists the materialized views contained within the group.

CHAPTER 10 IBM DB2 for z/OS (formerly OS/390)

To create a PDM with support for features specific to the IBM DB2 for z/OS DBMS family, select the appropriate version in the DBMS field of the New Model dialog. To view these extensions to the PowerDesigner metamodel in the Resource Editor, select **Database > Edit Current DBMS** and expand the **Profile** node.

Note: The DBMS definition file for IBM DB2 v8 for OS/390 is 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

The following sections list the extensions provided for DB2 for z/OS.

Note: We do not provide documentation for the properties on the **Physical Options** and certain other tabs, though minimal information is available for them in the Resource Editor. For information about these properties, consult your DBMS reference documentation.

Columns

The following extensions 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

Name	Description
Character subtype	[v6.x and higher] Specifies a subtype for a character string column. Scripting name: ExtSubtypeData [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
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
Implicitly hidden	[v9.x and higher] Specifies that the column is not visible in the result for SQL statements unless you explicitly refer to the column by name. Scripting name: ImplicitlyHidden
As security label	[v8 and higher] Specifies that the column will contain security label values. This also indicates that the table is defined with multi-level security with row level granularity. Scripting name: SecurityLabel

Domains

The following extensions 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

References

The following extensions are available on the DB2 tab:

Name	Description
Enforced	[v8 and higher] Indicates whether or not the referential constraint is enforced by the database manager during normal operations, such as insert, update, or delete. Scripting name: Enforced

Sequences

The following extensions are available on the DB2 tab:

Name	Description
Datatype	Specifies a computed value for "As" option. Allows to select a data type in a list. Scripting name: AsDatatype
Length	Specifies the length of the data type Scripting name: AsDatatypeLength
Start with	Specifies the first value for the sequence. Scripting name: InitialStartWith
Increment by	Specifies the interval between consecutive values of the sequence. Scripting name: InitialIncrementBy
Cache	Specifies the numerical value of the cache option. Scripting name: CacheValue
No Cache	Specifies a computed boolean value for order option. Scripting name: NoCacheBool
Cycle	Specifies a computed boolean value for cycle opion. Scripting name: CycleBool
Order	Specifies a computed boolean value for order option. Scripting name: OrderBool
Minimum value	Specifies the numerical value of the minvalue option. Scripting name: LimitsMinvalueValue
Maximum value	Specifies the numerical value of the maxvalue option. Scripting name: LimitsMaxvalueValue
No minimum	Specifies a computed boolean value for no minvalue option. Scripting name: NoMinLimit
No maximum	Specifies a computed boolean value for no maxvalue option. Scripting name: NoMaxLimit

Trusted Contexts (DB2)

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 for z/OS v9.x and higher and DB2 for Common Server v9.5 and higher. 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 a 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	[DB2 for z/OS only] 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

Name	Description
Default security label	[DB2 for z/OS only] 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 (DB2)

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 for z/OS v9.x and higher. 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 a 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

Name	Description
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

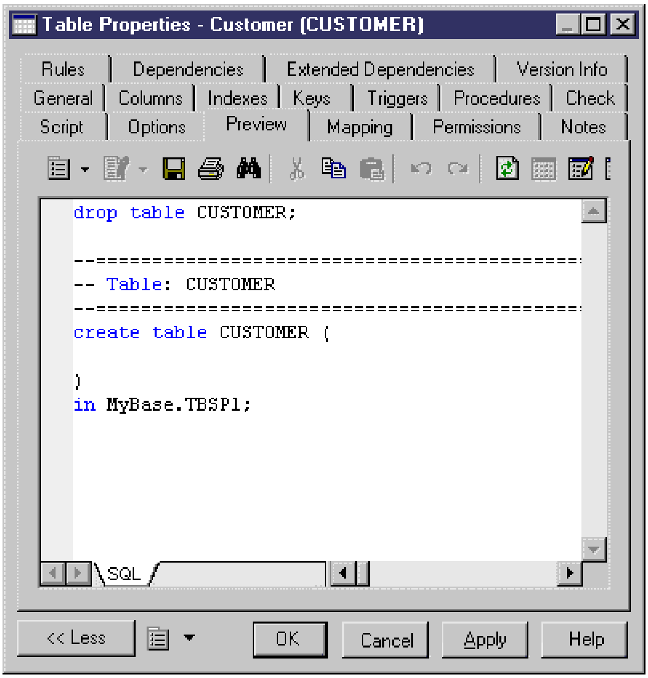
Tablespace Prefix (DB2)

In IBM databases for z/OS, 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.



Materialized Query Tables (DB2)

Materialized query tables are supported for IBM DB2 for z/OS 10 and higher. PowerDesigner models materialized query tables as views with a stereotype of <<Materialized query table>>.

Creating a Materialized Query Table

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

- Select **Model > Materialized Query Tables** to access the List of Materialized Query Tables, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New > Materialized Query Table**.

Materialized Query Table Properties

You can modify an object's properties from its property sheet. To open a materialized query table property sheet, double-click its diagram symbol or its Browser entry in the Materialized Query Tables folder.

The following extensions are available on the General tab:

Name	Description
Result table	Specifies whether the materialized view is a query table or result table. Scripting name: WithNoData
Maintained by	[Query table] Specifies how the data in the materialized query table is maintained. Scripting name: MaintainedBy
Query optimization	[Query table] Specifies whether this materialized query table can be used for optimization. Scripting name: QueryOptimization
Column default	[Result table] Specifies whether or not to copy column defaults. Scripting name: ColumnDefault
Identity	[Result table] Specifies whether or not to copy identity column attributes. Scripting name: Identity

The following tabs are also available:

- Partitions - lists the partitions contained within the materialized query table

Masks (DB2)

Masks are supported for IBM DB2 for z/OS 10 and higher. PowerDesigner models masks as extended objects with a stereotype of <<Mask>>.

Creating a Mask

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

- Select **Model > Masks** to access the List of Masks, and click the **Add a Row** tool.
- Right-click the model or package in the Browser, and select **New > Mask**.

Mask Properties

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

The following extended attributes are available on the **General** tab:

Name	Description
Column	Specifies the column to which the mask applies. A mask must not already exist for the column. Scripting name: MaskColumn
Enabled	Specifies if the column mask is to be enabled for column access control. Scripting name: MaskEnabled

The following extended attributes are available on the **Expression** tab:

Name	Description
Table correlation name	Specifies a correlation name that can be used within CASE expression to designate the table. Scripting name: TableCorrelation
Case expression	Specifies a CASE expression that determines the value that is returned for the column. The result of the CASE expression is returned in place of the column value in a row. Scripting name: CaseExpression

Row Permissions (DB2)

Auxiliary tables are supported for IBM DB2 for z/OS 10 and higher. PowerDesigner models row permissions as extended objects with a stereotype of <<Row permission>>.

Creating a Mask

You can create a row permission in any of the following ways:

- Select **Model > Row Permissions** to access the List of Row Permissions, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New > Row Permission**.

Row Permission Properties

You can modify an object's properties from its property sheet. To open a row permission property sheet, double-click its Browser entry in the Row Permissions folder.

The following extended attributes are available on the **General** tab:

Name	Description
Table	Specifies the table on which the row permission is created. Scripting name: Table
Enabled	Specifies that the row permission is to be enabled or disabled for row access control. Scripting name: RowPermissionEnabled

The following extended attributes are available on the Search condition tab:

Name	Description
Correlation name	Specifies a correlation name that can be used within search-condition to designate the table. Scripting name: TableCorrelation
Search condition	Specifies a condition that can be true, false, or unknown for a row of the table. Search condition follows the same rules used by the search condition in a WHERE clause of a subselect. Scripting name: SearchCondition

CHAPTER 11 IBM DB2 for Common Server

To create a PDM with support for features specific to the IBM DB2 for Common Server DBMS family, select the appropriate version in the DBMS field of the New Model dialog. To view these extensions to the PowerDesigner metamodel in the Resource Editor, select **Database > Edit Current DBMS** and expand the **Profile** node.

Note: The DBMS definition file for IBM DB2 v8.x Common Server is deprecated.

For a list of DB2 objects and their equivalents in PowerDesigner, see *Chapter 10, IBM DB2 for z/OS (formerly OS/390)* on page 351.

The following sections list the extensions provided for DB2 for Common Server.

Note: We do not provide documentation for the properties on the **Physical Options** and certain other tabs, though minimal information is available for them in the Resource Editor. For information about these properties, consult your DBMS reference documentation.

Tables

The following extensions 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
Data	Identifies the tablespace in which the table will be created. Scripting name: In
Cycle	Specifies whether or not the number of data partitions with no explicit tablespace can exceed the number of specified data partitions. Scripting name: DisplayCycle
Long	Identifies the table space in which the values of any long columns (LONG VARCHAR, LONG VARGRAPHIC, LOB data types, distinct types with any of these as source types, or any columns defined with user-defined structured types with values that cannot be stored inline) will be stored. Scripting name: InLongIn
Index	Identifies the tablespace in which any indexes on the table will be created. Scripting name: InIndexIn

Columns

The following extensions 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
Always Generate value	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
As row change timestamp	[v9.5 and higher] Specifies that the column is a timestamp column for the table. A value is generated for the column in each row that is inserted, and for any row in which any column is updated. Scripting name: AsRowChangeTimestampClause
Expression	Specifies that the definition of the column is based on an expression. Scripting name: ExtGenExpr (up to v9.0: ExtGenExpr)
Compact	Specifies COMPACT options for LOB data type columns. Scripting name: Compact
Logged	Specifies LOGGED options for LOB data type columns. Scripting name: Logged
Inline length	This option is only valid for a column defined using a structured type and indicates the maximum byte size of an instance of a structured type to store inline with the rest of the values in the row. Scripting name: InlineLength

Name	Description
Compress	Specifies that system default values (that is, the default values used for the data types when no specific values are specified) are to be stored using minimal space. If the VALUE COMPRESSION clause is not specified, a warning is returned and system default values are not stored using minimal space. Scripting name: CompressSystemDefault
Hidden	Specifies whether or not the column is to be defined as hidden. The hidden attribute determines whether the column is included in an implicit reference to the table, or whether it can be explicitly referenced in SQL statements. Scripting name: HiddenBool
Security label	Identifies a security label that exists for the security policy that is associated with the table. Scripting name: SecurityLabel

References

The following extensions are available on the DB2 tab (v8.0 and higher):

Name	Description
Enforced	Indicates whether or not the referential constraint is enforced by the database manager during normal operations, such as insert, update, or delete. Scripting name: Enforced
Enable query optimization	Specifies whether the constraint can be used for query optimization under appropriate circumstances. Scripting name: QueryOptimization

Views

The following extensions 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

Name	Description
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
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

Tablespaces

The following extensions are available on the DB2 tab:

Name	Description
Type	Specifies the tablespace type, as defined in the extended attribute type ExtTablespaceTypeList. Scripting name: ExtTablespaceType

Abstract Data Types

The following extensions 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 extensions 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 extensions 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

Database Partition Groups (DB2)

Database partition groups are supported for DB2 for Common Server v9.x and higher.

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 a 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 (DB2)

Index extensions are supported for DB2 for Common Server 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
Source key parameters	Specifies the parameter (and its data type) that is associated with the source key column. Scripting name: SourceKeyParameters
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
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
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

Security Policies (DB2)

Security policies define criteria that determine who has write and/or read access to individual rows and columns of tables.

Every protected table must have exactly one security policy associated with it. Rows and columns in that table can only be protected with security labels that are part of that security policy and all access of protected data follows the rules of that policy. You can have multiple security policies in a single database but you cannot have more than one security policy protecting any given table.

Security policies are supported for DB2 for Common Server v9.5 and higher. PowerDesigner models security policies as extended objects with a stereotype of <<SecurityPolicy>>.

Creating a Security Policy

You can create a security policy in any of the following ways:

- Select **Model > Security Policies** to access the List of Security Policies, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New > Security Policy**.

Security Policy Properties

You can modify an object's properties from its property sheet. To open a security policy property sheet, double-click its Browser entry in the Security Policies folder.

The following extended attributes are available on the **General** tab:

Property	Description
Use group authorization	Specifies that security labels and exemptions granted directly or indirectly to groups are considered for any access attempt. Scripting name: GroupAuthorization
Use role authorization	Specifies that security labels and exemptions granted directly or indirectly to roles are considered for any access attempt. Scripting name: RoleAuthorization
Restrict Not Authorized Write Security Label	Specifies the action that is to be taken when a user is not authorized to write the explicitly specified security label that is provided in the INSERT or UPDATE statement issued against a table that is protected with this security policy. A user's security label and exemption credentials determine the user's authorization to write an explicitly provided security label. Scripting name: Restrict

The following tabs are also available:

- Components - lists the security label components associated with the security policy

Security Labels (DB2)

Security labels are database objects that describe a set of security criteria, and which are granted to users to allow them to access protected data.

Every security label is part of exactly one security policy and includes one value for each component in that security policy.

Security labels are supported for DB2 for Common Server v9.5 and higher. PowerDesigner models security labels as extended objects with a stereotype of <<SecurityLabel>>.

Creating a Security Label

You can create a security label in any of the following ways:

- Select **Model > Security Labels** to access the List of Security Labels, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New > Security Label**.

Security Label Properties

You can modify an object's properties from its property sheet. To open a security label property sheet, double-click its Browser entry in the Security Labels folder.

The following extended attributes are available on the DB2 tab:

Property	Description
Policy	Specifies the security policy with which the label is associated. Scripting name: Policy

The following tabs are also available:

- Components - lists the security label components associated with the security label.

Security Label Components (DB2)

Security label components are database objects that model your organization's security structure.

A security label component represents a criteria to decide if a user should have access to a given piece of data, such as how well trusted the user is, what department she is in, or whether she is involved in a particular project.

Security label components are supported for DB2 for Common Server v9.5 and higher. PowerDesigner models security label components as extended objects with a stereotype of <<SecurityLabelComponent>>.

Creating a Security Label Component

You can create a security label component in any of the following ways:

- Select **Model > Security Label Components** to access the List of Security Label Components, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New > Security Label Component**.

Security Label Component Properties

You can modify an object's properties from its property sheet. To open a security label component property sheet, double-click its Browser entry in the Security Label Components folder.

The following extended attributes are available on the DB2 tab:

Property	Description
Component type	Specifies the type of component. You can choose between: • TREE: Each element represents a node in a tree structure • ARRAY: Each element represents a point on a linear scale • SET: Each element represents one member of a set Scripting name: Type
Constant list	Specifies one or more string constant values that make up the set of valid values for the component. The order in which the array elements appear is significant, with the first element ranking higher than the second element, and so on. Scripting name: List

Event Monitors (DB2)

Event monitors show activity from start to finish, and often consist of both a start and end event record. The most common uses for event monitors are for connections, locks, and statements. PowerDesigner models event monitors as extended objects with a stereotype of <<EventMonitor>>.

Creating an Event Monitor

You can create an event monitor in any of the following ways:

- Select **Model > Event Monitors** to access the List of Event Monitors, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select **New > Event Monitor**.

Event Monitor Properties

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

The following extended attributes are available on the General tab:

Name	Description
Workload management event monitor	Specifies that the event monitor is used for workload management. Selecting this option affects the types that are available in the Type field. Scripting name: WlmEventMonitor
Type	Specifies the type of event to record. Click the button to the right of the field to select multiple types. Scripting name: Type
Event condition	[connections, transactions, or statements type] Defines a filter that determines which connections cause a CONNECTION, STATEMENT or TRANSACTION event to occur. Scripting name: EventCondition
Details	[deadlock type] Specifies that the event monitor is to generate a more detailed deadlock connection event for each application that is involved in a deadlock. Scripting name: DeadlocksDetails

The following extended attributes are available on the DB2 tab:

Name	Description
Write to	Specifies the location where the event monitor will record its information. If you are writing to a table, you can additionally associate the event monitor with one or more event monitor groups on the EVMGroup tab. Event monitor groups identify the logical data group for which a target table is being defined, and PowerDesigner models them as extended sub-objects with a stereotype of <<EventMonitor>>. Scripting name: WriteToObject
Blocked	[table, file] Specifies that each agent that generates an event should wait for an event buffer to be written out to disk if the agent determines that both event buffers are full. This option should be selected to guarantee no event data loss. Scripting name: Blocked
Buffer size	[table, file] Specifies the size of the event monitor buffers (in units of 4K pages). All event monitor file I/O is buffered to improve the performance of the event monitors. Scripting name: BufferSize

Name	Description
Path	[file] The name of the directory in which the event monitor should write the event files data. The path must be known at the server. Scripting name: Path
Max files	[file] Specifies that there is a limit on the number of event monitor files that will exist for a particular event monitor at any time. Scripting name: MaxFiles
Maximum file size	[file] Specifies that there is a limit to the size of each event monitor file. Scripting name: MaxFileSize
Append	[file] Specifies that if event data files already exist when the event monitor is turned on, then the event monitor will append the new event data to the existing stream of data files. Scripting name: Append
Pipe name	[pipe] The name of the pipe to which the event monitor will write the data. The naming rules for pipes are platform specific. Scripting name: PipeName
Start	Specifies that the event monitor must be activated manually or is to be automatically activated whenever the database partition on which the event monitor runs is activated. Scripting name: Start
Scope	Either the event monitor reports on all database partitions (global) or only on the database partition that is running (local). Scripting name: Scope
Database partition	[pipe, file] Specifies the database partition on which the event monitor is to run. Scripting name: DBPartitionNum

Event Monitor Group Properties

You can create and manage event monitor groups from the EVMGroup tab of an event monitor. PowerDesigner models event monitor groups as extended sub-objects with a stereotype of <<EVMGroup>>.

The following extended attributes are available on the General tab:

Name	Description
Group	Identifies the logical data group for which a target table is being defined. Scripting name: Group
Table	Specifies the name of the target table. Scripting name: Table
PCTDeactivate	If a table is being created in a DMS table space, the PCTDEACTIVATE parameter specifies how full the table space must be before the event monitor automatically deactivates. Scripting name: PCTDeactivate
Tablespace	Defines the table space in which the table is to be created Scripting name: Tablespace
Trunc	Specifies that the STMT_TEXT and STMT_VALUE_DATA columns are defined as VARCHAR(n), where n is the largest size that can fit into the table row. Scripting name: Trunc
Inclusion criteria	Specifies which elements will be included in the table. Scripting name: Elements
Elements	Identifies a monitor element that will be included in or excluded from monitoring Scripting name: ElementList

Federated Systems (DB2)

A federated system consists of a DB2 instance that operates as a federated server, a database that acts as the federated database, one or more data sources, and clients (users and applications) that access the database and data sources. PowerDesigner provides support for federated servers for DB2 for Common Server v9.0 and higher through nicknames, servers, wrappers, and user mappings.

Nicknames (DB2)

A nickname is an identifier that an application uses to reference a data source object, such as a table or view. In a federated system, you use can nicknames to access data source objects and improve the performance of queries on remote data sources. Nicknames are supported for DB2 for Common Server v9.7 and higher.

Creating a Nickname

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

- Right-click the model node in the Browser and select **New Nickname for External Table**. In the dialog, select a table from a PDM open in the workspace and click **OK**. PowerDesigner will create a shortcut to the external table along with the necessary nickname and server objects.
- Select **Model > Nicknames** to access the List of Nicknames, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New > Nickname**.

Nickname Properties

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

The following extended attributes are available on the **General** tab:

Property	Description
Server	Specifies the server that contains the table the nickname is refering to (see *Servers (DB2)* on page 377). Use the tools to the right of the list to create, browse for, or view the properties of the currently selected server. Scripting name: Server
Remote schema	Specifies the schema to which the table or view belongs. If left empty, the server authorization name is used. Scripting name: RemoteSchema
Remote table	Specifies the remote table name. Scripting name: RemoteTable
Relational definition	Selecting **Yes** displays the **Relational Definition** tab, which contains a field to allow you to specify an appropriate definition in SQL. Scripting name: RemoteTable

The following extended attributes are available on the **Options** tab:

Property	Description
Code page	Specifies the code page of the file at the data source. This option is valid only for federated databases that use Unicode. Scripting name: CODEPAGE

Property	Description
Column delimiter	Specifies a single character to use as the delimiter that separates columns in the table-structured file. Scripting name: COLUMN_DELIMITER
Data source	Specifies the name of the script to invoke. Scripting name: DATASOURCE
File path	Specifies the fully qualified directory path and file name of the Excel spreadsheet to access. Scripting name: FILE_PATH
Key column	Specifies the name of the column on which the file is sorted. Scripting name: KEY_COLUMN
Namespaces	Specifies the namespaces that are associated with the namespace prefixes that are used in the XPATH and TEMPLATE options for each column. Scripting name: NAMESPACES
No empty string	Specifies whether the remote data source server can contain empty strings. Scripting name: NO_EMPTY_STRING
Numeric string	Specifies how to treat numeric strings. When set to Y for a column, the query optimizer recognizes that the column contains no blanks that could interfere with the sorting of the data in the column. Scripting name: NUMERIC_STRING
Range	Specifies the range of Excel cells to use. Scripting name: RANGE
Remote object	Specifies the name of the BioRS databank that is associated with the nickname. This name determines the schema and the BioRS databank for the nickname. Scripting name: REMOTE_OBJECT
SOAP action	Specifies the URI SOAPACTION attribute from the Web Services Description Language (WSDL) format. Scripting name: SOAPACTION
Sorted	Specifies whether the file at the data source is or is not sorted in ascending order. Scripting name: SORTED

Property	Description
Streaming	Specifies whether the source document should be separated into logical fragments for processing. Scripting name: STREAMING
Template	Specifies the nickname template fragment to use to construct a SOAP request. Scripting name: TEMPLATE
Timeout	Specifies the maximum time, in minutes, to wait for a response from the data source server. Scripting name: TIMEOUT
Validate	Specifies whether the source document is validated to ensure that it conforms to an XML schema or document type definition (DTD) before data is extracted from it. Scripting name: VALIDATE
Validate data file	For sorted files, this option specifies whether the wrapper verifies that the key column is sorted in ascending order and checks for null keys. Scripting name: VALIDATE_DATA_FILE
XPath	Specifies the XPath expression that identifies the XML elements that represent individual tuples. Scripting name: XPATH
XML root	Specifies the XML root element to add to the values of an XML column that references an XML sequence. Scripting name: XML_ROOT
Additional options	Can be used to specify any additional options. Scripting name: OtherOptions

Servers (DB2)

The instance owner supplies a name to identify the data source, along with the type and version of the data source, the database name for the data source (RDBMS only), and metadata that is specific to the data source. This information is called a server definition. Data sources answer requests for data and are servers in their own right. Servers are supported for DB2 for Common Server v9.7 and higher.

Creating a Server

Note: A server can be created automatically when you create a nickname (see *Nicknames (DB2)* on page 374) using the **New Nickname for External Table** command.

You can manually create a server in any of the following ways:

- Select **Model > Servers** to access the List of Servers, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New > Server**.
- Use the **Create** tool next to the **Server** field on the **General** tab of a nickname or user mapping property sheet (see *Servers (DB2)* on page 377).

Server Properties

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

The following extended attributes are available on the **General** tab:

Property	Description
Authorization / Password	Required only for DB2 family data sources. Specify the authorization ID and password under which any necessary actions are performed at the data source when the `CREATE SERVER` statement is processed. This authorization ID is not used when establishing subsequent connections to the server. Scripting name: Authorization, Password
Type / Version	Specify the type and version of the data source. Scripting name: Type, Version
Wrapper	Specifies the wrapper (see *Wrappers (DB2)* on page 381) that the DB2 federated server uses to interact with the server object. Use the tools to the right of the list to create, browse for, or view the properties of the currently selected wrapper. Scripting name: Wrapper
Model	Specifies the PDM containing the structure of the database on the server being referenced. Use the tools to the right of the list to browse for an object or view the properties of the currently selected PDM. Scripting name: Model

The following extended attributes are available on the **Options** tab:

Property	Description
Fold login / Fold password	Specify the case of user IDs and passwords that the DB2 federated server sends to the data source server for authentication, and whether they can be null. Scripting name: FOLD_ID, FOLD_PW
Enable plan hints	Specifies whether plan hints, which are statement fragments that provide extra information for data source optimizers to help decide whether to use an index, which index to use, or which table join sequence to use. This information can, for certain query types, improve query performance. Scripting name: PLAN_HINTS
Ignore user data types	Specifies whether the DB2 federated server should determine the built-in type that underlies a UDT without strong typing. Scripting name: IGNORE_UDT
Push down	Specifies whether the DB2 federated server will consider letting the data source evaluate operations. Scripting name: PUSHDOWN
Collating sequence	Specifies whether the data source uses the same default collating sequence as the DB2 federated server, based on the NLS code set and the country information. Scripting name: COLLATING_SEQUENCE
Date compatibility	Specifies whether the DATE compatibility semantics associated with the `TIMESTAMP(0)` data type are applied to the connected database. Scripting name: DATE_COMPAT
No trailing blanks	Specifies whether data sources which have variable character data types pad the length with trailing blanks. Scripting name: VARCHAR_NO_TRAILING_BLANKS
Enforce savepoint	Specifies whether the DB2 federated server should enforce detecting or building of application savepoint statements. Scripting name: IUD_APP_SVPT_ENFORCE
CPU ratio / IO ratio	Indicate how much faster or slower a data source's CPU and I/O system runs than those of the the DB2 federated server. Scripting name: CPU_RATIO, IO_RATIO

Property	Description
Packet size	Specifies the packet size of the Sybase interface file in bytes. If the data source does not support the specified packet size, the connection will fail. Increasing the packet size when each record is very large (for example, when inserting rows into large tables) significantly increases performance. Scripting name: PACKET_SIZE
Timeout	Specifies the number of seconds the DB2 federated server will wait for a response from Sybase Open Client for any SQL statement. The value of seconds is a positive whole number in DB2 Universal Database's integer range.The timeout value that you specify depends on which wrapper you are using. The default behavior of the TIMEOUT option for the Sybase wrappers is 0, which causes DB2 to wait indefinitely for a response. Scripting name: TIMEOUT
Login timeout	Specifies the number of seconds for the DB2 federated server to wait for a response from Sybase Open Client to the login request. Scripting name: LOGIN_TIMOUT
Communication rate	Specifies the communication rate between the DB2 federated server and the data source server in megabytes per second. Scripting name: COMM_RATE
Database name	Specifies the database that you want the the DB2 federated server to access on the data source. For DB2, this value corresponds to a specific database within an instance or, with DB2 for z/OS or OS/390, the database LOCATION value. Not required for Oracle instances, which contain only one database. Scripting name: DBNAME
Sybase OCI path	Specifies the path and name of the Sybase Open Client interfaces file. On Windows NT servers, the default is `%DB2PATH%\interfaces`. Scripting name: IFILE
Node	Specifies the name by which the data source is defined as an instance to its RDBMS. Scripting name: NODE
Additional options	Can be used to specify any additional options. Scripting name: OtherOptions

Wrappers (DB2)

Wrappers are mechanisms by which the federated server interacts with data sources. The federated server uses routines stored in a library called a wrapper module to implement a wrapper. Wrappers are supported for DB2 for Common Server v9.7 and higher.

