

SYBASE®

Data Modeling

PowerDesigner® 15.3

Windows

DOCUMENT ID: DC38058-01-1530-01

LAST REVISED: November 2010

Copyright © 2010 by Sybase, Inc. All rights reserved.

This publication pertains to Sybase software and to any subsequent release until otherwise indicated in new editions or technical notes. Information in this document is subject to change without notice. The software described herein is furnished under a license agreement, and it may be used or copied only in accordance with the terms of that agreement.

To order additional documents, U.S. and Canadian customers should call Customer Fulfillment at (800) 685-8225, fax (617) 229-9845.

Customers in other countries with a U.S. license agreement may contact Customer Fulfillment via the above fax number. All other international customers should contact their Sybase subsidiary or local distributor. Upgrades are provided only at regularly scheduled software release dates. No part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without the prior written permission of Sybase, Inc.

Sybase trademarks can be viewed at the Sybase trademarks page at <http://www.sybase.com/detail?id=1011207>. Sybase and the marks listed are trademarks of Sybase, Inc. A ® indicates registration in the United States of America.

Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Unicode and the Unicode Logo are registered trademarks of Unicode, Inc.

All other company and product names used herein may be trademarks or registered trademarks of the respective companies with which they are associated.

Use, duplication, or disclosure by the government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of DFARS 52.227-7013 for the DOD and as set forth in FAR 52.227-19(a)-(d) for civilian agencies.

Sybase, Inc., One Sybase Drive, Dublin, CA 94568

Contents

PART I: Building Data Models	1
CHAPTER 1: Getting Started with Data Modeling	3
Conceptual Data Models	3
Logical Data Models	3
Physical Data Models	4
DBMS-Specific Features	5
Creating a Data Model	6
Data Model Properties	8
Database Properties (PDM)	9
Archiving a PDM	10
Opening a V6 PAM into a CDM	11
Customizing your Modeling Environment	11
Setting CDM/LDM Model Options	12
Assertion Template	14
Migration Settings (LDM)	15
Setting PDM Model Options	16
Column and Domain Model Options	17
Reference Model Options	18
Other Object Model Options	20
Setting Data Model Display Preferences	21
Supported CDM/LDM Notations	21
Viewing and Editing the DBMS Definition File	25
Changing the DBMS	26
Extending your Modeling Environment	27
Creating Links with Extended Dependencies	28
CHAPTER 2: Building Conceptual and Logical Diagrams	29

Conceptual Diagram Basics	29
Conceptual Diagram Objects	30
Creating a Conceptual Diagram	31
Example: Building a Data Dictionary in a CDM	31
Logical Diagram Basics	41
Logical Diagram Objects	42
Creating a Logical Diagram	43
Importing a Deprecated PDM Logical Model	43
Importing Multiple Interconnected PDM Logical Models	44
Data Items (CDM)	45
Creating a Data Item	45
Data Item Properties	45
Controlling Uniqueness and Reuse of Data Items	47
Entities (CDM/LDM)	47
Creating an Entity	47
Entity Properties	48
Copying an Entity	48
Copying an Entity Within a Model	49
Copying an Entity to a Different Model	49
Displaying Attributes and Other Information on an Entity Symbol	49
Attributes (CDM/LDM)	52
Creating an Attribute	52
Attribute Properties	53
Deleting Attributes (CDM)	54
Identifiers (CDM/LDM)	54
Creating an Identifier	55
Identifier Properties	55
Relationships (CDM/LDM)	56
Creating a Relationship	56
Relationship Properties	56
Relationship Property Sheet Cardinalities Tab . . .	58
Relationship Property Sheet Joins Tab (LDM)	60

Enabling Many-to-many Relationships in an LDM	62
.....	62
Creating a Reflexive Relationship	63
Defining a Code Option for Relationships	63
Changing a Relationship into an Associative Entity	64
Changing a Relationship Directly into an	
Associative Entity	64
Changing a Relationship into an Associative	
Entity Using the Change to Entity Wizard	64
Relationship Examples	65
Identifier Migration Along Relationships	67
Associations and Association Links (CDM)	67
Creating an Association with Links	68
Creating an Association Without Links	68
Association Properties	69
Association Link Properties	69
Creating a Reflexive Association	70
Defining a Dependent Association	70
Changing an Association into an Associative Entity	71
Creating an Association Attribute	72
Inheritances (CDM/LDM)	72
Creating an Inheritance	73
Creating an Inheritance with the Inheritance	
Tool	73
Inheritance Properties	74
Inheritance Property Sheet Generation Tab	74
Inheritance Property Sheet Children Tab	76
Making Inheritance Links Mutually Exclusive	77
Identifier Migration Through an Inheritance (LDM)	77
CHAPTER 3: Building Physical Diagrams	79
Physical Diagram Objects	79
Creating a Physical Diagram	81
Tables (PDM)	81

Creating a Table	82
Table Properties	82
Linking a Table to an Abstract Data Type	84
Creating an XML Table or View	85
Naming a Table Constraint	86
Creating External Tables	87
Denormalizing Tables and Columns	87
Creating Horizontal Partitions	87
Creating Vertical Partitions	89
Creating Table Collapsings	91
Denormalizing Columns	93
Denormalization Object Properties	94
Example: Intermodel Generation and Horizontal Partitions	95
Example: Intermodel Generation and Vertical Partitions	97
Removing Partitionings and Table Collapsings	98
Using PowerBuilder Extended Attributes	99
Generating PowerBuilder Extended Attributes .	99
Reverse Engineering PowerBuilder Extended Attributes	100
Displaying Column, Domain, and Data Type Information on a Table Symbol	101
Columns (PDM)	103
Creating a Column	103
Column Properties	103
Obtaining Column Statistics from your Database	105
Setting Data Profiling Constraints	105
Specifying Constraints Through Business Rules	107
Creating Data Formats For Reuse	107
Specifying Advanced Constraints	108
Populating Columns with Test Data	109
Test Data Profile Properties	110
Assigning Test Data Profiles to Columns	113

Creating a Computed Column	113
Selecting a Data Type for a Column	115
Attaching a Column to a Domain	116
Copying a Column to Another Table	116
Naming a Column Constraint	117
Configuring the Display of the List of Columns	117
Keys (PDM)	118
Creating a Key	119
Key Properties	119
Primary Keys	120
Rebuilding Primary Keys	120
Alternate Keys	121
Foreign Keys	122
Adding Parent Table Columns to a Key	122
Naming Key Constraints	122
Indexes (PDM)	123
Creating an Index	123
Index Properties	127
Rebuilding Indexes	128
Indexes in Query Tables	130
Users, Groups, and Roles (PDM)	131
Creating a User, Group, or Role	131
User, Group, and Role Properties	131
Assigning an Owner to an Object	132
Specifying Default Owners for Object Types	133
Granting System Privileges	133
Generating Privileges	135
Granting Object Permissions	136
Defining Column Permissions	138
Inserting a User into a Group	140
Assigning a User to a Role	141
Synonyms (PDM)	142
Creating a Synonym	143
Synonym Properties	145
Creating a View for a Synonym	145

Defaults (PDM)	146
Creating a Default	146
Default Properties	147
Assigning a Default to a Column or a Domain	147
Rebuilding Defaults	148
Domains (CDM/LDM/PDM)	148
Creating a Domain	149
Domain Properties	149
List of Standard Data Types	151
Cascading Updates to Columns/entity Attributes	
Associated with the Domain	153
Enforcing Non-divergence from a Domain in a Data	
Model	154
Sequences (PDM)	156
Creating a Sequence	156
Creating a Sequence and Specifying its Physical	
Options	156
Applying and Enabling a Sequence on a Column	
.....	158
Sequence Properties	158
Changing the DBMS of a Model which Contains	
Sequences and Auto-incremented Columns	159
Sequences and Intermodel Generation	160
Abstract Data Types (PDM)	160
Creating an Abstract Data Type	161
Abstract Data Type Properties	161
Creating Object and SQLJ Object Abstract Data Types	
.....	162
Linking an Abstract Data Type to a Java Class	164
Reverse-engineering a PDM Linked to an OOM	
.....	167
References (PDM)	168
Creating a Reference	168
Automatic Reuse and Migration of Columns	169
Examples	170

Reference Properties	172
Reference Property Sheet Joins Tab	173
Reference Property Sheet Integrity Tab	175
Rebuilding References	177
Displaying Referential Integrity and Cardinality on a Reference Symbol	178
Views (PDM)	179
Creating a View	179
Creating a View from the Tools Menu	179
View Properties	180
View Column Properties	182
Working with View Queries	183
Query Property Sheet SQL Tab	184
Query Property Sheet Tables Tab	184
Query Property Sheet Columns Tab	185
Query Property Sheet Where Tab	185
Query Property Sheet Group By Tab	185
Query Property Sheet Having Tab	186
Query Property Sheet Order By Tab	186
Materialized Views	186
Using Extended Dependencies for Views	187
Defining a Generation Order for Views	187
View References (PDM)	190
Creating a View Reference	190
View Reference Properties	190
Defining View Reference Joins	191
Business Rules (CDM/LDM/PDM)	193
Creating a Business Rule	194
Business Rule Properties	194
Applying a Business Rule to a Model Object	195
Example: Creating and Attaching a Constraint Rule ..	196

CHAPTER 4: Building Multidimensional Diagrams . . . 201

Multidimensional Diagram Objects	202
---	------------

Creating a Multidimensional Diagram	203
Cubes (PDM)	203
Creating a Cube	203
Cube Properties	204
Retrieving Multidimensional Objects	204
Rebuilding Cubes	205
Generating Extraction Scripts	207
Extraction Script Generation Options Tab	209
Generating Cube Data	210
Cube Data Generation Options	211
Dimensions (PDM)	212
Creating a Dimension	212
Dimension Properties	212
Attributes (PDM)	213
Creating an Attribute	214
Attribute Properties	214
Attribute Property Sheet Detail Attributes Tab	214
Facts (PDM)	215
Creating a Fact	215
Fact Properties	216
Measures (PDM)	216
Creating a Measure	216
Measure Properties	217
Hierarchies (PDM)	217
Creating a Hierarchy	217
Hierarchy Properties	218
Associations (PDM)	218
Creating an Association	219
Association Properties	219
CHAPTER 5: Building Triggers and Procedures	221
Triggers (PDM)	221
Creating Triggers	221

Implementing Referential Integrity with Triggers	222
Creating a Trigger from a Template	222
Creating a Trigger from Scratch	223
Trigger and DBMS Trigger Properties	224
Rebuilding Triggers	226
Rebuilding Triggers Automatically	227
Rebuilding Triggers Manually	227
Modifying Triggers	228
Inserting a Template Item into a Trigger or Trigger Template	229
Declaring a Template Item in a Trigger Definition	231
Trigger Naming Conventions	232
Calling a Related Procedure in a Trigger Template ...	233
Multiple Triggers	233
Indicating Trigger Order	234
Defining Triggers with Multiple Events	234
DBMS Triggers (PDM)	234
Creating DBMS Triggers	235
Trigger Templates (PDM)	235
Creating a Trigger Template	236
PowerDesigner Pre-defined Trigger Templates	238
Modifying a Trigger Template	240
Trigger Template Properties	240
Trigger Template Items (PDM)	241
Creating a Trigger Template Item	242
PowerDesigner Pre-defined Trigger Template Items ..	245
Modifying a Trigger Template Item	247
Trigger Template Item Properties	247
Stored Procedures and Functions (PDM)	248
Creating a Stored Procedure or Function	248
Procedure Properties	250
Tracing Trigger and Procedure Dependencies	250
Creating Procedure Dependencies Manually	254

Rebuilding Trigger and Procedure Dependencies	255
Attaching a Stored Procedure to a Table	256
Rebuilding Procedures Attached to Tables	257
Procedure Templates (PDM)	258
Creating a Procedure Template	259
PowerDesigner Pre-defined Procedure Templates ...	260
Modifying a Procedure Template	261
Procedure Template Properties	261
Creating SQL/XML Queries with the Wizard	262
Generating Triggers and Procedures	266
Defining a Generation Order for Stored Procedures	267
Creating and Generating User-defined Error Messages	269
Creating a Message Table	269
Generating a User-defined Error Message	270
CHAPTER 6: Building Web Services	273
Web Services (PDM)	274
Web Services in Sybase ASA, ASE, and IQ	274
Web Services in IBM DB2	275
Creating a Web Service	275
Web Service Properties	275
Web Operations (PDM)	277
Creating a Web Operation	277
Web Operation Properties	277
Web Operation Result Columns	280
Web Parameters (PDM)	280
Creating a Web Parameter	280
Web Parameter Properties	280
Testing Web Services	281
Generating Web Services	282

Generating Web Services for Sybase ASA, ASE, and IQ	282
Generating Web Services for IBM DB2	282
Reverse Engineering Web Services	285
CHAPTER 7: Building a Physical Implementation . . .	289
Lifecycles (PDM)	289
Modeling a Lifecycle	290
Generating Data Archiving Scripts to Implement your Lifecycle	291
Lifecycle Properties	293
Archiving Data From External Databases	295
Linking an External Database by Generation . . .	295
Linking an External Database through the Mapping Editor	296
Linking an External Database via the Data Source Wizard	296
Phases (PDM)	297
Creating a Phase	297
Phase Properties	297
Tablespaces and Storages (PDM)	298
Creating a Tablespace or Storage	299
Tablespace and Storage Properties	299
Partitions (PDM)	300
Creating a Partition	300
Partition Properties	301
Physical Options	301
Defining Default Physical Options	302
Defining Physical Options for Individual Objects	304
PART II: Working with Data Models	307
CHAPTER 8: Checking a Data Model	309

Abstract Data Type Checks (PDM)	309
Abstract Data Type Procedure Checks (PDM)	310
Association Checks (CDM)	311
Association Checks (PDM)	312
Column Checks (PDM)	313
Cube Checks (PDM)	315
Database Checks (PDM)	316
Database Package Checks (PDM)	316
Database Package Subobject Checks (PDM)	317
Data Format Checks (CDM/LDM/PDM)	318
Data Item Checks (CDM)	318
Data Source Checks (PDM)	320
Default Checks (PDM)	320
Dimension Checks (PDM)	321
Domain Checks (CDM/LDM/PDM)	322
Entity Attribute Checks (CDM/LDM)	323
Entity Identifier Checks (CDM/LDM)	324
Entity Checks (CDM/LDM)	325
Fact Checks (PDM)	326
Fact Measure and Dimension Hierarchy and Attribute Checks (PDM)	327
Horizontal and Vertical Partitioning and Table Collapsing Checks (PDM)	328
Index and View Index Checks (PDM)	328
Inheritance Checks (CDM/LDM)	329
Join Index Checks (PDM)	330
Key Checks (PDM)	331
Lifecycle and Lifecycle Phase Checks (PDM)	332
Package Checks (CDM/LDM/PDM)	333
Procedure Checks (PDM)	335
Reference and View Reference Checks (PDM)	336
Relationship Checks (CDM/LDM)	337
Sequence Checks (PDM)	338
Synonym Checks (PDM)	338
Table and View Checks (PDM)	339

Tablespace and Storage Checks (PDM)	342
Trigger and DBMS Trigger Checks (PDM)	342
User, Group, and Role Checks (PDM)	343
View Checks (PDM)	344
Web Service and Web Operation Checks (PDM)	344

CHAPTER 9: Generating and Reverse-Engineering

Databases 345

Generating a Database from a PDM	345
Connecting to a Database	345
Generating a Database	346
Database Generation Dialog General Tab	349
Database Generation Dialog Options Tab	351
Database Generation Dialog Format Tab	354
Database Generation Dialog Selection Tab	356
Database Generation Dialog Summary Tab	357
Database Generation Dialog Preview Tab	358
Generating a Microsoft Access 97 Database	358
Customizing Scripts	358
Generating Test Data to a Database	361
Test Data Generation General Tab	362
Test Data Generation Number of Rows Tab	364
Test Data Generation Format Tab	365
Estimating Database Size	366
Modifying a Database	368
Apply Model Changes to Database Dialog	
Options	370
Displaying Data from a Database	372
Executing SQL Queries	372
Reverse Engineering a Database into a PDM	373
Reverse Engineering from Scripts	373
Reverse Engineering from a Live Database	375
Reverse Engineering Options Tab	378
Reverse Engineering Encoding Format	380

Database Reverse Engineering Selection Window	381
Reverse Engineering Target Models Tab	382
Reverse Engineering a Microsoft Access 97 Database	383
Optimizing Live Database Reverse Engineering Queries	384
Reverse Engineering Database Statistics	384
CHAPTER 10: Generating Other Models from a Data Model	387
Generating Other Models from a CDM	388
Generating PDM Table Keys from CDM Entity Identifiers	388
Generating Tables from Entities with Inheritance Links	390
Generating Other Models from an LDM	391
Generating Other Models from a PDM	391
Configuring the Generated Model Options	393
Generating an XSM from a PDM Via the XML Builder Wizard	393
CHAPTER 11: Working with SQL Statements in PowerDesigner	397
Previewing SQL Statements	397
Writing SQL Code in the PowerDesigner SQL Editor	399
SQL Editor Tools	401
Writing SQL using GTL	401
Writing SQL using PDM Variables and Macros	403
PDM Variables	404
Formatting Variables	416
PDM Macros	417

CHAPTER 12: Migrating from ERwin to PowerDesigner	429
.....	
Supported ERwin File Types	429
ERwin Import Limitations	430
Preparing to Import Your ERwin Models	430
Importing Individual ERwin Files	430
Importing Multiple ERwin Files	432
After Importing	433
PowerDesigner vs ERwin Terminology	433
Post-import Checklist	434
Differences to Expect	436
Getting Started Using PowerDesigner for Former ERwin Users	437
 PART III: DBMS Definition Reference	 439
 CHAPTER 13: HP Neoview	 441
Materialized View Groups (Neoview)	444
 CHAPTER 14: IBM DB2 for z/OS (formerly OS/390)	 447
.....	
Trusted Contexts (DB2)	449
Auxiliary Tables (DB2)	450
Tablespace Prefix (DB2)	451
 CHAPTER 15: IBM DB2 for Common Server	 453
Database Partition Groups (DB2)	459
Index Extensions (DB2)	460
Security Policies (DB2)	460
Security Labels (DB2)	461
Security Label Components (DB2)	462

Event Monitors (DB2)	463
CHAPTER 16: Microsoft SQL Server	467
Horizontal Partitioning (SQL Server)	482
Partition Functions (SQL Server)	482
Partition Schemes (SQL Server)	483
Common Language Runtime (CLR) Integration (SQL Server)	484
CLR Assemblies (SQL Server)	484
CLR Aggregate Functions (SQL Server)	485
CLR User-Defined Types (SQL Server)	486
CLR Procedures, Functions, and Triggers (SQL Server)	487
Encryption (SQL Server)	488
Certificates (SQL Server)	488
Asymmetric Keys (SQL Server)	490
Symmetric Keys (SQL Server)	491
Full Text Search (SQL Server)	492
Full-Text Catalogs (SQL Server)	492
Full-Text Indexes (SQL Server)	493
XML Indexes (SQL Server)	494
XML Data Types (SQL Server)	495
XML Schema Collections (SQL Server)	496
Database Mirroring (SQL Server)	497
End Points (SQL Server)	499
Service Broker (SQL Server)	500
Message Types (SQL Server)	500
Contracts (SQL Server)	501
Message Contracts (SQL Server)	502
Queues (SQL Server)	503
Event Notifications (SQL Server)	505
Services (SQL Server)	506
Routes (SQL Server)	507
Remote Service Bindings (SQL Server)	508

Resource Governor (SQL Server)	509
Workload Groups (SQL Server)	509
Resource Pools (SQL Server)	510
Schemas (SQL Server)	511
Synonyms (SQL Server)	512
Analysis Services (SQL Server 2000)	512
Generating Cubes	514
Reverse Engineering Cubes	515
Analysis Services (SQL Server 2005)	517
Specifying a Data Source for Cubes	517
Generating Cubes for Microsoft SQL Server 2005	518
Reverse Engineering Microsoft SQL Server 2005 Cubes	522
 CHAPTER 17: Oracle	 529
Object and SQLJ Object Data Types (Oracle)	535
Bitmap Join Indexes (Oracle)	535
Automatically Creating Bitmap Join Indexes Through Rebuilding	535
Manually Creating Bitmap Join Indexes	536
Bitmap Join Index Properties	536
Database Packages (Oracle)	537
Database Package Procedures	539
Database Package Variables	539
Database Package Cursors	540
Database Package Exceptions	541
Database Package Types	541
Database Package Parameters	542
Database Package Templates	542
Rebuilding Table Database Packages	543
Transparent Data Encryption (Oracle)	544
Clusters (Oracle)	545
Database Links (Oracle)	545
Materialized View Logs (Oracle)	547

CHAPTER 18: Sybase ASE	549
Proxy Tables (ASE)	552
Encryption Keys (ASE)	552
CHAPTER 19: Sybase IQ	555
Information Lifecycle Management (IQ)	560
Events (IQ/SQL Anywhere)	560
DBSpaces (IQ)	562
Multiplex Servers (IQ)	563
Login Policies (IQ/SQL Anywhere)	564
Full Text Searches (IQ/SQL Anywhere)	565
Text Configurations (IQ/SQL Anywhere)	565
Text Indexes (IQ/SQL Anywhere)	566
Indexes (IQ)	567
Rebuilding IQ Indexes	567
Index Types	568
Join Indexes (IQ/Oracle)	569
Automatically Creating Join Indexes Through	
Rebuilding	571
Adding References to a Join Index	572
Generating IQ Data Movement Scripts	572
Model Properties Data Movement Tab	573
Creating a Data Source to Populate Your IQ Data	
Warehouse	573
Data Source Properties Data Movement Tab	574
Specifying Data Movement Options	574
Table Properties Data Movement Tab	574
Specifying Mappings Between the Tables in Your	
Data Source and Your AS IQ Database	575
Generating the Data Movement Script	575
CHAPTER 20: Sybase SQL Anywhere	577

Auto-increment Columns	580
Mirror Servers (SQL Anywhere)	581
Spatial Data (SQL Anywhere)	583
Spatial Reference Systems (SQL Anywhere)	583
Spatial Units of Measure (SQL Anywhere)	585
Events, Login Policies, and Full Text Searches (SQL Anywhere)	586
Proxy Tables (ASE/SQL Anywhere)	586
Creating a Proxy Table	587
Defining the Remote Server of a Proxy Table	588
Generating the Remote Server and Proxy Tables Creation Scripts	588
CHAPTER 21: Teradata	591
Transform Groups (Teradata)	598
Database Permissions (Teradata)	599
Primary Indexes (Teradata)	599
Error Tables (Teradata)	600
Join Indexes (Teradata)	600
Hash Indexes (Teradata)	601
Glop Sets (Teradata)	602
Replication Groups (Teradata)	603
Replication Rules and Rule Sets (Teradata)	603
CHAPTER 22: Other Databases	607
Informix SQL	607
Ingres	608
Interbase	609
Microsoft Access	609
MySQL	610
Netezza	612
NonStop SQL	612
PostgreSQL	613
Red Brick Warehouse	617

Index619

PART I

Building Data Models

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

CHAPTER 1 Getting Started with Data Modeling

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

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 6. For detailed information about conceptual diagrams, see *Conceptual Diagram Basics* on page 29.

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:

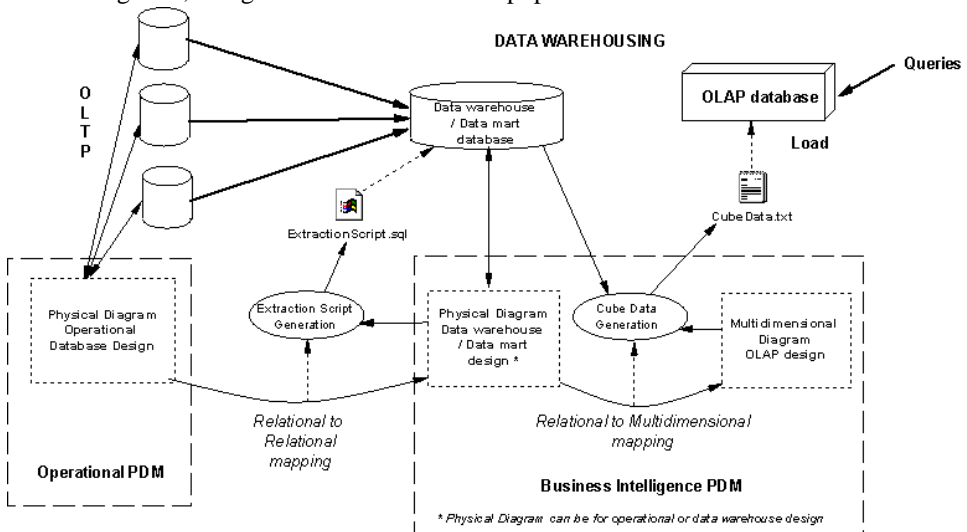
- Start with a CDM containing entities, attributes, relationships, domains, data items and business rules. If need be, you may develop the CDM in several design steps starting from a high level model to a low level CDM
- Generate an LDM. Create indexes and specify FK column names and other common features
- 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 6. For detailed information about logical diagrams, see *Logical Diagram Basics* on page 41.

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.