Creating a Wrapper

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

- Select **Model** > **Wrappers** to access the List of Wrappers, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New** > **Wrapper**.
- Use the **Create** tool next to the **Wrapper** field on the **General** tab of a server property sheet (see *Servers (DB2)* on page 377).

Wrapper Properties

You can modify an object's properties from its property sheet. To open a wrapper property sheet, double-click its Browser entry in the Wrappers folder. The following extended attributes are available on the **Options** tab:

Property	Description
Library	Specifies the name of the file that contains the wrapper library module. Scripting name: Library
Fenced	Specifies that the wrapper is fenced or trusted by DB2. A fenced wrapper operates under some restrictions. Scripting name: DB2_FENCED
Language / Class or library	Specify the language and implementation of the user mapping plug-in. Valid languages are Java (default) and C. For a plug-in written in Java, you must specify a case-sensitive string for the class name that corresponds to the user mapping repository class. For example, `UserMappingRepositoryLDAP`. For a plug-in written in C, you must specify any valid C library name. Scripting name: DB2_UM_PLUGIN_LANG, DB2_UM_PLUGIN
Additional options	Can be used to specify any additional options. Scripting name: OtherOptions

User Mappings (DB2)

A user mapping is an association between an authorization ID on the federated server and the information that is required to connect to the remote data source. User mappings are supported for DB2 for Common Server v9.7 and higher.

Creating a User Mapping

You can create a user mapping in any of the following ways:

- Select **Model > User Mappings** to access the List of User Mappings, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New > User Mapping**.

User Mapping Properties

You can modify an object's properties from its property sheet. To open a user mapping property sheet, double-click its Browser entry in the User Mappings folder.

The following extended attribute is available on the **General** tab:

Property	Description
Server	Specifies the name of the server object (see *Servers (DB2)* on page 377) for the data source that the authorization-name can access. The server name is the local name for the remote server that is registered with the federated database. Scripting name: Server

The following extended attributes are available on the **Options** tab:

Property	Description
Accounting string	Specifies a DRDA accounting string. Valid values include any string that has 255 characters or fewer. Scripting name: ACCOUNTING_STRING
Remote user ID / password	Specify the remote user ID to which the local user ID is mapped, and its password in the remote system. If you do not specify a password, the password used to connect to the federated database is used. Scripting name: REMOTE_AUTHID, REMOTE_PASSWORD
Use trusted context	Specifies whether the user mapping is trusted. Scripting name: USE_TRUSTED_CONTEXT
Additional options	Can be used to specify any additional options. Scripting name: OtherOptions

CHAPTER 12 Microsoft SQL Server

To create a PDM with support for features specific to the MS SQL Server DBMS family, select the appropriate version in the DBMS field of the New Model dialog. To view these extensions to the PowerDesigner metamodel in the Resource Editor, select **Database > Edit Current DBMS** and expand the **Profile** node.

Note: The DBMS for SQL Server v7.x is deprecated.

The following sections list the extensions provided for MS SQL Server.

Note: In addition to the extensions listed below, PowerDesigner supports the following features for SQL Server 2005 and higher:

- 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.

Abstract Data Types

The following extensions are available on the Microsoft tab:

Name	Description
Assembly	Specifies the assembly to bind with the abstract data type. Scripting name: Assembly

Abstract Data Type Attributes

The following extensions are available on the Microsoft tab:

Name	Description
Nullable	Specifies that the type column allows null value. Scripting name: Nullable
Computed	Specifies that the type column is computed. Scripting name: Specifies that the type column is computed.
Identity	Specifies that the new column is an identity column. Scripting name: Identity

Name	Description
Expression	Specifies an expression that defines the value of a computed column. Scripting name: Expression
Persisted	Specifies that the SQL Server Database Engine will physically store the computed values in the table, and update the values when any other columns on which the computed column depends are updated. Scripting name: Persisted
Seed	Specifies the value used for the very first row loaded into the table. Scripting name: Seed
Increment	Specifies the incremental value added to the identity value of the previous row loaded. Scripting name: Increment
Default	Specifies the value provided for the column when a value is not explicitly supplied during an insert. Scripting name: Default
Row GUID	Specifies that the new column is a row GUID column Scripting name: RowGuidCol
Collation	Specifies the collation for the column. Scripting name: Collate

Columns

The following extensions are available on the Microsoft tab:

Name	Description
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
Sparse	[v2008 and higher] Specifies that the column is a sparse column. The storage of sparse columns is optimized for null values. Sparse columns cannot be designated as NOT NULL. Scripting name: Sparse

Name	Description
Filestream	[v2008 and higher] Specifies that when the FILESTREAM storage attribute is specified for a column, all values for that column are stored in a FILESTREAM data container on the file system Scripting name: Filestream
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
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
Collation name	[v2005 and higher] A single string that specifies the collation name for a SQL collation. Scripting name: ExtCollation
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
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

Name	Description
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 extensions 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 extensions are available on the Microsoft tab:

Name	Description
Hidden	[v2000] Indicates whether the dimension is hidden from clients. Scripting name: IsHidden
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
WriteEnabled	[v2005 and higher] Indicates whether dimension writebacks are available (subject to security permissions). Scripting name: WriteEnabled

Dimension Attributes

The following extensions 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
Member-sWithData	[v2005 and higher] Determines whether to display data members for non-leaf mem-bers in the parent attribute. Scripting name: MembersWithData
OrderBy	[v2005 and higher] Describes how to order the members contained in the attribute. Scripting name: OrderBy
MemberNa-mesUnique	[v2005 and higher] Determines whether member names under the parent element must be unique. Scripting name: MemberNamesUnique
IsAggregat-able	[v2005 and higher] Specifies whether the values of the DimensionAttribute element can be aggregated. Scripting name: IsAggregatable
Attribute-HierarchyEn-abled	[v2005 and higher] Determines whether an attribute hierarchy is enabled for the attribute. Scripting name: AttributeHierarchyEnabled
Attribute-HierarchyVi-sible	[v2005 and higher] Determines whether the attribute hierarchy is visible to client applications. Scripting name: AttributeHierarchyVisible

Databases

The following extensions 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
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

Name	Description
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
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 418.

Data Sources

The following extensions 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 extensions are available on the Configuration tab:

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
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 extensions are available on the Microsoft tab:

Name	Description
Hidden	[v2000] Indicates whether the hierarchy is hidden from client applications. Scripting name: IsHidden
AllMember-Name	[v2005 and higher] Contains the caption in the default language for the All member of a Hierarchy element. Scripting name: AllMemberName
MemberName-sUnique	[v2005 and higher] Determines whether member names under the parent element must be unique. Scripting name: MemberNamesUnique
AllowDuplicate-Names	[v2005 and higher] Determines whether duplicate names are allowed in a Hierarchy element. Scripting name: AllowDuplicateNames

Fact Measures

The following extensions 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
AggregateFunction	[v2005 and higher] Defines the common prefix to be used for aggregation names throughout the associated parent element. Scripting name: AggregateFunction

Name	Description
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

Note: For additional information about special SQL Server index types, see *XML Indexes (SQL Server)* on page 415 and *Spatial Indexes (SQL Server)* on page 413.

The following extensions 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

Name	Description
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
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 extensions 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 extensions 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

Storages

The following extensions are available on the Microsoft tab:

Name	Description
Contains filestream	Specifies that the filegroup stores FILESTREAM binary large objects (BLOBs) in the file system. Scripting name: FileStream

Tables

The following extensions 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 TABLE CHECK constraint from being enforced during the distribution process used by replication. Scripting name: ExtCktNotForReplication
Table is partitioned	Specifies that the table is partitioned. Scripting name: PartitionedTable
Filegroup	[unpartitioned tables] Specifies the name of the filegroup. Scripting name: FileGroup
Text/Image	[unpartitioned tables] Specifies the name of the filegroup where text and image are stored. Scripting name: TextImageOn
Filestream	[unpartitioned tables] Specifies the name of the filegroup used for filestream. Scripting name: FilestreamOnFilegroup
Compression	[unpartitioned tables] Specifies the compression type of the table (none, row or page). Scripting name: TableCompression

Name	Description
Partition scheme	[partitioned tables, v2005 and higher] Specifies the name of the partition scheme. You must also specify the name of the partitioned column Scripting name: PartitionScheme, PartitionSchemeColumn
Filestream partition scheme	[partitioned tables, v2005 and higher] Specifies the name of the partition scheme. Scripting name: FilestreamPartitionScheme, FilestreamPartitionSchemeColumn
Compression	[partitioned tables] Specifies the partitions that use the compression. Scripting name: DataCompression

Triggers

The following extensions are available on the Microsoft tab:

Name	Description
Option	Is a concatenation of the `WITH ENCRYPTION` (which is illegal for CLR triggers, and which prevents the trigger from being published) and `EXECUTE AS` (which specifies the security context under which the trigger is executed) options. Scripting name: Option

An additional property is available for CLR triggers (see *CLR Procedures, Functions, and Triggers (SQL Server)* on page 406).

Users

The following extensions are available on the General tab (v2005 and higher):

Name	Description
Implicit schema	Specifies that the stored procedure `sp_grantdbaccess` will be used instead of a create user statement during database generation. Scripting name: ImplicitSchema
Default schema	Specifies the first schema searched to resolve the names of objects for this user. If the Implicit schema option is selected, then the default schema is initialized to the name of the user. Scripting name: DefaultSchema

Views

The following extensions 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

Horizontal Partitioning (SQL Server)

MS SQL Server 2005 and higher supports horizontal partitioning, 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.

To partition a table or an index, specify a partition scheme and column on the Microsoft tab of its property sheet.

Partition Functions (SQL Server)

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 a 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
Length	Specifies the length of input parameter data type. Scripting name: InputParameterLength

Name	Description
Precision	Specifies the precision of input parameter data type Scripting name: InputParameterPrec
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 (SQL Server)

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 a 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

Common Language Runtime (CLR) Integration (SQL Server)

CLR integration (for SQL Server 2005 and higher) 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 (SQL Server)

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 a 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 (SQL Server)

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 a 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
Schema	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	[v2005] Specifies the name of the input parameter. Scripting name: InputParameterName
Type	[v2005] 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
Length	Specifies the length of return data type. Scripting name: ReturnTypeLength

Name	Description
Precision	Specifies the precision of return data type. Scripting name: ReturnTypePrec

For v2008 and higher, the **Parameters** tab allows you to list the name, type, length, and precision of any parameters.

CLR User-Defined Types (SQL Server)

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.

Since UDTs are accessed by the system as a whole, their use for complex data types may negatively impact performance, and complex data is generally best modeled using traditional rows and tables. UDTs in SQL Server are well suited to date, time, currency, and extended numeric types, geospatial applications, and 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, and have an OOM containing an appropriate class open in the workspace, in order to specify the supertype:

1. Select **Model > Abstract Data Types** to access the List of Abstract Data Types, and click the **Add a Row** tool (or 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.

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

Name	Description
Mandatory	Specifies whether the type can hold a null value. Scripting name: Mandatory

CLR Procedures, Functions, and Triggers (SQL Server)

In Microsoft SQL Server 2005, you can write user-defined procedures, functions, and triggers in any Microsoft .NET Framework programming language. PowerDesigner models these objects as standard procedures and triggers that use a CLR template, and are linked to a method from an associated OOM.

Creating a CLR Procedure, Function, or Trigger

To create a CLR procedure, function, or trigger you must have already created an assembly, and you must have an OOM open in the workspace, in order to specify an associated class method:

1. Create a standard procedure or function and, on the Definition Tab of its property sheet, select CLR Procedure, CLR Function, or CLR Trigger from the template list. A Class method field will be displayed to the right of the template list.
2. Click the Select Method tool to the right of the Class method field, in order to specify the associated method.
3. Click the Microsoft tab and select an assembly from the list to bind to the procedure or function.

CLR Procedure, Function, and Trigger Properties

You can modify an object's properties from its property sheet. To open a CLR procedure, function, or trigger property sheet, double-click its diagram symbol or its Browser entry in the Procedures or Triggers 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

Encryption (SQL Server)

SQL Server 2005 and higher provide a security infrastructure that supports hierarchical encryption and key management.

PowerDesigner supports encryption with certificates and asymmetric and symmetric keys.

Certificates (SQL Server)

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 a 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	[v2005] Specifies the name of a user as the owner of the certificate. Scripting name: Authorization
Assembly	[v2005] Specifies a signed assembly that has already been loaded into the database. Scripting name: Assembly
Assembly File	[v2005] 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	[v2005] If the EXECUTABLE option is used, the file is a DLL that has been signed by the certificate. Scripting name: Executable

Name	Description
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	[v2005] Use this option only if you want to encrypt the certificate with a password. Scripting name: EncryptionPassword
Start 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
Active for begin dialog	Specifies that the certificate is available to the initiator of a Service Broker dialog conversation. Scripting name: ActiveForBeginDialog

Asymmetric Keys (SQL Server)

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 asymmetric keys 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 a 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
Source type	[v2008 and higher] Specifies the type of source (File, Executable file, Assembly or Provider) Scripting name: Source
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
Provider	[v2008 and higher] Specifies the name of the EKM (Extensible Key Management) provider. Scripting name: Provider
Executable	[v2005] 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
Create disposition	[v2008 and higher] Creates a new key or use an existing one. Scripting name: CreateDisposition

Name	Description
Provider key name	[v2008 and higher] Specifies the key name from the external provider. Scripting name: ProviderKeyName
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 (SQL Server)

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 a 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
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

Name	Description
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)

SQL Server 2005 and higher 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 (SQL Server)

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 a 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 (SQL Server)

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, you must have already created a catalog:

1. Create a standard index and, on the General tab, select FULLTEXT in the Type field.
2. Click the Microsoft tab and select a catalog from the list and then specify the type of change tracking required.

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

Name	Description
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

Spatial Indexes (SQL Server)

SQL Server 2008 and higher supports spatial data types and indexes. PowerDesigner supports these new features through table indexes with the type set to SPATIAL.

Creating a Spatial Index

To create a spatial index:

1. Create a table containing a column of type `geography` or `geometry`.
2. Create a standard index and, on the **General** tab, select `SPATIAL` in the **Type** field. The **Columns** tab is renamed to **Spatial Options**.
3. Click the **Spatial Options** tab, select your spatial column in the **Indexed column** field, and complete the remaining properties.

Spatial Index Properties

You can modify an object's properties from its property sheet. To open a spatial index property sheet, double-click its Browser entry. The following extended attributes are available on the **Spatial Options** tab:

Name	Description
Indexed column	Specifies the spatial column on which the index is based Scripting name: IndexedColumn
Tessellation scheme	Specifies the tessellation scheme for the spatial index. Scripting name: TesselationType
Bounding box	Specifies a numeric four-tuple that defines the four coordinates of the bounding box: the x-min and y-min coordinates of the lower, left corner, and the x-max and y-max coordinates of the upper right corner. Scripting name: BoundingBoxDefn

Name	Description
Cells per object	Specifies the number of tessellation cells (any integer between 1 and 8192, inclusive) per object that can be used for a single spatial object in the index by the tessellation process. Scripting name: CellsPerObject
Grids	Specifies the density of the grid at each level of a tessellation scheme. Scripting name: GridsDefn
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
Index padding	Specifies index padding. Scripting name: PadIndex
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 (up to 64) used in a parallel plan execution. Scripting name: MaxDop
Allow row locks	Specifies whether row locks are allowed. Scripting name: AllowRowLocks
Allow page locks	Specifies whether page locks are allowed. Scripting name: AllowPageLocks
Store sort result	Specifies to store temporary sort results in tempdb. Scripting name: SortInTempDB
Do not recompute statistics	Specifies to recompute distribution statistics. Scripting name: StatisticsNoRecompute
Drop if exist	Specifies that the named, preexisting clustered, nonclustered, or XML index is dropped and rebuilt. Scripting name: DropExisting

XML Indexes (SQL Server)

SQL Server 2005 provides improvements in indexing XML data. PowerDesigner supports these new features through table indexes with the type set to XML.

Creating an XML Index

To create an XML index:

1. Create a standard index and, on the General tab, select XML in the Type field.
2. Click the Microsoft tab and specify any appropriate additional options.

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

Name	Description
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
Allow page locks	Specifies whether page locks are allowed. Scripting name: AllowPageLocks

XML Data Types (SQL Server)

SQL Server 2005 and higher allows you to store XML documents and fragments in a database. PowerDesigner supports this feature through new column properties and the XML schema collection object.

Using an XML Data Type in a Table Column

To specify a column for storing XML, you must have already created an XML schema collection:

1. Create a standard column and, on the General tab, select XML in the Data type field.
2. Click the Microsoft tab, select an XML schema collection and content type.

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

XML Schema Collections (SQL Server)

An XML schema collection provides validation of and data type information about the XML to be stored in the column. PowerDesigner models XML schema collections as extended objects with a stereotype of <<XMLSchemaCollection>>.

Schemas provide information about the types of attributes and elements in the XML data type instance, and 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.

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 a 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

Name	Description
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

Database Mirroring (SQL Server)

SQL Server 2005 and higher 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 endpoints and extensions on the database object.

Creating a Database for Mirroring

To create a database to model database mirroring:

1. Right-click the model in the Browser and select **Properties**.
2. On the General tab, click the **Create** tool to the right of the **Database** field.
3. Click the Mirroring tab and specify any appropriate properties.

Mirroring Properties

You can modify an object's properties from its property sheet. To open a database property sheet, double-click its Browser entry.

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 (SQL Server)

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 a 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 (SQL Server)

SQL Server 2005 and higher provides the service broker, which 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 (SQL Server)

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 a 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

Name	Description
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 (SQL Server)

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 a 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 (SQL Server)

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 422).
- Select **Model > Message Contracts** to access the List of Message Contracts, and click the **Add a Row** tool.
- Right-click the model (or a 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 (SQL Server)

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 a 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 (SQL Server)

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 a 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 427. 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 (SQL Server)

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 a 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 (SQL Server)

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 a 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	[v2005] Specifies the name of the remote service to which the route points. Scripting name: Service
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

Name	Description
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 (SQL Server)

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 a 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

Name	Description
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 (SQL Server)

Resource Governor, available in SQL Server 2008 and higher, 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 (SQL Server)

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 a 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

Name	Description
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 (SQL Server)

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 a 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

Schemas (SQL Server)

For SQL Server 2005 and higher, schemas are distinct namespaces, separate from the users who created them, and can be transferred between users. PowerDesigner models schemas as users with a stereotype of <<Schema>>.

Creating a Schema

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

- Select **Model > Users and Roles > Schemas** to access the List of Schemas, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New > Schema**.

Schema Properties

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

The following extended attributes are available on the **General** tab:

Name	Description
Owner	Specifies the name of the database-level principal user that owns the schema. This user may own other schemas, any of which may be his default schema. Scripting name: SchemaOwner

Synonyms (SQL Server)

PowerDesigner supports synonyms for SQL Server 2005 and higher 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 general information about synonyms, see *Synonyms (PDM)* on page 157.

Analysis Services (SQL Server 2000)

The OLAP Services feature from SQL Server v7.0 is called *Analysis Services* in SQL Server 2000. To enable analysis services, select **Tools > General Options**, click the Add-ins category, select the Microsoft Analysis Services add-in (PdMsOlap.dll), and then click **OK** to install it and return to the model.

For information about analysis services in SQL Server 2005, see *Microsoft SQL Server 2005 Analysis Services* on page 438.

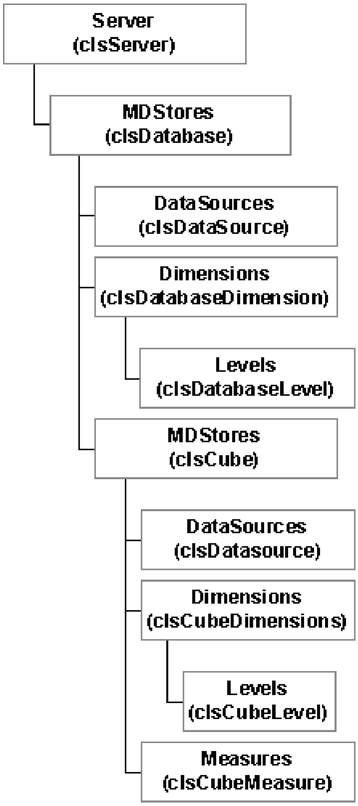
Analysis Services provide the following capabilities:

- 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.

DSO uses hierarchically arranged groups of objects to define basic elements of Analysis Services data storage, as implemented by the Analysis server:



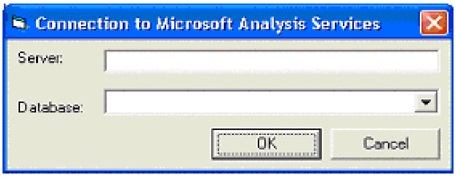
The following table lists the mappings between the objects contained within the DSO and PowerDesigner PDM metamodels:

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.)
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 clsCube-Level	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.)

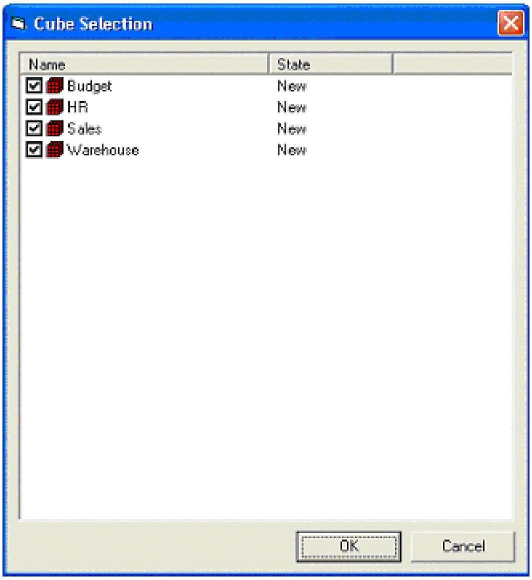
Generating Cubes

The Microsoft Analysis Services add-in lets you generate cubes.

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.



3. Select the cubes you want to generate, and then click OK.

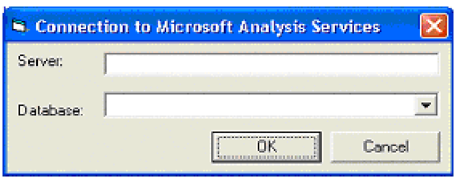
   The selected cubes are generated. If a cube already exists in the database, it is dropped before being recreated. If a dimension already exists, the selected cube reuses it. To be fully generated, a cube must have a complete mapping to a table before being generated.

Reverse Engineering Cubes

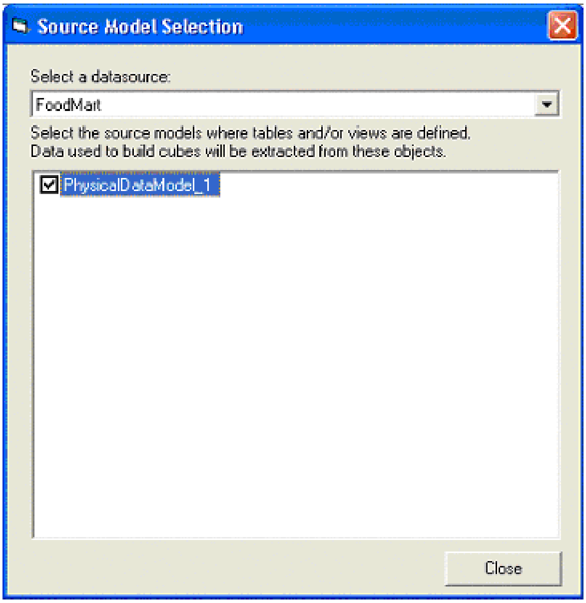
The Microsoft Analysis Services add-in lets you reverse engineer cubes.

Before reverse engineering cubes, you must create one or more PDMs to modelise the tables that will provide the data. PowerDesigner will create links from the retrieved cubes to these tables.

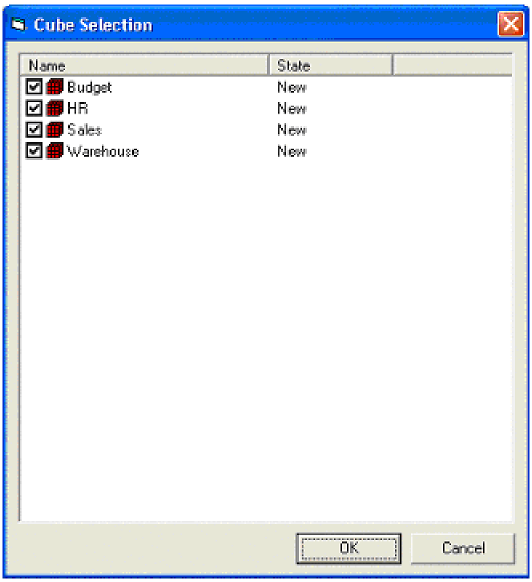
1. Select **Tools** > **Microsoft Analysis Services** > **Reverse Engineer Cubes** to open the connection dialog box.



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.



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.



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.

Analysis Services (SQL Server 2005)

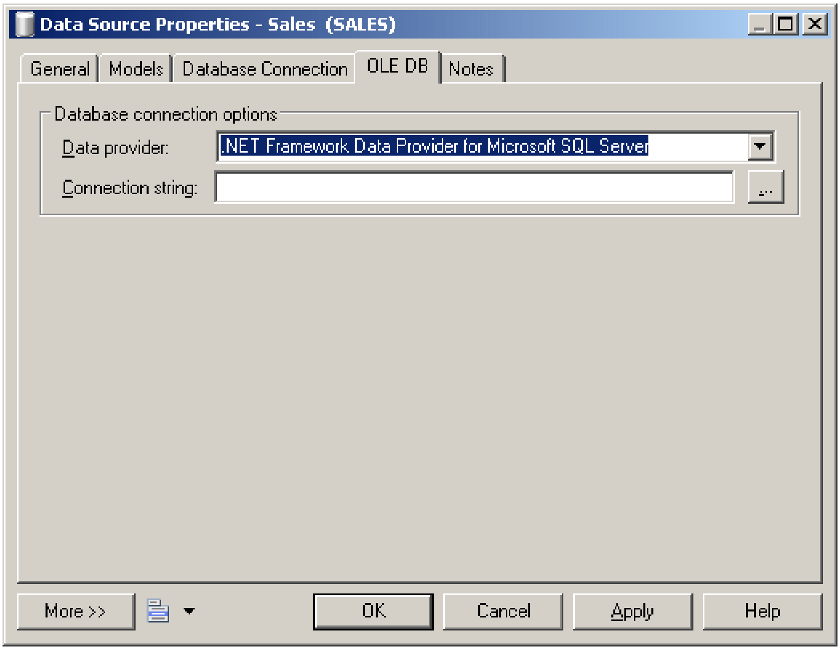
PowerDesigner allows you to retrieve multiple dimension objects in a PDM in order to build cubes, and to create a new multiple-dimension diagram. From this diagram, you can generate cubes to a Microsoft SQL Server 2005 Analysis Server (SSAS).To enable analysis services, select **Tools > General Options**, click the Add-ins category, 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.

Note: 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.

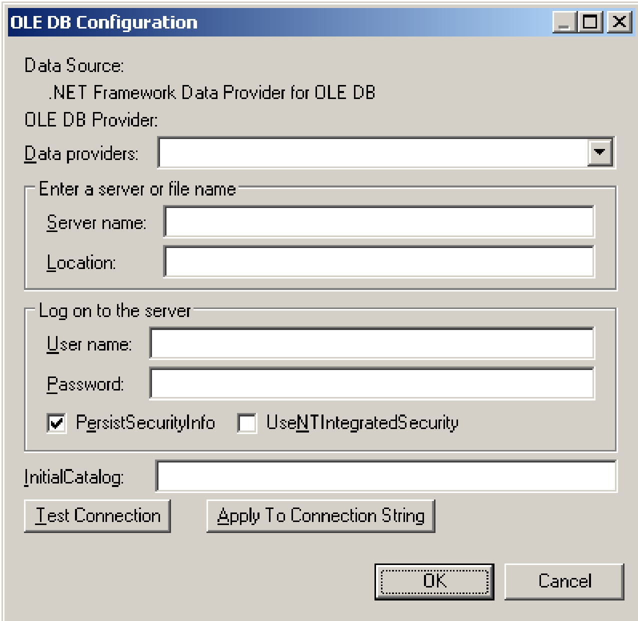
Specifying a Data Source for Cubes

Before generating cubes, you must define a data source with an OLE DB connection that will specify from where the cubes will be populated.

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.



3. Click the ellipsis tool to the right of the connection string field to open the provider-specific configuration dialog.



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.

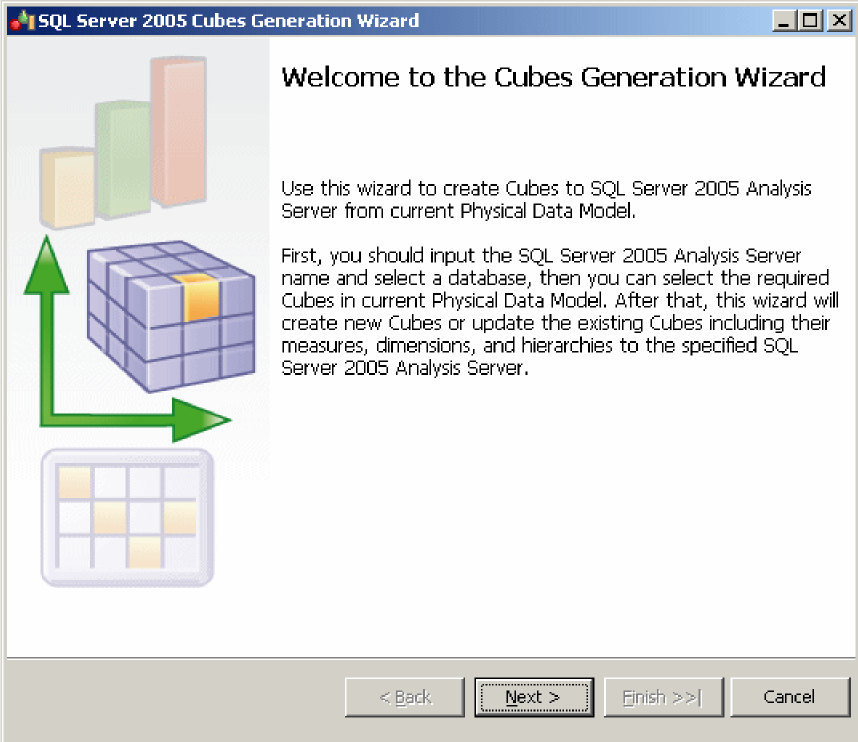
5. Click OK to return to your model.

When you have created the appropriate data sources, you can proceed with generating your cubes.

Generating Cubes for Microsoft SQL Server 2005

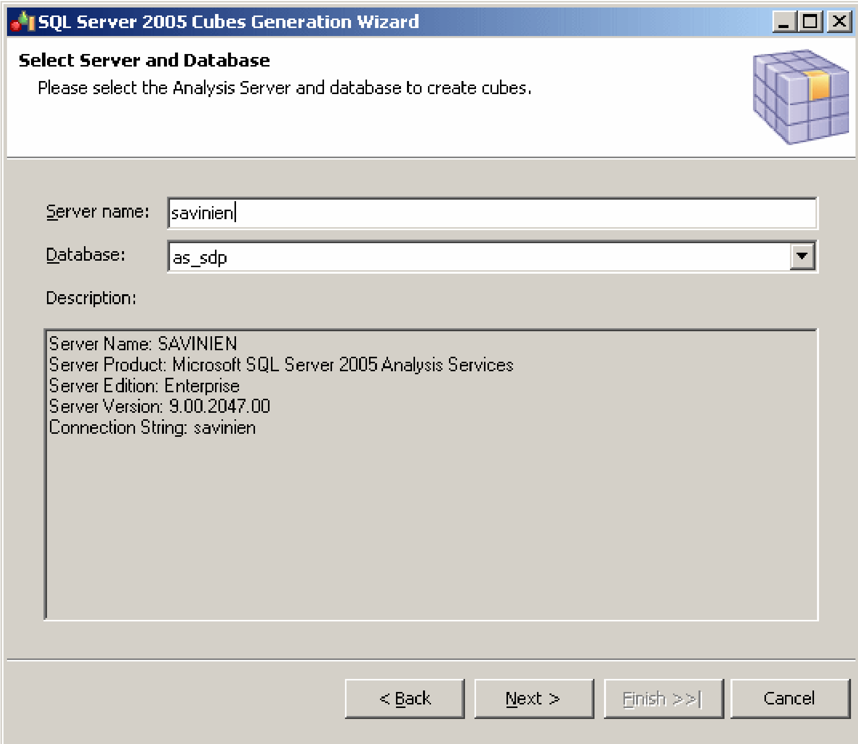
The Microsoft SQL Server 2005 Analysis Services add-in enables the generation of cubes.

1. Select **Tools** > **Microsoft SQL Server 2005 Analysis Services** > **Generate Cubes** to open the wizard.



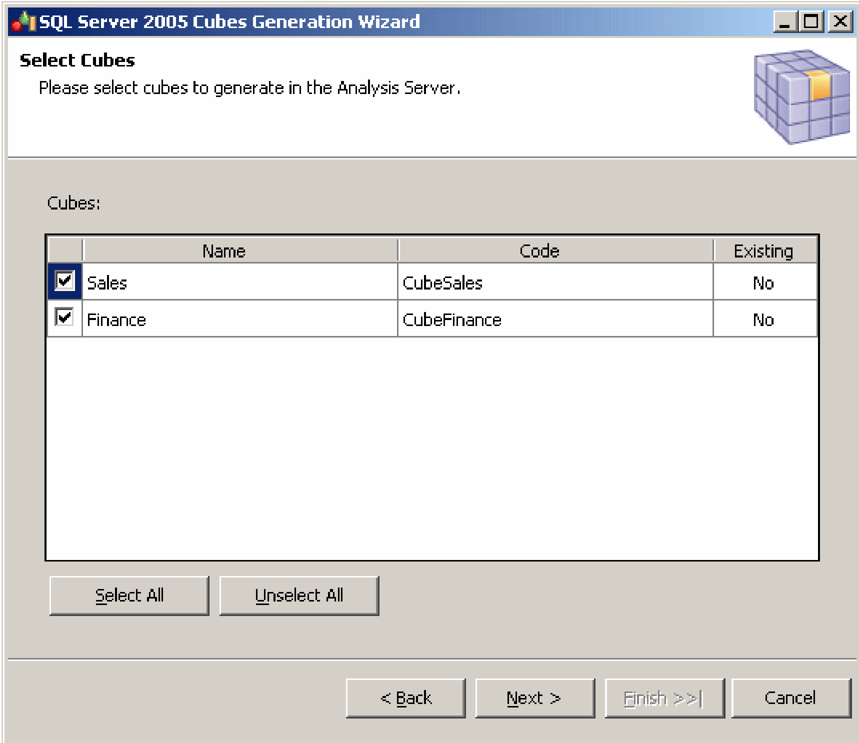
Click Next to continue.

2. Enter a server name, and select the database you want to generate to:



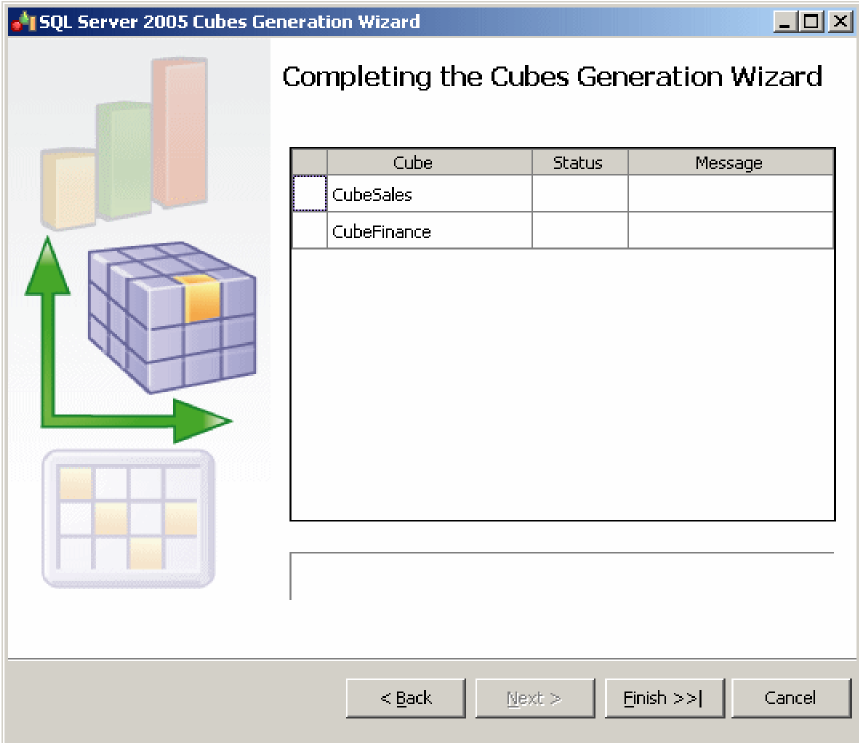
Click Next to continue.

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:



Click Next to continue.

4. The Generate Cubes page lists the cubes to be generated:



Click Finish to begin generation. Progress is displayed in the wizard, which will close automatically after successful completion.

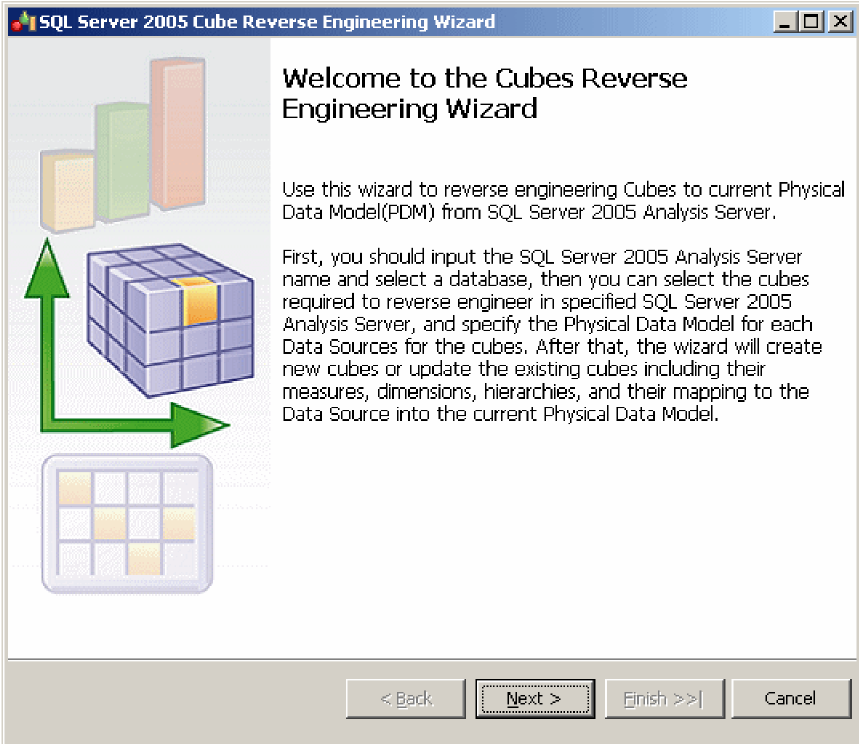
If a cube already exists in the database, it is dropped and recreated. If a related dimension already exists, it is reused. To fully generate a cube, your model must include a complete mapping to a table.

Reverse Engineering Microsoft SQL Server 2005 Cubes

The Microsoft SQL Server 2005 Analysis Services add-in enables the reverse engineering of cubes.

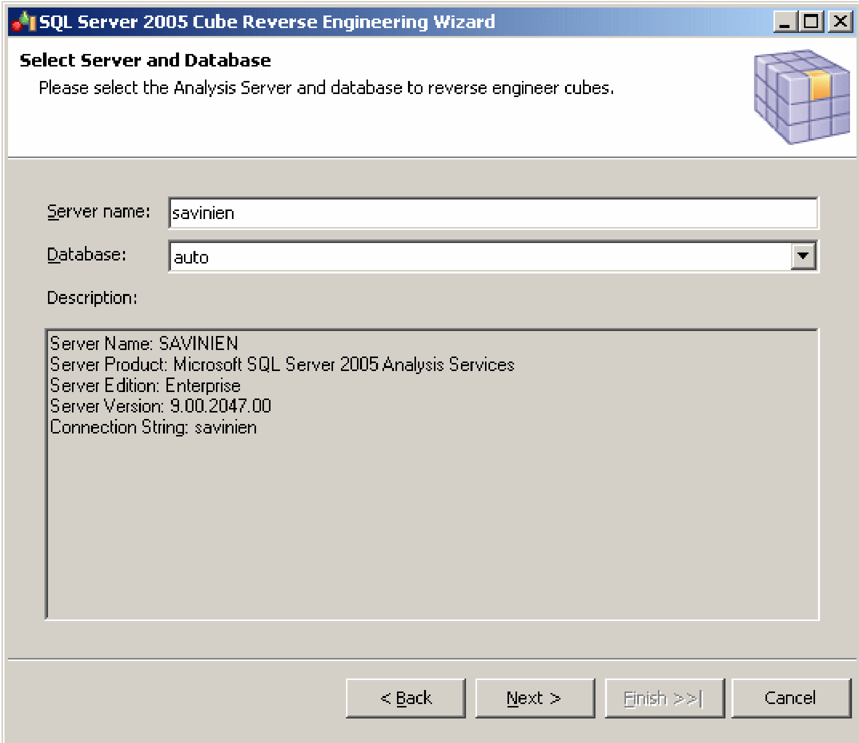
Before reverse-engineering cubes, you should create one or more PDMs to model the tables which provide its data. As part of the reverse-engineering process, PowerDesigner will create links from the reversed cubes to these tables.

1. Select **Tools** > **Microsoft SQL Server 2005 Analysis Services** > **Reverse Engineer Cubes** to open the wizard.



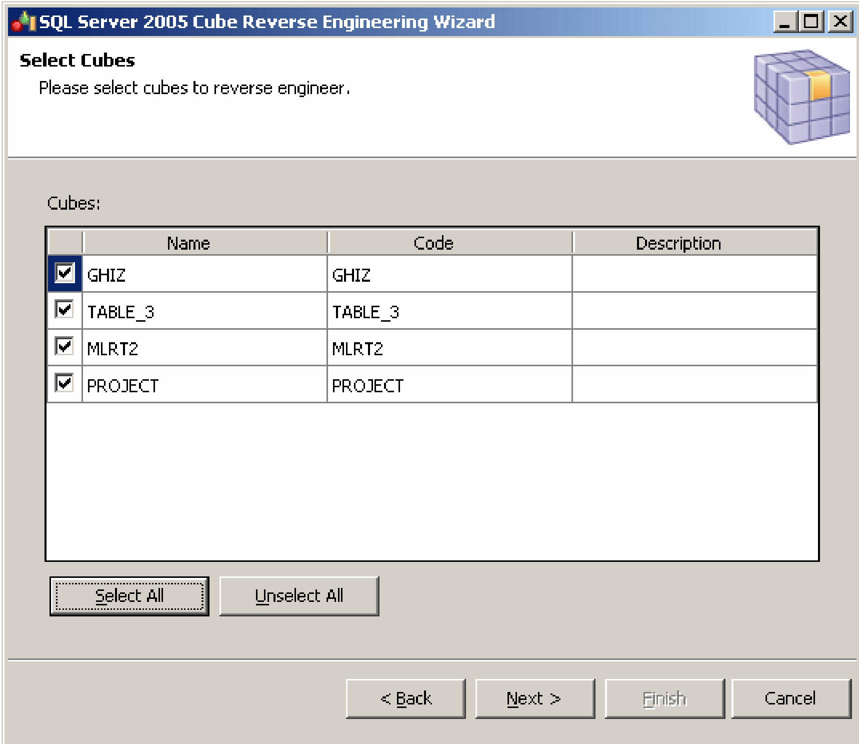
Click Next to continue.

2. Enter a server name, and select the database you want to reverse from:

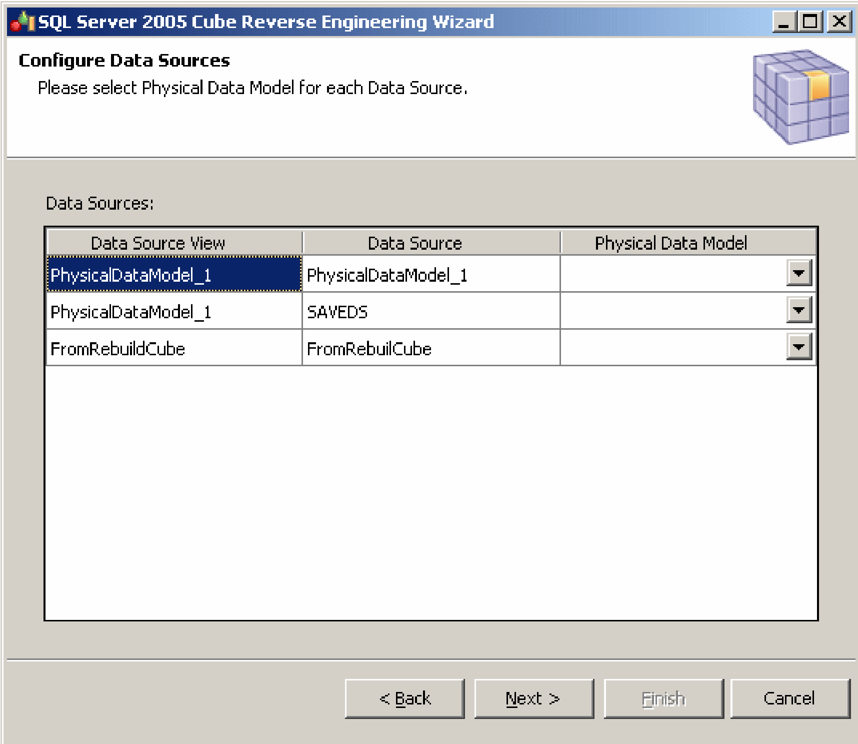


Click Next to continue.

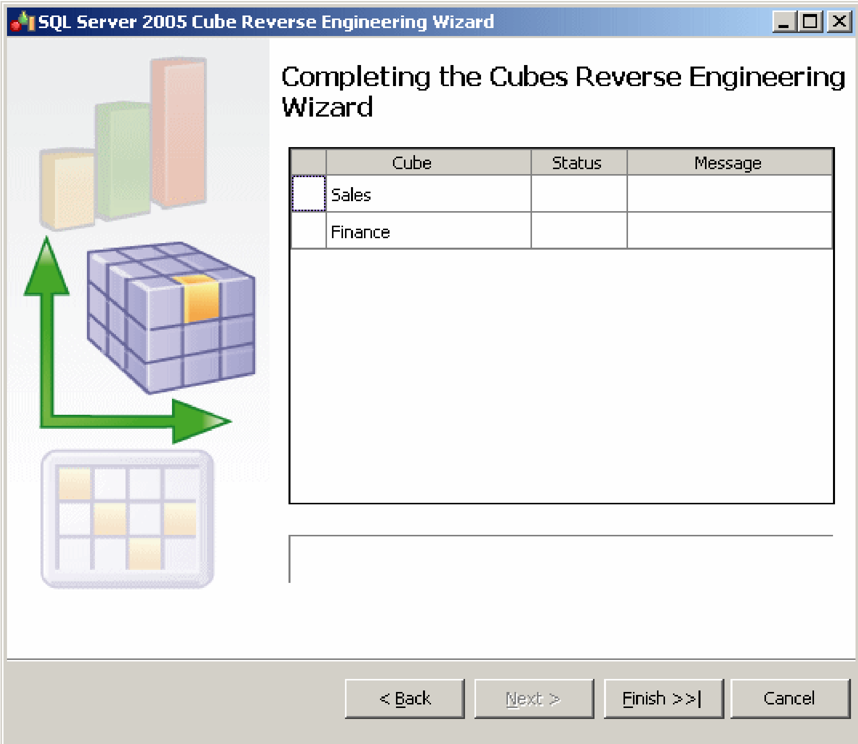
3. The Select Cubes page lists the available cubes. Select the cubes you want to reverse engineer and then click Next to continue:



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:



5. The Reverse Engineer Cubes page lists the cubes to be reversed:



Click Finish to begin reverse-engineering. Progress is displayed in the wizard, which will close automatically after successful completion.

CHAPTER 13 Netezza

To create a PDM with support for features specific to the Netezza DBMS family, select the appropriate version in the DBMS field of the New Model dialog. To view these extensions to the PowerDesigner metamodel in the Resource Editor, select **Database > Edit Current DBMS** and expand the **Profile** node.

The following sections list the extensions provided for Netezza.