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

- Operational PDM - You use PDM to design the structure of an operational database. Usually, in data modeling, the physical analysis follows the conceptual and/or logical

analysis, and addresses the details of the actual physical implementation of data in a database, to suit your performance and physical constraints.

- Business intelligence PDM - You can use a PDM to design the structure of a data environment, which consists of:
 - *Data warehouse* or *data mart* database – are populated with data transferred from operational databases, and gather together all the information that may be needed in an OLAP database, where queries for business analysis and decision making are performed. The data warehouse database gathers all the data manipulated in a company for example, whereas the data mart focuses on smaller entities in the company.

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

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

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

To create Physical Diagrams, see *Chapter 3, Building Physical Diagrams* on page 79. To create Multidimensional Diagrams, see *Chapter 4, Building Multidimensional Diagrams* on page 201.

DBMS-Specific Features

PowerDesigner provides customized support for a wide range of database families through DBMS-specific resource files.

These files have a .xdb extension, and are located in the "Resource Files\DBMS" directory inside your PowerDesigner installation directory.

Resource files support, among others, three kinds of customization for PowerDesigner model objects:

- Assignment of standard objects to the specific syntax of a DBMS
- Definition of Extended Attributes for certain objects to allow for DBMS-specific object properties
- Definition of additional DBMS-specific model objects.

You can view and edit the resource file for your selected DBMS in the Resource Editor by selecting **Database > Edit Current DBMS**.

For more information about working with resource files and the Resource File Editor, see the following chapters in the *Customizing and Extending PowerDesigner* manual:

- Resource Files and the Public Metamodel
- DBMS Resource File Reference
- Extending your models with Profiles

Creating a Data Model

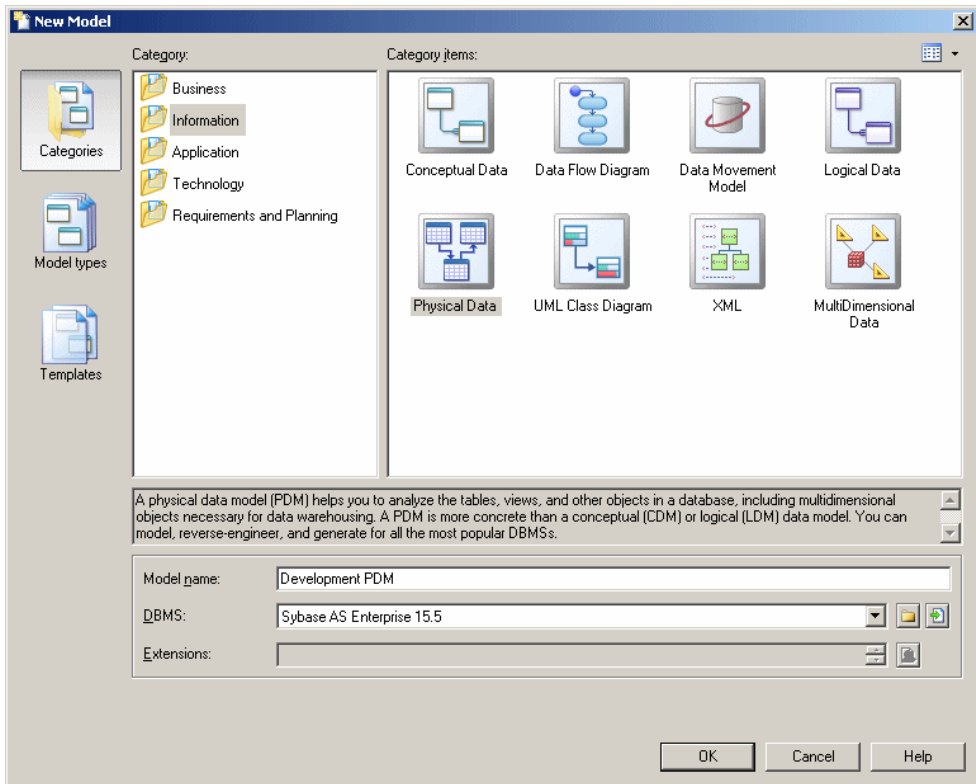
You create a new data model by selecting **File > New Model**.

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 a legacy PowerDesigner Process Analyst Model (.PAM) 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 373) 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	Specifies the name of the model, which should be clear and meaningful, and should convey the item's purpose to non-technical users.
Code	Specifies the technical name of the model, which is used for generating code or scripts.
Comment	Provides a description of the model.
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.
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 <code>EnableManyDatabases</code> entry in the Database category of the DBMS), this field is not present, and is replaced by a list of databases in the Model menu. A Database category is also displayed in the physical options of your database objects.
Default diagram	Specifies the diagram displayed by default when you open the model.

Database Properties (PDM)

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

A database has the following properties:

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

Using a Database in a Physical Option

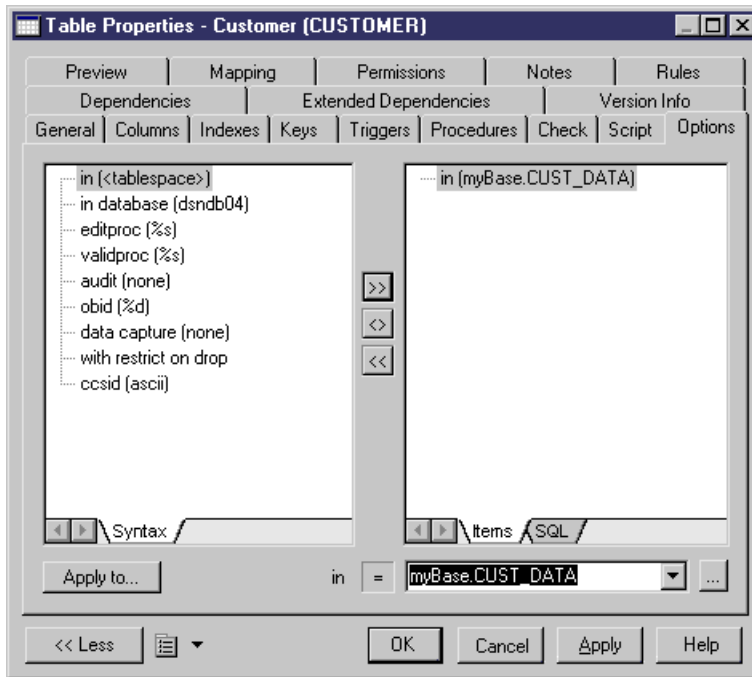
You can use a database in a physical option.

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

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

```
DBname . TBSPName
```

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



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

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

Archiving a PDM

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

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

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

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

or

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

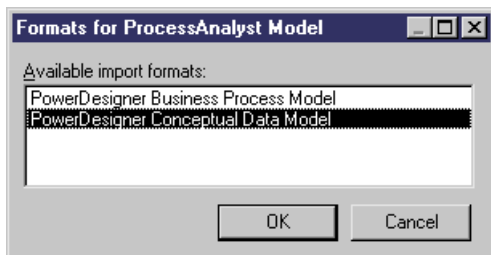
Opening a V6 PAM into a CDM

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

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

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

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



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

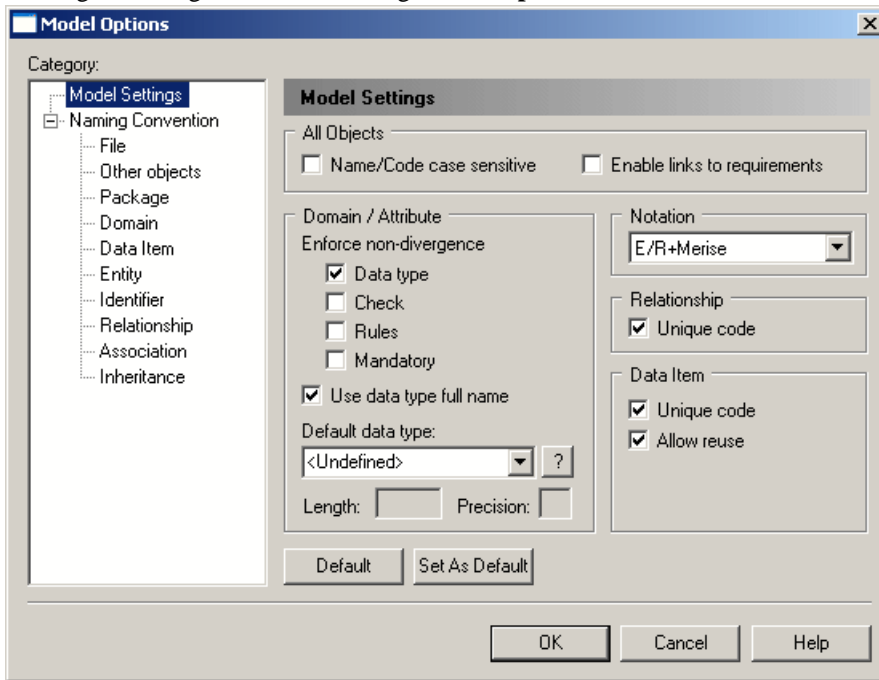
The recovered objects are imported into the CDM and appear in a default diagram.

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 the <i>Requirements Modeling</i> guide).

Option	Description
Enforce non-divergence	<p>Enforces non-divergence between a domain definition and the attributes using the domain. You can select any or all of the following attribute properties:</p> <ul style="list-style-type: none"> • Data type - Data type, length, and precision • Check - Check parameters, such as minimum and maximum • Rules - Business rules • Mandatory - Attribute mandatory property <p>When you apply these options, you are asked if you want to apply domain properties to attributes currently attached to the domain. If you click <i>OK</i>, the attribute properties are modified for consistency with the domain.</p> <p>When you modify the properties of a domain, the properties of the attributes attached to it are updated provided they are selected here.</p> <p>When you select an attribute property under Enforce non-divergence, that property cannot be modified in the lists of attributes and the property sheets of attributes.</p> <p>If you want to modify an attribute property that is defined as non-divergent, you must detach the attribute from its domain, or clear the appropriate Enforce non-divergence model option.</p>
Use data type full name	Specifies that the complete data type is displayed in entity symbols.
Default data type	Specifies a default data type to apply to domains and attributes if none is selected for them.
Notation	<p>You can choose between the following notations:</p> <ul style="list-style-type: none"> • 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. <p>For more information about these notations, see <i>Supported CDM/LDM notations</i> on page 21</p>
Unique code	Requires that data items or relationships have unique codes

Option	Description
Allow n-n relationships	[LDM only] Allows n-n relationships to be displayed.
Allow reuse	<p>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.</p> <p>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.</p> <p>When you delete an entity or entity attributes, these options determine whether or not the corresponding data items are also deleted, as follows:</p> <ul style="list-style-type: none"> • 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 "Naming Conventions" in *Chapter 8, Customizing Your Modeling Environment* of the *Core Features Guide*.

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 27

For detailed information about working with GTL, see *Chapter 4, Customizing Generation with GTL in Customizing and Extending PowerDesigner*

Migration Settings (LDM)

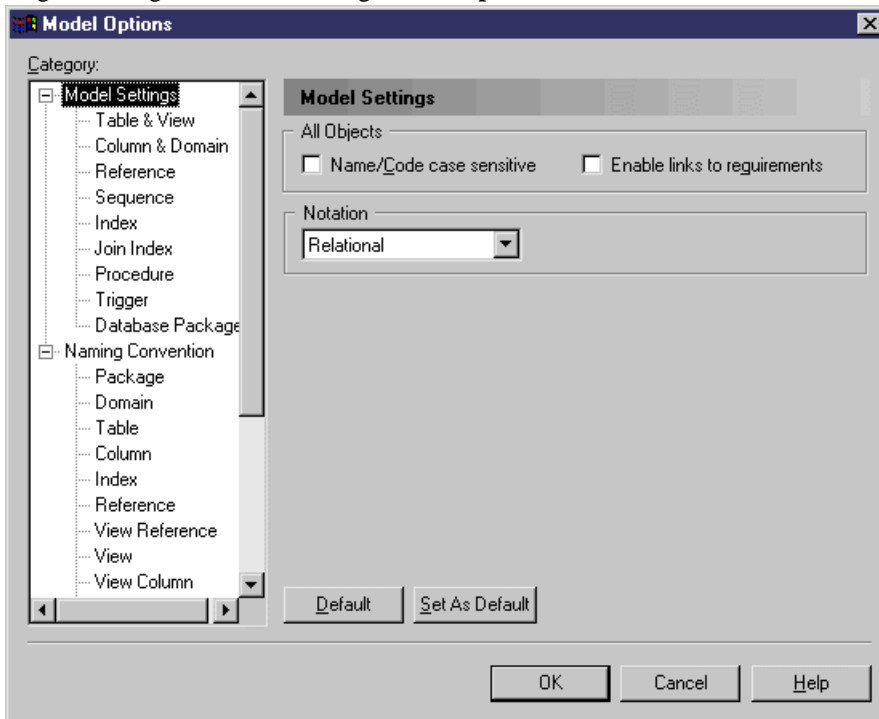
To set migration settings, select **Tools > Model Options**, and select the Migration settings sub-category under **Model Settings**.

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.
Foreign attribute name	<p>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:</p> <ul style="list-style-type: none"> • %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 <p>The following example checks the %PARENTROLE% value; if it is equal to the parent name (which is the replacement value) then the template "%PARENT%_%ATTRIBUTE%" will be used, otherwise template "%PARENTROLE%" will be used because the user has entered a parent role for the relationship:</p> <p>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.</p>
Use template	<p>Controls when the primary identifier attribute name template will be used. You can choose between:</p> <ul style="list-style-type: none"> • Always use template. • Only use template in case of conflict.

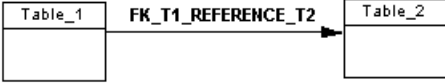
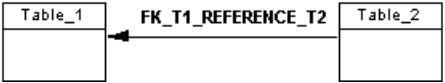
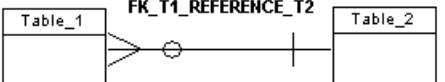
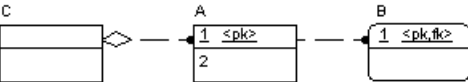
Setting PDM Model Options

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 (Tools > Check Model) to verify that your model does not contain any duplicate objects.
Enable links to requirements	Displays a Requirements tab in the property sheet of every object in the model, which allows you to attach requirements to objects in your model. These attached requirements are kept synchronized with your requirements model. For more information about requirements, see the <i>Requirements Modeling</i> guide.

Option	Function
<p>Notation</p>	<p>Specifies the use of one of the following notation types for the model. You can choose between:</p> <ul style="list-style-type: none"> Relational - Arrow pointing to primary key. This option is the default, and is used in this manual.  <ul style="list-style-type: none"> CODASYL - Arrow pointing to foreign key.  <ul style="list-style-type: none"> Conceptual - Cardinality displayed in IE format (crow's feet).  <ul style="list-style-type: none"> IDEFIX - Cardinality and mandatory status displayed on reference, primary columns in separate containers and dependent tables with rounded rectangles.  <p>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.</p>

For information about controlling the naming conventions of your models, see "Naming Conventions" in *Chapter 8, Customizing Your Modeling Environment* of the *Core Features Guide*.

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, any or all of: Data type, Check, Rules, Mandatory, and Profile.
Default data type	Specifies a default data type to be applied to columns and domains.
Column: Mandatory by default	Specifies that columns are created, by default, as mandatory.
Domain: Mandatory by default	Specifies that domains are created, by default, as mandatory.

Reference Model Options

To set model options for references, select **Tools > Model Options**, and select the Reference sub-category in the left-hand Category pane.

You can set the following options on this tab:

Option	Function
Unique code	Requires that references have unique codes. If this option is not selected then different references can have the same code (except when two references share the same child table).
Auto-reuse columns	<p>Enables the reuse of columns in a child table as foreign key columns if the following conditions are satisfied:</p> <ul style="list-style-type: none"> • Child column has same code as migrating primary key column • Child column is not already a foreign key column • Data types are compatible <p>For more information, see <i>Automatic Reuse and Migration of Columns</i> on page 169.</p>
Auto-migrate columns	<p>Enables the automatic migration of primary key columns from the parent table as foreign key columns to the child table. If you select both Auto-migrate columns and any of the following sub-options, then the relevant column property of the PK will also be migrated to the FK at reference creation:</p> <ul style="list-style-type: none"> • Domain • Check • Rules • Last position <p>For more information, see <i>Automatic Reuse and Migration of Columns</i> on page 169.</p>

Option	Function
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	<p>Specifies how reference links are created. You can select either:</p> <ul style="list-style-type: none"> • Primary key – automatically create links from primary key to foreign key columns at creation • User-defined – manually create your own links
Default implementation	<p>Specifies how referential integrity is implemented in the reference. You can select either:</p> <ul style="list-style-type: none"> • Declarative – referential integrity is defined by constraint in foreign declarations • Trigger – referential integrity is implemented by triggers <p>For more information on referential integrity, see <i>Reference Property Sheet Integrity Tab</i> on page 175.</p>
Default Constraints: Update	<p>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:</p> <ul style="list-style-type: none"> • 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

Option	Function
Default Constraints: Delete	<p>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:</p> <ul style="list-style-type: none"> • 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	<p>Specifies a default owner for the specified object from the list of users (see <i>Creating a User</i> on page 131). To create a user, click on the ellipsis button to open the List of Users, and click the Add a Row tool.</p> <p>If the user specified is subsequently deleted, this option (and the ownership of all associated objects) will be reset to none.</p>
Ignore identifying owner	<p>[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.</p> <p>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.</p>

Option	Function
Rebuild automatically triggers	<p>[triggers] Automatically rebuilds the triggers on the child and parent tables of a reference when you:</p> <ul style="list-style-type: none"> • 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 <p>If this option is not selected, you can manually instruct PowerDesigner to rebuild triggers at any time by selecting Tools > Rebuild Objects > Rebuild Triggers.</p>

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 "Display Preferences" in *Chapter 8, Customizing Your Modeling Environment* of the *Core Features Guide*.

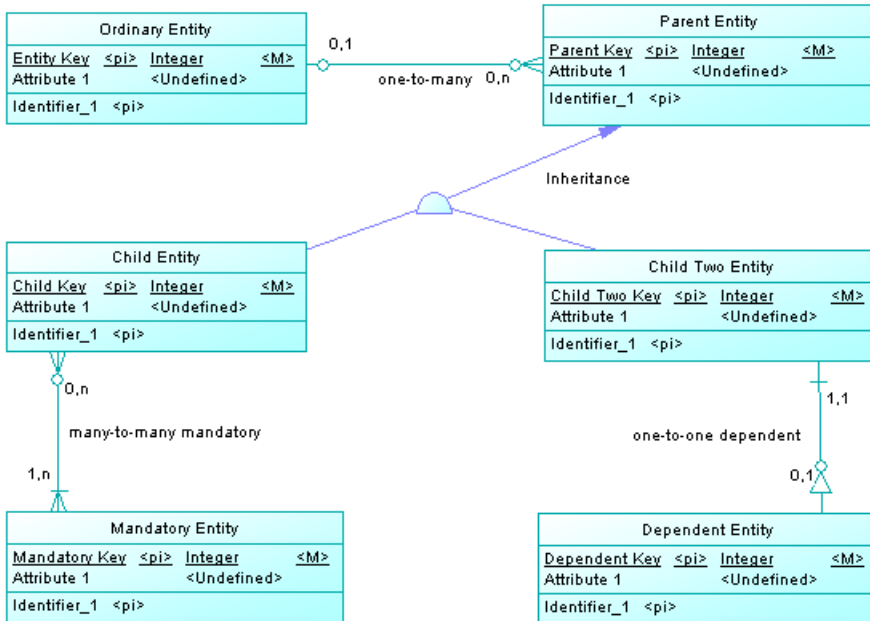
Supported CDM/LDM Notations

PowerDesigner supports the most popular data modeling notations in the CDM and LDM via the notation model option

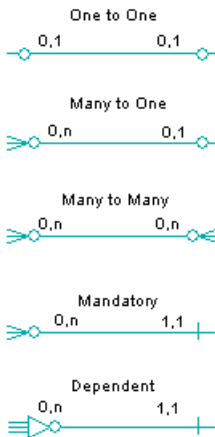
For more information, see *Setting CDM/LDM Model Options* on page 12.

Entity/relationship Notation

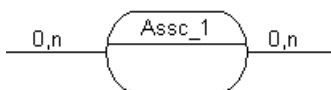
In the Entity/relationship notation, entities are represented as rectangles and divided in three compartments: name, attributes, and identifiers.



The termination points of relationships indicate the cardinality as follows:



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

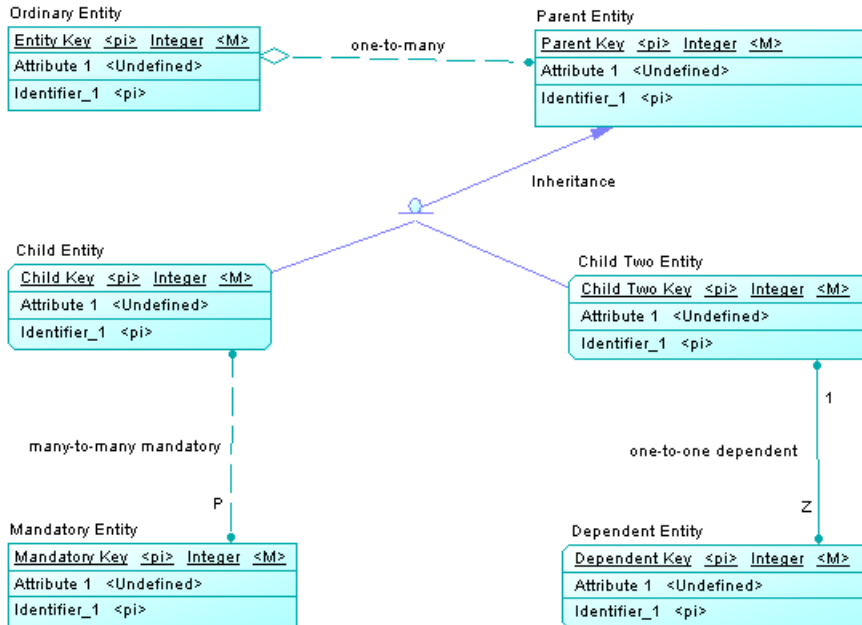


Inheritance symbols indicate if they are complete and if they have mutually exclusive children:

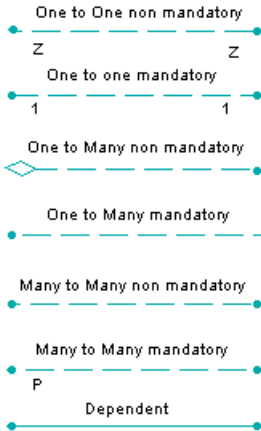
Complete	Mutually exclusive	Symbol
No	No	
Yes	No	
No	Yes	
Yes	Yes	

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:

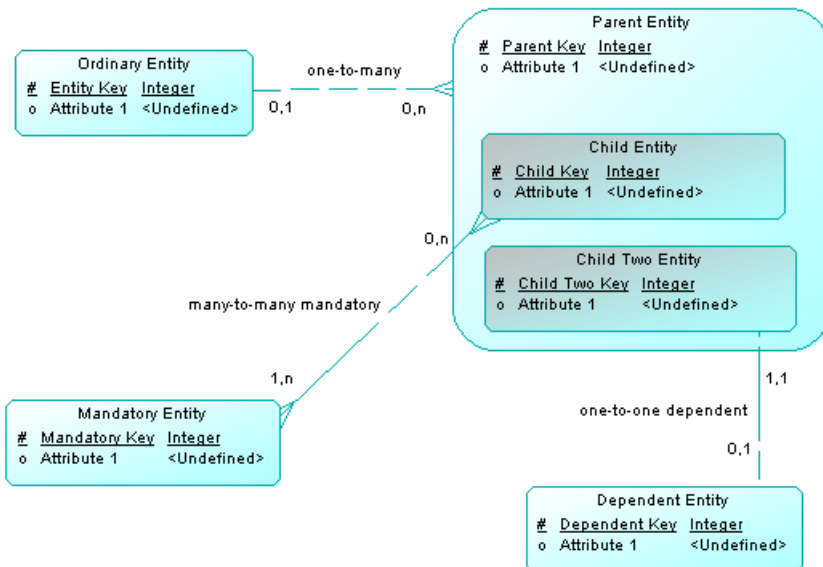


Inheritance symbols indicate if the inheritance is complete:

Complete	Symbol
Yes	
No	

Barker Notation

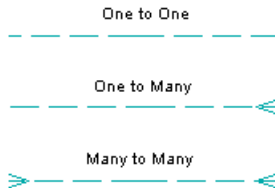
In the Barker notation, entities are drawn with round corners, and inheritances are displayed by placing children inside the parent entity.



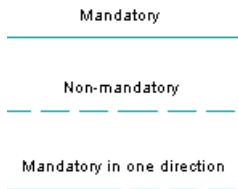
Only attributes are listed and a symbol specifies whether each attribute is a key, a mandatory or an optional attribute as follows:

- # Primary
- * Mandatory
- o Optional

Relationship symbols indicate the cardinality as follows:



The line style specifies if a relationship is mandatory:



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! We strongly recommend that you make a back up of the resource files delivered with PowerDesigner before editing them.

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

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

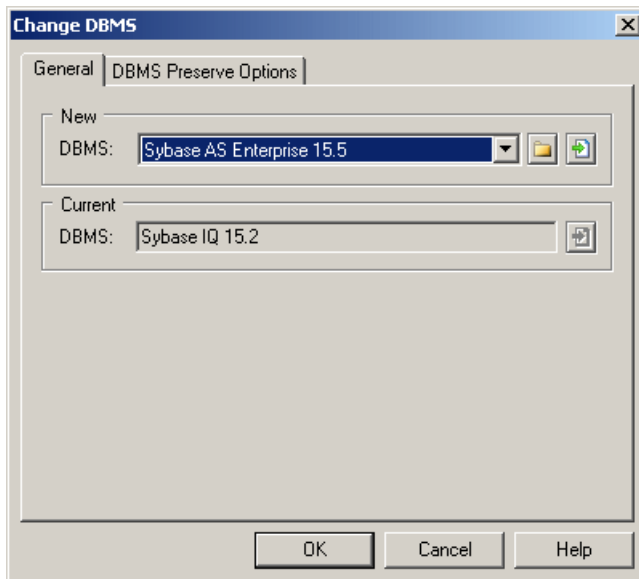
You can change the **DBMS** being modeled in your PDM at any time.

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** :



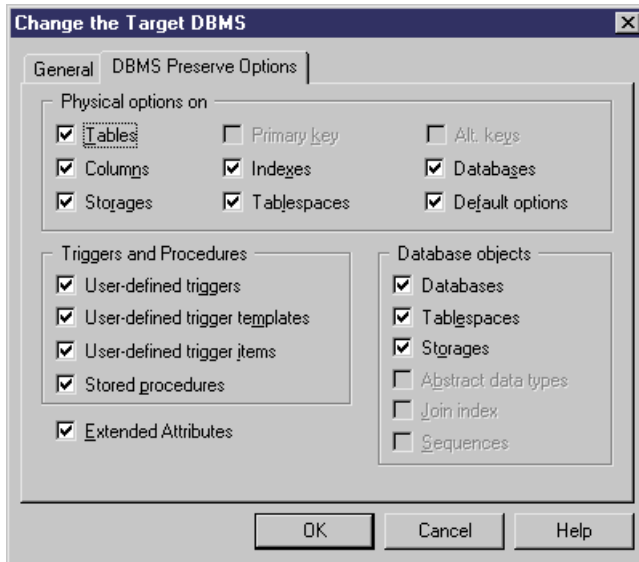
2. Select a **DBMS** from the list.

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.

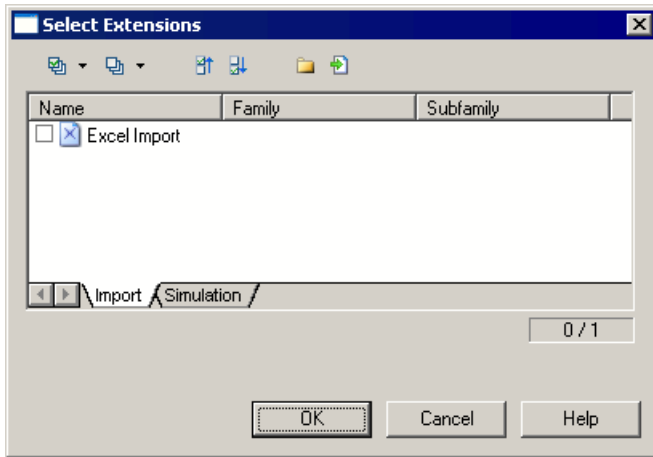
5. Click **OK** to return to the model.

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 **Import 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 "Extending Objects" in *Chapter 4, Objects of the Core Features Guide*. For detailed information about working with extensions, see *Chapter 1, Resource Files and the Public Metamodel* in *Customizing and Extending PowerDesigner*.

Creating Links with Extended Dependencies

Extended dependencies can be created between any model objects, including between objects in different models. They are used for documentation purposes only, and are not interpreted or checked by PowerDesigner.

For more information about extended dependencies, see "Creating Extended Dependencies" in *Chapter 9, Getting Started with Linking and Synching* of the *Core Features Guide*.

CHAPTER 2 Building Conceptual and Logical Diagrams

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

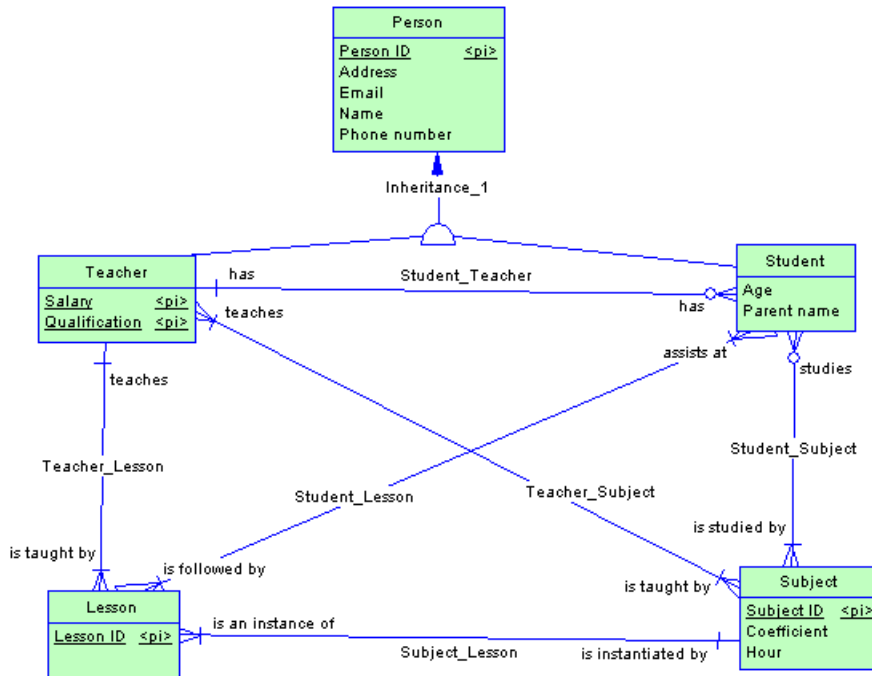
PowerDesigner provides you with a highly flexible environment in which to model your data systems. You can begin with either a CDM (see *Conceptual Diagram Basics* on page 29) or an LDM (see *Logical Diagram Basics* on page 41) to analyze your system and then generate a PDM (see the *Chapter 3, Building Physical Diagrams* on page 79) 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 10, Generating Other Models from a Data Model* on page 387.

Conceptual Diagram Basics

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.

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 <i>Domains (CDM/LDM/PDM)</i> on page 148.
Data Item	[none]	[none]	Elementary piece of information. See <i>Data Items (CDM)</i> on page 45.

Object	Tool	Symbol	Description
Entity			Person, place, thing, or concept that is of interest to the enterprise. See <i>Entities (CDM/LDM)</i> on page 47.
Entity Attribute	[none]	[none]	Elementary piece of information attached to an entity. See <i>Attributes (CDM/LDM)</i> on page 52.
Identifier	[none]	[none]	One or many entity attributes, whose values uniquely identify each occurrence of the entity. See <i>Identifiers (CDM/LDM)</i> on page 54.
Relationship			Named connection or relation between entities (ER modeling methodology). See <i>Relationships (CDM/LDM)</i> on page 56.
Inheritance			Relationship that defines an entity as a special case of a more general entity. See <i>Inheritances (CDM/LDM)</i> on page 72.
Association			Named connection or association between entities (Merise modeling methodology). See <i>Associations and Association Links (CDM)</i> on page 67.
Association Link			Link that connects an association to an entity. See <i>Associations and Association Links (CDM)</i> on page 67.

Creating a Conceptual Diagram

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

- Right-click the model in the Browser and select **New > Conceptual Diagram**.
- Right-click the background of any diagram and select **Diagram > New Diagram**.

To create a new CDM with a conceptual diagram, select **File > New Model**, choose to create a conceptual data model from the Model type list, choose **Conceptual Diagram**, and click **OK**.

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.

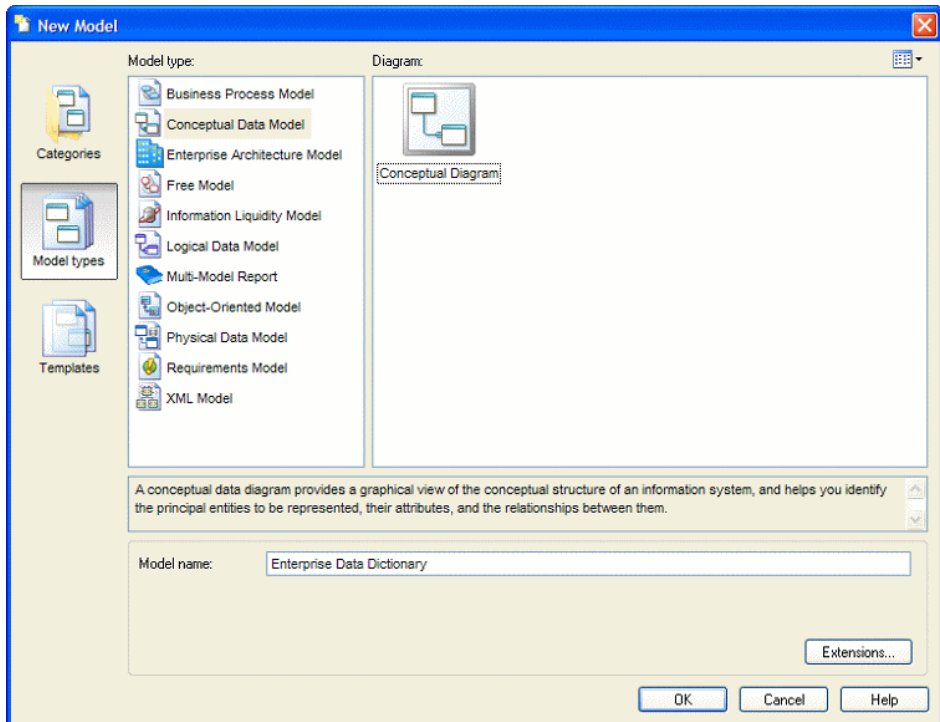
Data dictionaries ensure consistency of use by providing a single authoritative definition for all common data elements used across the enterprise. They are used to standardize data content, context, and definitions and to achieve consistency and reusability while increasing the quality of the data used 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

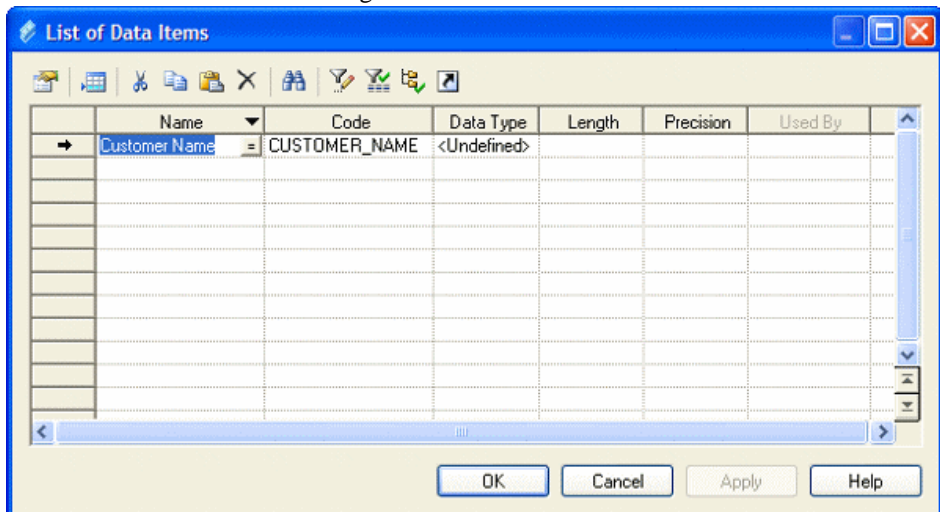
A data dictionary defined in a PowerDesigner CDM provides:

- 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 45.

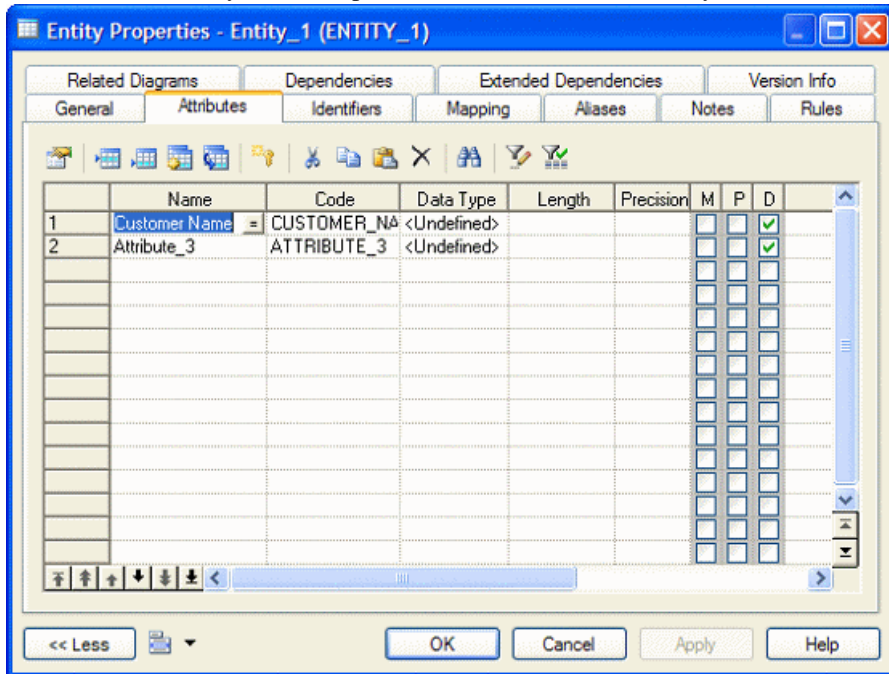
3. Select **Model > Entities** to open the List of Entities and enter some of the entities that you want to define. Entities represent more complex business structures composed of one or more attributes (which are associated with data items).

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 47.

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:

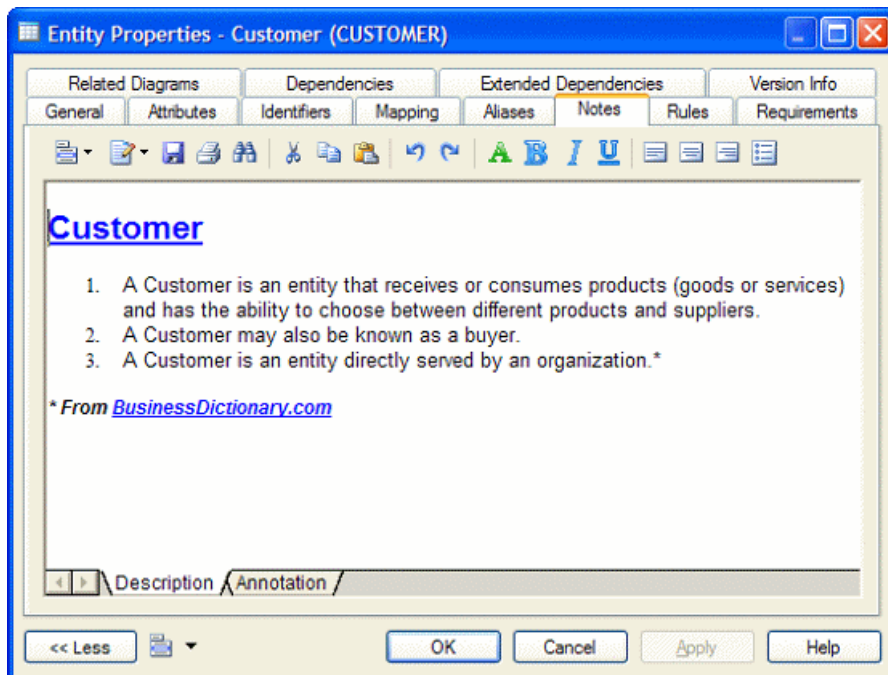


Create a new attribute by reusing an existing data item by clicking the **Reuse Data Item** tool and selecting the data item that you want to reuse. By default, PowerDesigner allows

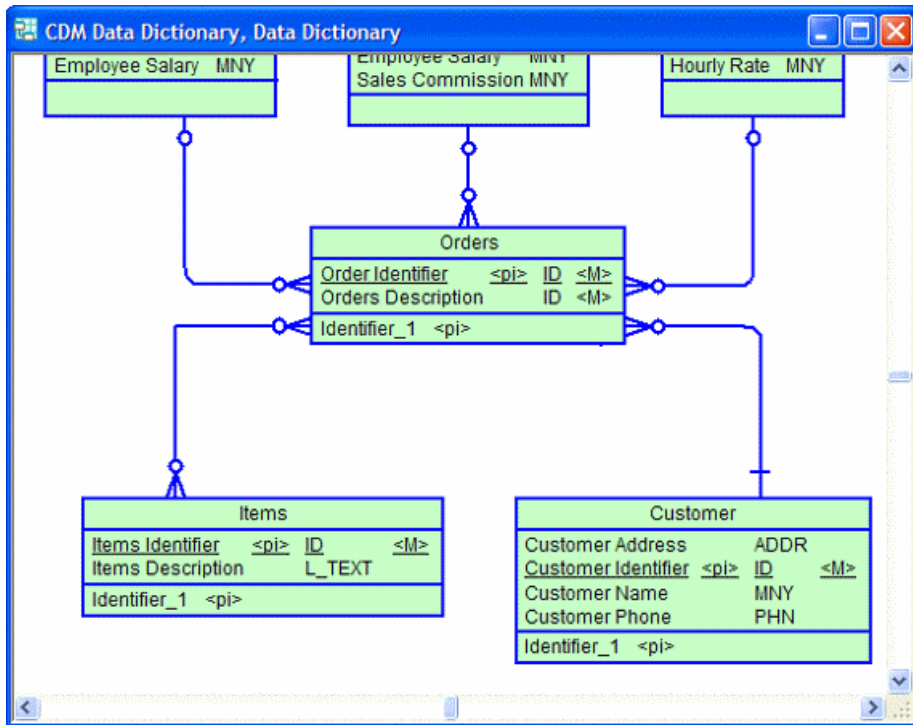
you to reuse a data item for more than one entity attribute so that, for example, you can define a `Zip Code` data item once, and reuse it in whatever entities contain addresses. If you then update the data item, your changes will simultaneously cascade down to all the entity attributes that use it. 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:

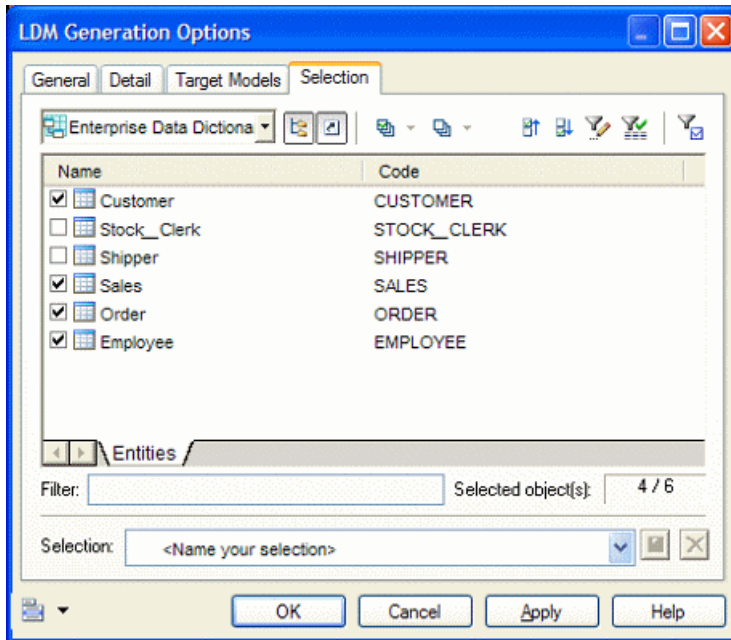


6. [optional] Select the **Relationship** tool in the pallet and create relationships between the entities in your data dictionary. Click and hold in one entity, then drag the cursor to a second entity and release the mouse button. Draw other relationships as necessary and then right-click anywhere in 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.

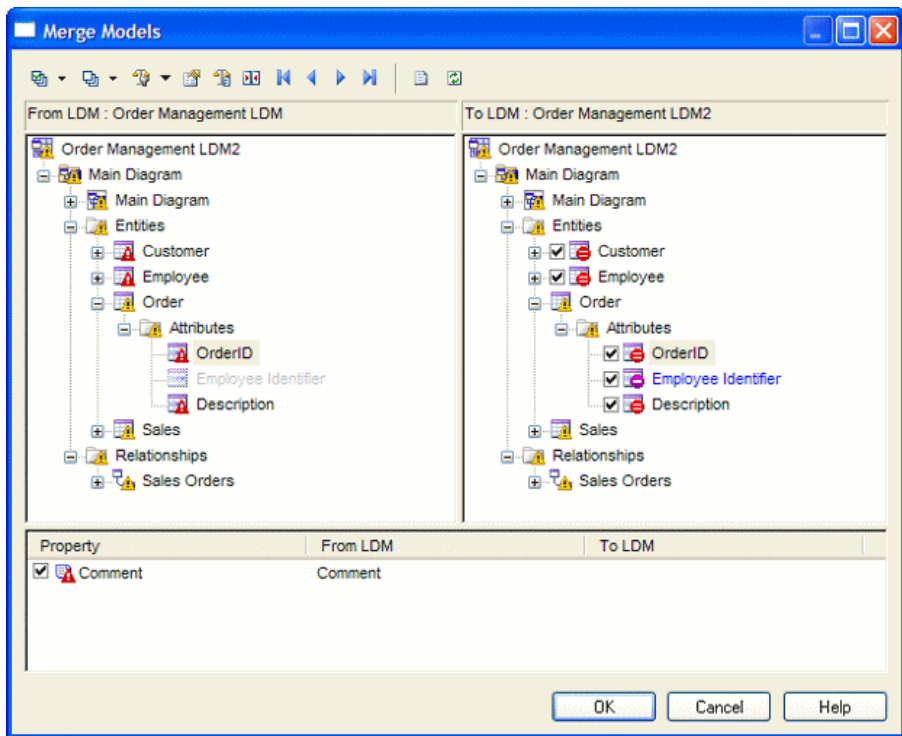


For detailed information about defining relationships, see *Relationships (CDM/LDM)* on page 56.

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**.



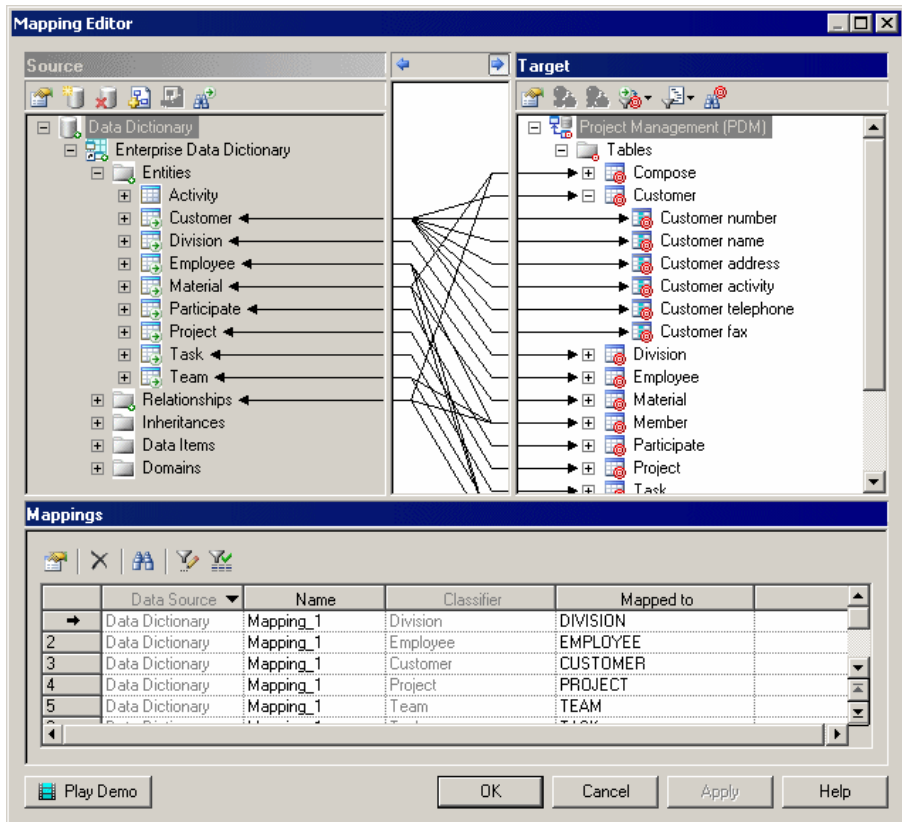
You can review the links created between the data dictionary and your other models in the Generation Links Viewer (select **Tools > Generation Links > Derived Models**). You can regenerate whenever necessary to propagate updates or additions in the data dictionary to your other models. The Merge Models dialog (see *Chapter 7, Comparing and Merging Models* in the *Core Features Guide*) 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 10, Generating Other Models from a Data Model* on page 387.