Columns (v5.0 and higher)

The following extensions are available on the **Standard Checks** tab:

Name	Description
Default constraint name	Specifies the constraint name for default constraint Scripting name: DefaultConstName
Not null constraint name	Specifies the constraint name for not null constraint. Scripting name: NotNullConstName

Tables

The following extensions are available on the **Options** tab:

Name	Description
Distribution type	Specifies the type of row distribution. You can choose between None, hash, and random (on **General** tab for v4.5). Scripting name: Distribution
Columns	[hash or random distribution] Specifies the hash distribution columns (on **General** tab for v4.5). Scripting name: DistributeOnExplicitColumnList
Organize on	Specifies whether or not the table is organized. Scripting name: Organized
Columns	[organized table] Specifies the list of columns. Scripting name: OrganizedColumnList
Options	Displays the options defined for the table. Scripting name: TableOption

Databases (v5.0 and higher)

The following extensions are available on the **General** tab:

Name	Description
Character set	Specifies the default character set and collation. The default and only supported value is Latin9. Scripting name: Charset
Collation	The collation is binary. You cannot specify other values. Scripting name: Collation

Users/Groups (v5.0 and higher)

The following extensions are available on the **Options** tab:

Name	Description
SysId	Specifies the SYSID clause to choose the group ID of the new user/group. Scripting name: SysId
Owner	The user that created this user/group. Scripting name: Owner
Rowset limit	Specifies the maximum number of rows any query run by this user (or group) can return. Scripting name: RowsetLimit
Query timeout	Specifies the amount of time a query can run before the system sends the administrator a message. Scripting name: QueryTimeout
Session idle time-out	Specifies the amount of time a session can be idle before the system terminates it. Scripting name: SessionTimeout
Session priority	[group only] Specifies the default priority for the group. Scripting name: DefPriority
Default priority	[user only] Specifies the default priority for the user. Scripting name: DefPriority
Maximum priority	Specifies the maximum priority for the user/group. Scripting name: MaxPriority

Name	Description
Minimum resource	[group only] Specifies the minimum percentage of the system that a resource group will use when it has jobs. Scripting name: ResourceMinimum
Maximum resource	[group only] Specifies the maximum percentage of the system that a resource group can use. Scripting name: ResourceMaximum
Job maximum	[group only] Specifies the maximum number of concurrent jobs that a single resource group can run. Scripting name: JobMaximum
Password	[user only] Specifies the password used for database connection. Scripting name: PasswordDisplay
Valid until	[user only] Specifies the password validity. Scripting name: ValidUntil
Expire	[user only] Specify is the password expires on next connection. Scripting name: ExpirePassword
Authentication	[user only] Overrides the authentication for the user to LOCAL if specified. DEFAULT is the connection setting or whatever authentication is set. Scripting name: Authentication

Sequences (v5.0 and higher)

The following extensions are available on the **Options** tab:

Name	Description
Datatype	Specifies the data type. The value can be any exact integer type such as byteint, smallint, integer, or bigint. Scripting name: As
Start with	Specifies the starting value. Scripting name: StartWith
Increment	Specifies the increment value. The integer value can be any positive or negative integer, but it cannot be zero. Scripting name: IncrementBy

Name	Description
Minimum	Specifies the minimum value of the sequence. Scripting name: Minvalue
No min value	Results in a value of 1. Scripting name: NoMinvalue
Maximum	Specifies the maximum value of the sequence. Scripting name: Maxvalue
No max value	Results in the largest value for the specified datatype. Scripting name: NoMaxvalue
Cycle	Specifies whether the sequence continues to generate values after reaching either its maximum value (in an ascending sequence) or its minimum value (in a descending sequence). Scripting name: Cycle

History Configurations (Netezza)

History configurations provide support for query history logging. PowerDesigner models history configurations as extended objects with a stereotype of <<HistoryConfiguration>>.

Creating an History Configuration

You can create an history configuration in any of the following ways:

- Select **Model > History Configurations** to access the List of history configurations, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select **New > History Configuration**.

History Configuration Properties

You can modify an object's properties from its property sheet. To open an history configuration property sheet, double-click its Browser entry in the History Configurations folder.

The following extended attributes are available on the Options tab:

Name	Description
History type	Specifies the type of the database to create, which can be QUERY or NONE. Specify NONE to disable history collection. This is a required option which does not have a default value. Scripting name: Histtype
Data to collect	Specifies the history data to collect. Specify multiple values using comma-separated values, or click the **Select** tool to the right of the field to select them. Scripting name: Collect
Database / User / Password	Specifies the history database to which the captured data will be written, along with the user and password to use for accessing and inserting data. Scripting name: Database, User, Password
Load interval	Specifies the number of minutes to wait before checking the staged area for history data to transfer to the loading area. Scripting name: Loadinterval
Load retry	Specifies the number of times that the load operation will be retried. The valid values are 0 (no retry), 1 or 2. Scripting name: Loadretry
Minimum / Maximum threshold	Specify the minimum and maximum amounts of history data in MB to collect before transferring the staged batch files to the loading area. Values of 0 disable these threshold checks. Scripting name: Loadminthreshold, Loadmaxthreshold
Disk full threshold	This option is reserved for future use. Any value you specify will be ignored. The default value is 0. Scripting name: Diskfullthreshold
Storage limit	Specifies the maximum size of the history data staging area in MB. Scripting name: Storagelimit
Enable history	Specifies to log information about queries to the query history database. Scripting name: Enablehist
Enable system	Specifies to log information about system queries. A system queries accesses at least one system table but no user tables. Scripting name: Enablesystem

Name	Description
Version	Specifies the query history schema version of the configuration. The version must match the version number specified in the nzhistcreatedb command; otherwise, the loader process will fail. Scripting name: Version
Definition	Specifies the attribute that stores the object definition. Scripting name:ObjectDefn

CHAPTER 14 Oracle

To create a PDM with support for features specific to the Oracle DBMS family, select the appropriate version in the DBMS field of the New Model dialog. To view these extensions to the PowerDesigner metamodel in the Resource Editor, select **Database > Edit Current DBMS** and expand the **Profile** node.

Note: The DBMSs for Oracle v8-9 are deprecated.

When working with Oracle triggers, you can use the `TRGBODY` and `TRGDESC` variables. For information about working with variables, see *Customizing and Extending PowerDesigner > DBMS Definition Files > PDM Variables and Macros*.

The following table lists Oracle dimension objects and their equivalents in PowerDesigner:

Oracle object	PowerDesigner object
Dimension	Dimension (see *Dimensions (PDM)* on page 222)
Hierarchy	Dimension hierarchy (see *Hierarchies (PDM)* on page 225)
Level	Dimension attribute used in a hierarchy (see *Fact and Dimension Attributes (PDM)* on page 223)
Attribute	Dimension attribute used as detail attribute (see*Fact and Dimension Attributes (PDM)* on page 223)

The following sections list the extensions provided for Oracle.

Note: We do not provide documentation for the properties on the **Physical Options** and certain other tabs, though minimal information is available for them in the Resource Editor. For information about these properties, consult your DBMS reference documentation.

Note: In Oracle, the `storage` composite physical option is used as a template to define all the storage values in a storage entry to avoid having to set values independently each time you need to re-use them same values in a storage clause. For this reason, the Oracle physical option does not include the storage name (%s).

Abstract Data Types Attributes

The following extensions are available on the Oracle tab for attributes of abstract data types of type OBJECT or SQLJ_OBJECT:

Name	Description
Declare REF	Generates a REF modifier on attribute to declare references, which hold pointers to objects. Scripting name: RefAttribute

Columns

The following extensions are available on the Oracle tab:

Name	Description
Deferrable	[v11g and higher] Specifies that in subsequent transactions you can use the SET CONSTRAINT clause to defer checking of this constraint until after the transaction is committed. Scripting name: CheckDeferrable, NotNullDeferrable
Initially deferred	[v11g and higher] Specifies that Oracle should check this constraint at the end of subsequent transactions. Scripting name: CheckInitiallyDeferred, NotNullInitiallyDeferred
Deferred option of check constraint	[up to v10gR2] Defines the deferred option of a column constraint check. It is used in the definition or create and add items statements. Scripting name: ExtColumnDeferOption
Constraint name/ 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	[up to v10gR2] 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

Name	Description
Identified by Password	[v10gR2 and higher] Identifies by password. Scripting name: IdentifiedByPassword

XML Virtual Columns

If the table type is set to `XML`, the **Columns** tab is replaced by the **XML Virtual Columns** tab.The following extensions are available on the General tab of XML virtual columns:

Name	Description
Expression	Specifies the SQL expression used to compute virtual column value. Scripting name: Expression

Database Packages

The following extensions 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

Models

The following extensions are available on the Oracle tab:

Name	Description
Password Encryption	[v10gR2 and higher] Specifies the master key for encoding and decoding encrypted data. Scripting name: PasswordEncryption

References

The following extensions are available on the Oracle tab:

Name	Description
Deferred option	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 extensions are available on the Oracle tab:

Name	Description
Materialized view log	Specifies the materialized view log associated with the table. Scripting name: MaterializedViewLog

The following extensions are available on the XML properties tab (for v11g and higher) when the table type is set to XML:

Name	Description
Definition	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

Name	Description
Storage type	Specifies that XMLType columns can be stored in LOB, object-relational, or binary XML columns. Scripting name: XMLTypeStorage
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 extensions 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

Name	Description
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

Note: If you do not have a login "System", when reversing tablespaces via a live database connection, physical options will not be reversed. If you want to cancel the reverse engineering of tablespace physical options, you should clear the `SqlAttrQuery` query in the Tablespace category in the Oracle DBMS.

Users

The following extensions are available on the General tab (for v9i and higher):

Name	Description
Identification type	Specifies how the user will be identified. You can choose between: • by - requires a password • externally - requires a distinguished name • globally - requires a distinguished name Scripting name: Identification
Distinguished name	[external or global identification types] Specifies the user's distinguished name (DN) in the directory or certificate. Scripting name: DistinguishedName
Password	[by identification type] Specifies the user password. Scripting name: ClearPassword

The following extensions are available on the Options tab (for v9i and higher):

Name	Description
Default table-space	Specifies the default tablespace for objects that the user creates. Scripting name: DefaultTablespace
Temporary table-space	Specifies the tablespace or tablespace group for the user's temporary segments. Scripting name: TemporaryTablespace

Name	Description
Quota definition	Specifies the maximum amount of space the user can allocate in the tablespace. Scripting name: QuotaDefinition
Profile	Specifies the profile to assign to the user. Scripting name: Profile
Password expire	Specifies that the user's password will expire. Scripting name: PasswordExpire
Account lock	Select lock to lock the user's account and disable access or unlock to enable access to the account. Scripting name: AccountLock

Views

The following extensions 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

Object and SQLJ Object Data Types (Oracle)

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.

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 to display additional **Attributes** and **Procedures** tabs.
3. Enter as many attributes and procedures as appropriate.
4. Click **OK** to close the property sheet and return to your model.

   Once you have defined your data type, you can base a table on it by opening the table property sheet, selecting `Object` in the **Type** field, and then selecting your new data type in the **Based on** field.

Bitmap Join Indexes (Oracle)

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. You can design a bitmap join index either automatically or manually. For detailed information about bitmap join indexes, see your Oracle documentation.

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.

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.

Note: 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.

1. Select **Model > Join Indexes** to open the List of Join Indexes, click the **Add a Row** tool, enter a bitmap join index name in the **Name** column, and then click the **Properties** tool to open the new bitmap join index property sheet.
2. Select a base table on the **General** tab.

   Note: You can, alternately, create a bitmap join index from a table property sheet by clicking the **Add a Row** tool. In this case, the **Base table** field is set automatically.
3. Click the **References** tab, and then 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.
4. 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.
5. Click **OK** to complete the creation of the bitmap join index and return to the model.

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 - Lists the columns used for the 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 the columns to use. When you create a bitmap join index by rebuilding, the list of columns is initialized with all columns of the tables involved in the join except foreign keys.
- References - Lists the references used for the index.
- Physical Options - 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.

Database Packages (Oracle)

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.
- Improved *performance* since the entire package is loaded into memory when an object from the package is called for the first time.

You can generate and reverse engineer database packages in the same way as other database objects (see *Chapter 6, Generating and Reverse-Engineering Databases* on page 281). 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.

Creating a Database Package

You can create a database package in any of the following ways:

- Select **Model > Database Packages** to access the List of Database Packages, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New > Database Package**.

Database Package Properties

To view or edit a database package's properties, double-click its Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

Property	Description
Name/Code/ Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
Owner	Specifies the name of the database package owner, which you choose from the list of users.
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).
Table	Specifies the table with which the database package is associated.
Template	Specifies the template on which the database package is based (see *Database Package Templates* on page 471). If you use a template, then the remaining tabs of the property sheet will be completed by the template. If you make any modifications to the other tabs, then the **User-Defined** button to the right of the field is depressed and the package is detached from the template and will no longer be automatically updated when you modify the definition of the table with which it is associated.
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

The following tabs are also available:

- Procedures – Lists the procedures associated with the database package (see *Database Package Procedures* on page 466).
- Variables - Lists the variables associated with the database package (see *Database Package Variables* on page 467).
- Cursors - Lists the cursors associated with the database package (see *Database Package Cursors* on page 468).
- Exceptions – Lists the exceptions associated with the database package (see *Database Package Exceptions* on page 469).
- Types - Lists the types associated with the database package (see *Database Package Types* on page 470).

- Initialization - 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.
- Preview - Displays the SQL code that will be generated for the database package.

Database Package Procedures

You create database package procedures on the **Procedures** tab of a database package using the **Add a Row** tool. To copy a procedure from elsewhere in the model, use the **Create from Procedure** tool.

Note: To rebuild database package procedure dependencies (along with other procedure dependencies), select **Tools > Rebuild Objects > Rebuild Procedures Dependencies** (see *Rebuilding Trigger and Procedure Dependencies* on page 146).

To view or edit a database package procedure's properties, double-click its Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

Property	Description
Name/Code/Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
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.
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

The following tabs are also available:

- Parameters – Lists the input and output parameters required by the procedure (see *Database Package Parameters* on page 471).
- 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, and 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. You create database package variables on the **Variables** tab of a database package using the **Add a Row** tool.

To view or edit a database package variable's properties, double-click its Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

Property	Description
Name/Code/Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
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.

Property	Description
Default value	Default value of the variable.
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.
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

Database Package Cursors

A cursor is a multi-row query, which lets you name a work area and access its stored information. You create database package cursors on the **Cursors** tab of a database package using the **Add a Row** tool.

To view or edit a database package cursor's properties, double-click its Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

Property	Description
Name/Code/ Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
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.
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

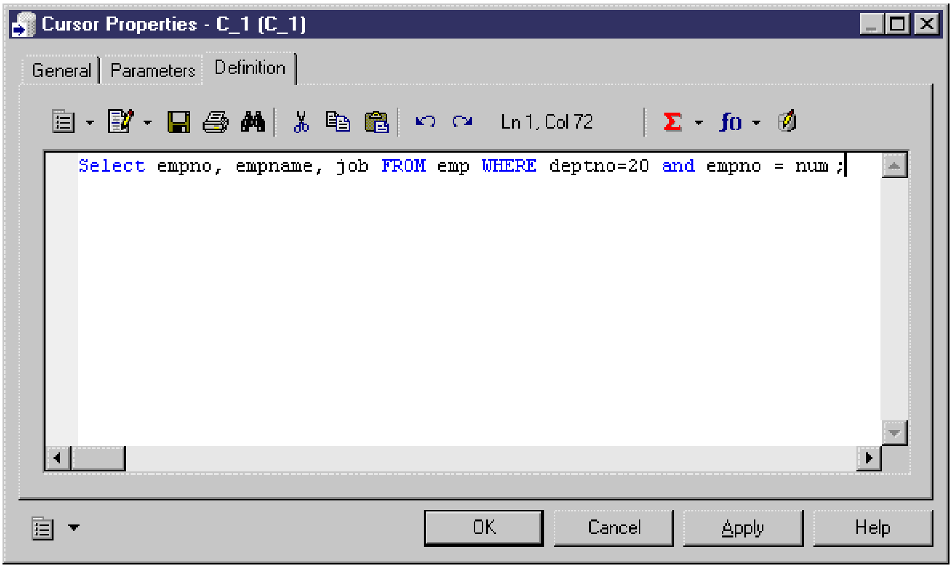
The following tabs are also available:

- Parameters – Lists the input and output parameters required by the cursor (see *Database Package Parameters* on page 471).