- 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:

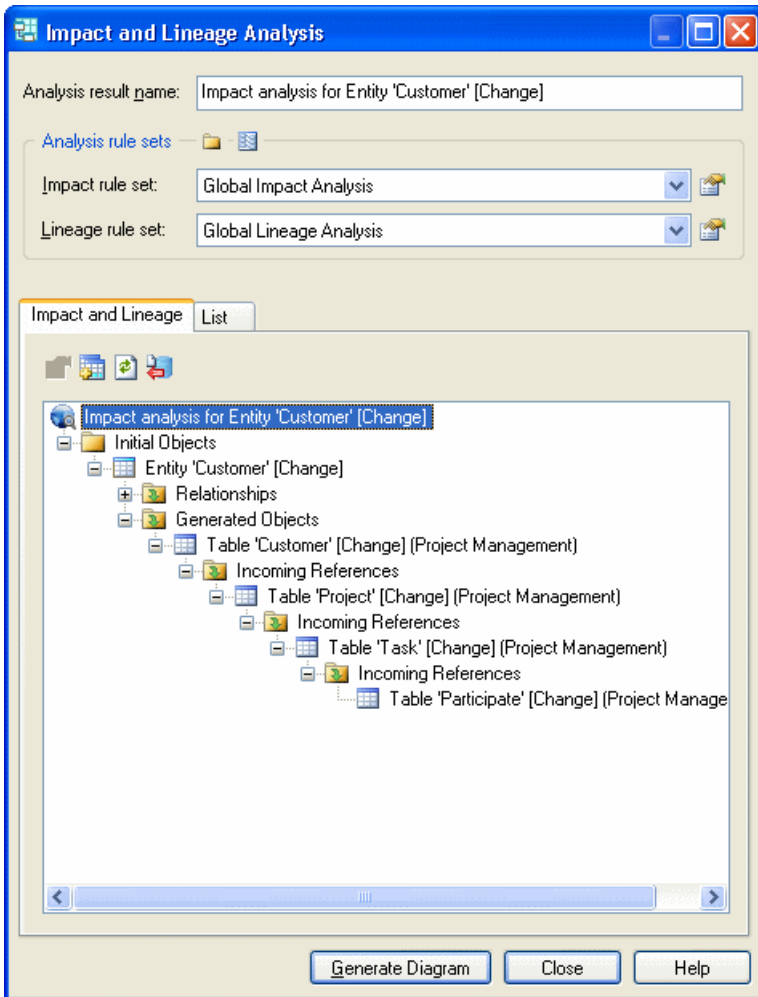


You can create additional mappings as necessary by dragging and dropping entities and attributes from the data dictionary onto objects in the target model. Note that mappings created in this way will not automatically propagate changes.

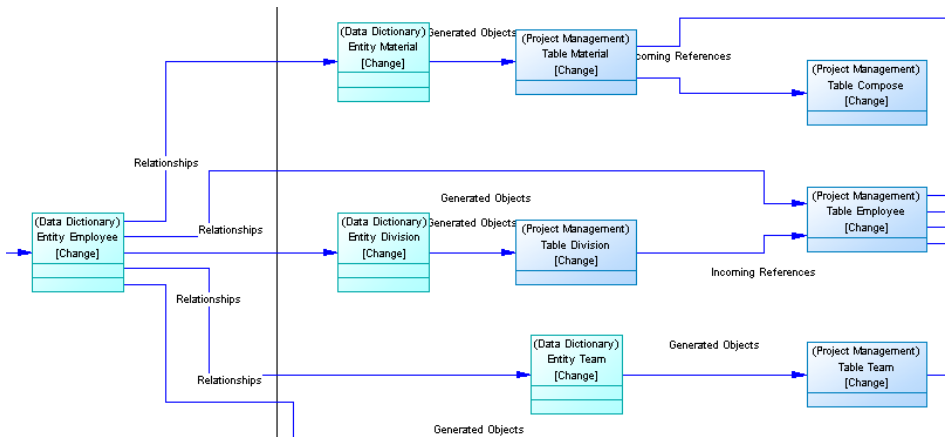
For detailed information about using the Mapping Editor, see *Chapter 12, Object Mappings* in the *Core Features Guide*.

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.

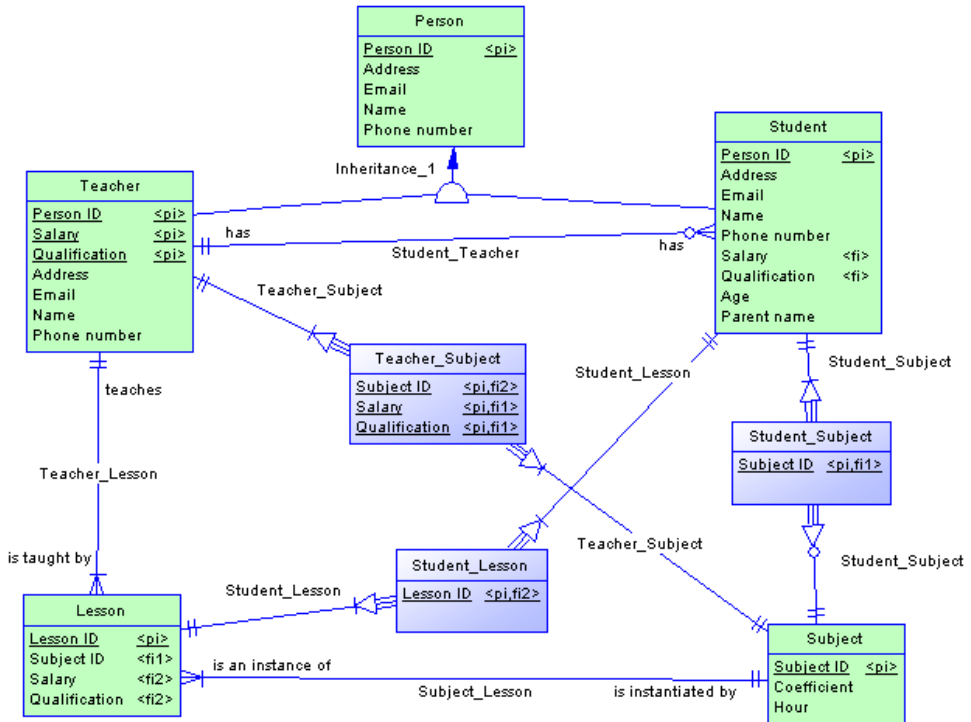
For detailed information about working with impact analysis, see *Chapter 13, Impact and Lineage Analysis* in the *Core Features Guide*.

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 (see the *Working with the Repository* book).
10. Share your data dictionary with other members of your organization through the PowerDesigner Portal (see *Chapter 3, Browsing the Repository via the Web* in *Working with the Repository*) or by publishing it to HTML or RTF (see *Chapter 6, Reports* in the *Core Features Guide*).

Logical Diagram Basics

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.

The following logical diagram represent the same system as that in our CDM example (see *Conceptual Diagram Basics* on page 29).









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 <i>Domains (CDM/LDM/PDM)</i> on page 148.
Entity			Person, place, thing, or concept that is of interest to the enterprise. See <i>Entities (CDM/LDM)</i> on page 47.
Entity Attribute	[none]	[none]	Elementary piece of information attached to an entity. See <i>Attributes (CDM/LDM)</i> on page 52.

Object	Tool	Symbol	Description
Identifier	[none]	[none]	One or many entity attributes, whose values uniquely identify each occurrence of the entity. See <i>Identifiers (CDM/LDM)</i> on page 54.
Relationship			Named connection or relation between entities (ER modeling methodology). See <i>Relationships (CDM/LDM)</i> on page 56.
n-n Relationship			[LDM only] Named cardinality represented with an intermediary entity. See <i>Relationships (CDM/LDM)</i> on page 56.
Inheritance			Relationship that defines an entity as a special case of a more general entity. See <i>Inheritances (CDM/LDM)</i> on page 72.

Creating a Logical Diagram

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

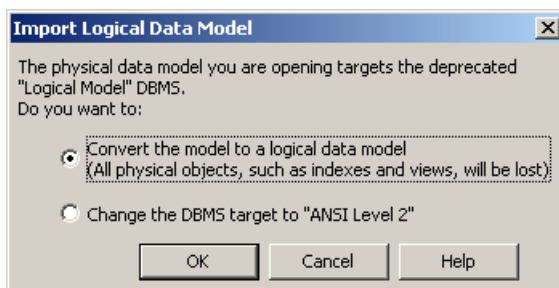
- Right-click the model in the Browser and select **New > Logical Diagram**.
- Right-click the background of any diagram and select **Diagram > New Diagram**.

To create a new LDM with a logical diagram, select **File > New Model**, choose to create a logical data model from the Model type list, choose **Logical Diagram**, and click **OK**.

Importing a Deprecated PDM Logical Model

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

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

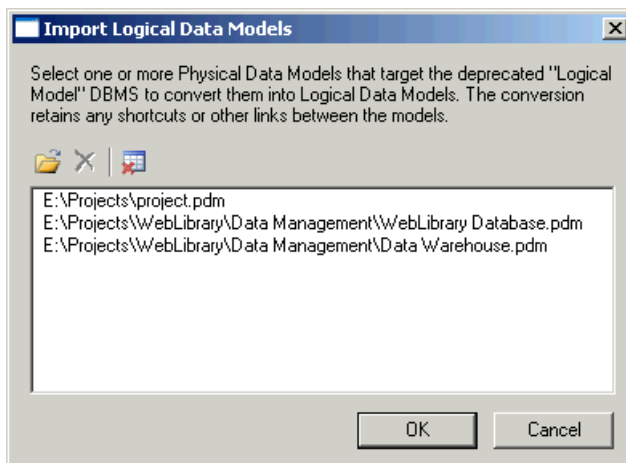
4. Click OK to open the model.

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 *Chapter 10, Generating Models and Model Objects in the Core Features Guide*).

Importing Multiple Interconnected PDM Logical Models

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

1. Select **File > Import > Legacy Logical Data Model** 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 47) in order to create an entity attribute (see *Attributes (CDM/LDM)* on page 52), which is associated with the data item.

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

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

Example

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

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

Creating a Data Item

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

- Select **Model > Data Items** to access the List of Data Items, and click the Add a Row tool.
- Create an entity attribute (see *Attributes (CDM/LDM)* on page 52). A data item will be automatically created.
- Right-click the model or package in the Browser, and select **New > Data Item**.

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

Data Item Properties

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

The General tab contains the following properties:

Property	Description
Name	Specifies the name of the item, which should be clear and meaningful, and should convey the item's purpose to non-technical users.
Code	Specifies the technical name of the object, which is used for generating code or scripts, which may be abbreviated, and should not generally include spaces.
Comment	Specifies a descriptive label for the data item.
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined.
Data type	Specifies the code indicating the data format, such as N for numeric or A for alphanumeric, followed by the number of characters.
Length	Specifies the maximum number of characters.
Precision	Specifies the number of places after the decimal point, for data values that can take a decimal point.
Domain	Specifies the name of the associated domain (See <i>Domains (CDM/LDM/PDM)</i> on page 148). If you attach a data item to a domain, the domain supplies a data type to the data item, and can also apply length, decimal precision, and check parameters.

The following tabs are also commonly used:

- Standard Checks - contains checks which control the values permitted for the data item (see *Check Parameters (CDM/LDM/PDM)* on page 105).
- Additional Checks - allows you to specify additional constraints (not defined by standard check parameters) for the data item.
- Rules - lists the business rules associated with the data item (see *Business Rules (CDM/LDM/PDM)* on page 193).

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	<p>Each data item must have a unique code.</p> <p>If you try to select this option and some existing data items are already sharing a code, the following error will be displayed:</p> <p>Unique Code option could not be selected because two data items have the same code: <i>data_item_code</i></p> <p>To be able to select the option, you must first assign unique codes to all data items.</p>	<p>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.</p> <hr/> <p>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.</p> <hr/>
Allow re-use	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 12.

Entities (CDM/LDM)

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

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

Creating an Entity

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

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

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

Entity Properties

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

The General tab contains the following properties:

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

The following tabs are also available:

- Attributes - lists the attributes associated with the entity (see *Attributes (CDM/LDM)* on page 52).
- Identifiers - lists the attributes associated with the entity (see *Identifiers (CDM/LDM)* on page 54).
- Rules - lists the business rules associated with the entity (see *Business Rules (CDM/LDM/PDM)* on page 193).
- Subtypes – [Barker only] lists the subtypes that inherit from the entity.

Copying an Entity

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

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

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

Copying an Entity Within a Model

You copy an entity within a model as follows:

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

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

Copying an Entity to a Different Model

You copy an entity to a different model as follows:

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

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

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:

Preference	Display description																																				
Attributes	<p>Specifies whether Attributes are displayed on entity symbols. If selected, you can choose between displaying:</p> <ul style="list-style-type: none"> All attributes - All attributes: <table border="1" data-bbox="455 340 744 505"> <tr><td colspan="2">Customer</td></tr> <tr><td>Customer number</td><td>ID</td></tr> <tr><td>Customer name</td><td>NAME</td></tr> <tr><td>Customer address</td><td>SHORT_TEXT</td></tr> <tr><td>Customer activity</td><td>SHORT_TEXT</td></tr> <tr><td>Customer telephone</td><td>PHONE</td></tr> <tr><td>Customer fax</td><td>PHONE</td></tr> </table> Primary attributes - Only primary identifier attributes: <table border="1" data-bbox="455 562 630 644"> <tr><td colspan="2">Customer</td></tr> <tr><td>Customer number</td><td>ID</td></tr> </table> Identifying attributes - All identifier attributes: <table border="1" data-bbox="460 706 650 788"> <tr><td colspan="2">Customer</td></tr> <tr><td>Customer number</td><td><pi></td></tr> </table> Display limit - Number of attributes shown depends on defined value. For example, if set to 5: <table border="1" data-bbox="455 878 744 1043"> <tr><td colspan="2">Customer</td></tr> <tr><td>Customer number</td><td>ID</td></tr> <tr><td>Customer name</td><td>NAME</td></tr> <tr><td>Customer address</td><td>SHORT_TEXT</td></tr> <tr><td>Customer activity</td><td>SHORT_TEXT</td></tr> <tr><td>Customer telephone</td><td>PHONE</td></tr> <tr><td>...</td><td>...</td></tr> </table> 	Customer		Customer number	ID	Customer name	NAME	Customer address	SHORT_TEXT	Customer activity	SHORT_TEXT	Customer telephone	PHONE	Customer fax	PHONE	Customer		Customer number	ID	Customer		Customer number	<pi>	Customer		Customer number	ID	Customer name	NAME	Customer address	SHORT_TEXT	Customer activity	SHORT_TEXT	Customer telephone	PHONE
Customer																																					
Customer number	ID																																				
Customer name	NAME																																				
Customer address	SHORT_TEXT																																				
Customer activity	SHORT_TEXT																																				
Customer telephone	PHONE																																				
Customer fax	PHONE																																				
Customer																																					
Customer number	ID																																				
Customer																																					
Customer number	<pi>																																				
Customer																																					
Customer number	ID																																				
Customer name	NAME																																				
Customer address	SHORT_TEXT																																				
Customer activity	SHORT_TEXT																																				
Customer telephone	PHONE																																				
...	...																																				
Identifiers	<p>All identifier attributes for the entity are listed at the bottom of the entity symbol:</p> <table border="1" data-bbox="417 1121 529 1199"> <tr><td colspan="2">Customer</td></tr> <tr><td colspan="2">James_Joyce</td></tr> </table>	Customer		James_Joyce																																	
Customer																																					
James_Joyce																																					
Stereotype	Stereotype of the entity.																																				
Comment	<p>Comment of the entity. When selected, all other check boxes are deselected, except for Stereotype:</p> <table border="1" data-bbox="413 1347 615 1433"> <tr><td colspan="2">Customer</td></tr> <tr><td colspan="2">This entity can be shared</td></tr> </table>	Customer		This entity can be shared																																	
Customer																																					
This entity can be shared																																					

Entity Attributes

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

Preference	Display description																								
Data type	<p>Data type for each entity attribute:</p> <table border="1" data-bbox="419 253 626 418"> <thead> <tr> <th colspan="2">Customer</th> </tr> </thead> <tbody> <tr> <td>Customer number</td> <td>N5</td> </tr> <tr> <td>Customer name</td> <td>A30</td> </tr> <tr> <td>Customer address</td> <td>A80</td> </tr> <tr> <td>Customer activity</td> <td>A80</td> </tr> <tr> <td>Customer telephone</td> <td>A12</td> </tr> <tr> <td>...</td> <td>...</td> </tr> </tbody> </table>	Customer		Customer number	N5	Customer name	A30	Customer address	A80	Customer activity	A80	Customer telephone	A12										
Customer																									
Customer number	N5																								
Customer name	A30																								
Customer address	A80																								
Customer activity	A80																								
Customer telephone	A12																								
...	...																								
Domain or data type	<p>Domain for each entity attribute. You can only display domains when the Data type check box is selected.</p>																								
Domain	<p>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:</p> <ul style="list-style-type: none"> • Data types - Displays only the data type, if any: <table border="1" data-bbox="474 673 705 746"> <thead> <tr> <th colspan="2">CUSTOMER</th> </tr> </thead> <tbody> <tr> <td>Customer Number</td> <td><UNDEF></td> </tr> <tr> <td>Customer Name</td> <td>A30</td> </tr> </tbody> </table> • Domains - Displays only the domain, if any: <table border="1" data-bbox="478 833 704 906"> <thead> <tr> <th colspan="2">CUSTOMER</th> </tr> </thead> <tbody> <tr> <td>Customer Number</td> <td>Identifier</td> </tr> <tr> <td>Customer Name</td> <td><None></td> </tr> </tbody> </table> • Data types and Domain - Displays both data type and domain, if any: <table border="1" data-bbox="478 992 790 1065"> <thead> <tr> <th colspan="2">CUSTOMER</th> </tr> </thead> <tbody> <tr> <td>Customer Number</td> <td><UNDEF> Identifier</td> </tr> <tr> <td>Customer Name</td> <td>A30 <None></td> </tr> </tbody> </table> • Data types and Replace by domains - Displays either data type or domain, if any, and domain if both are present: <table border="1" data-bbox="474 1187 733 1260"> <thead> <tr> <th colspan="2">CUSTOMER</th> </tr> </thead> <tbody> <tr> <td>Customer Number</td> <td>IDENTIFIER</td> </tr> <tr> <td>Customer Name</td> <td>A30</td> </tr> </tbody> </table> 	CUSTOMER		Customer Number	<UNDEF>	Customer Name	A30	CUSTOMER		Customer Number	Identifier	Customer Name	<None>	CUSTOMER		Customer Number	<UNDEF> Identifier	Customer Name	A30 <None>	CUSTOMER		Customer Number	IDENTIFIER	Customer Name	A30
CUSTOMER																									
Customer Number	<UNDEF>																								
Customer Name	A30																								
CUSTOMER																									
Customer Number	Identifier																								
Customer Name	<None>																								
CUSTOMER																									
Customer Number	<UNDEF> Identifier																								
Customer Name	A30 <None>																								
CUSTOMER																									
Customer Number	IDENTIFIER																								
Customer Name	A30																								
Mandatory	<p><M> indicators are displayed next to each mandatory attribute:</p> <table border="1" data-bbox="419 1348 626 1513"> <thead> <tr> <th colspan="2">Customer</th> </tr> </thead> <tbody> <tr> <td>Customer number</td> <td><M></td> </tr> <tr> <td>Customer name</td> <td><M></td> </tr> <tr> <td>Customer address</td> <td><M></td> </tr> <tr> <td>Customer activity</td> <td></td> </tr> <tr> <td>Customer telephone</td> <td></td> </tr> <tr> <td>...</td> <td>...</td> </tr> </tbody> </table>	Customer		Customer number	<M>	Customer name	<M>	Customer address	<M>	Customer activity		Customer telephone											
Customer																									
Customer number	<M>																								
Customer name	<M>																								
Customer address	<M>																								
Customer activity																									
Customer telephone																									
...	...																								

Preference	Display description														
Identifier indicators	<p><pi> indicators are displayed next to primary identifiers and <ai> indicators next to non-primary identifiers:</p> <table border="1"> <thead> <tr> <th colspan="2">Customer</th> </tr> </thead> <tbody> <tr> <td>Customer number</td> <td><pi></td> </tr> <tr> <td>Customer name</td> <td></td> </tr> <tr> <td>Customer address</td> <td></td> </tr> <tr> <td>Customer activity</td> <td></td> </tr> <tr> <td>Customer telephone</td> <td></td> </tr> <tr> <td>...</td> <td>...</td> </tr> </tbody> </table>	Customer		Customer number	<pi>	Customer name		Customer address		Customer activity		Customer telephone	
Customer															
Customer number	<pi>														
Customer name															
Customer address															
Customer activity															
Customer telephone															
...	...														
Stereotype	Displays the stereotype of the entity attributes														

Note: For information about selecting other properties to display, see "Display Preferences" in *Chapter 8, Customizing Your Modeling Environment of the Core Features Guide*.



Attributes (CDM/LDM)


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

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

Creating an Attribute

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

Tool	Description
	<p>Add a Row – Creates a new attribute and associated data item.</p> <p>If you have enabled the Allow Reuse model option (see <i>Setting CDM/LDM Model Options</i> on page 12), the new data item can be used as an attribute for other objects.</p> <p>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.</p>
	<p>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.</p> <p>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.</p>

Tool	Description
	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 the Objects chapter in the *Core Features Guide*.

Attribute Properties

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

The General tab contains the following properties:

Property	Description
Name	Specifies the name of the item, which should be clear and meaningful, and should convey the item's purpose to non-technical users.
Code	Specifies the technical name of the object, which is used for generating code or scripts, which may be abbreviated, and should not generally include spaces.
Comment	Specifies a descriptive label for the attribute.
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined.
Entity/ Association/ Inheritance	[read-only] Specifies the parent object. Click the tool to the right of the field to open its property sheet.
Data Item	[CDM only, read-only] Specifies the related data item. Click the tool to the right of the field to open its property sheet.
Inherited from	[LDM only, read-only] Specifies the parent entity from which the attribute is migrated through an inheritance.
Data type	Specifies the data type of the attribute, such as numeric, alphanumeric, boolean, or others. Click the question mark button to open the list of data types (see <i>Domains (CDM/LDM/PDM)</i> on page 148).
Length	Specifies the maximum length of the data type.
Precision	Specifies the maximum number of places after the decimal point.
Domain	Specifies the name of the associated domain (see <i>Domains (CDM/LDM/PDM)</i> on page 148). If you attach an attribute to a domain, the domain supplies a data type to the attribute, and can also apply length, decimal precision, and check parameters.

Property	Description
Primary Identifier	[entity attributes only] Indicates whether or not the attribute is the primary identifier of the entity.
Displayed	[entity and association attributes only] Displays the attribute in the object symbol.
Mandatory	Specifies that every object occurrence must assign a value to the attribute. Identifiers (see <i>Identifiers (CDM/LDM)</i> on page 54) are always mandatory.
Foreign identifier	[LDM only, read-only] Indicates whether or not the attribute is the foreign identifier of the entity.

The following tabs are also available:

- Standard Checks - contains checks which control the values permitted for the attribute (see *Setting Data Profiling Constraints* on page 105).
- Additional Checks - allows you to specify additional constraints (not defined by standard check parameters) for the attribute.
- Rules - lists the business rules associated with the attribute (see *Business Rules (CDM/LDM/PDM)* on page 193).

Deleting Attributes (CDM)

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

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

Identifiers (CDM/LDM)

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

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

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

Creating an Identifier

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

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

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

Identifier Properties

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

The General tab contains the following properties:

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

The following tabs are also available:

- Attributes - lists the attributes (see *Attributes (CDM/LDM)* on page 52) associated with the identifier: Click the Add Attributes tool to add an attribute.

Relationships (CDM/LDM)

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

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

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

Relationships and Associations

Relationships are used in the Entity Relationship (ER), Barker, and IDEF1X modeling methodologies. In the Merise methodology associations (see *Associations and Association Links (CDM)* on page 67) are used to link entities. PowerDesigner lets you use either relationships or associations exclusively, or combine the two methodologies in the same model.

This section analyzes relationships in the Entity Relationship methodology, for more information on IDEF1X, see *Setting CDM/LDM Model Options* on page 12.

Creating a Relationship

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

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

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

Relationship Properties

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

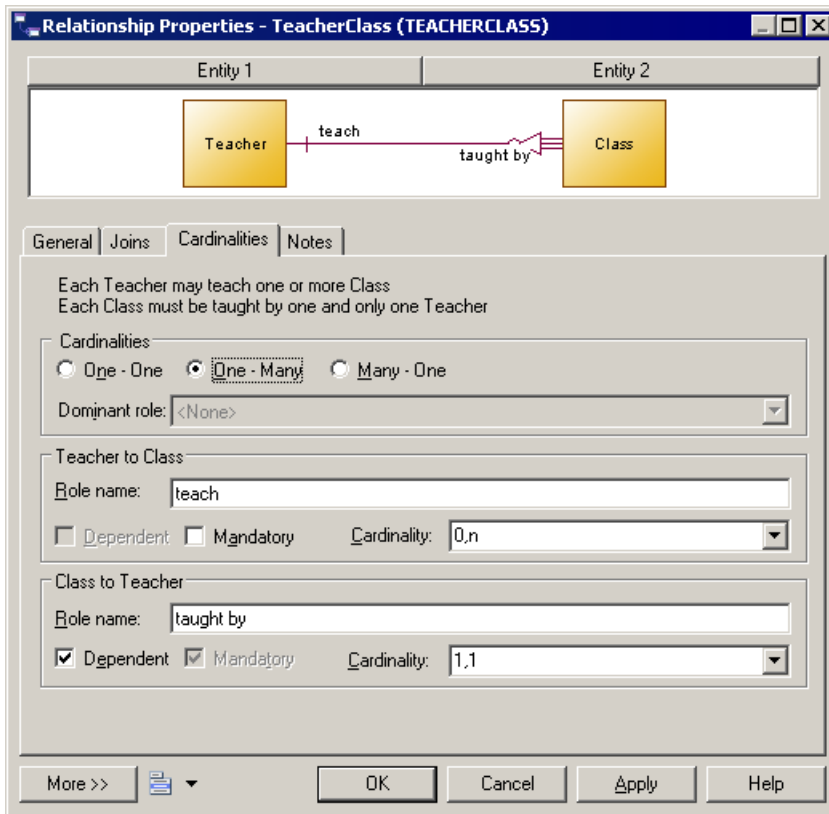
The General tab contains the following properties:

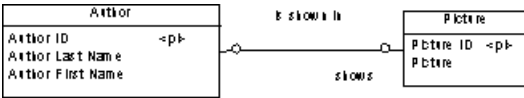
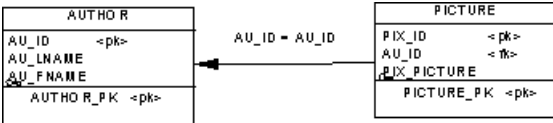
Property	Description
Name	Specifies the name of the item, which should be clear and meaningful, and should convey the item's purpose to non-technical users.

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

Relationship Property Sheet Cardinalities Tab

The Cardinalities tab allows you to specify the nature of the relationship between the two entities.



Property	Description
Cardinality	<p>Specifies the number of instances (none, one, or many) of an entity in relation to another entity. You can choose from the following values:</p> <ul style="list-style-type: none"> • One-to-one (symbol: <1..1>) - One instance of entity A can correspond to only one instance of entity B. • One-to-many (symbol: <1..n>) - One instance of entity A can correspond to more than one instance of entity B. • Many-to-one (symbol: <n..1>) - More than one instance of entity A can correspond to the same one instance of entity B. • Many-to-many (symbol: <n..n>) - More than one instance of entity A can correspond to more than one instance of entity B. To use n..n relationships in an LDM, see <i>Enabling many-to-many relationships in an LDM</i> on page 62.
Dominant role	<p>[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.</p> <p>In the following example, the author is the dominant entity:</p>  <p>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:</p> 

For information about the termination points of the relationships in each of the supported notations, see *Supported CDM/LDM Notations* on page 21.

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

Property	Description
Role name	<p>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.</p> <p>To modify the sentences generated from your role names, edit your model's assertion template (see <i>Assertion Template</i> on page 14).</p>
Dependent	<p>Specifies that the entity is dependent on and partially identified by the other entity.</p> <p>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:</p> <pre> classDiagram class Task { Task name <-pk> Task cost } class Project { Project number <-pk> Project name Project label Start date End date } Task "0..n" -- "1..1" Project </pre>
Mandatory	<p>Specifies that each instance of the entity requires at least one instance of the other entity.</p> <p>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.</p> <p>Implied by dependent</p>
Cardinality	<p>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:</p> <ul style="list-style-type: none"> • 0..1 – Zero to one instances • 0..n – Zero to many instances • 1..1 – Exactly one instance • 1..n – one to many instances

Relationship Property Sheet Joins Tab (LDM)

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

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




Linking Attributes in a Primary or Alternate Identifier

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

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

Reuse and Migration Option for a Selected Relationship

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

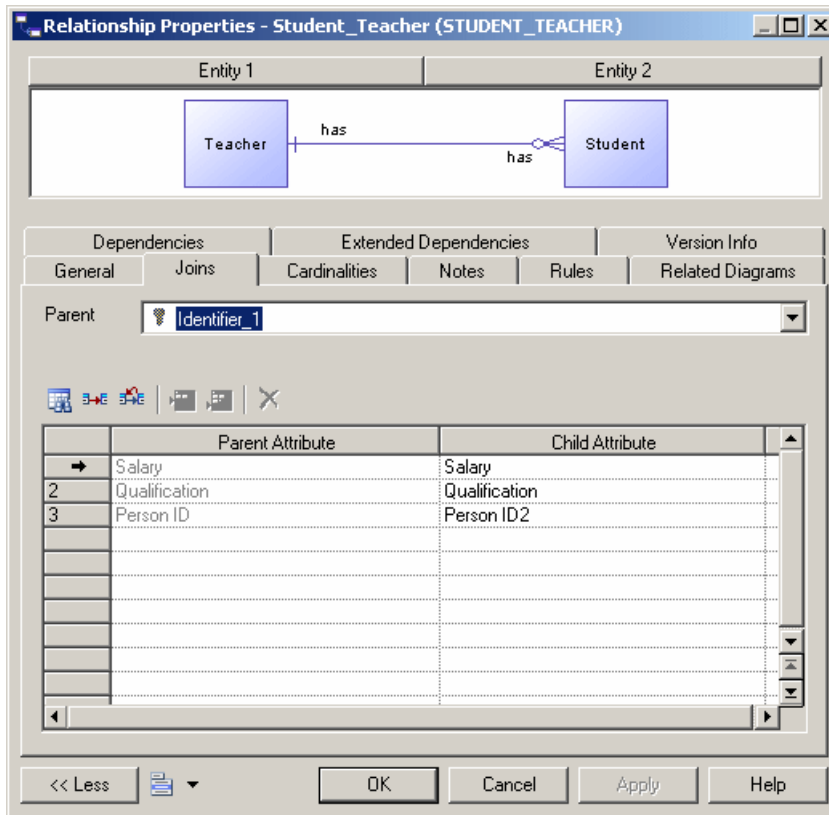
Tool	Description
	Reuse Attributes - Reuse existing child attributes with same code as parent entity attributes.
	Migrate Attributes - Migrate identifier attributes to foreign identifier attributes. If attributes do not exist they are created.
	Cancel Migration - Delete any migrated attributes in child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

Defining Joins in a Relationship

You define joins in a relationship as follows:

1. Double-click a relationship in the diagram to open its property sheet and then click the Joins tab.
2. Select a key in the Parent Identifier list to create joins on its attributes. If you select <NONE>, the attribute lists are empty and you must specify your own attributes to join.

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



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

3. [optional] If you selected <NONE> from the Parent Identifier list, click the Parent Attribute column and select an attribute from the list, then click the Child Attribute column and select a child attribute.
4. [optional] Select the Auto arrange join order check box to sort the list by the identifier attribute order. If this option is not selected, you can re-arrange the attributes using the arrow buttons.
5. Click OK.

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.

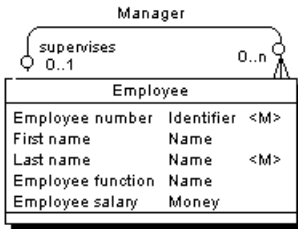
1. Select **Tools > Model Options** to open the Model Options dialog box.

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

Creating a Reflexive Relationship

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

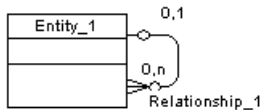
In the following example, the reflexive relationship *Supervise* expresses that an employee (Manager) can supervise other employees.



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

A relationship symbol loops back to the same entity.



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: <i>relationship_code</i> .	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 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.

The associative entity retains the name and code of the relationship, and the two new relationships handle cardinality properties (see *Associations and Association Links (CDM)* on page 67)

Changing a Relationship Directly into an Associative Entity

You change a relationship directly into an associative entity as follows:

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

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

Changing a Relationship into an Associative Entity Using the Change to Entity Wizard

You change a relationship into an associative entity using the Change to Entity wizard as follows:

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

The associative entity with two relationships replaces the relationship.

Relationship Examples

This section shows the graphic representation of various relationship properties.

One-to-many relationship	Description
	<p>Each division may have zero or more employees</p> <p>Each employee may belong to zero or one division</p>
	<p>Each division must have one or more employees</p> <p>Each employee may belong to zero or one division</p>
	<p>Each division may have zero or more employees</p> <p>Each employee must belong to one and only one division</p>
	<p>Each division must have one or more employees</p> <p>Each employee must belong to one and only one division</p>
	<p>Each division may have zero or more employees</p> <p>Each employee must belong to one and only one division</p> <p>Each employee is identified uniquely by division number and employee number</p>
	<p>Each division must have one or more employees</p> <p>Each employee must belong to one and only one division</p> <p>Each employee is identified uniquely by division number and employee number</p>

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

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

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.

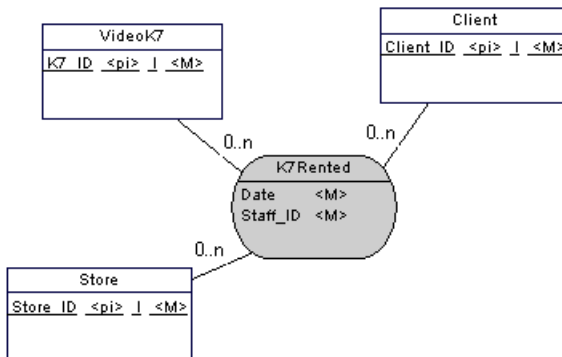
Associations and Association Links (CDM)

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.

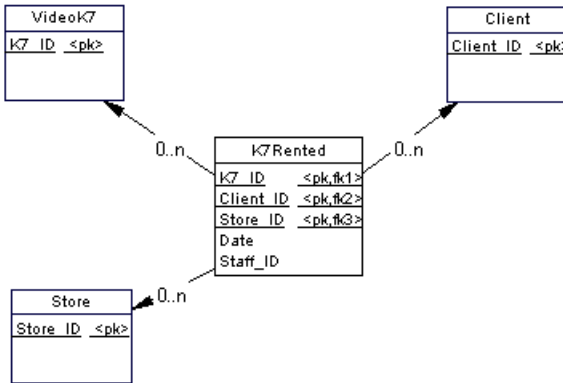
Each instance of an association corresponds to an instance of each entity linked to the association.

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

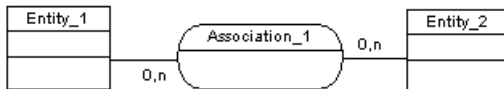
An association is connected to an entity by an association link, which symbolizes the role and the cardinality between an association and an entity.

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

An association symbol is created between the two entities.



Creating an Association Without Links

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

- Use the Association tool in the diagram Palette.
- Select **Model > Associations** to access the List of Associations, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select **New > Association**.

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

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

Association Properties

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

The General tab contains the following properties:

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

Association Link Properties

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

The General tab contains the following properties:

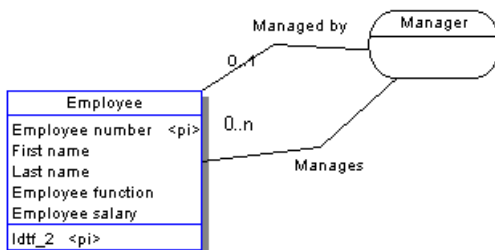
Property	Description
Entity	Specifies the entity connected by the association link. Use the tools to the right of the list to create, browse for, or view the properties of the currently selected entity.
Association	Specifies the association connected by the association link.
Role	Specifies the label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

Property	Description
Cardinality	<p>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:</p> <ul style="list-style-type: none"> • 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 <p>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)</p>
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined.

Creating a Reflexive Association

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

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



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

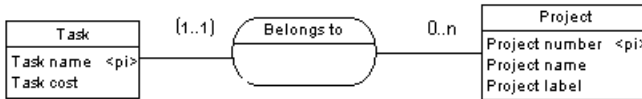
Defining a Dependent Association

In a dependent association, one entity is partially identified by another. Each entity must have an identifier. In some cases, however, the attributes of an entity are not sufficient to identify an

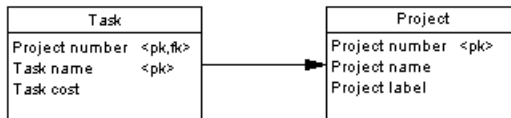
occurrence of the entity. For these entities, their identifiers incorporate the identifier of another entity with which they have a dependent association.

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.

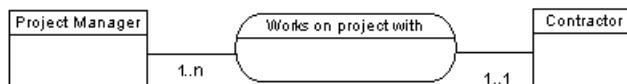
1. Double-click an association link symbol to display the association link property sheet.
2. Select the Identifier check box and then click OK to return to the model.

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

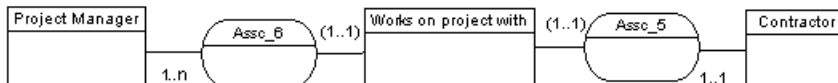
Changing an Association into an Associative Entity

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.

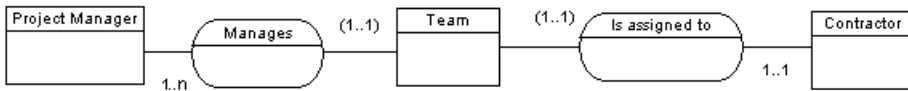
Two entities *PROJECT MANAGER* and *CONTRACTOR* are linked by the association *WORKS ON PROJECT WITH*:



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

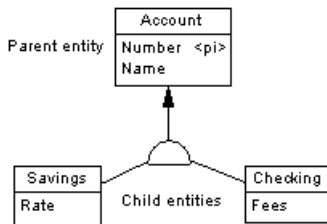
The tools used for creating association attributes on this tab are the same as those for creating entity attributes.

For more information, see *Creating an attribute* on page 52.

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:

IDEF 1X	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. You represent an inheritance by placing one entity symbol on top of another. Barker inheritances are always complete and mutually exclusive. The supertype lists its subtypes on the Subtypes tab (see *Entity properties* on page 48).

Creating an Inheritance

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

- Use the Inheritance tool in the diagram Palette (see *Creating an inheritance with the Inheritance tool* on page 73).
- Select **Model > Inheritances** to access the List of Inheritances, and click the Add a Row tool. You will be required to specify a parent entity.
- Right-click the model or package in the Browser, and select **New > Inheritance**. You will be required to specify a parent entity.

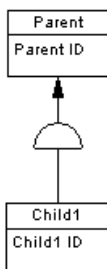
For general information about creating objects, see the Objects chapter in the *Core Features Guide*.

Creating an Inheritance with the Inheritance Tool

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

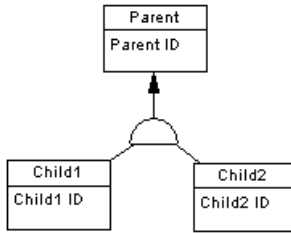
1. Select the Inheritance tool in the Palette.
2. Click the child entity and, while continuing to hold down the mouse button, drag the cursor to the parent entity. Release the mouse button inside the child entity.

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



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

The symbol links all the child entities to the parent.



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

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

Inheritance Properties

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

The General tab contains the following properties:

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

Inheritance Property Sheet Generation Tab

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

The Generation tab contains the following properties:

Property	Description
Generation Mode	<p>Specifies which parts of the inheritance will be generated. You can specify one or both of the following:</p> <ul style="list-style-type: none"> • Generate parent - Generates a table corresponding to the parent entity, which: <ul style="list-style-type: none"> • Inherits entity attributes of each child entity • Is affected by child entity relationships • Contains reference to any table which has a many-to-one relationship with child entity • Generate children - Generates a table corresponding to each child entity. The primary key of each child table is the concatenation of the child entity identifier and the parent entity identifier. When this option is selected, you must choose between: <ul style="list-style-type: none"> • 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 <p>Note that, if you do not select Generate Children, you can control the generation of individual child tables using the option Generate in the property sheet of each child entity.</p>

Property	Description
Specifying attributes	<p>In the case of parent-only generation, you can choose to define a <i>specifying attribute</i>, an entity attribute that is defined for a parent entity which differentiates occurrences of each child.</p> <p>In the example below, the TITLE entity has two children, NONPERIODICAL and PERIODICAL.</p> <p>As only the parent table TITLE will be generated in a PDM, a specifying entity attribute PERIODICAL is defined for the inheritance link to differentiate between the two child entities.</p> <p>In the PDM, each of the child entity attributes will generate columns in the table TITLE, and the specifying entity attribute PERIODICAL will generate a corresponding column PERIODICAL. The values of this column indicate whether an instance of TITLE is a periodical or not.</p> <div data-bbox="400 621 813 980" style="text-align: center;"> <pre> graph BT Nonperiodical --> Title Periodical --> Title style Title fill:#fff,stroke:#000 style Nonperiodical fill:#fff,stroke:#000 style Periodical fill:#fff,stroke:#000 </pre> </div> <p>The tools available on this tab for creating specifying attributes are the same as those for creating entity attributes. For more information, see <i>Creating an attribute</i> on page 52.</p>

Inheritance Property Sheet Children Tab

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

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

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

4. Click OK to return to the diagram.

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

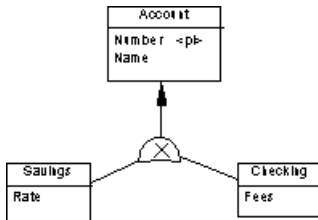
Making Inheritance Links Mutually Exclusive

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

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

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

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



Identifier Migration Through an Inheritance (LDM)

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

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



CHAPTER 3 Building Physical Diagrams









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.

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

Physical Diagram Objects

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 <i>Tables (PDM)</i> on page 81.
Column	[none]	[none]	Data structure that contains an individual data item within a row (record), model equivalent of a database field. See <i>Columns (PDM)</i> on page 103.
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 <i>Keys (PDM)</i> on page 118.
Alternate key	[none]	[none]	Column or columns whose values uniquely identify each row in a table, and which is not a primary key. See <i>Keys (PDM)</i> on page 118.
Foreign key	[none]	[none]	Column or columns whose values depend on and migrate from a primary or alternate key in another table. See <i>Keys (PDM)</i> on page 118.
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 <i>Indexes (PDM)</i> on page 123.
Default	[none]	[none]	[certain DBMSs] A default value for a column. See <i>Defaults (PDM)</i> on page 146.
Domain	[none]	[none]	Defines valid values for a column. See <i>Domains (CDM/LDM/PDM)</i> on page 148.

Object	Tool	Symbol	Description
Sequence	[none]	[none]	[certain DBMSs] Defines the form of incrementation for a column. See <i>Sequences (PDM)</i> on page 156.
Abstract data type	[none]	[none]	[certain DBMSs] User-defined data type. See <i>Abstract Data Types (PDM)</i> on page 160.
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 <i>References (PDM)</i> on page 168.
View			Data structure that results from a SQL query and that is built from data in one or more tables. See <i>Views (PDM)</i> on page 179.
View Reference			Link between a table and a view. See <i>View References (PDM)</i> on page 190.
Trigger	[none]	[none]	A segment of SQL code associated with a table or a view. See <i>Triggers (PDM)</i> on page 221.
Procedure			Precompiled collection of SQL statements stored under a name in the database and processed as a unit. See <i>Stored Procedures and Functions (PDM)</i> on page 248.
Database	[none]	[none]	The database of which the PDM is a representation. See <i>Database Properties (PDM)</i> on page 9.
Storage	[none]	[none]	A partition on a storage device. See <i>Configuring Tablespace and Storages</i> on page 298.
Tablespace	[none]	[none]	A partition in a database. See <i>Configuring Tablespaces and Storages</i> on page 298.
User	[none]	[none]	A person who can log in or connect to the database. See <i>Users, Groups, and Roles (PDM)</i> on page 131.
Role	[none]	[none]	A predefined user profile. See <i>Users, Groups, and Roles (PDM)</i> on page 131.
Group	[none]	[none]	Defines privileges and permissions for a set of users. See <i>Users, Groups, and Roles (PDM)</i> on page 131.
Synonym	[none]	[none]	An alternative name for various types of objects. See <i>Synonyms (PDM)</i> on page 142.

Object	Tool	Symbol	Description
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 <i>Web Services (PDM)</i> on page 274.
Web operation	[none]	[none]	Sub-object of a Web service containing a SQL statement and displaying Web parameters and result columns. See <i>Web Operations (PDM)</i> on page 277.

Creating a Physical Diagram

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

- Right-click the model in the Browser and select **New > Physical Diagram**.
- Right-click the background of any diagram and select **Diagram > New Diagram > Physical Diagram**.

To create a new PDM with a physical diagram, select **File > New Model**, choose to create a physical data model from the Model type list, choose **Physical Diagram** as the first diagram, and click **OK**.

Tables (PDM)

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

- Columns are named properties of a table that describe its characteristics (see *Columns (PDM)* on page 103).
- Indexes are data structures associated with a table that are logically ordered by key values (see *Indexes (PDM)* on page 123).
- Keys are columns, or combinations of columns, that uniquely identify rows in a table. Each key can generate a unique index or a unique constraint in a target database (see *Keys (PDM)* on page 118).
- Triggers are segments of SQL code associated with tables, and stored in a database. They are invoked automatically whenever there is an attempt to modify data in associated tables (see *Chapter 5, Building Triggers and Procedures* on page 221).

You can use database-specific physical options to specify physical parameters for tables and many other objects.

Creating a Table

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

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

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

Table Properties

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

The General tab contains the following properties:

Property	Description
Name	Specifies the name of the item which should be clear and meaningful, and should convey the item's purpose to non-technical users
Code	Specifies the technical name of the item used for generating code or scripts, which may be abbreviated, and should not generally include spaces
Comment	Specifies a descriptive label for the table
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined
Owner	Specifies the name of the table owner. You choose an owner from a list of users. A table can only have one owner at a time. This is normally the table creator
Number	<p>Specifies the estimated number of records in the table in the physical database, which is used to estimate database size. This box is automatically populated during reverse engineering if you select the Statistics check box in the Reverse Engineering dialog box (see <i>Reverse Engineering from a Live Database</i> on page 375).</p> <p>You can enter your own value in this field, or refresh its statistics (along with those for all of the table's columns) at any time by right-clicking the table symbol or its entry in the Browser and selecting Update Statistics from the contextual menu. You can also update the statistics for all tables by selecting Tools > Update Statistics (see <i>Reverse Engineering Database Statistics</i> on page 384).</p>
Generate	Specifies that the table is generated in the database
Dimensional type	<p>Specifies the multidimensional type of the table. You can choose between:</p> <ul style="list-style-type: none">• Dimension - see <i>Dimensions (PDM)</i> on page 212• Fact - see <i>Facts (PDM)</i> on page 215

Property	Description
Type	Specifies the type of the table. You can choose between: <ul style="list-style-type: none"> • Relational • Object - for abstract data types • XML - for storing and retrieving data using an XML format. For more information, see <i>Creating an XML Table or View</i> on page 85

Lifecycle tab

The Lifecycle tab is available if data lifecycle modeling (see *Lifecycles (PDM)* on page 289) 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 293).

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 <i>Lifecycle Properties</i> on page 293).
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:

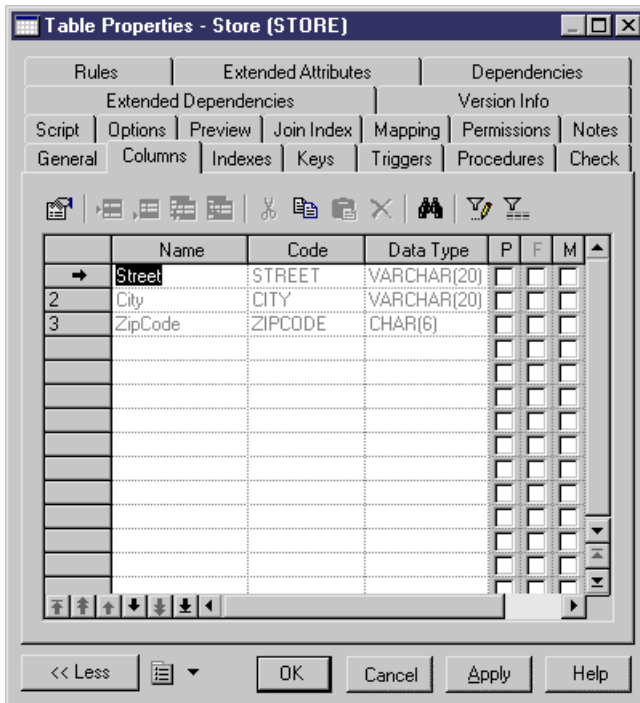
- Columns - lists the columns associated with the table (see *Columns (PDM)* on page 103).
- Indexes - lists the indexes associated with the table (see *Indexes (PDM)* on page 123).
- Keys - lists the keys associated with the table (see *Keys (PDM)* on page 118).
- Triggers - lists the triggers associated with the table (see *Triggers (PDM)* on page 221).
- Procedures - lists the procedures associated with the table (see *Stored Procedures and Functions (PDM)* on page 248).