- Definition - 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 ;
```



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. You create database package exceptions on the **Exceptions** tab of a database package using the **Add a Row** tool.

To view or edit a database package exception's properties, double-click its Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

Properties	Description
Name/Code/ Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.

Properties	Description
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.
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

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 create database package types on the **Types** tab of a database package using the **Add a Row** tool.

To view or edit a database package type's properties, double-click its Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

Property	Description
Name/Code/ Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
DB Package	Name of the database package to which the type belongs.
Type	Allows you to declare the type as type or subtype. A subtype contains all the attributes and methods of the parent type, it can contain additional attributes and can override methods from the type.
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.
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

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 and cursors can use input and output parameters. For example, in a CREDIT procedure, you could define the parameters Account Number and Amount. You create database package parameters on the **Parameters** tab of a database package procedure or cursor using the **Add a Row** or **Insert a Row** tools.

To view or edit a database package parameter's properties, double-click its Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

Property	Description
Name/Code/ Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
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.
Keywords	Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas.

Database Package Templates

Instead of modeling each individual database package by hand, you can use a template and have PowerDesigner generate packages specific to each table. Database packages defined through a template are updated automatically when you make changes to the table definition, and you can quickly create packages for multiple tables from the Rebuild Table Database Packages dialog.

Database package templates are written in the PowerDesigner Generation Template Language (GTL). PowerDesigner provides a template for generating CRUD procedures, and you can create your own templates as necessary.

To define a database package from a template, simply select the template on the **General** tab of the database package property sheet.

Creating a Database Package Template

The available database package templates are defined in the DBMS resource file. Select **Database > Edit Current Database**, click the **Database Package Templates** tab. To create a database package template, click the **Add a Row** tool

Database Package Template Properties

To open a template property sheet, select it in the list and click the **Properties** tool.

The **General** tab contains the following properties:

Property	Description
Name/Code/Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
DBMS	Specifies the DBMS version.

The following tabs are also available:

- Definition - Contains a GTL template, which will generate a database package creation script based on the properties of the associated table. For detailed information about working with GTL, see *Customizing and Extending PowerDesigner > Customizing Generation with GTL* .

Rebuilding Table Database Packages

Database packages defined through templates are automatically updated when you modify the definition of the table with which they are associated. You can use the Rebuild Table Database Packages dialog to add database packages to tables that lack them or to overwrite any modifications you have made to packages associated with a template.

1. Select **Tools > Rebuild Objects > Rebuild Table Database Packages** to open the Rebuild Table Database Packages dialog.
2. Select a rebuild mode:

- **Delete and Rebuild** - deletes all table database packages associated with templates (including those which have been modified) and recreates them from the template
- **Add Missing Database Packages** - preserves existing database packages and creates packages only for those tables that lack them

3. Select the templates to use in the rebuild. You can select as many templates as necessary and the rebuild will create a database package for each template for each table.
4. [optional] Click the **Selection** tab and select the tables for which you want to rebuild database packages. By default all the tables in the model are selected.
5. Click **OK** to begin the rebuild.

Transparent Data Encryption (Oracle)

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.

Note: In order to access the master key used to encrypt the table keys, you must create a master password to open the wallet. To do this, right-click the model in the Browser, and select **Properties**. Click the **Oracle** tab, and enter your wallet password in the **Password Encryption** field. Click **OK** to return to the model. The password will be used to create alter statements for opening and closing the wallet.

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.

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 (Oracle)

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>>.

Note: 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 a 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. You can can define the following extended attributes for cluster columns:

Name	Description
Data type	Specifies the data type for the cluster index. Scripting name: Datatype
Length	Specifies the length for the cluster index. Scripting name: DatatypeLength

Name	Description
Precision	Specifies the precision for the cluster index. Scripting name: DatatypePrec
Sort	This clause instructs Oracle Database to sort the rows of the cluster on this column before applying the hash function. Scripting name: RowSort

- Indexes – lists the indexes defined for the cluster. You can can define the following extended attributes for cluster columns:

Name	Description
Owner	Specifies the owner of the cluster index Scripting name: Owner
Unique	Specifies whether the cluster index is unique. Scripting name: Unique
Bitmap	Specifies if the index is to be created with a bitmap for each distinct key, rather than indexing each row separately. Scripting name: Bitmap
Sort	By default, Oracle Database sorts indexes in ascending order when it creates the index. You can specify NOSORT to indicate to the database that the rows are already stored in the database in ascending order, so that Oracle Database does not have to sort the rows when creating the index. Scripting name: Sort

Database Links (Oracle)

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 a 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 extensions are available on the General tab:

Name	Description
Public	Specifies whether the database link is available to all users. If False, then the database link is private and is available only to you. Scripting name: Public

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 (Oracle)

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.

CHAPTER 15 SAP Sybase Adaptive Server Enterprise

To create a PDM with support for features specific to the SAP® Sybase® Adaptive Server® Enterprise DBMS family, select the appropriate version in the DBMS field of the New Model dialog. To view these extensions to the PowerDesigner metamodel in the Resource Editor, select **Database > Edit Current DBMS** and expand the **Profile** node.

Note: The DBMS definition file for Sybase AS Enterprise v12.5.3a is deprecated.

The following sections list the extensions provided for ASE.

Note: We do not provide documentation for the properties on the **Physical Options** and certain other tabs, though minimal information is available for them in the Resource Editor. For information about these properties, consult your DBMS reference documentation.

Tables

The following extensions are available on the Partitions tab (v15.0 and higher):

Name	Description
Partition	Indicates how records are distributed on table partitions. You must choose between: • Range - partitioned according to specified ranges of values in the partitioning column or columns (Scripting name: PartitionByRange). • Hash - partitioned by a system-supplied hash function (Scripting name: PartitionByHash). • List - partitioned according to literal values specified in the named column (Scripting name: PartitionByList). • Round robin - partitioned in a sequential manner (Scripting name: PartitionByRoundrobin). Each of the partitioning methods enables a list of partitions for you to complete, except round robin by partition number, which requires only that you specify the number of available partitions on a particular storage. Scripting name: Partition
Columns	[range and hash] Specifies an ordered list of columns used to determine into which partition a row belongs. Scripting name: PartitionByRangeColumnListColumn, PartitionByHashColumnListColumn

Name	Description
Column	[list] Specifies the column used to determine into which partition a row belongs. Scripting name: PartitionByListColumnColumnName
List	[round robin] Specifies the table partitions Scripting name: PartitionByRoundrobinSegmentEnumOnAbsence
Partition number	[round robin] Specifies the number of partitions for the table. Scripting name: PartitionByRoundrobinSegmentEnumOnPresence
Quantity	[round robin by partition number] Number of partitions for the table Scripting name: PartitionByRoundrobinSegmentEnumPartitionNum
Storage (segment)	[round robin by partition number] Specifies the name of the segment on which to place the table partition. Scripting name: PartitionByRoundrobinSegmentEnumOnSegmentName
[list of partitions]	[all but round robin by partition number] Specifies the list of partitions to be used Scripting name: PartitionByRangePartitionListPartitionDefinition, PartitionByHashPartitionListPartitionDefinition, PartitionByListPartitionListPartitionDefinition,PartitionByRoundrobinPartitionListPartitionDefinition

Columns

The following extensions 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	[v15.0 and higher] Specifies that the computed column is materialized. Scripting name: Materialized
Encrypted	[v12.5.3a and higher] Specifies that the column is encrypted. Enabled only for columns with a datatype that supports encryption. Scripting name: Encrypted

Name	Description
Encryption key	[v12.5.3a and higher] Specifies an encryption key. Use the tools to the create or select a key (see *Encryption Keys* on page 483). Scripting name: EncryptionKey
Default decrypt value	[v15.5.0 and higher] Specifies the default constant value that is returned to users who do not have decrypt permissions. Scripting name: DecryptDefault
Compressed	[v15.7 and higher] Specifies that the data in the column is compressed. Scripting name: Compressed
Compression Level	[v15.7 and higher] Specifies the level of column data compression. Scripting name: CompressionLevel

Databases

The following extensions are available on the General tab:

Name	Description
For cluster	[v15.5.0 and higher] Specifies that the database will support clustering. Scripting name: ForCluster
Type	[v15.5.0 and higher] Specifies the whether the database is of type: • [for standard databases] `inmemory`, `temporary`, or `inmemory temporary` • [for cluster databases] `temporary`, `global temporary`, or `system temporary` . Scripting name: DatabaseType

Keys

The following extensions are available on the Sybase tab:

Name	Description
Key index is descending	[v12.0 and higher] Specifies if the index created for a constraint is to be created in descending order for each column. Scripting name: DescKey

Model

The following extensions 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 extensions 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 extensions 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

Proxy Tables (ASE)

Sybase supports modeling for Sybase ASE proxy tables.

For more information, see *Proxy Tables (ASE/SQL Anywhere)* on page 548.

Encryption Keys (ASE)

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 a 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	Specifies the owner of the encryption key. Scripting name: Owner
Key length	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	Specifies the algorithm used to generate the encryption key. Currently, Advanced Encryption Standard (AES) is the only algorithm supported. Scripting name: Algorithm

Name	Description
Initialization vector	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	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
Default encryption key	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

The following tabs are also available:

- Key Copies - [v15.0.2 and higher] ASE allows users to access encrypted columns using their copy of a single key. A key copy is designated for an individual user with a private password known only to the user, ASE does not save the passwords on disk, so that even the SA cannot access the protected data. PowerDesigner models key copies as extended sub-objects with a <<KeyCopy>> stereotype, and the following extensions are available on the Sybase tab of its property sheet:
  - User - identifies the user for whom the key copy is made.
  - Password - specifies the password used to encrypt the key copy.

CHAPTER 16 SAP Business Suite

To create a PDM with support for features specific to the SAP Business Suite, select the DBMS on which your installation is running in the DBMS field of the New Model dialog, click **OK** to create an empty PDM, and then select **Tools > SAP Business Suite > Import SAP Business Suite Data Dictionary**. To view these extensions to the PowerDesigner metamodel in the Resource Editor, select **Model > Extensions**, select the `SAP Business Suite` extension, click the **Properties** tool, and expand the **Profile** node.

PowerDesigner supports importing any recent version of SAP® Business Suite.

The following sections list the extensions provided for SAP Business Suite.

Model

The following extensions are available on the **Data Dictionary** tab:

Name	Description
Host name	Specifies the host name or IP address of the server on which the Business Suite installation is running. Scripting name: HostName
User name	Specifies the user who connects to the Business Suite server. Scripting name: UserName

ABAP Components

ABAP components are based on standard PowerDesigner packages with a `ABAP Component` stereotype. The following extensions are available on the **Data Dictionary** tab:

Name	Description
Created by	Specifies the user who created the object and when the change was made. Scripting name: CreatedBy, CreatedOnDate, CreatedOnTime
Changed by	Specifies the user who last changed the object and when the change was made. Scripting name: ChangedBy, ChangedOnDate, ChangedOnTime
Parent	Specifies the ABAP component that is the parent of the present component. Scripting name: Parent

Name	Description
Customized by / Release notes by	Specify the users who customized the component and wrote the release notes for the customization. Scripting name: CustomizingResponsible, ReleaseNoteResponsible
Released	Provides release information about the component. Scripting name: Released
Support web / desktop	Specify that the component can be displayed in the SAP NetWeaver Portal and in the desktop NetWeaver client. Scripting name: SupportWeb, SupportDesktop
Visible	Specifies that the component should be visible to users. Scripting name: Visible

ABAP Packages

ABAP packages are based on standard PowerDesigner packages with an `ABAP Package` stereotype. The following extensions are available on the **Data Dictionary** tab:

Name	Description
Created by	Specifies the user who create the object and when it was created. Scripting name: CreatedBy, CreatedOnDate, CreatedOnTime
Changed by	Specifies the user who last changed the object and when the change was made. Scripting name: ChangedBy, ChangedOnDate
Parent	Specifies the ABAP component or package that is the parent of the package. Scripting name: ApplicationComponent
Software component	Specifies the software component to which the package is a support package. Scripting name: SoftwareComponent
Main package	Specifies whether the package is a: • <empty> - Development Package • X - Main Package, which does not contain any development elements itself; and provides a structure for its children • S - Structure Package Scripting name: MainPackage

Name	Description
Namespace	Specifies a deprecated method for organizing package structures. Scripting name: Namespace
Owner	Specifies the user currently responsible for the package. Scripting name: Owner

Tables/Structures

The following extensions are available on the **Data Dictionary** tab:

Name	Description
Changed by	Specifies the user who last changed the object and when the change was made. Scripting name: ChangedBy, ChangedOnDate, ChangedOnTime
Parent package	Specifies the ABAP package that is the parent of the table or structure. Scripting name: PackageCode
Physical table	Specifies the database table on which the table or structure is based. Scripting name: PhysicalTableCode
Is extended	Specifies that the table contains extensions. Scripting name: IsExtended

Columns (Fields)

Business Suite fields are represented as columns in PowerDesigner. The following extensions are available on the **Data Dictionary** tab:

Name	Description
Field type	Specifies whether the field is: • <Empty> - Built-in type • E - Data element • S - Structure • L - Table type Scripting name: FieldType
Domain	Specifies the domain attached to the column. Scripting name: Domain

Name	Description
ABAP data type	Specifies the data type used by the runtime environment: • B, S, I - 1-byte, 2-byte, or 4-byte integer • C, N, B, F, G - Character, numerical, binary, float, or string • D, T - Date, or time • X, Y - Fixed or variable length raw Scripting name: ABAPDataType
Data dictionary data type	Specifies the data type used in the Dictionary. Scripting name: DataDictionaryDataType
Data element	Specifies the data element attached to the field, if of type E. Scripting name: DataElement
Include structure	Specifies the included structure attached to the field, if of type S. Scripting name: IncludeStructure
Lookup table	Specifies that lookup table from which to draw key values, if of type T. Scripting name: LookupTable
Reference field / table	For numerical or currency columns, specify the reference field and table. Scripting name: ReferenceField, ReferenceTable
Input help	Specifies the origin of input help: • <empty> - No input help exists • X - Explicit search help attachment to field • P - Input help implemented with check table • D - Explicit search help attachment to data element • F - Input help with fixed values • T - Input help based on data type Scripting name: InputHelp

Indexes

The following extensions are available on the **Data Dictionary** tab:

Name	Description
Changed by	Specifies the user who last changed the object and when the change was made. Scripting name: ChangedBy, ChangedOnDate, ChangedOnTime

Name	Description
DB index name	Specifies the associated database index name. Scripting name: DBIndexName
DB include exclude	Specifies that a list of database systems is used as: • I - List of inclusions: create index on these DB systems. • E - List of exclusions: do not create index on these DB systems. Scripting name: DBIncludeExclude
List of database systems 1-4	Specify lists of database systems for inclusion or exclusion by the index. Scripting name: DBSYSSEL1, DBSYSSEL2, DBSYSSEL3, DBSYSSEL4
Extension index	Specifies that the index is an extension index. Scripting name: IsExtensionIndex
Status	Specifies the status of the index in the database: • <empty> - Create on database. • O - Do not create on database. • D - Create on database depending on DB Scripting name: Status

Data Elements

Data elements are based on PowerDesigner extended objects with a `Data Element` stereotype. The following extensions are available on the **Data Dictionary** tab:

Name	Description
Changed by	Specifies the user who last changed the object and when the change was made. Scripting name: ChangedBy, ChangedOnDate, ChangedOnTime
Package code	Specifies the package containing the data element. Scripting name: PackageCode
Default name	Specifies the default name for components using the data element. Scripting name: DefaultName
Original language	Specifies the language in which the data element was defined. Scripting name: OriginalLanguage

Name	Description
Data dictionary data type	Specifies the data type of the column in terms of the ABAP Dictionary. Scripting name: DataDictionaryDataType, DataType
Length / Output length	Specifies the supported number of characters and the number that can be displayed in ABAP forms. Scripting name: OutputLength, Length
Precision	Specifies the supported number of decimal places. Scripting name: Precision
Reference kind	Specifies the category of dictionary type: • <empty> - Direct type • E - Elementary type • S - Structured type • L - Table type • R - Reference type • D - Domain Scripting name: ReferenceKind
Conversion routine	Specifies function modules that are executed when values are input to and displayed in the ABAP screen field. Scripting name: ConversionRoutine
Value table	Specifies that the permitted values for the data element are PK values of the selected table. Scripting name: ValueTable
Signed	Specifies that negative values are supported. Scripting name: Signed
Lowercase	Specifies that lowercase letters are supported. Scripting name: Lowercase
Fixed values	Specifies that permitted values are limited to those specified. Scripting name: FixedValues

Domains

The following extensions are available on the **Data Dictionary** tab:

Name	Description
Changed by	Specifies the user who last changed the object and when the change was made. Scripting name: ChangedBy, ChangedOnDate, ChangedOnTime
Package code	Specifies the package containing the domain. Scripting name: PackageCode
Data dictionary data type	Specifies the data type of the column in terms of the ABAP Dictionary. Scripting name: DataDictionaryDataType
Base domain	Specifies the domain that the present domain extends. Scripting name: BaseDomain
Value table	Specifies that the permitted values for the domain are PK values of the selected table. Scripting name: ValueTable
Conversion routine	Specifies function modules that are executed when values are input to and displayed in the ABAP screen field. Scripting name: ConversionRoutine
Fixed values	Specifies that permitted values are limited to those specified. Scripting name: FixedValues
Signed	Specifies that negative values are supported. Scripting name: Signed

Views

The following extensions are available on the **Data Dictionary** tab:

Name	Description
Changed by	Specifies the user who last changed the object and when the change was made. Scripting name: ChangedBy, ChangedOnDate, ChangedOnTime
Root table code	Specifies the primary table of an aggregate. Scripting name: RootTableCode

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>View type</td><td>Specifies that the view is a:
• H - Help view
• D - Database view
• P - Projection view
• S - Structure view, data selection not possible
• C - Maintenance view
• E - Entity view (no longer supported)
• V - View variant
• A - Append view
Scripting name: ViewType</td></tr>
<tr><td>Delivery class</td><td>Specifies that the delivery class of the view is:
• A - Application table
• C - Customer table, maintained by customer
• L - Table for storing temporary data
• G - Customer table, SAP can add rows
• E - Control table
• S - System table, maintained by SAP
• W - System table
Scripting name: DeliveryClass</td></tr>
<tr><td>Maintenance status</td><td>Specifies the maintenance status of the view:
• <empty> - Modifiable
• R - Read only
• U - Read and change
• M - Time dependent view
Scripting name: MaintenanceStatus</td></tr>
</table>

View Columns (View Fields)

The following extensions are available on the **Data Dictionary** tab:

Name	Description
ABAP form name	Specifies the field's name in ABAP forms. Scripting name: ABAPFormName

Name	Description
Base table / field	Specify the table and field from which the field are drawn. Scripting name: BaseTable, BaseField
Data element	Specifies the data element attached to the field. Scripting name: DataElement
Is key	Specifies that the field belongs to a key area. Scripting name: IsKey
Lock mode	Specifies the lock mode for the field: • E - Write lock • S - Read lock • X - Exclusive lock Scripting name: LockMode
Maintenance status	Specifies the maintenance status of the field: • <empty> - View is available as normal • R - View field can only be read • S - View field is used to form subsets • H - View field is not transferred to the maintenance screens Scripting name: MaintenanceStatus

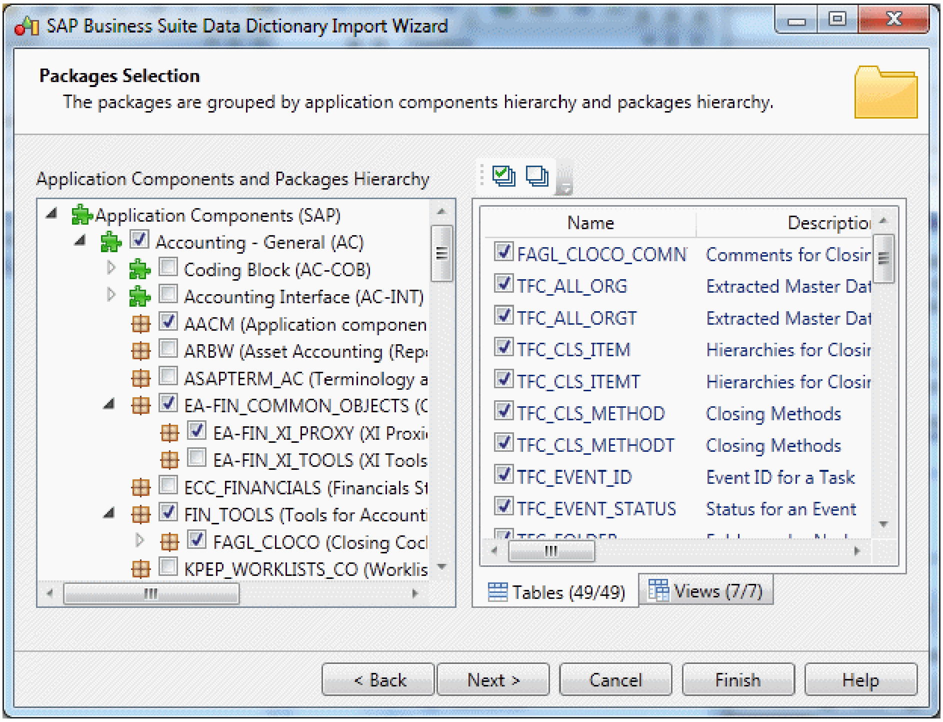
Importing an SAP Business Suite Data Dictionary

An SAP Business Suite installation is built on a complex database structure, which comprises many thousand tables with often cryptic names, and may include large numbers of extensions. In certain environments, there may be multiple servers, each with different extensions. PowerDesigner allows you to browse the application component and packages in the hierarchy, and to import them and their supporting logical objects for analysis, comparison, and merging of data dictionaries.

1. Create a new PDM targeting the DBMS hosting your Business Suite server. For a server running SAP MaxDB, use the SAP HANA DBMS.
2. Select **Tools > SAP Business Suite > Import SAP Business Suite Data Dictionary** to open the wizard, and click **Next** on the Welcome page.
3. Enter your Business Suite connection parameters and then click **Next** to connect.

Use the tools to the right of the **Connection name** field to create a new connection profile, review the properties of the existing profile, or delete it. Business Suite connection profiles are stored in the registry.

4. Select application components and packages in the left pane to add their tables and views to the import and then click **Next**.

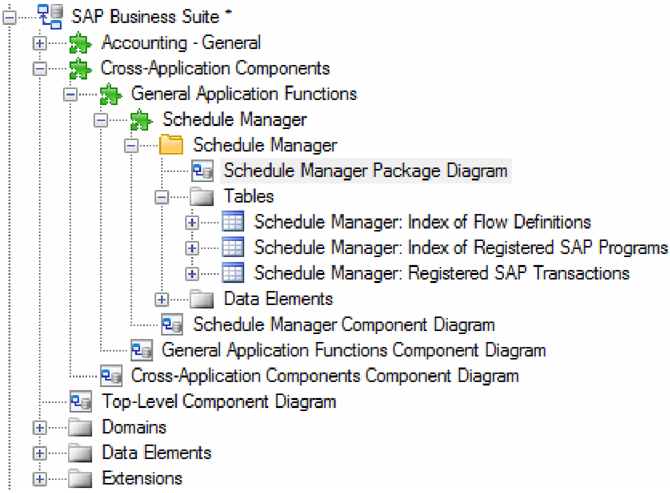


When you select a component or package to import, its supporting tables and views are added to the subtabs in the right pane, and the total number of tables and views to be imported is updated. Tables are selected for import by default, but views are not. You can select or deselect tables and views for import as necessary.

Note: Since an ERP system can contain many thousands of tables, we recommend that you import only a limited subset of components or packages at a time. You can relaunch the wizard and import additional components or packages as many times as necessary.

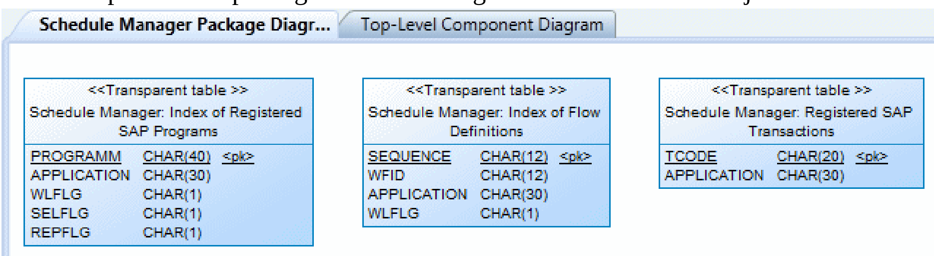
5. Review the objects that will be imported and then click **Finish**.

The component and package structure is imported, with tables located in their appropriate package, and global domains and data elements are listed at the root of the model.



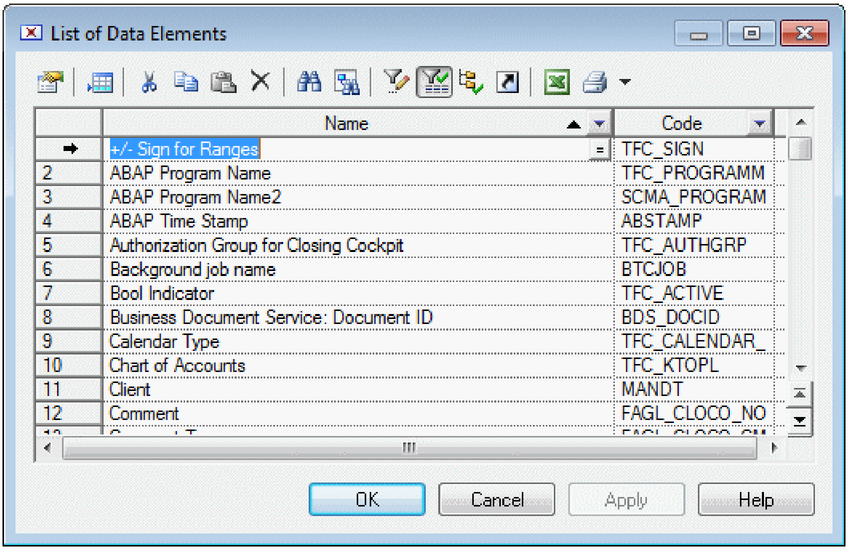
PowerDesigner stores the technical name of each object in its **Code** field, and uses the more intuitive short description of the object as its name. Thus, for example, the table identified as `SCMATRANSACT` in the data dictionary is displayed as `Schedule Manager: Registered SAP Transactions` in PowerDesigner.

Each component and package contains a diagram which shows the objects it contains:



Note: Not all packages contain tables. To view the structure of only those components and packages with diagrams that do contain tables, right-click the model in the Browser or a diagram background and select **View ABAP Diagrams Containing Tables** . Select a diagram in the tree and click **OK** to open it.

6. Review the imported metadata as appropriate. Configurable and filterable lists of each type of object are available from the **Model** menu. For example, to display the List of Data Elements, select **Model** > **Data Elements**:

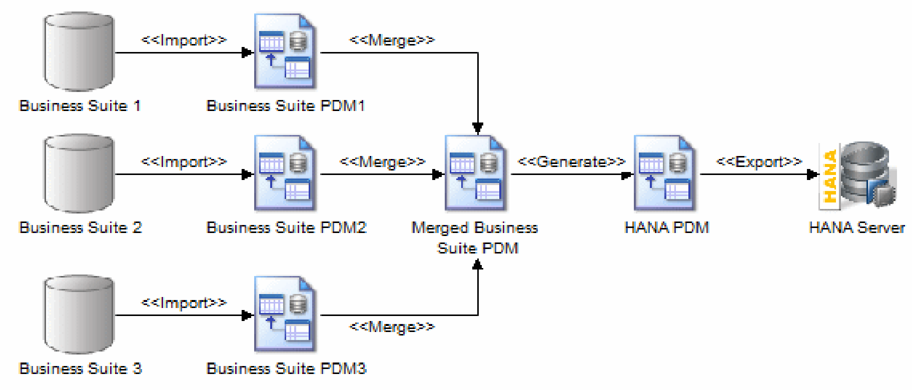


Note: To view lists of global objects, such as domains and data elements, ensure that you are in the context of the model root (by double-clicking the `Top-Level Component Diagram`) before opening the list. To view all the components, packages, or tables in the model, ensure that you are at root, open the list, and click the **Include Sub-Packages** tool in the list toolbar. For detailed information about working with object lists, see *Core Features Guide > Modeling with PowerDesigner > Objects > Object Lists*

7. [optional] Perform a new import to enrich your model. You can perform as many imports as necessary, and delete components, packages, or other objects as appropriate, to simplify your model and focus on the areas that interest you.
8. [optional] To compare two or more Business Suite installations, import each one into its own PDM, and select **Tools > Compare Models**. For detailed information about working in this dialog, see *Core Features Guide > Modeling with PowerDesigner > Comparing and Merging Models*.
9. [optional] PowerDesigner supports the merging of Business Suite PDMs and their generation to HANA to provide the basis for establishing a business intelligence environment for reporting on your Business Suite transactional data (see *Generating an SAP Business Suite Data Dictionary to HANA* on page 497).

Generating an SAP Business Suite Data Dictionary to HANA

PowerDesigner can help you prepare a HANA table structure to allow consolidated reporting on one or more SAP Business Suite installations.



1. Create a PDM for each SAP Business Suite installation, and import the logical tables that you want to define warehouse reporting on (see *Importing an SAP Business Suite Data Dictionary* on page 493).
2. Analyze and purify your models, deleting components, packages, tables, and columns that are not of interest to your reporting project.

   Note: You should not edit the properties of Business Suite objects (except for the **Comment** field or **Notes** tab) or create new objects, in order to ensure the integrity of the metadata that will be generated to the HANA schema.

3. Select a model to act as the core warehouse model, and then select **Tools > Merge Models** and merge the other models into it one after the other to create a superset of all the components, packages, tables, and columns that you want to generate to HANA.

   For detailed information about merging models, see *Core Features Guide > Modeling with PowerDesigner > Comparing and Merging Models*.

4. Select **Tools > SAP Business Suite > Generate HANA Physical Data Model**.

   PowerDesigner generates a new PDM targeting the HANA DBMS.

   Note: PowerDesigner generates the Business Suite component and package structure to the HANA PDM. This structure does not represent HANA packages, and your tables will all be exported to the HANA catalog.

5. Export your tables to your HANA server (see *Exporting Objects to the HANA Repository* on page 504).

Implement loading of your transactional data to your HANA warehouse using your standard ETL solution.

CHAPTER 17 SAP HANA

To create a PDM with support for features specific to the SAP HANA® DBMS family, select the appropriate version in the DBMS field of the New Model dialog. To view these extensions to the PowerDesigner metamodel in the Resource Editor, select **Database > Edit Current DBMS** and expand the **Profile** node.

PowerDesigner supports round trip reverse-engineering and generation of SAP HANA® v1.0 SP05 and SP06 tables and analytic and attribute views.

The following sections list the extensions provided for SAP HANA.

Tables

The following extensions are available on the **General** tab:

Name	Description
Type	Specifies the table type. You can choose between: • Row - [default] If the majority of table access involves selecting a few records, with all attributes selected, ROW-based storage is preferable. • Column - If the majority of table access will be through a large number of tuples, with only a few selected attributes, COLUMN-based storage should be used. • History column - Creates a table with a session type HISTORY, to support time travel queries, which are queries against historical states of the database. • Global temporary - The table definition is globally available while data is visible only to the current session. The table is truncated at the end of the session. • Local temporary - The table definition and data is visible only to the current session. The table is truncated at the end of the session. Scripting name: FullType

The following extensions are available on the **HANA** tab:

Name	Description
Logging type	Specifies whether table logging is activated. You can choose between: • logging - [default] • nologging - specifies that logging is deactivated. As a result, the definition of the table is persistent and globally available and data is temporary and global. The resource manager should therefore explicitly drop a NOLOGGING table. Scripting name: LoggingType
Retention period	[if nologging] Specifies the retention time in seconds of the table created as nologging. Scripting name: Retention
Auto-Merge	Specifies that automatic delta merge is triggered. Scripting name: AutoMerge
Unload priority	Specifies the priority for unloading the table from memory from 0 to 9. Scripting name: UnloadPriority
Schema flexibility	Specifies that the table schema is flexible. Scripting name: WithSchemaFlexibility
Location	Specifies that partitions will be created on the listed instances using round robin scheme. Scripting name: PartitionLocation
Multiple	Specifies that the location targets multiple HANA instances. Scripting name: HasMultipleLocations
Options text	Specifies the SQL text of the table options. Options entered here will be set in their relevant fields, and changes to the fields are reflected here. Scripting name: FullTableOptions

The following extensions are available on the **Partitions** tab:

Name	Description
Type	Specifies the partition scheme type: • Hash - Distributes rows to partitions equally for load balancing and to overcome the 2 billion row limitation. Specify an expression listing the columns to partition on and the number of partitions to create. You may specify a second scheme of type Hash or Range. • Range - Creates partitions for specific values or value ranges. Specify an expression and range specifier. • RoundRobin - Distributes rows to partitions equally without specifying partitioning columns. Specify the number of partitions to create. You may specify a second scheme of type Range. Scripting name: FirstPartitionElement, etc

Columns

The following extensions are available on the **Detail** tab:

Name	Description
Stored as	Specifies the stored data type. Scripting name: StoreDataType
DDIC type	Specifies the application data type. Scripting name: DDICDataType

Indexes

The following extensions are available on the **General** tab:

Name	Description
Descending	Specifies that the index should be created in descending order. Scripting name: DescIndex

Keys

The following extensions are available on the **General** tab:

Name	Description
Key type	Specifies the key type. Scripting name: KeyType

Roles

The following extensions are available on the **General** tab:

Name	Description
Global visibility	Specifies that the role is available globally. Scripting name: GlobalVisibility
Global ID	[if global visibility] Specifies the external role name for the global user. Scripting name: GlobalID

References

The following extensions are available on the **HANA** tab:

Name	Description
Cardinality	Specifies the type of cardinality. Scripting name: HANACardinality
Join type	Specifies the join type. Scripting name: HANAJoinType
Language Column	Specifies the language column. Scripting name: HANALanguageColumn

Users

The following extensions are available on the **General** tab:

Name	Description
Identification	Specifies the type of identification (global, local or external). Scripting name: Identification
Distinguished name	Specifies the user's distinguished name (DN) in the directory or certificate. Scripting name: DistinguishedName
Password	Specifies the clear copy of the password. Scripting name: CopyPassword
Implicit Schema	Specifie"s that the database generation will use the stored procedure sp_grantdbaccess instead of a create user statement. Scripting name: ImplicitSchema
Default Schema	Specifies the first schema searched to resolve the names of objects for this user. Scripting name: DefaultSchema

Packages

The following extensions are available on the **HANA** tab of HANA packages:

Name	Description
Structure package	Specifies that the package is a structural package Scripting name: Structural
Object Name	Specifies the HANA object name. Scripting name: _ObjectName_

Facts (Analytic Views) and Dimensions (Attribute Views)

The following extensions are available on the **HANA** tab:

Name	Description
Default Client / Language / Member	Specify the HANA default client, language, and (dimension only) member. Scripting name: DefaultClient, DefaultLanguage, DefaultMember
Multidimensional reporting	[facts] Specifies that multidimensional reporting is enabled. Scripting name: MultidimensionalReporting
Package / Name / Version	Specifies the HANA package, object name, and version. Scripting name: _ObjectPackage_, _ObjectName_, _ObjectVersion_
Last Updated Date / at	Specifies when the dimension or fact was last edited. Scripting name: _LastUpdatedDate_, _LastUpdatedTime_

Dimension Attributes and Fact Attributes

The following extensions are available on the **HANA** tab:

Name	Description
Default Member / Info Object	Specify the HANA default member and info object. Scripting Name: DefaultMember, InfoObject
Drill Down Enabled	Specifies the drill down is enabled for the attribute. Scripting Name: DrillDownEnabled
Hidden	Specifies that the attribute is hidden. Scripting Name: IsHidden

Name	Description
Key Attribute / Principal Key / Attribute Hierarchy Active	[Dimension attribute only] Specify that the attribute is a key attribute, a principal key attribute, and that the attribute hierarchy is active. Scripting Name: KeyAttribute, PrincipalKey, AttributeHierarchyActive
Data Type / Length / Scale	Specify the data type, length and scale of the attribute. Scripting Name: AttributeDataType, Length, AttributeScale

Fact Measures

The following extensions are available on the **HANA** tab:

Name	Description
Data Type / Length / Scale	Specify the data type, length and scale of the measure. Scripting Name: MeasureDataType, MeasureLength, MeasureScale

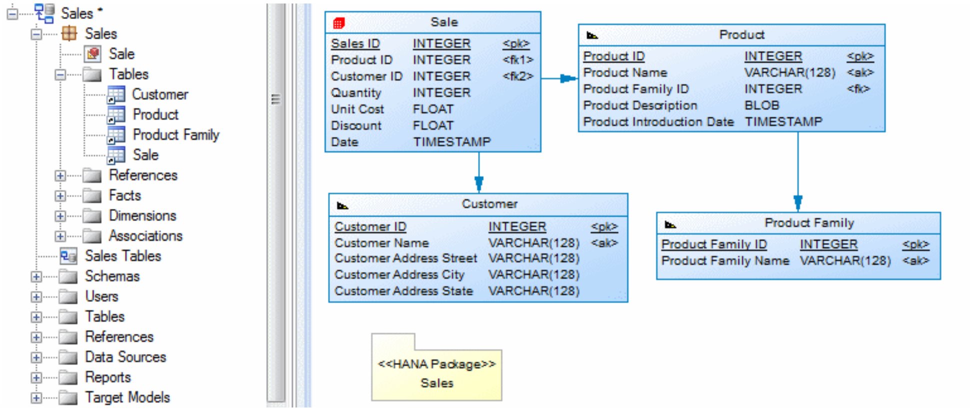
Exporting Objects to the HANA Repository

While HANA tables are generated directly to the catalog, analytic and attribute views are exported to the HANA repository from where they will be deployed. PowerDesigner provides a wizard to allow you to export your views and tables to the HANA repository and catalog respectively in a single action.

Note: This feature requires a 32-Bit Java installation.

In your PDM, the HANA catalog is represented by the root of the model, while the HANA repository is represented by a structure of HANA packages. In order to generate your tables and views correctly, you must place your tables at the root (or in standard PowerDesigner packages), and your facts (analytic views) and dimensions (analytic views) in their appropriate HANA packages.

In the following example, the tables in the `Sales Tables` physical diagram are at the root of the model, and appear as shortcuts inside the `Sales` Hana package, which contains the corresponding fact and dimensions:



Tables and analytic and attribute views imported from HANA are automatically placed at the root and in HANA packages as appropriate. When generating cubes from tables in your model (see *Generating Cubes* on page 217), launch the wizard from within a HANA package. If you have generated cubes at the model root, drag the multidimensional diagram into a HANA package to move its contents.

1. Select **Database > Apply Model Changes to HANA Repository** to open the wizard, and click **Next** on the Welcome page.

   The wizard checks your model for consistency and displays any errors which may compromise the generation.

2. Enter your HANA repository host name and instance number, along with your user name and password, and then click **Next** to connect.