- Security Procedures - [data lifecycle modeling only] lists the procedures which control access to the table (see *Stored Procedures and Functions (PDM)* on page 248).
- Check - specifies the constraints associated with the table (see *Setting Data Profiling Constraints* on page 105)
- Physical Options - lists the physical options associated with the table (see *Physical Options* on page 301).
- Preview - displays the SQL code associated with the table (see *Previewing SQL Statements* on page 397).

Linking a Table to an Abstract Data Type

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

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



For more information on abstract data types, see *Abstract Data Types (PDM)* on page 160.

Creating an XML Table or View

Some DBMS support tables and views of XML type.

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

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

Property	Description
Schema	<p>Allows you to enter the target namespace or the name of an XML model. The schema must be registered in the database to be used for validating XML documents. You can:</p> <ul style="list-style-type: none"> • type a user-defined schema name • click the Select a registered schema button to connect to a database and select a registered schema <p>If you select an element from an XML model open in the PowerDesigner workspace, the Schema property is automatically initialized with the XML model target namespace. Note that this schema must also be registered in the database to be used for validating XML documents</p>
Element	<p>Allows you to select a root element in the XML document. You can:</p> <ul style="list-style-type: none"> • type a user-defined element name • click the Select an element button to select an element from the XML models open in the workspace or from the schema registered in the database

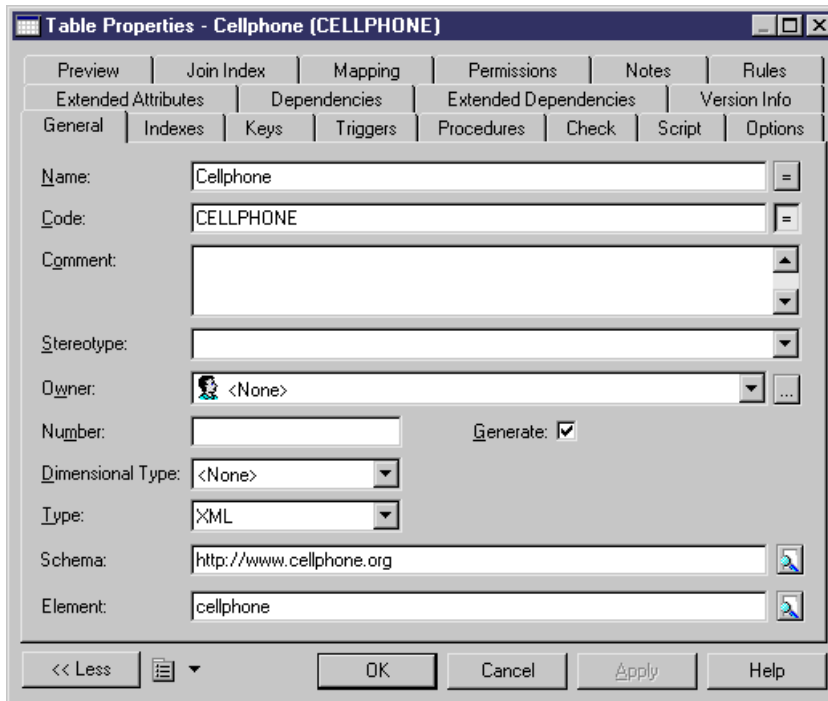
1. Right-click the Table category in the Browser and select New.

The property sheet of a new table is displayed.

2. Type a table name and a table code.
3. In the Type list, select XML.

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

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



6. Click OK.

Naming a Table Constraint

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

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

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

1. Double-click a table in the diagram to display its property sheet, and click the Check tab.
2. Click the User Defined button to the right of the Constraint Name box, and type changes to the constraint name in the Constraint Name box.

Note: You can always return to the default constraint name by re-clicking the User-Defined button.

3. Click OK.

For more information, see *Check Parameters (CDM/LDM/PDM)* on page 105.

Creating External Tables

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

External tables are metamodel extensions. In SybaseAdaptive Server® Anywhere and Adaptive Server® Enterprise, external tables are called proxy tables and an extension file is provided to help you design them (see *Proxy Tables (ASE/SQL Anywhere)* on page 586).

Denormalizing Tables and Columns

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

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

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

Denormalization may be accomplished in several ways:

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

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

Creating Horizontal Partitions

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

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

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

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

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

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

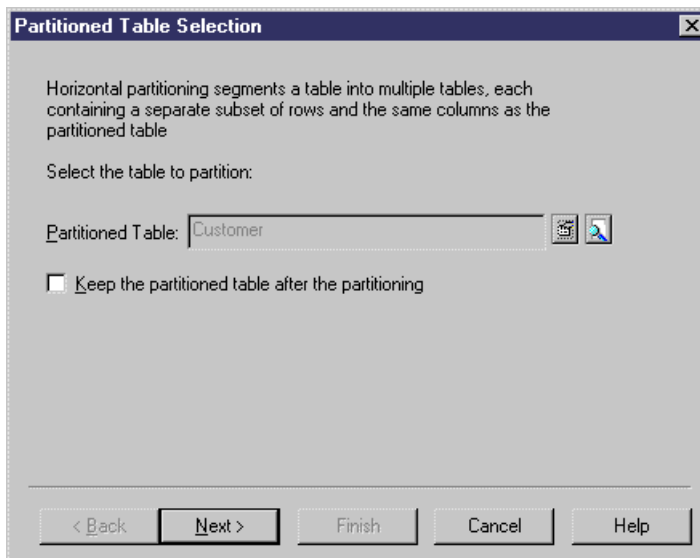
Annual Sales_2001	
Amount	numeric
Order ID	varchar(10) <pk>

Horizontal partitioning has the following pros and cons:

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

You can partition tables horizontally using the Horizontal Partitioning Wizard.

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



2. Select the table to partition and select the check box if you want to keep the original table after partitioning. Then click Next to go to the Partition Definition page.
3. The Partition Definition page allows you to create as many partitions as you need with the Insert and Add a row tools. The name of each partition must be unique in the model. A table will be created for each partition you specify, and will take the name of the relevant partition. Then click Next to go to the Discriminant Column Selection page.
4. The Discriminant Column Selection page allows you to specify the columns that will be used as partition criteria using the Add Columns tool. These columns will not be included in the partitions. Then click Next to go to the Partitioning Information page.
5. The Partitioning Information page allows you to specify a name and code for the transformation object that will be created together with the partitions. Then click Finish.

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

Creating Vertical Partitions

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

The table Customer contains the following columns:

Customer	
<u>Cust_ID</u>	<pk>
Name	
Last_Name	
Street	
City	
Zip_Code	
Country	
Credit_Card_Number	
Account_Number	

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

Customer_credit_details	Customer_address
<u>Cust_ID</u>	<u>Cust_ID</u>
Credit_Card_Number	Name
Account_Number	Last_Name
	Street
	City
	Zip_Code
	Country

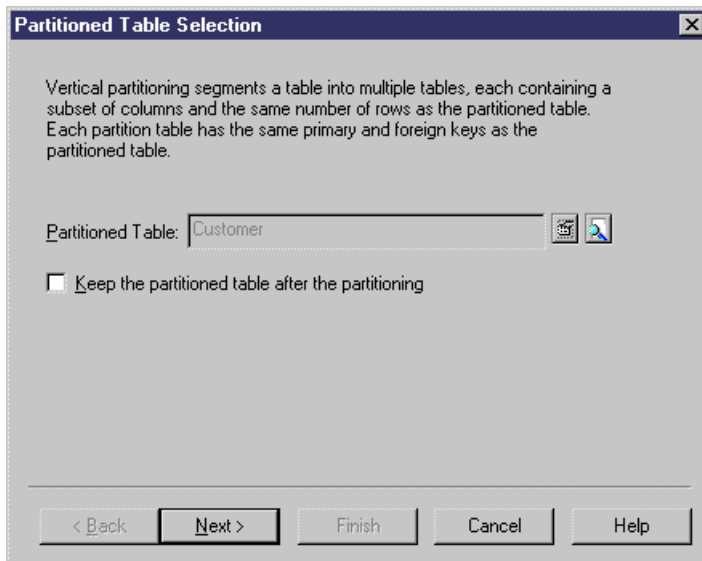
Vertical partitioning has the following pros and cons:

Pros	Cons
Improve the query response time	Requires additional joins and unions to retrieve data from multiple tables

Pros	Cons
Allows you to split data requiring different levels of protection, you can store confidential information in a special partition	Requires more intelligent queries to determine which table contains the requested data
Reduce the load time of indexed tables	Requires additional metadata to describe the partitioned table

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

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



2. Select the table to partition and select the check box if you want to keep the original table after partitioning. Then click Next to go to the Partition Definition page.
3. The Partition Definition page allows you to create as many partitions as you need with the Insert and Add a row tools. The name of each partition must be unique in the model. A table will be created for each partition you specify, and will take the name of the relevant partition. Then click Next to go to the Discriminant Column Selection page.
4. The Discriminant Column Selection page allows you to specify which columns will be included in each partition table. Drag columns from the Available columns pane, and drop them onto the appropriate partition table in the Columns distribution pane, or use the Add

and Remove buttons at the bottom of each pane. When all your columns are allocated, click Next to go to the Partitioning Information page.

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

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

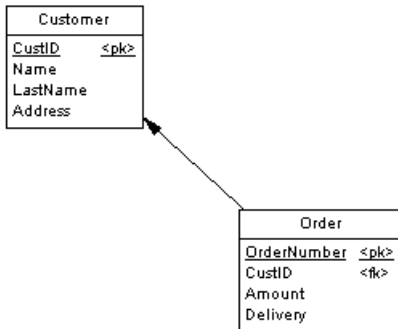
Creating Table Collapsings

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

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

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

Tables Customer and Order are linked together.



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

Cust_Order : 1
Name
LastName
Address
<u>OrderNumber</u> <pk>
CustID
Amount
Delivery

Cust_Order : 2
Name
LastName
Address
<u>OrderNumber</u> <pk>
CustID
Amount
Delivery

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

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

2. Specify a name and code for the target table to be created, and then click Next to go to the Input Table Selection page.
3. The Input Table Selection page allows you to select the tables to collapse with the Add Tables tool. Select the check box if you want to keep the original tables after collapsing, and then click Next to go to the Table Collapsing Information page.

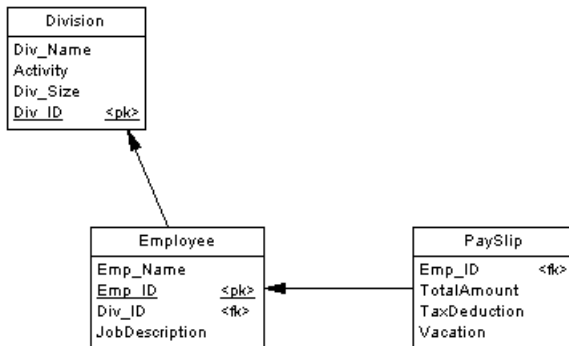
- The Table Collapsing Information page allows you to specify a name and code for the transformation object that will be created together with the table collapsing. Then click Finish.

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

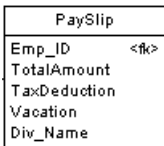
Denormalizing Columns

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

In this example, you want to have the division name printed on the pay slip of each employee, however, you do not want to create a join between those tables.



You can denormalize columns in order to have column Div_Name in table PaySlip:



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

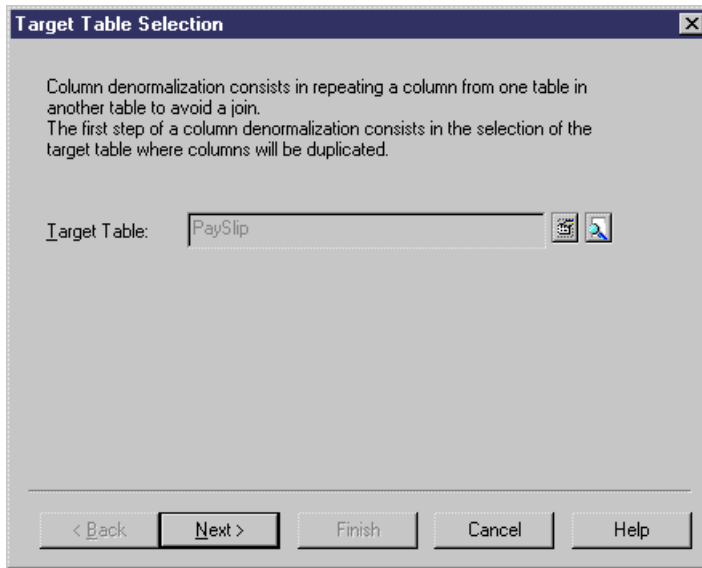
You can move and paste a denormalized column into another model or package in the standard way.

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

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

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

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



2. Select the table in which you want the denormalized columns to be added, and then click Next to go to the Column Selection page.
3. The Column Selection page allows you to select the columns to replicate. Select one or more columns to replicate, and then click Finish.

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

Denormalization Object Properties

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

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

The following properties are available on the General tab:

Property	Description
Name	Specifies the name of the partitioning object. It is recommended to provide a clear name in the Partitioning Wizard

Property	Description
Code	Specifies the code of the partitioning object
Comment	Specifies additional information about the partitioning object.
Partitioned table	[partitionings only] Specifies the name of the table used to create the table partitions.
Discriminant Columns	[horizontal partitionings only] Specifies the name and code of the columns used as partition criteria
Target table	Specifies the name of the table resulting from the collapsing of selected tables

The following tabs are also available:

- Partitions - [partitionings only] Lists the tables associated with the partitioning. The following actions can be performed on this tab:
 - Open the property sheets of the partition tables.
 - Add more partitions and edit the properties of the corresponding tables.
 - Add comments to identify the different partitions
 - Delete partitions and their corresponding tables. When you delete a partition, you are prompted to specify whether you want to delete the corresponding table. You can delete a partition and keep its table, but you cannot delete a table and keep an empty partition
- Partition Columns - [vertical partitionings only] Displays the distribution of columns between the partition tables. You can drag and drop columns to reallocate them between tables.
- Source Tables - [table collapsings only] Lists the tables that were collapsed. These tables will no longer exist unless you selected to keep the original tables in the Table Collapsing Wizard.

Example: Intermodel Generation and Horizontal Partitions

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

For example, the Sales CDM contains the entity Customer:

Customer	
Name	A30
City	A30
ID	N10,2

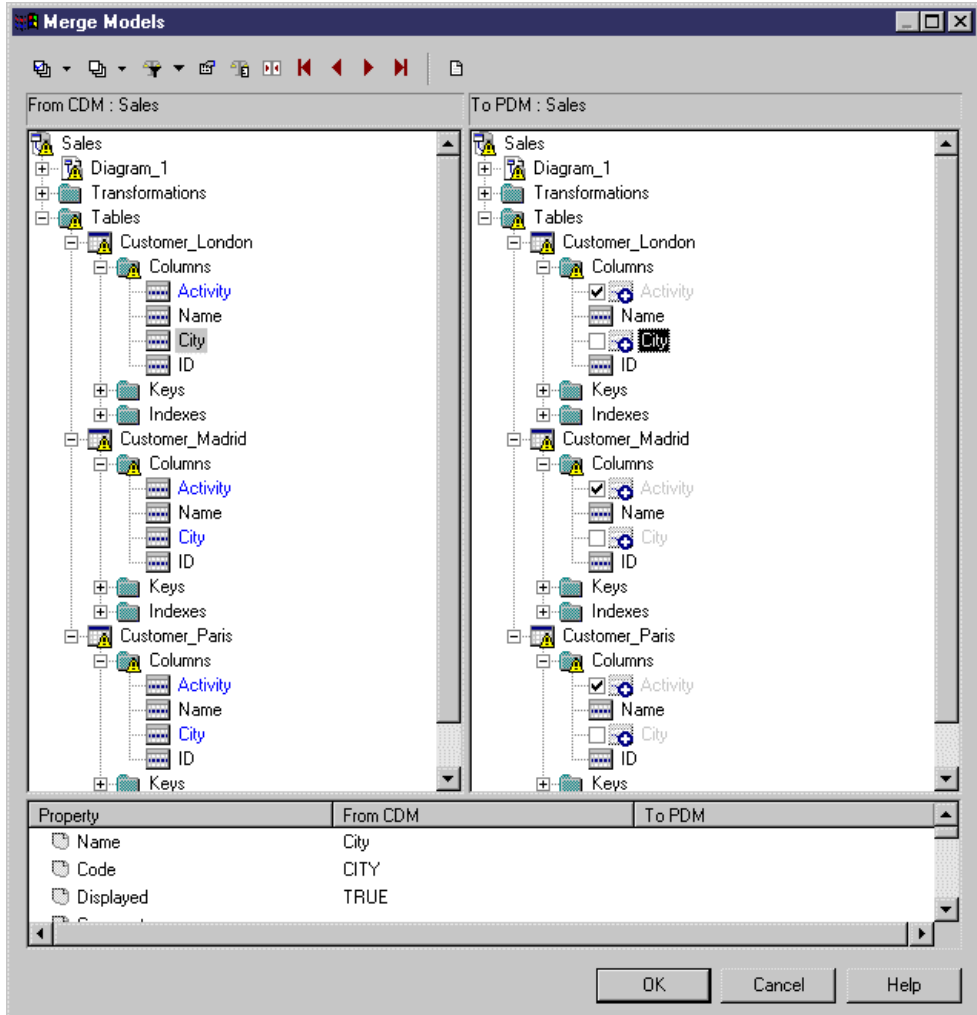
You generate the Sales PDM from the CDM, and the Customer entity is generated to the Customer table:

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

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

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

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



Example: Intermodel Generation and Vertical Partitions

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

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

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

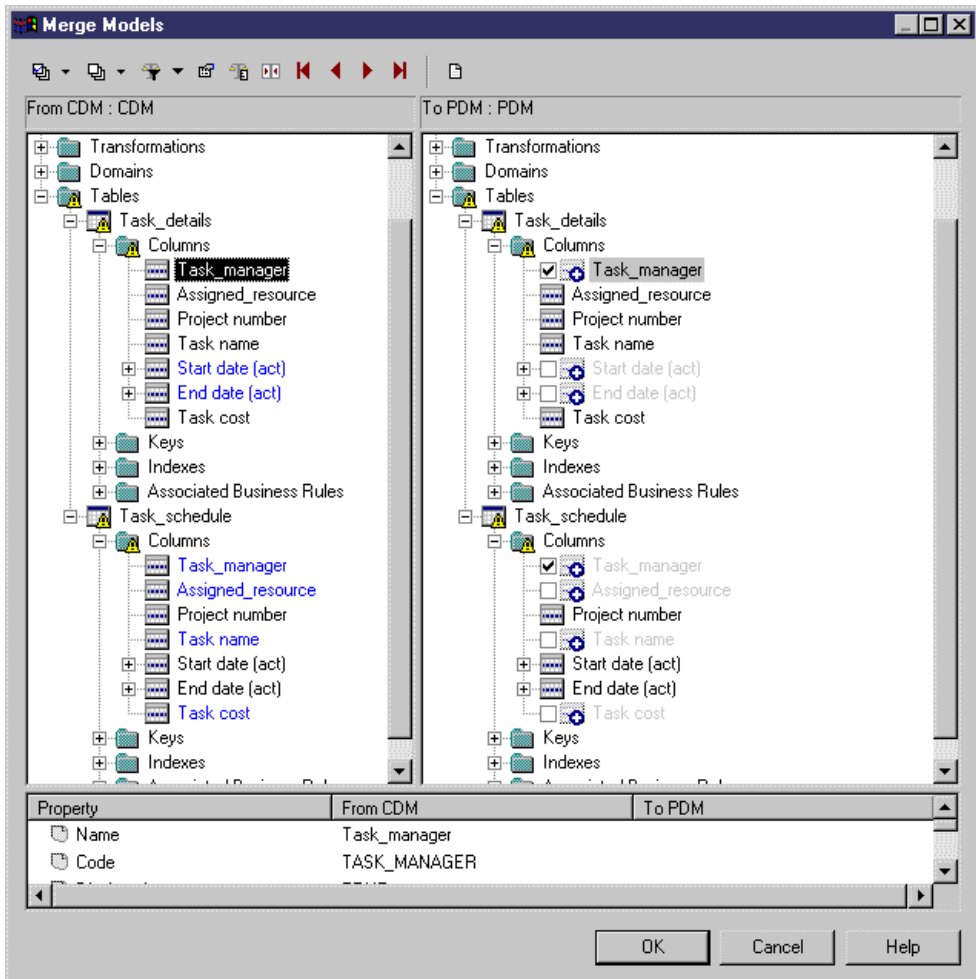
The CDM is generated in a PDM, entity Task becomes table Task:

Task	
Assigned_resource	
<u>Project number</u>	<pk>
Task name	
Start date	
End date	
Task cost	

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

Task_details		Task_schedule	
Assigned_resource		Project number	<pk>
Project number	<pk>	Start date (act)	
Task name		End date (act)	
Task cost			

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



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, and then click the :

- **Cancel** tool - to remove the denormalization as well as the associated tables. Note that this tool is only available if the selected denormalization object is based upon a table generated from another model. You can recover the original table by regenerating it from the source model.
- **Delete** tool - to remove the denormalization but keep the associated tables.

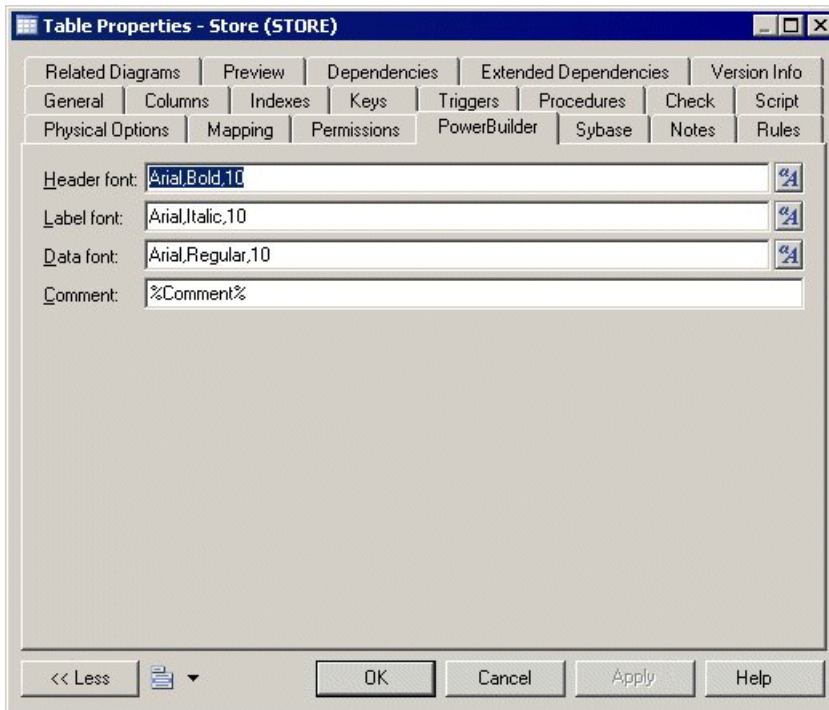
Note: You cannot move or paste a denormalization object to another model or package.

Using PowerBuilder Extended Attributes

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

PowerDesigner supports the modeling of this information through an extension file. To enable the PowerBuilder extensions in your model, select **Model > Extensions**, click the **Import** 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:



Generating PowerBuilder Extended Attributes

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

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

Object	Property
Table	Comment
Column	Comment Label Header Initial value

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

1. Select **Tools > PowerBuilder > Generate Extended Attributes** to open the PowerBuilder Extended Attributes Generation dialog box.
2. Click the Connect to a Data Source tool to open the Connect to a Data Source window.
3. Select a machine or file data source and click Connect.

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

4. Select the tables you want to generate.
5. Click OK to start generation.

The Output window displays the generation messages.

Reverse Engineering PowerBuilder Extended Attributes

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

During reverse engineering (see *Generating PowerBuilder Extended Attributes* on page 99), certain 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.

1. Select **Tools > PowerBuilder > Reverse Extended Attributes**.

The PowerBuilder Extended Attributes Reverse Engineering dialog box is displayed.

2. Click the Connect to a Data Source tool to open the Connect to a Data Source dialog box.
3. Select a machine or file data source and click Connect.

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

4. Select the tables you want to reverse engineer.
5. Click OK to start reverse engineering.

The Output window displays the reverse engineering messages.







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	<table border="1"> <thead> <tr> <th colspan="2">Publisher</th> </tr> </thead> <tbody> <tr> <td>Publisher ID</td> <td>char(12)</td> </tr> <tr> <td>Publisher Name</td> <td>varchar(40)</td> </tr> <tr> <td>City</td> <td>varchar(20)</td> </tr> <tr> <td>State</td> <td>char(2)</td> </tr> </tbody> </table>	Publisher		Publisher ID	char(12)	Publisher Name	varchar(40)	City	varchar(20)	State	char(2)								
Publisher																				
Publisher ID	char(12)																			
Publisher Name	varchar(40)																			
City	varchar(20)																			
State	char(2)																			
Replace by domains	Domain codes for each column attached to a domain	<table border="1"> <thead> <tr> <th colspan="2">Publisher</th> </tr> </thead> <tbody> <tr> <td>Publisher ID</td> <td>AN_IDENTIFIER</td> </tr> <tr> <td>Publisher Name</td> <td>NAMES</td> </tr> <tr> <td>City</td> <td>SHORT_TEXT</td> </tr> <tr> <td>State</td> <td>char(2)</td> </tr> </tbody> </table>	Publisher		Publisher ID	AN_IDENTIFIER	Publisher Name	NAMES	City	SHORT_TEXT	State	char(2)								
Publisher																				
Publisher ID	AN_IDENTIFIER																			
Publisher Name	NAMES																			
City	SHORT_TEXT																			
State	char(2)																			
Domains	Domain of an attribute in the table. This display option interacts with the selection for Data types. As a result, there are four display options	See Display domain and data type for options and examples																		
Key Indicators	<pk>, <fk>, and <ak> indicators next to primary key, foreign key, and alternate key columns respectively. When the Keys preference is also selected, the key names are listed at the bottom of the table symbol	<table border="1"> <thead> <tr> <th colspan="2">Publisher</th> </tr> </thead> <tbody> <tr> <td>Publisher ID</td> <td><pk></td> </tr> <tr> <td>Author ID</td> <td><fk></td> </tr> <tr> <td>Title ISBN</td> <td><fk></td> </tr> <tr> <td>Publisher Name</td> <td><ak></td> </tr> <tr> <td>City</td> <td></td> </tr> <tr> <td>State</td> <td></td> </tr> <tr> <td> Primary Key</td> <td><pk></td> </tr> <tr> <td> Pub_Name</td> <td><ak></td> </tr> </tbody> </table>	Publisher		Publisher ID	<pk>	Author ID	<fk>	Title ISBN	<fk>	Publisher Name	<ak>	City		State		 Primary Key	<pk>	 Pub_Name	<ak>
Publisher																				
Publisher ID	<pk>																			
Author ID	<fk>																			
Title ISBN	<fk>																			
Publisher Name	<ak>																			
City																				
State																				
 Primary Key	<pk>																			
 Pub_Name	<ak>																			

Preference	Displays	Example																						
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	<table border="1"> <thead> <tr> <th colspan="2">Sale</th> </tr> </thead> <tbody> <tr> <td>Sale Invoice ID</td> <td><i1></td> </tr> <tr> <td>Store ID</td> <td><i2></td> </tr> <tr> <td>Title ISBN</td> <td><i3></td> </tr> <tr> <td>Sale Date</td> <td></td> </tr> <tr> <td>Sale Amount</td> <td></td> </tr> <tr> <td>Sale Terms</td> <td></td> </tr> <tr> <td>Sale Quantity</td> <td></td> </tr> <tr> <td> SALE_PK</td> <td><i1></td> </tr> <tr> <td> STORE_SALES_FK</td> <td><i2></td> </tr> <tr> <td> SALES_TITLE_FK</td> <td><i3></td> </tr> </tbody> </table>	Sale		Sale Invoice ID	<i1>	Store ID	<i2>	Title ISBN	<i3>	Sale Date		Sale Amount		Sale Terms		Sale Quantity		SALE_PK	<i1>	STORE_SALES_FK	<i2>	SALES_TITLE_FK	<i3>
Sale																								
Sale Invoice ID	<i1>																							
Store ID	<i2>																							
Title ISBN	<i3>																							
Sale Date																								
Sale Amount																								
Sale Terms																								
Sale Quantity																								
SALE_PK	<i1>																							
STORE_SALES_FK	<i2>																							
SALES_TITLE_FK	<i3>																							
NULL/NOT NULL	Column indicator: null, not null, identity, or with default (DBMS-dependent)	<table border="1"> <thead> <tr> <th colspan="2">PUBLISHER</th> </tr> </thead> <tbody> <tr> <td>PUB_ID</td> <td>not null</td> </tr> <tr> <td>AU_ID</td> <td>null</td> </tr> <tr> <td>TITLE_ISBN</td> <td>null</td> </tr> <tr> <td>PUB_NAME</td> <td>null</td> </tr> <tr> <td>CITY</td> <td>null</td> </tr> <tr> <td>STATE</td> <td>null</td> </tr> </tbody> </table>	PUBLISHER		PUB_ID	not null	AU_ID	null	TITLE_ISBN	null	PUB_NAME	null	CITY	null	STATE	null								
PUBLISHER																								
PUB_ID	not null																							
AU_ID	null																							
TITLE_ISBN	null																							
PUB_NAME	null																							
CITY	null																							
STATE	null																							

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	<table border="1"> <thead> <tr> <th colspan="2">SALE</th> </tr> </thead> <tbody> <tr> <td>SALE ID</td> <td><Undefined></td> </tr> <tr> <td>STORE ID</td> <td>char(12)</td> </tr> </tbody> </table>	SALE		SALE ID	<Undefined>	STORE ID	char(12)
SALE								
SALE ID	<Undefined>							
STORE ID	char(12)							
Domains	Only the domain, if it exists	<table border="1"> <thead> <tr> <th colspan="2">SALE</th> </tr> </thead> <tbody> <tr> <td>SALE_ID</td> <td>SALE_IDENTIFIER</td> </tr> <tr> <td>STORE_ID</td> <td><None></td> </tr> </tbody> </table>	SALE		SALE_ID	SALE_IDENTIFIER	STORE_ID	<None>
SALE								
SALE_ID	SALE_IDENTIFIER							
STORE_ID	<None>							
Data types and Domains	Both data type and domain, if they exist	<table border="1"> <thead> <tr> <th colspan="2">SALE</th> </tr> </thead> <tbody> <tr> <td>SALE ID</td> <td><Undefined> SALE_IDENTIFIER</td> </tr> <tr> <td>STORE ID</td> <td>char(12) <None></td> </tr> </tbody> </table>	SALE		SALE ID	<Undefined> SALE_IDENTIFIER	STORE ID	char(12) <None>
SALE								
SALE ID	<Undefined> SALE_IDENTIFIER							
STORE ID	char(12) <None>							
Data types and Replace by domains	<p>If domain exists and data type does not exist, then displays domain.</p> <p>If domain does not exist and data type exists, then displays data type.</p>	<table border="1"> <thead> <tr> <th colspan="2">SALE</th> </tr> </thead> <tbody> <tr> <td>SALE ID</td> <td>SALE_IDENTIFIER</td> </tr> <tr> <td>STORE ID</td> <td>char(12)</td> </tr> </tbody> </table>	SALE		SALE ID	SALE_IDENTIFIER	STORE ID	char(12)
SALE								
SALE ID	SALE_IDENTIFIER							
STORE ID	char(12)							

Note: For information about selecting other properties to display, see "Display Preferences" in *Chapter 8, Customizing Your Modeling Environment* of the *Core Features Guide*.

Columns (PDM)

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

Creating a Column

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

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

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

Column Properties

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

The General tab contains the following properties:

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

Property	Description
Precision	Specifies the maximum number of places after the decimal point.
Identity	Specifies that the column is populated with values generated by the database. Identity column are often used as primary keys(not available for all DBMSs).
Domain	Specifies the name of the associated domain (see <i>Domains (CDM/LDM/PDM)</i> on page 148). Use the tools to the right of the list to create a domain, browse the tree of available domains, or view the properties of the currently selected domain.
Computed	Specifies that the column is computed from an expression using values from other columns in the table (not available for all DBMSs).
Primary key	Specifies that the values in the column uniquely identify table rows.
Foreign key	Specifies that the column depends on and migrates from a primary key column in another table.
Mandatory	Specifies that the column must be assigned a value that is not null.
With default	Specifies that a default value is assigned to the column when a null value is inserted (not available for all DBMSs).

Detail Tab

The Detail tab contain the following properties:

Property	Description
Null Values	Specifies the proportion of column entries which contain null values. You can enter a number or percentage in this and the other fields in the Column fill parameters groupbox or obtain them from your database (see <i>Obtaining Column Statistics from your Database</i> on page 105).
Distinct Values	<p>Specifies the proportion of column entries which contain distinct values. You can enter a number or percentage in this field or derive its value from database statistics. For example, if you set the percentage of distinct values to 100 % for one table column to 80% for a second column and then generate the table with 10 rows, all 10 rows in the first column will have distinct values, while only 8 rows in the second column will have distinct values.</p> <p>When you apply a test data profile with a list generation source to a column with a given percentage of distinct values, PowerDesigner uses the values from the test data profile list. If there are not enough values declared in the list, a warning message is displayed in the Output window to inform you that the distinct value parameter cannot be enforced due to lack of distinct values in the list of values.</p>
Average Length	Specifies the average length of a value. You can enter a number in this field or derive its value from database statistics.

Property	Description
Profile	Specifies a test data profile to use to generate test data for the column. Click the ellipsis button to the right of this field to access the List of Test Data Profiles (see <i>Populating Columns with Test Data</i> on page 109).
Computed Expression	Specifies an expression used to compute data for the column (see <i>Creating a Computed Column</i> on page 113).

The following tabs are also available:

- Standard Checks - specifies constraints on column data (see *Setting Data Profiling Constraints* on page 105).
- Additional Checks - provides an editable SQL statement, which can be used to generate more complex constraints (see *Specifying Advanced Constraints* on page 108).

Obtaining Column Statistics from your Database

You can enter values in the **Null values**, **Distinct values** and **Average length** fields on the **Detail** tab on your column property sheets, or obtain appropriate values from your database.

To obtain column statistics as part of your standard reverse engineering process, select the **Statistics** option in the Reverse Engineering dialog box (see *Reverse Engineering from a Live Database* on page 375).

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 384).

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	<p>Specifies the range of acceptable values. You can set a:</p> <ul style="list-style-type: none"> • 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 enter a default value or select a keyword from the list, which is defined in the <code>Script\Keywords\ReservedDefault</code> entry of the DBMS definition file.
Characteristics	<p>Specifies the shape of acceptable data. You can choose a:</p> <ul style="list-style-type: none"> • Format - A number of standard formats are available in the list and you can create your own format for reuse elsewhere or simply enter a format in the field. • Unit - A standard measure. This field is informational only and is not generated. • No space - Space characters are not allowed. • Cannot modify - The value cannot be updated after initialization.
Character case	<p>Specifies the acceptable case for the data. You can choose between:</p> <ul style="list-style-type: none"> • Mixed case [default] • Uppercase • Lowercase • Sentence case • Title case
List of values	<p>Specifies the various values that are acceptable.</p> <p>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.</p> <p>Select the Complete check box beneath the list to exclude all other values not appearing in the list.</p>

Note: When specifying strings in the list of values, quotation marks will be added around the values in the generated script. However, quotation marks will not be added if you surround the value by tilde characters, if the value is a keyword (such as NULL) defined in the DBMS, or if PowerDesigner does not recognize your data type as a string. Additional quotation marks will not be added if you have supplied them. The generation of single or double quotation marks depends on the target DBMS. The following examples show how various string values will be 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

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
%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 193) 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 concatenated 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

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

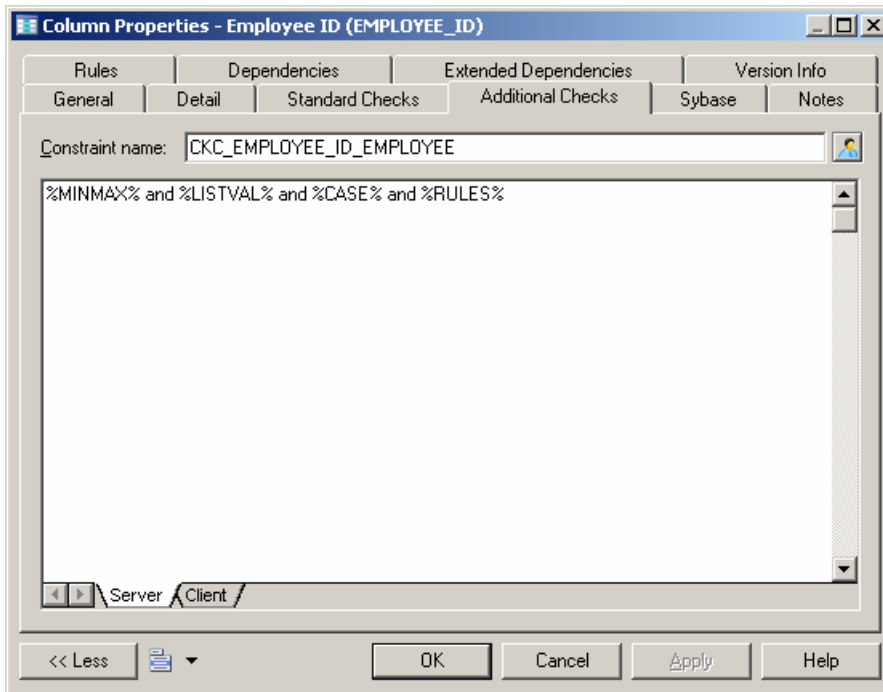
The General tab contains the following properties:

Property	Description
Name	Specifies the name of the item, which should be clear and meaningful, and should convey the item's purpose to non-technical users.
Code	Specifies the technical name of the object, which is used for generating code or scripts, which may be abbreviated, and should not generally include spaces.
Comment	Provides a description of the format.
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined.
Type	Specifies the type of the format. You can choose between: <ul style="list-style-type: none"> • 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.

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.

In addition, for columns, you can override the default Constraint name:



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 test character data that is 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.

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 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

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

The General tab contains the following properties:

Property	Description
Name	Specifies the name of the item which should be clear and meaningful, and should convey the item's purpose to non-technical users
Code	Specifies the technical name of the item used for generating code or scripts, which may be abbreviated, and should not generally include spaces
Comment	Specifies a descriptive label for the test data profile.
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined
Class	Specifies the kind of data to be generated from the profile. You can choose between: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • 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.

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	<p>Specifies the characters that can be generated (by default, all alphanumeric characters and spaces), separated by commas. You can specify:</p> <ul style="list-style-type: none"> • 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' <p>To allow any character, select the All checkbox.</p>
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	<p>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:</p> <ul style="list-style-type: none"> • 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.
Values	Specifies whether the values are to be generated randomly or sequentially.

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 148, 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. 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.
 - *Average Length* - [read only] Used for estimating the size of the database (see *Estimating Database Size* on page 366). 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.

Creating a Computed Column

A computed column is a column whose content is computed from values in other columns in the table. Computed columns are not available in all DBMSs.

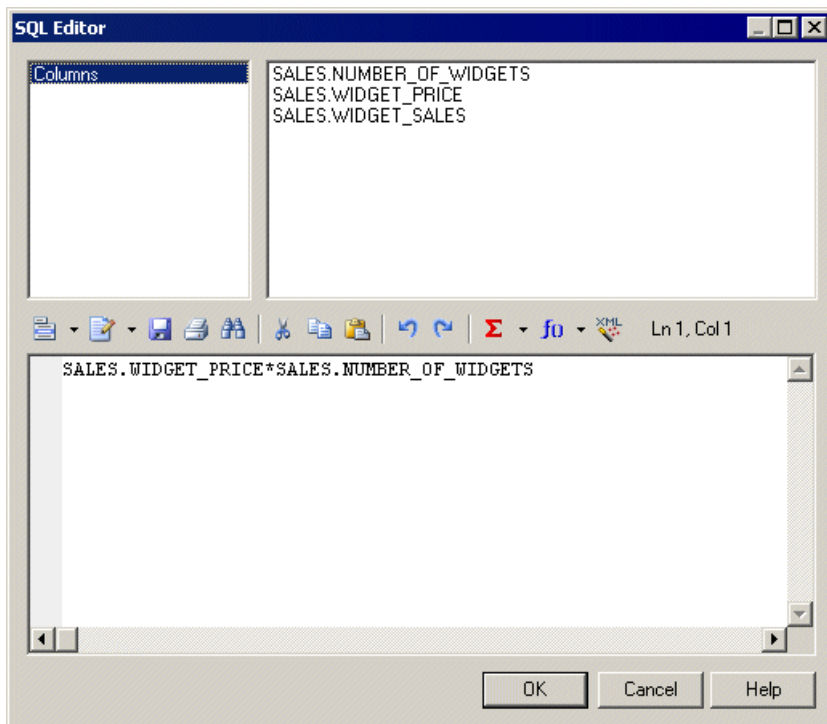
Simple computed expressions can be entered directly in the **Computed expression** field on the **Detail** tab of the column property sheet. For more complex expressions, click the Edit tool

to the right of the field to access the SQL Editor (see *Writing SQL Code in the PowerDesigner SQL Editor* on page 399).

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:

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

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.
4. Click the **Edit** tool to the right of the **Computed Expression** field to open the SQL Editor and enter the appropriate expression to compute the values for the column.



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

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

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

Selecting a Data Type for a Column

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

- When you attach the column to a domain - The domain dictates a data type, a length, and a level of precision, as well as optional check parameters
- When you manually select a data type - You also specify a length, a level of precision, and optional check parameters

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

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

The screenshot shows the 'Column Properties' dialog for a column named 'TOTAL SALES'. The 'General' tab is selected, and the 'Table' is set to 'HISTORY'. The 'Data type' is 'numeric(8,2)', with a length of 8 and a precision of 2. The 'Domain' is set to 'T_AMOUNT'. The 'Displayed' checkbox is checked, while 'Computed', 'Mandatory', 'Identity', 'Primary key', and 'Foreign Key' are unchecked. The dialog includes standard window controls and action buttons like 'OK', 'Cancel', 'Apply', and 'Help'.

3. Select a data type from the Data Type list or click the Question mark button to open and choose a data type from the Standard Data Types dialog box.
4. If required, enter a data type length and precision.

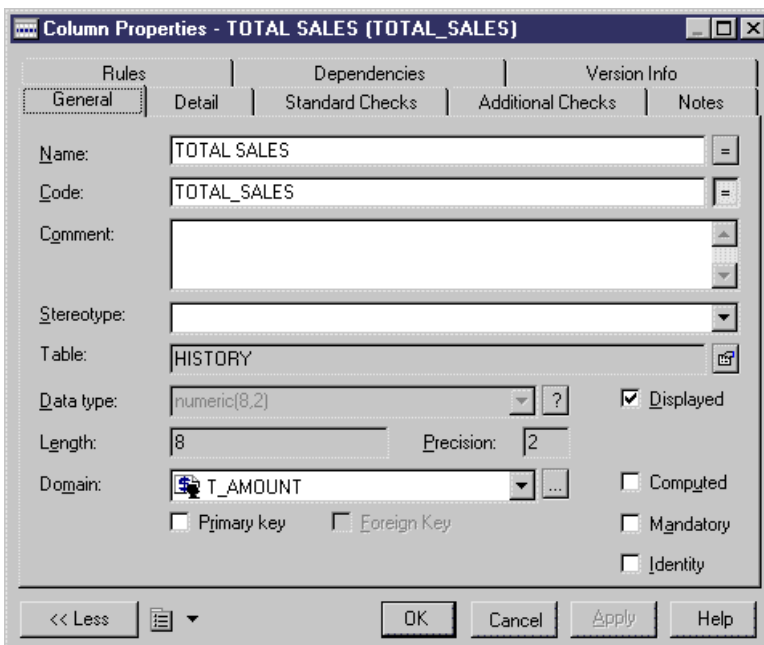
Note: If you do not want to select a data type immediately, you can choose the <undefined> data type. When you generate the database, this data type is replaced by the default data type for your database, as defined in the DBMS.

5. Click OK in each of the dialog boxes.

Attaching a Column to a Domain

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

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

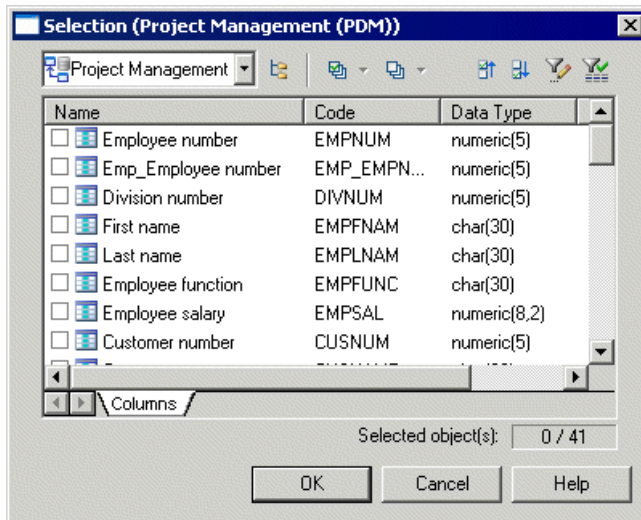


3. Select a domain from the Domain list and then click OK.

Copying a Column to Another Table

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

1. Double-click a table to open its property sheet, and click the Columns tab.
2. Click the Add Columns tool to open a selection box listing the columns attached to all other tables in the model.



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

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

4. Click OK.

Naming a Column Constraint

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

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

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

1. Open the property sheet of a column and click the Additional Checks tab.
2. Type changes to the constraint name in the Constraint Name box.

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

Note: You can always return to the default constraint name by clicking the User-Defined button.

3. Click OK in each of the dialog boxes.

Configuring the Display of the List of Columns

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

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

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

Each arrow type corresponds to the following list orders:

Arrow type	Listed order
Down arrow	Alphabetically
Up arrow	Reverse alphabetically

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

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

Keys (PDM)



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

The physical diagram supports the following types of keys:

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

Example

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

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

- TITLE_ID is the primary key and consists of the column TITLE ISBN which identifies each book title in the table

- `TITLE_NAME` is an alternate key containing the columns `TITLE NAME` and `TITLE TYPE`. It allows each title to be identified by its name and type. The fact that it is an alternate key indicates that there is a constraint that no two titles of the same type can have the same name

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

Creating a Key

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

- *Primary keys* on page 120
- *Alternate keys* on page 121
- *Foreign keys* on page 122

Key Properties

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

The `General` tab contains the following properties:


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

Primary Keys

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

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

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

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

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

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

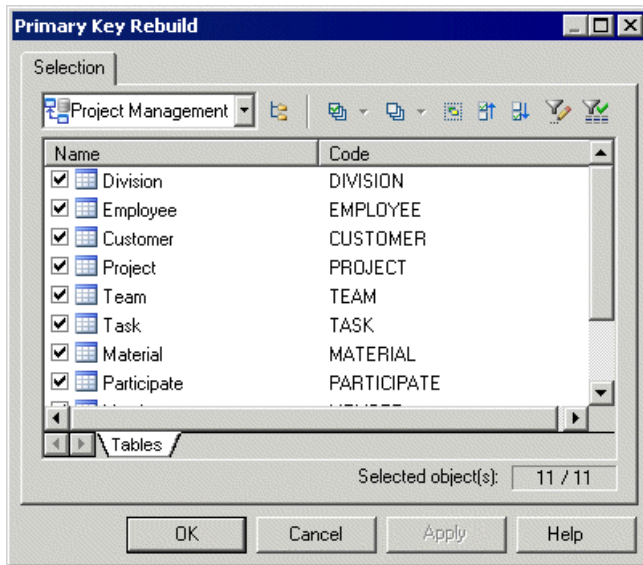
Rebuilding Primary Keys

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

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

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

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, select the Include Sub-Packages icon next to the list, and then select a sub-package from the dropdown list.

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

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

3. Click OK.

Alternate Keys

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

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

1. Double-click a table in the diagram to open its property sheet.
2. Click the Keys tab, which lists all the keys defined for the table.
3. Click the Add a Row tool, and type a name for the newly created key.

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

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

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

5. Click the property tool and confirm the object creation in order to open the property sheet for the new key.
6. Click the columns tab, which lists all the columns to which the key is attached. At key creation, the list is empty.
7. Click the Add Columns tool to open a selection box listing all the columns in the table (except those attached to the primary key).
8. Select one or more columns and click OK in each of the dialog boxes.

Foreign Keys

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

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

For information about the auto-migration of a foreign key, see *Automatic reuse and migration of columns* on page 169.

For more information on defining references, see *References (PDM)* on page 168.

Adding Parent Table Columns to a Key

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

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

Naming Key Constraints

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

A primary key constraint is a named check that enforces the uniqueness and the presence of values in a primary key column. If you do not specify a constraint name, PowerDesigner creates a default constraint name automatically.

1. Open the property sheet of the table and click the **Keys** tab, select a primary, alternate, or foreign key, and click the **Properties** tool to open its property sheet.

- Enter the required name in the **Constraint Name** box.

You can use the following variables in the constraint name:

Variable	Description
%TABLE%	Code of the table
%AK%	Code of the alternate key
%AKNAME%	Name of the alternate key
%REFRNAME%	Name of the reference
%REFRCODE%	Code of the reference
%PARENT%	Code of the parent table
%CHILD%	Code of the child table

For a complete list of PowerDesigner variables, see *Writing SQL using PDM Variables and Macros* on page 403.

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

- Click **OK** in each of the dialog boxes.

Indexes (PDM)

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

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

Example

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

Creating an Index

You can create the several types of index.

- A user-defined index - Associated with one or more columns
- An index linked to a key - Automatically updated when the key column or columns are modified. An index linked to a key is unique because it uses the same unique set of columns as the key.