   Use the tools to the right of the **Connection** field to create a new connection profile, review the properties of the existing profile, or delete it. HANA connection profiles are stored in the registry.

   Note: The account with which you connect must have at least the `CONTENT_ADMIN`, `MODELING`, and `PUBLIC` roles.

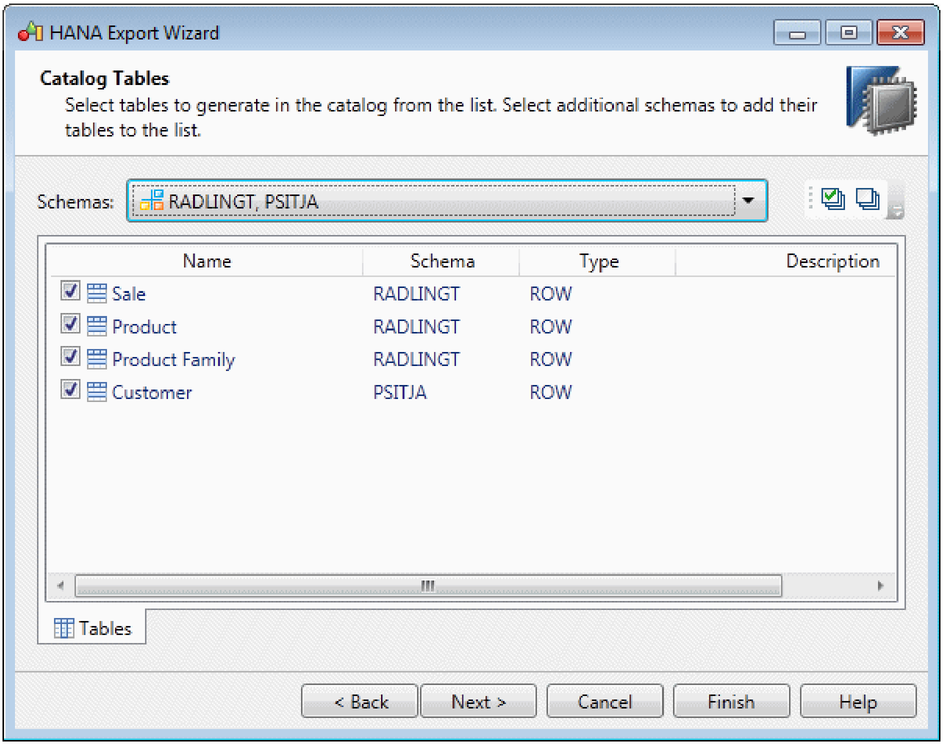
3. Select HANA packages in your model in the left pane to make their contents available to export. Select the facts to export in the right pane, and then click **Next**.

   When you select a fact to export, its supporting dimensions are automatically selected for import.

   Note: If you have previously imported objects from HANA, the archive model helps to determine model changes since that point (see *Archive PDMs* on page 322).

4. Select the catalog tables to export, and then click **Next**.

   PowerDesigner automatically selects any catalog tables required by the selected facts and dimensions.



5. Review the objects that will be exported and then click **Finish** to generate them to the HANA repository.

Note: If PowerDesigner detects conflicts between changes made in the model and changes to the same objects on the server, then a merge dialog (see *Core Features Guide > Modeling with PowerDesigner > Comparing and Merging Models*) will open to allow you to select, for each conflict, which of the conflicting changes will prevail. The resolutions that you select will first be applied to the model, and then your changes will be exported to the the server.

Importing Objects from the HANA Repository

While HANA tables are generated directly to the catalog, analytic and attribute views are exported to the HANA repository from where they will be deployed. PowerDesigner provides a wizard to allow you to import analytic and attribute views from the HANA repository, along with their supporting catalog tables.

Note: This feature requires a 32-Bit Java installation.

1. Select **Database > Update Model from HANA Repository** to open the wizard, and click **Next** on the Welcome page.
2. Enter your HANA repository host name and instance number, along with your user name and password, and then click **Next** to connect.

   Note: The account with which you connect must have at least the `CONTENT_ADMIN`, `MODELING`, and `PUBLIC` roles.

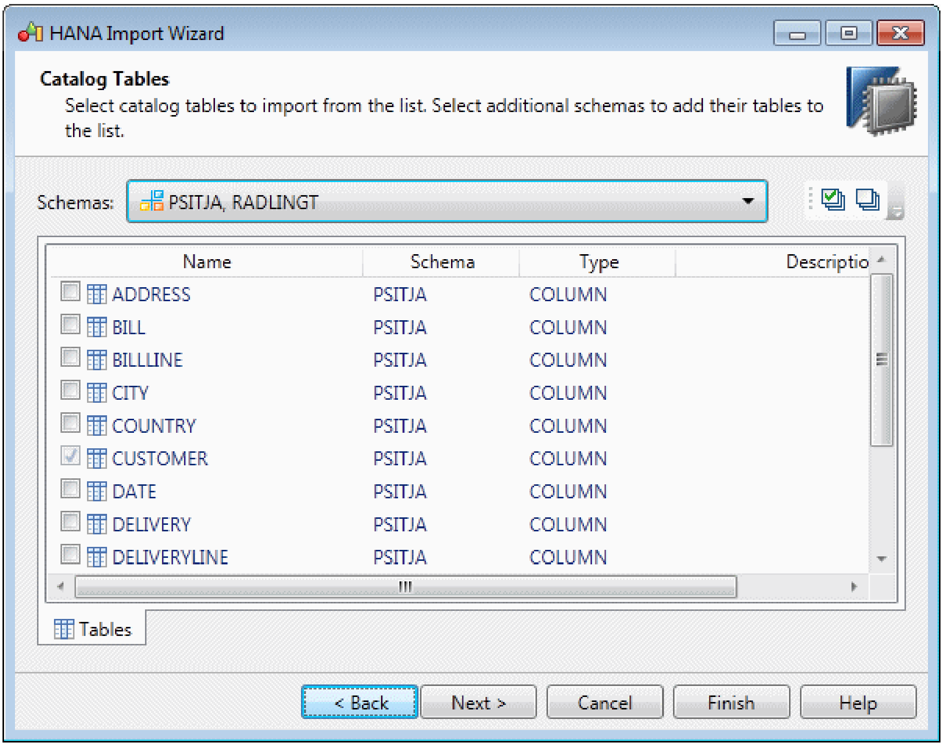
3. Select packages in the repository in the left pane to make their contents available to import. Select the analytic views to import in the right pane, and then click **Next**.

   When you select an analytic view to import, its supporting attribute views are automatically selected for import.

   Note: The archive model retains a snapshot of the structure of your objects at import time to help in determining model changes when re-exporting to HANA (see *Archive PDMs* on page 322).

4. Select catalog tables to import from the list, and then click **Next**.

   PowerDesigner automatically selects any catalog tables required by the selected analytic and attribute views. Select additional schemas to make their tables available for selection.



5. Review the objects that will be imported and then click **Finish**.

6. If objects are already present in the model, a merge dialog will open (see *Core Features Guide > Modeling with PowerDesigner > Comparing and Merging Models*) to allow you to review the specific changes that will be make. Approve or reject the proposed changes, and then click **OK** to perfom the import.

   PowerDesigner will import schemas, users, and tables to the root of the model and analytic and attribute views to their appropriate HANA packages. When the import is complete, click **Close** to exit the wizard.

CHAPTER 18 SAP Sybase IQ

To create a PDM with support for features specific to the SAP® Sybase® IQ DBMS family, select the appropriate version in the DBMS field of the New Model dialog. To view these extensions to the PowerDesigner metamodel in the Resource Editor, select **Database > Edit Current DBMS** and expand the **Profile** node.

Note: The DBMS definition files for Sybase IQ v12.x are deprecated.

The following sections list the extensions provided for IQ.

Note: We do not provide documentation for the properties on the **Physical Options** and certain other tabs, though minimal information is available for them in the Resource Editor. For information about these properties, consult your DBMS reference documentation.

Tables

The following extensions are available on the Sybase IQ tab (v12.4.3 and higher):

Name	Description
DBSpace	[v15.0 and higher] Specifies the dbspace in which to create the table (see *Dbspaces (IQ)* on page 516). Scripting name: DBSpace
Global temporary table	[v12.4.3 to 15.2] Specifies that the table is a global temporary table. Scripting name: ExtGlobalTemporaryTable
Scope	[v15.3 and higher] Specifies that the table is either a global or local temporary table. Scripting name: TemporaryTableScope
On commit	[v15.0 and higher] Action on commit. Scripting name: OnCommit
Not transactional	[v15.0 and higher] A table created using NOT TRANSACTIONAL is not affected by either COMMIT or ROLLBACK. Scripting name: NotTransactional
Remote location	[v15.0 and higher] Used to create a table at the remote location. Scripting name: At

Name	Description
Partition key	[v15.0 and higher] Specifies the partition key column. Scripting name: PartitionKey

Columns

The following extensions are available on the Sybase tab (v12.4.3 and higher):

Name	Description
DBSpace	[v15.4 and higher] Specifies the database file (dbspace) in which to create the column (see *Dbspaces (IQ)* on page 516). Scripting name: DBSpace
Number of distinct value (Iq unique)	Defines the cardinality of the column (to optimize the indexes internally). Scripting name: ExtIqUnicity

In addition, from v15.0 and higher, the **Partitions** tab allows you to override the allocations of partitioned column values to different dbspaces (see *Table and Column Partitions (IQ)* on page 518).

Indexes

The following extensions are available on the Sybase tab (v15.0 and higher):

Name	Description
With nulls not distinct	[v15.4 and higher, when Unique] Specifies that more than one null value is permitted despite the index requiring unique values. Scripting name: WithNullsNotDistinct
Tablespace	[Non-text indexes] Specifies the index dbspace (see *Dbspaces (IQ)* on page 516). Scripting name: In
Notify	[Non-text indexes] Gives notification messages after n records are successfully added for the index. Scripting name: Notify
Word length	[WD indexes] Specifies the maximum word length that is permitted in the WD index. Scripting name: Limit

Name	Description
Delimited by	[WD indexes] Specifies separators to use in parsing a column string into the words to be stored in that column's WD index. Scripting name: DelimitedBy
Configuration	[Text indexes] Specifies the text configuration (see *Text Configurations (IQ/ SQL Anywhere)* on page 530) to be used to control the building of the text index. Scripting name: Configuration
Immediate refresh	[Text indexes v15.2 and higher] Specifies that the index is refreshed immediately each time data is written to the table. Scripting name: Refresh

Keys and References

The following extensions are available on the General tab (v15.0 and higher):

Name	Description
DBSpace	Specifies the DBSpace where the object is stored (see *Dbspaces (IQ)* on page 516). Scripting name: PortNumber

Data Sources

The following extensions are available on the Data Movement (Lifecycle) tab (v15.0 and higher), and are required when the first phase of a lifecycle policy must manage data in an external database:

Name	Description
Remote server name	Specifies the name of the server where the remote database is located. Scripting name: Server
Remote database name	Specifies the name of the remote database from which data must be loaded. Scripting name: DatabaseName
Server class	Specifies the type of connection that must be made to the external database. Select the appropriate value from the list. Scripting name: ServerClass

Name	Description
Connection string	Specifies the connection string used to connect to the external database in the format: • JDBC - `<host>:<port>[/database name]` • ODBC - `<odbc name>` Scripting name: JDBCConnectionString/ODBCConnectionString
User/group	Specifies the user or group name with which to log into the external database. Scripting name: ExternalLogin

Procedures

The following extensions are available on the Sybase IQ tab (v15.0 and higher):

Name	Description
Temporary	[standard functions] Specifies that the function is visible only by the connection that created it, and that it is automatically dropped when the connection is dropped. Scripting name: TempFunction
Return data type	Specifies the procedure return data type. Scripting name: ReturnDttp
Routine characteristics	[standard functions] Transact-SQL-like error handling and deterministic options. Scripting name: RoutineCharacteristics
Sql security	[standard functions] Defines whether the function is executed as the INVOKER, the user who is calling the function, or as the DEFINER, the user who owns the function. Scripting name: SqlSecurity
URL	[web functions] Specifies the URL of the web service. Scripting name: URL
Type	[web functions] Specifies the format used when making the web service request. Scripting name: URLType

Name	Description
Header	[HTTP web functions] When creating HTTP web service client functions, use this clause to add or modify HTTP request header entries. Scripting name: Header
Soap header	[SOAP web functions] When declaring a SOAP web service as a function, use this clause to specify one or more SOAP request header entries. Scripting name: SoapHeader
Certificate	[web functions] To make a secure (HTTPS) request, a client must have access to the certificate used by the HTTPS server. The necessary information is specified in a string of semicolon-separated key/value pairs. Scripting name: Certificate
Client port	[HTTP web functions] Identifies the port number on which the HTTP client procedure communicates using TCP/IP. Scripting name: ClientPort
Namespace	[SOAP web functions] Identifies the method namespace usually required for both SOAP:RPC and SOAP:DOC requests. Scripting name: Namespace
Proxy	[web functions] Specifies the URI of a proxy server. Scripting name: Proxy

Users

The following extensions are available on the General tab (v15.0 and higher):

Name	Description
Force change	Controls whether the user must specify a new password when they log in. This setting overrides the password_expiry_on_next_login option setting in the login policy. Scripting name: ForcePasswordChange
Login policy	Specifies the login policy to assign to the user (see *Login Policies* on page 522). Scripting name: LoginPolicy

Web Services

The following extensions 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 extensions 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

Reference Architecture Modeling (IQ)

PowerDesigner provides a special EAM model to help you determine the architecture required to deploy a Sybase IQ data warehouse solution to meet your anticipated workload. An advisor wizard generates architectures based on one or more hardware servers, and comparison tools help you choose the best architecture based on your requirements for cost and speed.

For detailed information, see *Enterprise Architecture Modeling > Sybase IQ Reference Architecture Model*.

Information Lifecycle Management (IQ)

Sybase IQ v15.0 and higher provides data placement capabilities and supports hierarchical storage management with relocation of less critical data to cheaper storage. PowerDesigner offers a simple modeling structure to cost effectively manage "aging" of data inside the data center from 1st tier high performance storage for frequently accessed data through 2nd tier near-line storage for data that is infrequently accessed to 3rd tier archive storage for data that must remain available for regulatory audits.

For detailed information about using PowerDesigner to model your IQ information lifecycle management, see *Lifecycles (PDM)* on page 189.

Events (IQ/SQL Anywhere)

Sybase IQ (v12.7 and higher) and SQL Anywhere (v10 and higher) support events, which allow you to automate and schedule actions. PowerDesigner models events as extended objects with a stereotype of <<Event>>.

Creating an 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 a package) in the Browser, and select **New > Event**.

Event Properties

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
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

Name	Description
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

Dbspaces (IQ)

Sybase IQ distributes user data across multiple disks at the application level by representing each device as a dbspace. A dbspace can be an operating system file or a raw disk partition. Dbspaces can contain both user data and internal database structures used for startup, recovery, backup, and transaction management.

PowerDesigner allows you to allocate tables and tables partitions, columns and column partitions, indexes, join indexes, keys, and references to specific dbspaces from each object's property sheet.

Creating a Dbspace

PowerDesigner models dbspaces as tablespaces with additional properties. You can create a dbspace in any of the following ways:

- Select **Model > Tablespaces** to access the List of Tablespaces, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New > Tablespace**.

Dbspace Properties

PowerDesigner models dbspaces as tablespaces (see *Tablespaces and Storages (PDM)* on page 198) with the following additional properties on the **General** tab (v15.0 and higher):

Property	Description
Catalog store	Specifies that the dbspace is created for the catalog store and will contain a single dbfile. If you select this option, you must specify a path to the file. Scripting name: CatalogStoreDisplay
File path	Specifies a physical file path for the dbspace. Scripting name: As
Online	Specifies that the dbspace is online. Scripting name: Online
Read-only	Specifies that the online dbspace is read-only. Scripting name: ReadOnly
Striping	Specifies that the dbspace is available for striping. Scripting name: Striping
Stripe size (in kb)	Specifies the size of the stripes. Scripting name: Stripesizekb

In addition, the following tabs are available:

- Cost - allows you to specify the cost per GB of storage for the dbspace (see *Tablespace and Storage Properties* on page 200).
- DBFiles - lists the dbfiles associated with the dbspace.

DBSpace Files

PowerDesigner models dbspace files as extended objects with a stereotype of `<<DBSpaceFile>>` with the following additional properties on the **General** tab (v15.0 and higher):

Property	Description
Path	Specifies the file path to the dbspace file. Scripting name: FilePath

Property	Description
Read-only	Specifies that the resource is read-only. Scripting name: ReadOnly
Size	Specifies that the size of the dbspace file. Scripting name: Size, SizeUnit
Reserve	Specifies the size of space to reserve, so that the dbspace can be increased in size in the future. Scripting name: Reserve, ReserveUnit

Table and Column Partitions (IQ)

A partition is a physical division of the contents of a database table, based on values in the column designated as the partition key, and allocated to a particular dbspace. You can override the allocation of values in certain columns by specifying column partitions.

Creating a Table Partition

In order to create table partitions, you must first select a column as the **Partition key** on the **Sybase IQ** tab of the table property sheet (see *Chapter 18, SAP Sybase IQ* on page 509), in order to display the **Partitions** tab.

You can create as many partitions as necessary for the table on this tab using the **Insert Row** and **Add a Row** tools.

Note: Some PowerDesigner features automate the creation of partitions (see *Denormalizing Tables and Columns* on page 80 and *Modeling a Lifecycle* on page 190. If you associate a table with a lifecycle (see *Lifecycles (PDM)* on page 189), PowerDesigner will delete all existing table partitions in order to create the necessary partitions to move data between lifecycle phases.

Table Partition Properties

To view or edit a partition's properties, double-click its Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator. The following properties are available on the **General** tab:

Property	Description
Parent object	[read only] Specifies the table of which the partition forms a part.

Property	Description
Name/Code/ Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
Values	Specifies the upper bound of the partition, based on the value of the column specified as the partition key. The `max` keyword can only be set on the last partition.
DBSpace	Specifies the dbspace with which the partition is associated (see *Dbspaces (IQ)* on page 516). Select a dbspace from the list or click the tools to the right of this field to create, delete, or search for a dbspace, or to open the property sheet of the selected dbspace.

Overriding Partition DBspaces for a Particular Column

You can override the allocation of values in a particular column from the table partition dbspace to an alternate dbspace. The column will continue to be partitioned based on the same partition key ranges, but the column values for each range will be allocated to the alternate dbspaces.

You create column partitions on the **Partitions** tab of the column property sheet. Click the **Properties** tool to specify the following properties:

Property	Description
Parent object	[read only] Specifies the column to which the partition belongs.
Comment	Provides more detailed information about the object.
Partition	Specifies the table partition for which this partition will redirect column values to an alternate dbspace.
Dbspace	Specifies the dbspace (see *Dbspaces (IQ)* on page 516) to which column values contained within this table partition should be allocated.

Logical Servers and Policies (IQ)

Sybase IQ v16 and higher supports logical servers, which provide the only means to access the multiplex server nodes. PowerDesigner models logical servers and logical server policies as extended objects with a stereotype of `<<LogicalServer>>` and `<<LogicalServerPolicy>>` respectively.

Creating a Logical Server

You can create a logical server in any of the following ways:

- Select **Model > Logical Servers** to access the List of Logical Servers, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New > Logical Server**.

Creating a Logical Server Policy

You can create a logical server policy in any of the following ways:

- Select **Model > Logical Server Policies** to access the List of Logical Policies, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New > Logical Server Policy**.

Logical Server and Logical Server Policy Properties

You can modify an object's properties from its property sheet. To open a logical server or logical server policy property sheet, double-click its Browser entry in the Logical Servers or Logical Server Policies folder.

The following extended attributes are available on the **General** tab:

Name	Description
With stop server	Automatically shuts down all servers in the logical server when the `TEMP_DATA_IN_SHARED_TEMP` option is changed directly or indirectly. Scripting name: WithStopServer

The following extended attributes are available on the **Sybase IQ** tab of logical servers:

Name	Description
Membership	Lists the multiplex nodes (see *Multiplex Servers (IQ)* on page 521) of the logical server. Select the **Add for logical coordinator membership** option to specifies a logical server membership to the current coordinator. Scripting name: Membership, MembershipForLogicalCoordinator
Policy	Specifies the logical server policy applied to the server. Scripting name: Policy

The following extended attributes are available on the **Options** tab of logical server policies:

Name	Description
DQP enabled	Specifies how query processing is distributed: • 0 - Not distributed • 1 - [default] Distributed as long as a writable shared temporary file exists. • 2 - Distributed over the network, and the shared temporary store is not used Scripting name: DqpEnabled-disp
Allow coordinator as member	[ROOT policy only] Specifies that the coordinator can be a member of any user-defined logical server. Enabled by default. Scripting name: AllowCoordinatorAsMember-disp
Login redirection	Enables login redirection for logical servers governed by specified login policy. By default, login redirection is disabled at the logical server level, allowing external connection management. Scripting name: LoginRedirection-disp
Redirection waiters threshold	Specifies how many connections can queue before IQ redirects a connection to this logical server to another server. Scripting name: RedirectionWaitersThreshold-disp
Temp data in shared temp	Enables temporary table data and eligible scratch data writes to the shared temporary store, provided that the shared temporary store has at least one read-write file added. Scripting name: TempDataInSharedTemp-disp

Multiplex Servers (IQ)

Sybase IQ v15.0 and higher supports multiplex, a highly scalable shared disk grid technology that allows concurrent data loads and queries via independent data processing nodes connected to a shared data source.

PowerDesigner models multiplex servers as extended objects with a stereotype of <<MultiplexServer>>.

Creating a Multiplex Server

You can create a multiplex server in any of the following ways:

- Select **Model > Multiplex Servers** to access the List of Multiplex Servers, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New > Multiplex Server**.

Multiplex Server Properties

You can modify an object's properties from its property sheet. To open a multiplex server property sheet, double-click its Browser entry in the Multiplex Servers folder.

The following extended attributes are available on the **Sybase** tab:

Name	Description
Database	Specifies the database file with which the server is associated. Scripting name: Database
Host port list	Specifies the machine where the database engine will run. Scripting name: HostPortList
Role	Specifies the server's role in the multiplex environment. Scripting name: Role
Status	Specifies whether the server is included or excluded. If a multiplex secondary server will be shut down for an extended period of time, that server should be excluded. Excluding the server allows the coordinator to ignore this server when performing version cleanup. Scripting name: Status
Failover	Specifies that the server is a failover server. Scripting name: Failover

Login Policies (IQ/SQL Anywhere)

Sybase IQ (v15.0 and higher) and SQL Anywhere (v12 and higher) define the rules to be followed when establishing a user's database connection in a database object called a login policy. PowerDesigner models login policies as extended objects with a stereotype of <<LoginPolicy>>.

Creating a Login Policy

You can create a login policy in any of the following ways:

- Select **Model** > **Login Policies** to access the List of Login Policies, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New** > **Login Policy**.

Login Policy Properties

You can modify an object's properties from its property sheet. To open a login policy property sheet, double-click its Browser entry in the Login Policies folder.

The following extended attributes are available on the **Sybase** tab (**Options** tab from v16 and higher):

Name	Description
Password life time	Specifies the maximum number of days before a password must be changed. Scripting name: PasswordLifeTime
Password grace time	Specifies the number of days before password expiration during which login is allowed but the default post_login procedure issues warnings. Scripting name: PasswordGraceTime
Password expires	Specifies that the user's password will expire in the next login. Scripting name: PasswordExpiryOnNextLogin
Locked	Specifies that users are prohibited from establishing new connections. Scripting name: Locked
Maximum connections	Specifies the maximum number of concurrent connections allowed for a user. Scripting name: MaxConnections
Maximum failed logins	Specifies the maximum number of failed attempts, since the last successful attempt, to login to the user account before the account is locked. Scripting name: MaxFailedLoginAttempts
Auto unlock time	[v16 and higher] Specifies the time period after which locked accounts not granted the MANAGE ANY USER system privilege are automatically unlocked. Scripting name: AutoUnlockTime
Maximum days since login	Specifies the maximum number of days that can elapse between two successive logins by the same user. Scripting name: MaxDaysSinceLogin
Maximum non-dba connections	Specifies the maximum number of concurrent connections that a user without DBA authority can make. This option is only supported in the root login policy. Scripting name: MaxNonDBAConnections
Change password dual control	[v16 and higher] Specifies that two users, each granted the CHANGE PASSWORD system privilege, are required to change the password of another user. Scripting name: ChangePasswordDualControl
Default logical server	[v16 and higher] Specifies the server to which the user using this login policy connects when the connection string specifies no logical server. Scripting name: DefaultLogicalServer_disp

Name	Description
Root auto unlock time	[v16 and higher] Specifies the time period after which locked accounts granted the MANAGE ANY USER system privilege are automatically unlocked. Scripting name: RootAutoUnlockTime

[v16 and higher] The following extended attributes are available on the **LDAP** tab:

Name	Description
Primary / Secondary server	Specify the names of the primary and secondary LDAP servers (see *LDAP Servers (IQ)* on page 524). Scripting name: LDAPPrimaryServer, LDAPSecondaryServer
Auto fallback period	Specifies the time period, in minutes, after which automatic failback to the primary server is attempted. Scripting name: LDAPAutoFailbackPeriod
Failover to standard authentication	Permits standard authentication when authentication via the LDAP server fails due to system resources, network outage, connection timeouts, or similar system failures. Scripting name: LDAPFailoverToStd
Record LDAP DN refresh time	Updates the `ldap_refresh_dn` value in the system table with the current time, stored in Coordinated Universal Time (UTC) Scripting name: LDAPRefreshDN

LDAP Servers (IQ)

Sybase IQ v16 and higher supports delegating the authentication of users to LDAP servers. PowerDesigner models LDAP servers as extended objects with a stereotype of `<<LDAPServer>>`.

Creating an LDAP Server

You can create an LDAP server in any of the following ways:

- Select **Model > LDAP Servers** to access the List of LDAP Servers, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New > LDAP Server**.

LDAP Server Properties

You can modify an object's properties from its property sheet. To open an LDAP server property sheet, double-click its Browser entry in the LDAP Servers folder.

The following extended attributes are available on the **General** tab:

Name	Description
Activate LDAP server after creation	Activates the LDAP server configuration object for immediate use upon creation. Scripting name: WithActivate

The following extended attributes are available on the **Sybase** tab:

Name	Description
Search DN	Specifies the host (by name or by IP address), port number, and the search to be performed for the DN lookup for a given user ID, along with the user created in the LDAP server for use by SAP Sybase IQ, the password to use, and whether it is encrypted. Scripting name: URL, AccessAccount, Password, Encrypted
Attributes	Specifies the host (by name or IP address) and the port number of the LDAP server to use for authentication of the user, the connection timeout and number of retries, and whether TLS or Secure LDAP protocol is used for connections for both DN searches and authentication. Scripting name: AuthenticationURL, ConnectionTimeout, ConnectionRetries, TLS

Remote Servers (IQ)

Sybase IQ v15.0 and higher supports remote servers, which define where remote objects mapped to a local proxy table are located. PowerDesigner models remote servers as extended objects with a stereotype of <<RemoteServer>>.

Creating a Remote Server

You can create a remote server in any of the following ways:

- Select **Model > Remote Servers** to access the List of Remote Servers, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New > Remote Server**.

Remote Server Properties

You can modify an object's properties from its property sheet. To open a remote server property sheet, double-click its Browser entry in the Multiplex Servers folder.

The following extended attributes are available on the **General** tab:

Name	Description
Class	Specifies the remote server class. Scripting name: Class
Read-only	Specifies that the remote server is a read-only data source. Any update request is rejected by Sybase IQ. Scripting name: ReadOnly
Connection	Specifies the connection string in the format `machine-name:port-number [ /dbname ]` or as a data source name Scripting name: ConnectionInfo

External Logins (IQ)

Sybase IQ v15.3 and higher supports external logins, which are alternate login names and passwords that are used when communicating with a remote server. PowerDesigner models external logins as extended objects with a stereotype of <<ExternLogin>>.

Creating an External Login

You can create an external login in any of the following ways:

- Select **Model > Extern Logins** to access the List of External Logins, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New** > **External Login**.

External Login Properties

You can modify an object's properties from its property sheet. To open an external login property sheet, double-click its Browser entry in the External Logins folder.

The following extended attributes are available on the **General** tab:

Name	Description
Local login	Specifies the local login name to which the remote login is assigned. Scripting name: LocalLogin
Remote server	Specifies the name of the remote server. Scripting name: RemoteServer
Remote login	Specifies the user account on the remote server, which is associated with the local user login. Scripting name: RemoteLogin

Name	Description
Remote password	Specifies the password for the remote login Scripting name: RemotePassword

Spatial Data (IQ/SQL Anywhere)

Sybase IQ v15.4 and higher and SQL Anywhere v12 and higher can store spatial data (data that describes the position, shape, and orientation of objects in a defined space) using spatial reference systems.

Spatial Reference Systems (SQL Anywhere)

Sybase IQ v15.4 and higher and SQL Anywhere v12 and higher support spatial reference systems, which define the space in which geometries are described. PowerDesigner models spatial reference systems as extended objects with a stereotype of `<<SpatialReferenceSystem>>`.

Creating a Spatial Reference System

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

- Select **Model > Spatial Reference Systems** to access the List of Spatial Reference Systems, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New > Spatial Reference System**.

Spatial Reference System Properties

You can modify an object's properties from its property sheet. To open a spatial reference system property sheet, double-click its diagram symbol or its Browser entry in the Spatial Reference Systems folder.

The following extended attributes are available on the **General** tab:

Name	Description
Spatial reference system identifier	Specifies the SRID (srs-id) for the spatial reference system. Scripting name: SRS_Id
Organization	Specifies the organization that created the spatial reference system that the new spatial reference system is based on. Scripting name: Organization

Name	Description
Organization coordinate reference system ID	Specifies the numeric identifier the organization uses to identify the spatial reference system. Scripting name: OrganizationSRSId

The following extended attributes are available on the **Definition** tab:

Name	Description
Definition	Specifies default coordinate system settings. If any attribute is set in a clause other than the DEFINITION clause, the value specified in the other clause is used regardless of what is specified in the DEFINITION clause. Scripting name: Definition
Type	Specifies whether the system is Projected, Geographic, or Engineering. If a definition is given, this attribute is computed from the definition text. Scripting name: SRSType
Transform definition	Specify a description of the transform to use for the spatial reference system. Scripting name: TransformDefinition

The following extended attributes are available on the **Settings** tab:

Name	Description
Line interpretation	Specifies how the SRS interprets lines between points. Scripting name: LineInterpretation
Axis order	Specifies the order in which values are given for each axis. Scripting name: AxisOrder
Polygon format	Specifies how polygons are interpreted. Scripting name: PolygonFormat
Storage format	Specifies how data is stored. Scripting name: StorageFormat

The following extended attributes are available on the **Coordinate** tab:

Name	Description
Axis/Bounded/Unbounded	Specifies whether the axis is bounded or unbounded and, if it is bounded, the minimum and maximum values. Scripting names: BoundedCoordinate*Axis*, MinCoordinate*Axis*, MaxCoordinate*Axis*
Ellipsoid axis length	[round earth systems] Specifies the values to use for representing the Earth as an ellipsoid. Scripting names: SemiMajorAxisLength, SemiMinorAxisLength, InverseFlattening
Grid Size	[planar systems] Specifies the size of the grid to use when performing calculations. Scripting name: GridSize
Tolerance	[planar systems] Specifies the precision to use when comparing points. Scripting name: Tolerance
Linear/Angular unit of measure	Specify the linear and angular units of measure for the spatial reference system. Scripting name: LinearUnitOfMeasure, AngularUnitOfMeasure

Spatial Units of Measure (SQL Anywhere)

Sybase IQ v15.4 and higher and SQL Anywhere v12 and higher support spatial units of measure, which define the units in which geographic coordinates are measured, and how these units are converted to radians or meters. PowerDesigner models spatial units of measure as extended objects with a stereotype of `<<SpatialUnitOf Measure>>`.

Creating a Spatial Unit of Measure

You can create a spatial unit of measure in any of the following ways:

- Select **Model > Spatial Units of Measure** to access the List of Spatial Units of Measure, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New > Spatial Unit of Measure**.

Spatial Unit of Measure Properties

You can modify an object's properties from its property sheet. To open a spatial unit of measure property sheet, double-click its diagram symbol or its Browser entry in the Spatial Units of Measure folder.

The following extended attributes are available on the **General** tab:

Name	Description
Type	Specifies the kind of unit. Linear units are used for distances and angular units are used for angles. Scripting name: Type
Conversion factor	Specifies how to convert the defined units to the base unit of measure (radians or meters). Scripting name: ConversionFactor

Full Text Searches (IQ/SQL Anywhere)

Full text search can quickly find all instances of a term (word) in a database without having to scan table rows and without having to know which column a term is stored in. IQ (v15.2 and higher) and SQL Anywhere) support full text searches through text configurations and text indexes, which store complete positional information for every instance of every term in every indexed column.

Text Configurations (IQ/SQL Anywhere)

Text configuration objects are supported for IQ (v15.2 and higher) and SQL Anywhere (v12 and higher) to control the creation of text indexes. PowerDesigner models text configurations as extended objects with a stereotype of <<TextConfiguration>>.

Text configurations contain a set of configuration settings that control the characteristics of text index data such as what terms to ignore, and the minimum and maximum length of terms to include in the index. Once you have created a text configuration, you can select it to control a text index on the Sybase tab of your text index property sheet (see *Text Indexes* on page 531).

Creating a Text Configuration

You can create a text configuration in any of the following ways:

- Select **Model > Text Configurations** to access the List of Text Configurations, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New > Text Configuration**.

Text Configuration Properties

You can modify an object's properties from its property sheet. To open a text configuration property sheet, double-click its Browser entry in the Text Configurations folder.

The following extended attributes are available on the **General** tab:

Name	Description
Owner	Specifies the owner of the text configuration. Use the tools to the right of the field to create or choose an owner or to delete or inspect the properties of the current owner. Scripting name: Owner
Template	Specifies a text configuration to use as the template for creating this one. Scripting name: ParentConfiguration

The following extended attributes are available on the **Sybase** tab:

Name	Description
Minimum/Maximum Term Length	Specify the minimum and maximum length in characters of terms that will be included in the index. Scripting name: MinTermLength, MaxTermLength
Text breaker	Specifies the name of the algorithm to use for separating column values into terms. Scripting name: TextBreaker
Stoplist	Specifies terms to ignore when building a text index. Scripting name: StopList

Text Indexes (IQ/SQL Anywhere)

Text indexes are supported for IQ (v15.2 and higher) and SQL Anywhere (v12 and higher) to enable fast full text searching.

You create a text index by creating a standard index (see *Creating an Index* on page 109), and selecting the type `TEXT`. For information about the properties specific to text indexes, see *Chapter 18, SAP Sybase IQ* on page 509.

Indexes (IQ)

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

You should consider all criteria in combination, rather than individually. Try to anticipate for the data in each column, the number of unique and total values, the query results users will want, and whether the data will be used in ad hoc joins or join indexes.

The following types of index are available:

- HG – HighGroup indexes are used for GROUP BY, COUNT(DISTINCT) and SELECT DISTINCT statements when data has more than 1000 unique values
- HNG – 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 – LowFast indexes are used for columns that have a very low number of unique values. This index also facilitates join index processing (*Join Indexes (IQ/Oracle)* on page 534). It is one of the two indexes allowed for columns used in join relationships.
- CMP – Compare indexes are used for columns that store the binary comparison (<, >, or =) of any two distinct columns with identical data types, precision, and scale.
- TEXT – Full text indexes (see *Full Text Searches (IQ/SQL Anywhere)* on page 530).
- WD – Used to index keywords by treating the contents of a CHAR or VARCHAR column as a delimited list.
- DATE, TIME, and DTTM – For date and timestamp columns.

For detailed information about choosing index types, see your IQ documentation.

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.

When you rebuild indexes, PowerDesigner determines the index type based on information contained from the table statistics, using 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 you have not specified a number of rows for the table, PD assumes that the table will include at least 1 row of data.

The rebuild process creates a FASTPROJECTION index for all columns, unless any of the following criteria apply:

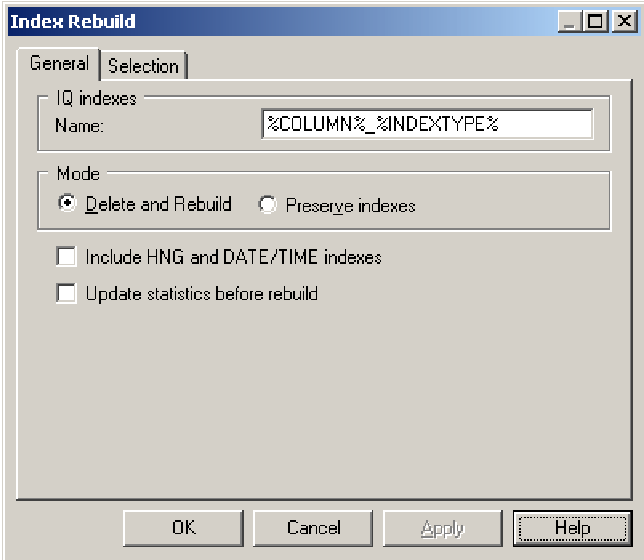
Criteria	Index type
If no statistics are provided and the column has an undefined data type	No index is created
Low number of unique values in a column Column used in join predicate	LOWFAST

Criteria	Index type
High number of unique values in a column No COUNT DISTINCT, SELECT DISTINCT, or GROUP BY queries required	HIGHNONGROUP
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

1. Select **Tools > Rebuild Objects > Rebuild Indexes** to open the Rebuild Indexes dialog box:



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 **Include HNG and DATE/TIME indexes** option to permit the creation of these specialized indexes for appropriate columns. If you do not select this option then only HG and LF indexes will be created.
5. [optional] Select the **Update statistics before rebuild** option 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.
6. [optional] Click the Selection tab and select or clear checkboxes to specify for which tables you want to rebuild indexes.
7. Click OK, and then Yes to confirm the rebuilding of your indexes.

Join Indexes (IQ/Oracle)

A join index is a special type of index, which represents a full outer join of two or more tables, where all rows from both tables are included in the result (with NULL returned for any column with no matching value). The query engine may use this full outer join as a starting point for queries that include left outer, right outer, and inner joins.

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 IQ allows you to create user-defined references to include the exact join required by your foreseen queries.

Creating a Join Index

You can create a join index in any of the following ways:

- Open the property sheet of a table, click the **Join Index** tab, and click the **Add a Row** tool. The join index is created with the selected table specified as the base table.
- Select **Model** > **Join Indexes**, and click the **Add a Row** tool.
- Right-click the model or package in the Browser, and select **New** > **Join Index**
- Automatically, for each fact table and the dimension table it references by selecting **Tools** > **Rebuild Objects** > **Rebuild Join Indexes** (see *Automatically Creating Join Indexes Through Rebuilding* on page 536).

Join Index Properties

You can modify an object's properties from its property sheet. To open a join index property sheet, double-click its Browser entry in the Join Indexes folder.

The General tab contains the following properties:

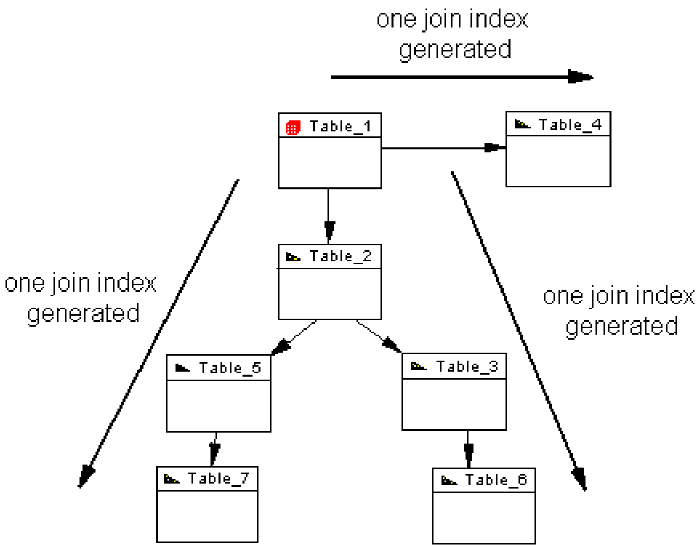
Property	Description
Name/Code/ Comment	Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.
Stereotype	Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file.
Owner	Specifies the user who is the owner of the join index (usually its creator). Use the tools to the right of the list to create, browse for, or view the properties of the currently selected user.
Comment	Descriptive label for the join index.
Base table	Specifies the name of the table or materialized view that stores the join index.
DBSpace	[IQ only] Specifies the DBSpace that will contain the join index.

The following tabs are also available:

- Columns - Lists the columns used for the join index.
- References - Lists the references used for the join index.

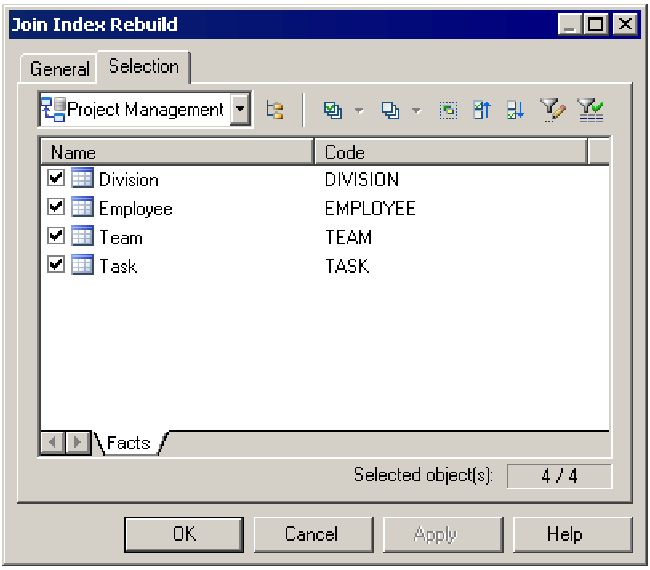
Automatically Creating Join Indexes Through Rebuilding

You can automatically generate a join index for each selected fact table and the dimension tables that it references. Each rebuilt join index contains the references that link the fact table to all the dimension tables located on a single axis proceeding from the fact table.



A join index is constrained and can only be defined for tables that are organized in a connected tree. A reference between two fact tables does not generate any join index.

1. Select **Tools** > **Rebuild Objects** > **Rebuild Join Indexes** to open the Rebuild Join Indexes dialog.
2. On the **General** tab, select the appropriate mode to use:
   - Delete and Rebuild - all existing indexes are deleted before join index rebuild.
   - Preserve - preserves all existing join indexes
3. Click the Selection tab, and select one or more fact tables from the list:



4. Click **OK**, and then **Yes** to confirm the rebuild.

   A join index is generated for each fact table. The generated join indexes are available in the list of join indexes (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.

1. Open the property sheet of the join index and, if necessary, specify the appropriate base table and DBSpace on the **General** tab.
2. Click 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 references in the list and click **OK** to add them to the join index.
3. Click **OK** to save your changes and return to the model.

IQ Data Movement Scripts

PowerDesigner can generate data movement scripts to populate your AS IQ data warehouse from other databases. The script can generate a flat file for loading to the IQ data warehouse and create Insert Location statements for use with a proxy database (for ASE and ASA only).

To create a data movement script, you must:

- [optional] Specify mappings between the tables in your data source and your AS IQ database

- Generate the data movement script

1. To enable the Data Movement extensions in your model, select **Model > Extensions**, click the **Attach an Extension** tool, select the `Data Movement IQ` (on the **General Purpose** tab), and click **OK** to attach it.
2. Right-click the model in the Browser and select **Properties** to open its property sheet, then click the **Data Movement** tab and set the following properties as appropriate to control the files used during data movement:

Property	Description
Field / Row delimiter	Specify the delimiters to be used between fields and between rows in the dump file.
Fully delimited file	Specifies that each row ends with a field delimiter before the row delimiter.
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.

Note: You can override these global data movement options for a specific table (and specify a table-specific dump file for importing) by opening its property sheet and enter table-specific values on the **Data Movement** tab.

3. In your IQ warehouse PDM, right-click the model in the Browser and select **New > Data Source** to create a data source to populate your IQ Data Warehouse. Enter a name for the source and then click the **Models** tab, click the **Add Models** tool, and select your source model.
4. Click the data source **Database Connection** tab, and select a data source, login, and password to connect to your source database.
5. Click the data source **Data Movement** tab, and enter the following properties as appropriate to access the remote server:

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.

Property	Description
Local user name	Specifies the database user name.

6. [optional] Select **Tools > Mapping Editor** and create mappings between your source and warehouse tables.

   For detailed information about using the Mapping Editor, see *Core Features Guide > Linking and Synchronizing Models > Object Mappings*.

Generating the Data Movement Script

You can generate the data movement script from the Tools menu.

1. Select **Tools > Extended Generation** , and specify a directory in which to generate your data movement files.
2. [optional] Click the **Selection** tab and specify for which Tables and/or Data Sources you want to generate a data movement script.
3. [optional] Click the **Options** tab and specify the following generation options as appropriate:
   - **Use Mappings** – Specifies to use mappings to control the data movement.
   - **Data Movement Method** – Specifies the type of script to generate:
     - `Insert Location` – [IQ or ASE only] Create a loadscript for connecting the source database to the IQ server. If the data source is not an IQ or ASE database, then no loadscript will be generated.
     - `External File` – Create a dump file from the source database together with a loadscript to upload it to the IQ server.
4. [optional] Click the **Generated Files** tab to review the names and locations of the files to be generated.
5. Click **OK** to begin the generation of the data movement script.

CHAPTER 19 SAP Sybase SQL Anywhere

To create a PDM with support for features specific to the SAP® Sybase® SQL Anywhere® (formerly AS Anywhere) DBMS family, select the appropriate version in the DBMS field of the New Model dialog. To view these extensions to the PowerDesigner metamodel in the Resource Editor, select **Database > Edit Current DBMS** and expand the **Profile** node.

Note: The DBMS definition file for Sybase AS Anywhere v9 is deprecated.

The following sections list the extensions provided for SQL Anywhere.

Note: We do not provide documentation for the properties on the **Physical Options** and certain other tabs, though minimal information is available for them in the Resource Editor. For information about these properties, consult your DBMS reference documentation.

Columns

The following extensions are available on the Sybase tab (v10 and higher):

Name	Description
Column is compressed	Specifies whether this column is stored in a compressed format. Scripting name: Compressed

Tables

The following extensions are available on the Sybase tab:

Name	Description
PCTFREE	Specifies the percentage of free space to reserve for each table page. 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. Enter an integer between 0 (no free space is to be left on each page) and 100 (high values cause 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
Dbspace (tablespace)	Specifies the dbspace in which the table is to be created Scripting name: DbspaceIn

Name	Description
Remote location	Creates a table at the specified remote location in addition to a proxy table on the current database that maps to the remote table. Supports the semicolon (;) as a field delimiter in the location-string. If no semicolon is present, a period is the field delimiter. Scripting name: At
Encrypted	Encrypts the table using the encryption key and algorithm specified at database creation time. Encrypting a table may take time, depending on the size of the table. Scripting name: Encrypted
Temporary table/ Global temporary table	Specifies either temporary table is a global or a local temporary table. Scripting name: [v10 and higher] TemporaryTable, [up to v9] ExtGlobalTemporaryTable
Not transactional	[temporary tables] Specifies that the temporary table is not affected by either COMMIT or ROLLBACK. This can provide performance improvements because operations on non-transactional temporary tables do not require entries in the rollback log. For example, NOT TRANSACTIONAL may be useful if procedures that use the temporary table are called repeatedly with no intervening COMMITs or ROLLBACKs. Scripting name: TemporaryTableOptionsNotTransactional
On commit	[temporary tables] Specifies that the rows of a temporary table are deleted on COMMIT. Scripting name: TemporaryTableOptionsOnCommit

Indexes

The following extensions are available on the Sybase tab:

Name	Description
Tablespace	[Non-text indexes] Specifies the index dbspace. Scripting name: In

Name	Description
Virtual index	[v10 and higher] 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
Notify	[Non-text indexes v12 and higher] Gives notification messages after n records are successfully added for the index. Scripting name: Notify
Word length	[Non-text indexes v12 and higher] Specifies the maximum word length that is permitted. Scripting name: Limit
Delimited by	[Non-text indexes v12 and higher] Specifies separators to use in parsing a column string into the words to be stored in the index. Scripting name: DelimitedBy
Text index	[v12 and higher] Specifies whether the index is a text index or not. Scripting name: TextIndex
Configuration	[Text indexes v12 and higher] Specifies the text configuration (see *Text Configurations* on page 530) to be used to control the building of the text index. Scripting name: Configuration
Immediate refresh	[Text indexes v12 and higher] Specifies that the index is refreshed immediately each time data is written to the table. Scripting name: Refresh

Users

The following extensions are available on the General tab (v12 and higher):

Name	Description
Force change	Controls whether the user must specify a new password when they log in. This setting overrides the password_expiry_on_next_login option setting in the login policy. Scripting name: ForcePasswordChange

Name	Description
Login policy	Specifies the login policy to assign to the user (see *Login Policies* on page 522). Scripting name: LoginPolicy

Web Services
The following extensions 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 service type] Specifies a name prefix. Only SOAP services whose names begin with this prefix are handled. Scripting name: Prefix

Web Operations
The following extensions 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

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.

Mirror Servers (SQL Anywhere)

Sybase SQL Anywhere (v12 and higher) supports database mirroring through the use of mirror servers. PowerDesigner models mirror servers as extended objects with a stereotype of <<MirrorServer>>.

Creating a Mirror Server

You can create a mirror server in any of the following ways:

- Select **Model > Mirror Servers** to access the List of Mirror Servers, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New > Mirror Server**.

Mirror Server Properties

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

The following extended attributes are available on the Options tab:

Name	Description
Type	Specifies the type of mirror server to create. You can choose between: • Primary - defines a virtual or logical server, whose name is the alternate server name for the database, which can be used by applications to connect to the server currently acting as the primary server. There can be only one PRIMARY server for a database. • Mirror - defines a virtual or logical server, whose name is the alternate server name for the database, which can be used by applications to connect to the server currently acting as the read-only mirror. There can be only one MIRROR server for a database. • Arbiter - assists in determining which of the PARTNER servers takes ownership of the database. The arbiter server must be defined with a connection string that can be used by the partner servers to connect to the arbiter. There can be only one ARBITER server for a database. • Partner - is eligible to become the primary server and take ownership of the database. You must define two PARTNER servers for database mirroring, and both must have a connection string and a state file. In a read-only scale-out system, you must define one PARTNER server. This server is the root server, and runs the only copy of the database that allows both read and write operations. • Copy - In a read-only scale-out system, this value specifies that the database server is a copy node. All connections to the database on this server are read-only. You do not have to explicitly define copy nodes for the scale-out system; you can choose to have the root node define the copy nodes when they connect. Scripting name: Type
Using auto parent	[copy only] Specifies that the primary server will assign a parent for this server. Scripting name: UsingAutoParent
Parent	[copy only] Specifies a tree of servers for a mirroring or scale-out system and indicates the servers from which the non-participating nodes obtain transaction log pages. Scripting name: ParentServer
Alternate parent	[copy only] Specifies an alternate parent for the copy node. Scripting name: AlternateParentServer
Primary	[copy only] Specifies that the parent server is the primary server. Scripting name: PrimaryParentServer
Connection string	Specifies the connection string to be used to connect to the server. Scripting name: ConnectionString

Name	Description
Log file	Specifies the location of the log file that is sent between mirror servers. Scripting name: LogFile
Preferred	[partner only] Specifies whether the server is the preferred server in the mirroring system, which assumes the role of primary server whenever possible. Scripting name: Preferred
State file	[arbiter, partner] Specifies the location of the file used for maintaining state information about the mirroring system. Scripting name: StateFile

Spatial Data (SQL Anywhere)

SQL Anywhere (v12 and higher) can store spatial data (data that describes the position, shape, and orientation of objects in a defined space) using spatial reference systems.

For more information, see *Spatial Data (IQ/SQL Anywhere)* on page 527.

Events, Login Policies, and Full Text Searches (SQL Anywhere)

PowerDesigner supports modeling for Sybase SQL Anywhere events (v10 and higher), login policies (v12 and higher), and full text searches (v12 and higher).

For detailed information, see *Events (IQ/SQL Anywhere)* on page 515, *Login Policies (IQ/SQL Anywhere)* on page 522, and *Full Text Searches (IQ/SQL Anywhere)* on page 530.

Certificates (SQL Anywhere)

Sybase SQL Anywhere (v16.0 and higher) supports X.509 certificates for transport-layer security. 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 a 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 **General** tab:

Name	Description
Type	Specifies the type of the certificate, which can be a string, variable, or file. Scripting name: CertificateSourceType
Certificate	Specifies the source of the certificate. Scripting name: CertificateSource

Proxy Tables (ASE/SQL Anywhere)

A proxy table is used to access data in a remote table; it has all the attributes of the remote table, but does not contain any data locally. PowerDesigner uses an extension file to provide support for generating the script for a proxy table in order to run it in a Sybase ASA or ASE database.

1. To enable the proxy table extensions in your model, select **Model > Extensions**, click the **Attach an Extension** tool, select the Proxy Tables file (on the **General Purpose** tab), and click **OK** to attach it.
2. For each proxy table, right-click a table in another PDM target model, drag it to the model where you want to create a proxy table, release the right mouse button and select one of the following:
   - **Create Shortcut Here** - Creates a non-modifiable reference to the original table.
   - **Replicate Here** - Creates a modifiable reference to the original table. You can desynchronize the Code property of the replica to give the proxy table a different name in the local model.

   For more information about shortcuts and replicas, see *Core Features Guide > Linking and Synchronizing Models > Shortcuts and Replicas*.

   Note: A custom check verifies that the proxy table is not the child table of a reference.
3. Right-click the model in the browser and select **New > Data Source** to create a new data source to provide access to the remote tables on the server, and ensure that the **GenerateAsProxyServer** property on the **Extended Attributes** tab is set to `True`.

   Note: A single data source can contain information for several models if they represent a single remote server.
4. Add the models from which you have drawn your proxy tables in the **Models** tab.

5. Click the **Database Connection** tab, and define the data source name, login and password and click **OK** to return to your model.

Generating the Remote Server and Proxy Tables Creation Scripts

You can generate the remote server and proxy tables creation scripts from the model containing proxy tables in order to run them in the database.

1. Select **Tools > Proxy Tables > GenerateProxy Tables** to open the Generation dialog, and click the **Options** tab.
2. Set an appropriate 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 `True` to generate proxy servers. You can deselect any proxy servers you do not want to generate.
4. Click **OK** to begin generation.

   The generated script is displayed in the Result dialog.
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.

CHAPTER 20 Teradata

To create a PDM with support for features specific to the Teradata DBMS family, select the appropriate version in the DBMS field of the New Model dialog. To view these extensions to the PowerDesigner metamodel in the Resource Editor, select **Database > Edit Current DBMS** and expand the **Profile** node.

Note: The DBMS definition files for Teradata V2R5 and V2R6 are deprecated.

The following sections list the extensions provided for Teradata.

Abstract Data Types

The following extensions are available on the Teradata tab (V2R6 and higher):

Name	Description
Predefined data type Dimension	[type:`distinct`] Indicates that character column comparison uses character case (upper and lower) to raise differences. Scripting name: PredefinedDataType [v14 and higher, type:`array`] Specifies the dimension(s) of the array as `[n1][n2]....` Scripting name: Dimension
Nullify	[v14 and higher, type:`array`] Initializes all of the elements of `array_type_name` to null when the type is constructed. Scripting name: DefaultNull

Abstract Data Type Procedures

The following extensions are available on the Teradata tab if the type is distinct (V2R6 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

Name	Description
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
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
Glop set	[v13 and higher]Specifies the glop set with which the method is associated. Scripting name: GlopSet
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 extensions are available on the Teradata tab:

Name	Description
Character set	Specifies the character set to be used. Scripting name: CharacterSet
Case specific	Specifies that character column comparison is case-sensitive. Scripting name: CaseSpecific
Compress	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
Always generate value	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
Partition	Specifies the partition to which the column is assigned.

Databases

The following extensions are available on the Teradata tab:

Name	Description
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
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

Name	Description
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 journal table	Specifies the default table that is to receive the journal images of data tables created in the new database. Scripting name: DefaultJournalTable
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

Indexes

The following extensions are available on the Teradata tab:

Name	Description
Primary Index	Specifies that the index is the primary index. Scripting name: PrimaryIndex

Name	Description
Partition by	[primary key] Lets you select the used function to evaluate partition condition. • case_n: Evaluates a list of conditions and returns the position of the first condition that evaluates to TRUE, provided that no prior condition in the list evaluates to UNKNOWN. • range_n: Evaluates an expression and maps the result into one of a list of specified ranges and returns the position of the range in the list. Scripting name: PartitionBy
Partition expression	[primary key] Specifies an SQL expression used to define the partition to which a partitioned primary index row is assigned when it is hashed to its AMP. Scripting name: PartitionExpression
Click on the check box to switch multiple / single partition mode	[primary key] Specifies whether the index is defined over multiple partitioning expressions. When this checkbox is selected, you can specify the partition functions and expressions in a list. Scripting name: DisplayMultiplePartitions
Ordering type	[not primary key] Select VALUES to optimize queries that return a contiguous range of values, especially for a covering index or a nested join. Select HASH to limit hash-ordering to one column, rather than all columns (the default) Scripting name: OrderingType
Column	[not primary key] Row ordering on each AMP by a single NUSI column: either value-ordered or hash-ordered. Scripting name: OrderByColumnList
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 extensions are available on the Teradata tab:

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
Commit row action	Specifies the action to take with the contents of a global temporary table when a transaction ends: • DELETE - clears the temporary table of all rows. • PRESERVE - retains the rows in the table after the transaction is committed. Scripting name: CommitRowAction
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
Primary index	Specifies the primary index of the table (see *Primary Indexes (Teradata)* on page 561). Scripting name: PrimaryIndex

Users

The following extensions 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

Name	Description
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
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

Name	Description
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 extensions are available on the Teradata tab:

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

Partitions (Teradata)

Teradata partitions allow you partition table data by range, case, or column. PowerDesigner models partitions as extended sub-objects with a stereotype of `Partition`.

Creating a Partition

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

- Open the property sheet of a table, select the **Partitions** tab and click the **Add a Row** tool. The **Partition** field on the **Teradata** tab is updated to reflect the partitions that you create

- Open the property sheet of a table, select the **Teradata** tab and enter your partition definition in the **Partition** field. Partition objects are created, deleted, or modified to reflect changes in this field.

Partition Properties

You can modify an object's properties from its property sheet. To open a partition property sheet, double-click its Browser entry in the Partitions folder under its parent table.

Name	Description
Table	Specifies the parent table of the partition. Scripting name: ParentObject
Type	Specifies the type of the partition: • Range n - Specify a range and interval in the Expression field. • Case n - Specify criteria for the partition in the Expression field. • Column - [if no primary index is defined on the table] Create objects in the Column Groups list, open their property sheets and associate columns with them. Select the **All but** option to create a single-column partition with autocompression and a system-determined COLUMN or ROW format for each column, if any, that is not specified in the column group list. Scripting name: PartitionType, AllBut
Expression	Specifies the partitioning expression for partitions of type Range n or Case n. Scripting name: Expression
Column Groups	Lists the groups of columns that will be partitioned for partitions of type Column. Select an item in the list and click the **Properties** tool to define its type, and the columns of the parent table to which it applies. You can specify partitioning by: • Row • Column • Auto - Teradata determines the optimum partitioning format. Select the **All but** option to compress data as physical rows that are inserted into that column partition of a column-partitioned table if an appropriate method can be calculated. Scripting name: PartitionColumns
Add	Specifies that the maximum number of partitions for a partitioning level is the number of partitions it defines plus the value of the BIGINT constant value specified in this field. Scripting name: AddConstant

Name	Description
Partition sql	Specifies the SQL statement that defines the partition. You can enter SQL in this field to generate appropriate PowerDesigner objects or create the objects and have them generate the SQL in this field. Changes to objects or the SQL are synchronized with the other. Scripting name: Gen

Transform Groups (Teradata)

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 a 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.

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

Name	Description
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

Database Permissions (Teradata)

You can define multiple databases in a PDM for Teradata, and also define *permissions* on the database object.

For more information on permissions, see *Granting Object Permissions* on page 154.

Primary Indexes (Teradata)

In Teradata, users tend to use indexes rather than key constraints.

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.

Error Tables (Teradata)

Teradata can record errors encounterd when writing to a data table in an error table associated with the data table. Error tables are supported for Teradata v12 and higher. PowerDesigner models error tables as extended objects with a stereotype of <<ErrorTable>>.

Creating an Error Table

You can create an error table in any of the following ways:

- Select **Model > Error Tables** to access the List of Error Tables, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New > Error Table**.

Error Table Properties

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

The following extended attributes are available on the **General** tab:

Name	Description
Owner	Specifies the name of the database containing the error table. Scripting name: Owner
Data table	Specifies the data table for which the error table is being created. Scripting name: DataTable
Use name at generation	Specifies that the error table will be generated with its name. Scripting name: HasName

Join Indexes (Teradata)

Join indexes are materialized views that improve access times for cross-table queries, and which are automatically updated when changed are made to the underlying tables. Join indexes are supported for Teradata v12 and higher. PowerDesigner models join indexes as views with a stereotype of <<JoinIndex>>.

Creating a Join Index

You can create a join index in any of the following ways:

- Select **Model > Join Indexes** to access the List of Join Indexes, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New > Join Index**.

To complete the view, specify a view query (see *View Queries* on page 115).

Join Index Properties

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

The following extended attributes are available on the **General** tab:

Name	Description
Fallback	Specifies that the join index uses fallback protection. Scripting name: Fallback
Checksum	Enables a table-specific disk I/O integrity checksum level. The checksum setting applies to primary data rows, fallback data rows, and all secondary index rows for the index. Scripting name: Checksum

Hash Indexes (Teradata)

Hash indexes are designed to improve query performance like join indexes, but may in addition enable you to avoid accessing the base table. Hash indexes are supported for Teradata v12 and higher. PowerDesigner models hash indexes as extended objects with a stereotype of <<HashIndex>>.

Creating a Hash Index

You can create a hash index in any of the following ways:

- Select **Model > Hash Indexes** to access the List of Hash Indexes, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New > Hash Index**.

Hash Index Properties

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

The following extended attributes are available on the **General** tab:

Name	Description
Table	Specifies the base table on which the hash index is defined. Scripting name: Table
Database	Specifies the name of the database containing the base table. By default the same as the database in which the hash index is created. Scripting name: Owner
Fallback	Specifies that the hash index uses fallback protection. Scripting name: Fallback

Name	Description
Checksum	Enables a table-specific disk I/O integrity checksum level. The checksum setting applies to primary data rows, fallback data rows, and all secondary index rows for the index. Scripting name: Checksum

The following extended attributes are available on the Teradata tab:

Name	Description
Columns	Specifies the base table columns on which the hash index is defined Scripting name: Columns
Distributed columns	Specifies an optional, explicitly specified column set on which the hash index rows are distributed across the AMPs. This is a subset of index column list. Scripting name: ByColumns
Order by columns	Specifies the row ordering on each AMP, which must be either value-ordered or hash-ordered. Scripting name: OrderByColumns
Ordering type	[if Order by columns are specified] Specifies the ordering type of the ORDER BY column. Scripting name: OrderByType

Glop Sets (Teradata)

Glop sets are sets of persistent data used in external procedures and functions. PowerDesigner supports glop sets for Teradata v13 and higher as extended objects with a stereotype of <<GlopSet>>.

Creating a Glop Set

You can create a glop set in any of the following ways:

- Select **Model > Glop Sets** to access the List of Glop Sets, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New > Glop Set**.

Glop Set Properties

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

The following extended attributes are available on the **General** tab:

Name	Description
Owner	Specifies the owner of the glop set. Scripting name: Owner

Replication Groups (Teradata)

Replication groups contain tables to be replicated. PowerDesigner supports replication groups for Teradata v13 and higher as extended objects with a stereotype of <<ReplicationGroup>>.

Creating a Replication Group

You can create a replication group in any of the following ways:

- Select **Model > Replication Groups** to access the List of Replication Groups, and click the **Add a Row** tool.
- Right-click the model (or a package) in the Browser, and select **New > Replication Group**.

Replication Group Properties

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

The following extended attributes are available on the **General** tab:

Name	Description
List of tables	Specifies the tables to be included in the replication group. You can enter table names here as a comma-separated list and on the **Tables** tab. Both lists are synchronized and if any table name does not currently exist in the model, then it will be created. Scripting name: TableList

Replication Rules and Rule Sets (Teradata)

Replication rules are patterns for matching table names to include in replication groups. Rules are collected into sets, which are in turn associated with replication groups. PowerDesigner supports replication rule sets and rules for Teradata v13 and higher as extended objects with a stereotype of <<ReplicationRuleSet>> and extended sub-objects with a stereotype of <<ReplicationRule>>.

Creating a Replication Rule Set

You can create a replication rule set in any of the following ways:

- Select **Model > Replication Rule Sets** to access the List of Replication Rule Sets, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select **New > Replication Rule Set**.

Creating Replication Rules

You create replication rules on the **Patterns** tab of a replication rule set. You can define the rule on the tab or by clicking the **Properties** tool to open the rule properties sheet. Rules have the following properties:

Name	Description
Object kind	Specifies the type of database object to be added to the replication rule set. Scripting name: ObjectKind
Like/And not like	Specifies pattern strings to match or exclude against the fully qualified names of the objects of certain SQL statements. The specified string literals can contain wildcard characters. Scripting name: LikeClause, NotLikeClause
Escape character	Specifies an escape character for the like and not like patterns. Scripting name: EscapeLike, EscapeNotLike
Sql	[property sheet only] Displays the SQL expression corresponding to the values entered in the other fields. Scripting name: Definition

Replication Rule Set Properties

You can modify an object's properties from its property sheet. To open a replication rule set property sheet, double-click its diagram symbol or its Browser entry in the Replication Rule Sets folder.

The following extended attributes are available on the General tab:

Name	Description
Default	Specifies that all the rules in the rule set are default rules. Scripting name: DefaultRules
Replication group	Specifies the name of the replication group to which the rule set is assigned. Scripting name: ReplicationGroup

CHAPTER 21 Other Databases

The following sections list extensions to other DBMS families supported by PowerDesigner.

Informix SQL

To create a PDM with support for features specific to the Informix SQL DBMS family, select the appropriate version in the DBMS field of the New Model dialog. To view these extensions to the PowerDesigner metamodel in the Resource Editor, select **Database > Edit Current DBMS** and expand the **Profile** node.

Note: The DBMSs for Informix v8-9 are deprecated.

The following sections list the extensions provided for Informix SQL.

Columns

The following extensions are available on the Informix tab:

Name	Description
Serial Start	Specifies the initial value of the column with a SERIAL datatype. Scripting name: ExtSerialStart

Indexes

The following extensions are available on the Extended Attributes tab:

Name	Description
IndexSpec	Specifies an internal index definition (indexkeys column). Scripting name: IndexSpec

Procedures

The following extensions are available on the Extended Attributes tab:

Name	Description
InternalID	Specifies an internal identifier in the server, which is used to retrieve the function of an index expression. Scripting name: InternalID

Ingres

To create a PDM with support for features specific to the Ingres DBMS family, select the appropriate version in the DBMS field of the New Model dialog. To view these extensions to the PowerDesigner metamodel in the Resource Editor, select **Database** > **Edit Current DBMS** and expand the **Profile** node.

The following sections list the extensions provided for Ingres.

Columns

The following extensions are available on the Extended Attributes tab:

Name	Description
NotDefault	Specifies that the column needs a value. This generates the "not default" clause in the sql statement. Scripting name: NotDefault

Users

The following extensions 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
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
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

Interbase

To create a PDM with support for features specific to the Interbase DBMS family, select the appropriate version in the DBMS field of the New Model dialog. To view these extensions to the PowerDesigner metamodel in the Resource Editor, select **Database** > **Edit Current DBMS** and expand the **Profile** node.

The following sections list the extensions provided for Interbase.

Indexes

The following extensions 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 extensions 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

To create a PDM with support for features specific to the MS Access DBMS family, select the appropriate version in the DBMS field of the New Model dialog. To view these extensions to the PowerDesigner metamodel in the Resource Editor, select **Database** > **Edit Current DBMS** and expand the **Profile** node.

Note: The DBMS definition file for Microsoft Access 2000 is deprecated.

The following sections list the extensions provided forMS Access.

Columns

The following extended attributes are available on the Access 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

Generating a Microsoft Access Database

PowerDesigner and MS Access use `.dat` files to exchange information. You must pass via the appropriate `access`*`version`* database delivered with PowerDesigner in order to convert the `.dat` files generated into Access database files.

1. Select **Database > Generate Database** to launch the standard Database Generation dialog (see *Generating a Database from a PDM* on page 290), set any appropriate options, and click **OK**.
2. Open the appropriate `access`*`version`* database in the PowerDesigner `\tools` directory.
3. Select the **Generate Access database from script file** radio button and enter or select a destination database file in the **Select database** field.
4. Select the `.dat` file generated by PowerDesigner in the **Script file** field.
5. Click the **Create** button to create the database file, and then click the **Open MDB** button to open the generated database.

Reverse Engineering a Microsoft Access Database

PowerDesigner and MS Access use `.dat` files to exchange information. You must pass via the appropriate `access`*`version`* database delivered with PowerDesigner in order to convert an Access database file into the `.dat` file required by PowerDesigner.

1. Open the appropriate `access`*`version`* database in the PowerDesigner `\tools` directory.
2. Select the **Reverse engineer Access database to script** radio button and select the database file to reverse in the **Select database** field.
3. Enter the `.dat` file to be generated in the **Script file** field.
4. Click the **Create** button to generate the `.dat` file and then reverse engineer this script in PowerDesigner (see *Reverse Engineering from Scripts* on page 312).

MySQL

To create a PDM with support for features specific to the MySQL DBMS family, select the appropriate version in the DBMS field of the New Model dialog. To view these extensions to the PowerDesigner metamodel in the Resource Editor, select **Database** > **Edit Current DBMS** and expand the **Profile** node.

Note: The DBMSs for MySQL v3.22 and 3.23 are deprecated. In v4.0 the attributes listed below are available on the **Extended Attributes** tab.

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'
```

The following sections list the extensions provided for MySQL.

Columns

The following extended attributes are available on the MySQL tab:

Name	Description
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
Unsigned	Indicates negative values are not allowed for the column. Scripting name: Unsigned
National	A way to indicate that a CHAR column should use UTF8 character set. Scripting name: National
Character set	Set of symbols and encodings. Scripting name: CharSet
Collation	Set of rules for comparing characters in a character set. Scripting name: Collate

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

Tables

The following extended attributes are available on the MySQL tab:

Name	Description
Temporary table	[v5.0 and higher] Used to create a temporary table. A temporary table is visible only to the current connection, and is dropped automatically when the connection is closed. Scripting name: Temporary

NonStop SQL

To create a PDM with support for features specific to the NonStop SQL DBMS family, select the appropriate version in the DBMS field of the New Model dialog. To view these extensions to the PowerDesigner metamodel in the Resource Editor, select **Database > Edit Current DBMS** and expand the **Profile** node.

The following sections list the extensions provided for NonStop SQL.

Columns

The following extensions are available on the Extended Attributes tab:

Name	Description
ExtType	Specifies an extended type for columns. Select either signed or unsigned in the Value column. Scripting name: ExtType

PostgreSQL

To create a PDM with support for features specific to the PostgreSQL DBMS family, select the appropriate version in the DBMS field of the New Model dialog. To view these extensions to the PowerDesigner metamodel in the Resource Editor, select **Database** > **Edit Current DBMS** and expand the **Profile** node.

The following sections list the extensions provided for PostgreSQL.

Databases

The following extensions 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

Domains

The following extensions are available on the PostgreSQL tab. To display this tab, select `BaseType` or `CompositeType` in the **Stereotype** field on the **General** tab and click **Apply**:

Name	Description
Definition	[Composite Type] 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
Length	[Base Type] Specifies the internal length of the new type. Scripting name: ExtTypeLength
Array Element type	[Base Type] Specifies the type of the array elements. Scripting name: ExtTypeElement
Array delimiter	[Base Type] Specifies the delimiter character for the array. Scripting name: ExtTypeDelimiter
By Value	[Base Type] Specifies that operators and functions which use this data type should be passed an argument by value rather than by reference. Scripting name: ExtTypePassedByValue
Input function	[Base Type] Specifies the 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
Output function	[Base Type] Specifies the name of a function, created by CREATE FUNCTION, which converts data from its internal form to a form suitable for display. Scripting name: ExtTypeOutput
Send function	[Base Type] Specifies the 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
Receive function	[Base Type] Specifies the 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

Groups

The following extensions 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 extensions 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 extensions 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 deferred	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 extensions 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 extensions 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 extensions are available on the General tab (v8 and higher):

Name	Description
Is schema	Specifies that the user is a schema. If TRUE, the user is allowed to create databases. Scripting name: Schema
Owner	[schemas] Specifies the owner of the schema. Scripting name: Owner

The following extensions are available on the PostgreSQL tab (v8 and higher):

Name	Description
User identifier (id)	Specifies 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
Create database	Specifies that the user can create databases. Scripting name: CreateDB

Name	Description
Create user	Specifies that the user can create users and turns the user into a superuser who can override all access restrictions. Scripting name: CreateUser
Validity	Specifies an absolute time after which the user's password is no longer valid. By default, the password will be valid forever. Scripting name: Validity
Encrypted password	Specifies that the password is stored encrypted in the system catalogs. Scripting name: EncryptedPassword

Red Brick Warehouse

To create a PDM with support for features specific to the Red Brick Warehouse DBMS family, select the appropriate version in the DBMS field of the New Model dialog. To view these extensions to the PowerDesigner metamodel in the Resource Editor, select **Database > Edit Current DBMS** and expand the **Profile** node.

The following sections list the extensions provided for Red Brick Warehouse.

Columns

The following extensions are available on the Red Brick tab:

Name	Description
Unique	Specifies 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 extensions are available on the Red Brick tab:

Name	Description
Macro Type	Specifies the type of macro. You can choose either Public or Temporary. If you do not select a type, a private macro is created by default. Scripting name: MacroType

Index

A

B

C

E

F

G

N

O

P

Q

R

S

T