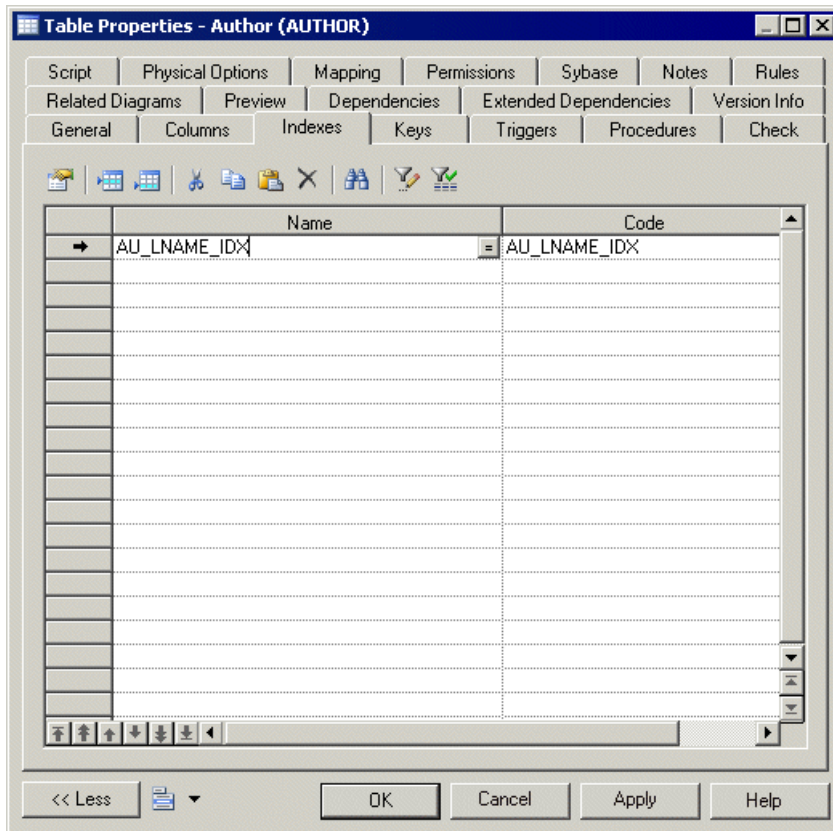
- A function-based index - [if supported by the DBMS] Precomputes the value of a function or expression based on one or more columns and stores it in the index. The function or the expression will replace the index column in the index definition. Function-based indexes provide an efficient mechanism for evaluating statements that contain functions in their WHERE clauses.

Use the following naming conventions for indexes:

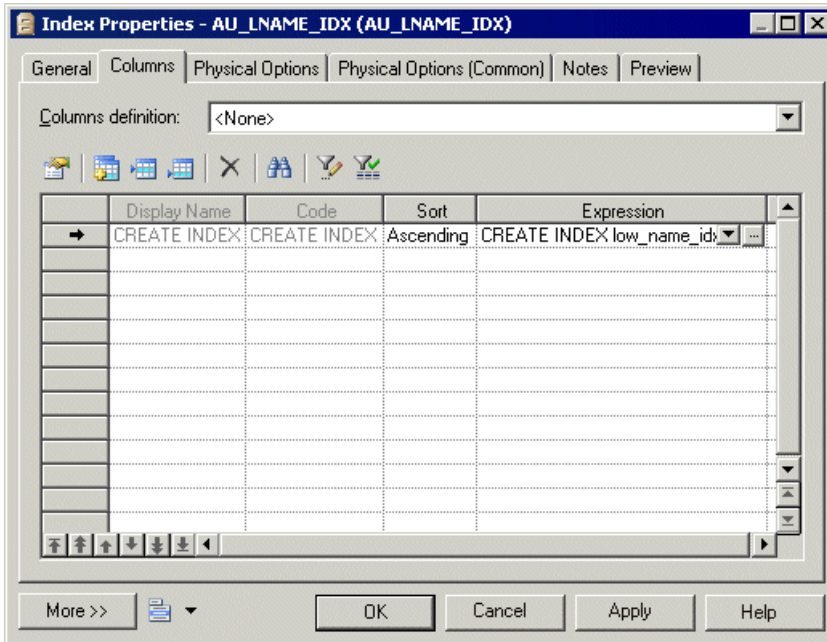
Index	Naming convention
Primary key	Table code followed by PK; for example EMPLOYEE _PK
Foreign key	Table code followed by FK; for example PROJECT _ FK
Alternate key	Table code followed by AK; for example EMPLOYEE _ AK

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

1. Double-click a table symbol to display its property sheet and click the Indexes tab.
2. Click the Add a Row tool and enter an index name and an index code.



3. Click the Properties tool to open the property sheet of the new index.
4. Type or select any appropriate index properties, and then click the Columns tab.
5. To:
 - Create a user defined index, click the Add Columns tool, select one or more columns from the list, and then click OK
 - Create an index linked to a key, select the primary key, an alternate key, or foreign key from the Columns definition list
 - Create a function-based index [if supported by the DBMS] ,click the Add a Row tool, then click in the Expression column and select the ellipsis button to open the SQL Editor. Enter an expression in the editor and then click OK



6. Select Ascending or Descending in the Sort column.
7. Click OK in each of the dialog boxes.

Reverse Engineering Function-based Index

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

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

```
'%SqlExpression.Xpr' || i.table_name || i.index_name ||
c.column_position || '%'
```

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

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

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

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

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

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

Index Properties

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

The General tab contains the following properties:

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

Property	Description
Type	<p>[Sybase® IQ, and Oracle only] Specifies the type of index. You can choose between:</p> <ul style="list-style-type: none"> • Bitmap – [Oracle] In a bitmap index, a bitmap for each key value is used instead of a list of row Ids • HG – [Sybase IQ] HighGroup indexes are used for GROUP BY, COUNT(DISTINCT) and SELECT DISTINCT statements when data has more than 1000 unique values • HNG – [Sybase IQ] HighNonGroup indexes make equality comparisons, SUM and AVG calculations very fast when data has more than 1000 unique values. Nonequality comparisons can also be done • LF – [Sybase IQ] LowFast indexes are used for columns that have a very low number of unique values. This index also facilitates join index processing. It is one of the two indexes allowed for columns used in join relationships • CMP – [Sybase IQ] Compare indexes are used for columns that store the binary comparison (<, >, or =) of any two distinct columns with identical data types, precision, and scale • WD – [Sybase IQ] Is used to index keywords by treating the contents of a CHAR or VARCHAR column as a delimited list
Unique	Specifies whether an index is a unique index
Cluster	<p>Specifies that the index is a clustered index. A table cannot have more than one clustered index.</p> <p>Note that clusters in Oracle 11 and higher are modeled as extended objects with a <<Cluster>> stereotype. For more information, see <i>DBMS-Specific Features</i> on page 5.</p>

The following tabs are also available:

- Columns - lists the columns with which the index is associated (see *Creating an index* on page 123).

Rebuilding Indexes

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

The following options are available when rebuilding indexes:

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

Option	Description
Foreign key indexes	Rebuilds foreign key indexes. The text box shows the naming convention for foreign keys. By default this is %REFR%_FK
Foreign key threshold	Specifies the minimum number of estimated records in a table that are necessary before a foreign key index can be created. The estimated number of records is defined in the Number box in the table property sheet. If the table has no specified number of occurrences, the foreign key indexes are generated by default
Mode	Specifies the extent of the rebuild. You can select: <ul style="list-style-type: none"> • Delete and Rebuild – deletes and rebuilds all indexes presently attached to primary, alternate, and foreign keys • Add missing indexes – preserves all indexes presently attached to primary, alternate, and foreign keys and adds any that are missing

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

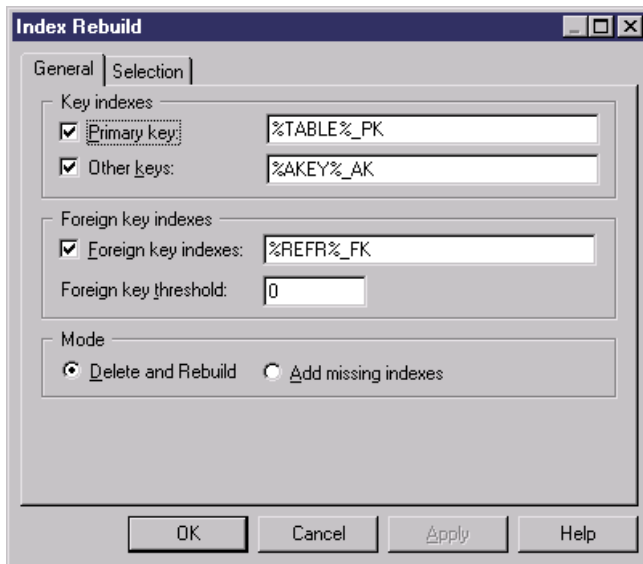
Variable	Value
%TABLE%	Generated code of the table. This is the table code generated in the database. It may be truncated if the code contains characters not supported by the DBMS
%TNAME%	Table name
%TCODE%	Table code
%TLABL%	Table comment

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

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

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

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



2. Set the appropriate options.
3. [optional] Click the Selection tab to specify which tables you want to rebuild indexes for.
4. Click OK. If you selected the Delete and Rebuild mode, a confirmation box asks you to confirm your choice. Click Yes to confirm the deletion and rebuild of the selected references.

Indexes in Query Tables

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

For more information about query tables, see *Creating a query table* on page 186.

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 511).

Creating a User, Group, or Role

You can create users, groups, and roles in any of the following ways:

- 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 package in the Browser, and select **New > Type**.

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

User, Group, and Role Properties

You can modify an object's properties from its property sheet. To open a user, group, or role property sheet, double-click its Browser entry in the Users, Groups, or Roles folder.

The General tab contains the following properties:

Property	Description
Name	Name of the user, group, or role.
Code	Code of the user, group, or role used as an identifier in the database.
Comment	Description of the user, group, or role.
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined.
Password	[users and groups] Password used for database connection.

The following tabs are also available:

- Privileges - lists the system privileges granted to the user (see *Granting System Privileges* on page 133).

- Permissions - lists the operations that the user is permitted to perform on various database objects (see *Granting Object Permissions* on page 136).
- 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.

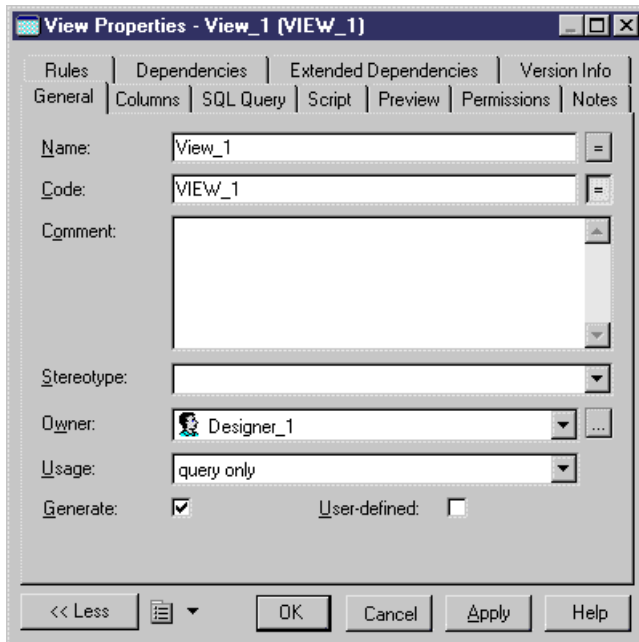
Assigning an Owner to an Object

In a database, the user who creates an object (tables, views, stored procedures, etc) is the owner of the object and is automatically granted all permissions on it.

When building a PDM, you must attach the user to the object in order to make it the owner. Each object can have only one owner. In a model where many users have access to the same objects, you can restrict object modifications to the owner and define permissions for the other users.

Owners can also be used during generation: when you select to generate for a selected owner, only the tables belonging to this owner are generated, whereas when you generate as ADMIN, you generate all the tables on behalf of all their owners.

1. Open the property sheet of the object.
2. Select a user in the Owner list. You can create a new user by clicking the ellipsis button to the right of the Owner list.



3. Click OK.

Specifying Default Owners for Object Types

You can specify a default owner for each type of object that supports the concept of ownership. The default owner will automatically be linked to all the objects of this type that you create after making this change.

1. Select **Tools > Model Options** and then select Table and View in the left-hand pane.
2. Select a user in the Default owner list in the Table groupbox. You can create a new user by clicking the ellipsis button to the right of the Default owner list.
3. Click OK.

For more information, see *Setting PDM Model Options* on page 16.

Granting System Privileges

A system privilege is a set of rights assigned to a database user, group, or role. You use system privileges to create user profiles with different levels of influence over the database content. The procedure for defining privileges is identical for users, groups, and roles.

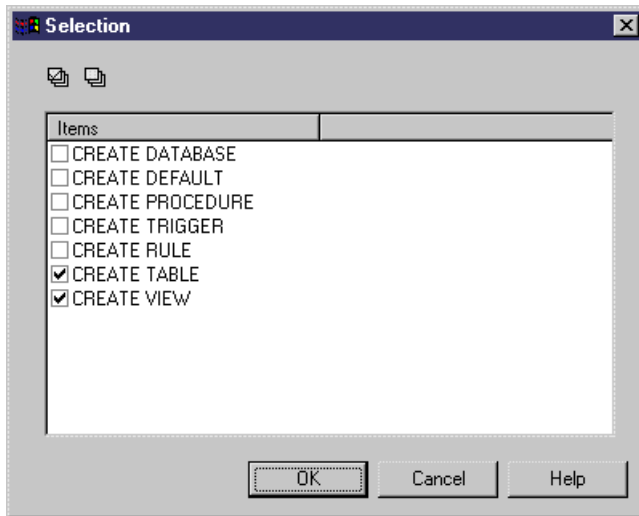
System privileges are used in association with object permissions (see *Granting Object Permissions* on page 136) to evaluate the rights of a user, group, or role. For example, even if a user has the modify privilege, he cannot modify an object on which he has no update permission.

System privileges are *granted* to a user. A user with an administrative profile is also allowed to *revoke* a privilege. By default, a user belonging to a group or having a role inherits the group or role privileges and inherited privileges appear in the **Privileges** tab of the user property sheet.

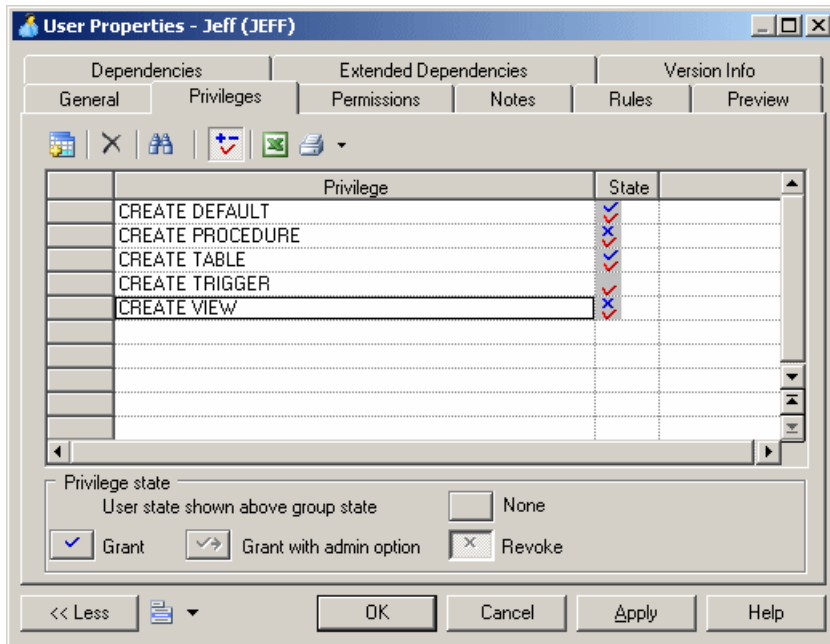
System privileges vary according to the DBMS you are using. The list of privileges also includes predefined roles (like connect, or resource) for an easier use. In some DBMS, system privileges are called permissions. In this manual, the term privilege is used for any right granted to a user, a group, or a role. Permissions are defined for objects.

Note: To review and edit the list of available privileges in the Resource Editor, 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.

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 display in 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:



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:
- **Grant** – [default] Assigns the privilege to the user.
 - **Grant with admin option** - Assigns the privilege to the user, and allows the recipient to pass on the privilege to other users, groups, or roles. For example, you assign the CREATE TABLE privilege for user Designer_1 and then click the Grant With Admin Option button to permit Designer_1 to grant this privilege to other users.
 - **Revoke** – Revokes the privilege inherited from a group or role for the current user or group.
 - **None** - Cancels any state and cleans up the current cell.



The following table summarizes the different privilege combinations:

Privilege combination	Description
<input checked="" type="checkbox"/>	Privilege granted to user
<input checked="" type="checkbox"/>	Privilege inherited from group
<input checked="" type="checkbox"/>	Privilege inherited from group and revoked to user
<input checked="" type="checkbox"/>	Privilege inherited from group overloaded by "with admin option"

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 346.

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 350.

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 give users the right to perform operations on particular database objects. The procedure for defining permissions is identical for users, groups, and roles.

System privileges are used in association with object permissions (see *Granting System Privileges* on page 133) to evaluate the rights of a user, group, or role.

PowerDesigner allows you to define permissions on tables, views, columns, procedures, packages, and other objects depending on your DBMS. Some or all of the following may be available:

Permission	Description
Select	To observe information contained in object
Insert	To insert rows into object
Alter	To alter table with ALTER TABLE command
Delete	To delete rows from object
References	To create indexes in tables and foreign key referencing tables
Update	To update row in object
Index	To create an index with the CREATE INDEX command
Execute	To execute procedure or function

For more information on the permissions allowed in your DBMS, see your DBMS documentation.

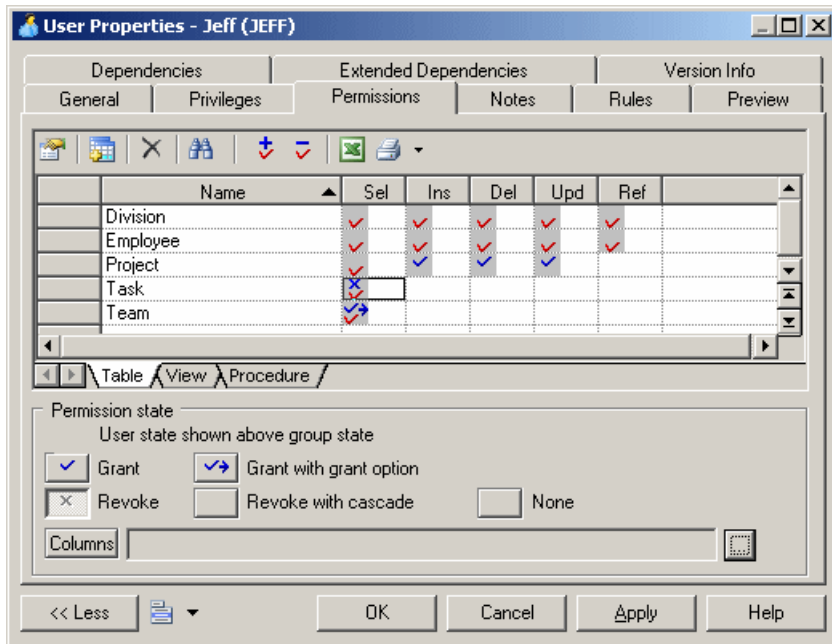
Note: The owner of an object (see *Assigning an Owner to an Object* on page 132) automatically has permission to carry out any operation on that object. These permissions do not appear in the **Permissions** tab of the object property sheet but they are implemented during generation and reverse engineering.

1. Open the property sheet of a user, role, or group, and click the **Permissions** tab. The columns in the list show the permissions available for a given type of object in the current

DBMS. A sub-tab is displayed for each type of object supporting permissions in the current DBMS.

Note: You can also assign permissions to an object from the **Permissions** tab of the object property sheet. In this case, there are sub-tabs listing the users, roles, and groups who have permissions on the object. This tab lets you see all the permissions granted for the object, while the **Permissions** tab in the property sheet of a user, role, or group lists all the objects for which it has permissions.

2. Click the **Add Objects** tool to add one or more of objects of the present type in the model, and click **OK** to add them to the list. If the user belongs to a group with permissions on the selected 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 display in 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:
 - **Grant** – Assigns the permission to the user.
 - **Grant with admin option** - Assigns the permission to the user, and allows the recipient to pass on the permission to other users, groups, or roles.
 - **Revoke** – Revokes the permission inherited from a group or role for the current user or group.
 - **Revoke with cascade** – Revokes the permission inherited from a group or role for the current user or group and revokes any permission granted by the user.
 - **None** - Cancels any state and cleans up the current cell.



The following table summarizes the available permission combinations:

Permission combination	Description
	Permission granted to user
	Permission inherited from group
	Permission granted to group and revoked to user
	Permission granted to group and overloaded by "with admin option"
	Permission granted to group and revoked with cascade to user

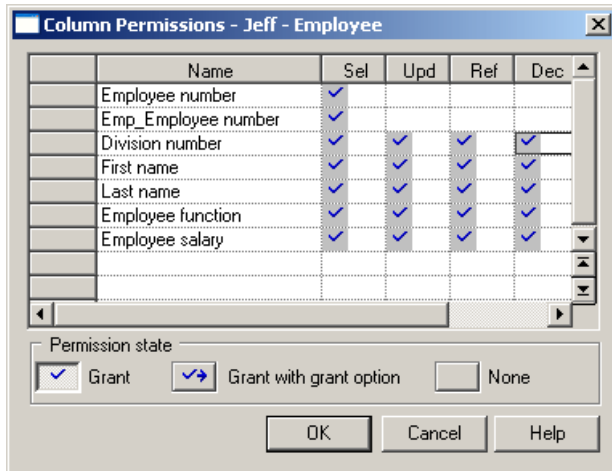
- [optional] For tables, you can specify permissions on individual columns (see *Defining Column Permissions* on page 138).
- When the permissions are correct, click **OK** to close the property sheet and return to the model.

Defining Column Permissions

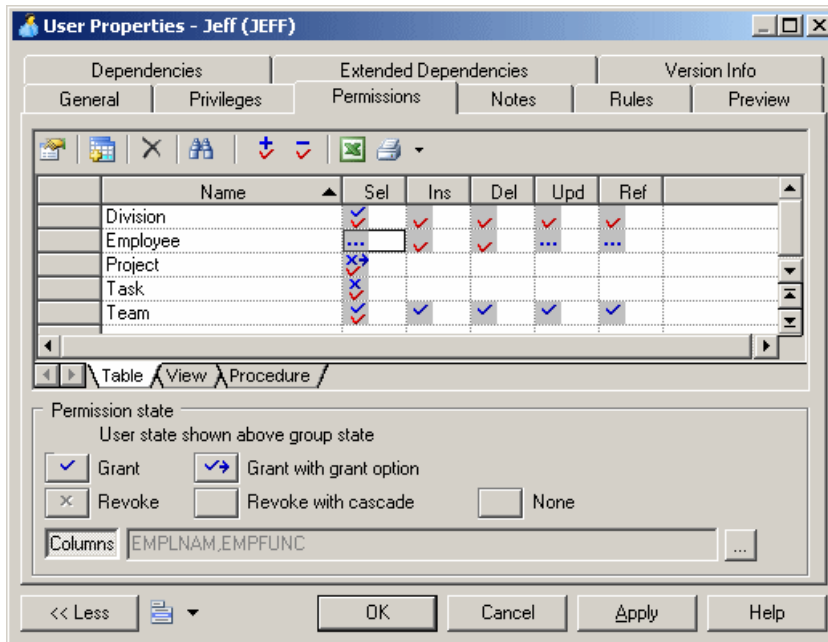
You can fine tune the permissions on a table by specifying permissions on a column-by-column basis. The available column permissions are specified in the DBMS resource file.

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:

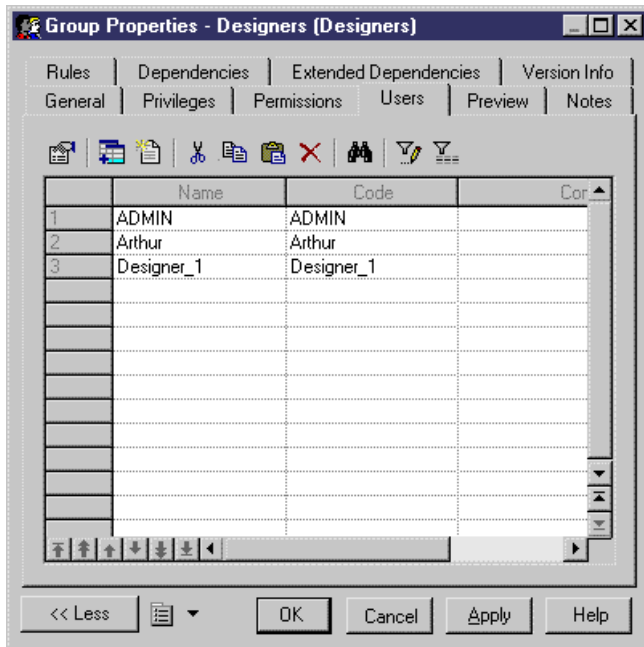


5. Click **OK** to close the property sheet and return to the model.

Inserting a User into a Group

Once you have created a group, you can insert users into it.

1. Select **Model > Users and Roles > Groups** to open the List of Groups.
2. Select a group in the list, click the Properties tool to open its property sheet and then click the Users tab.
3. Click the Add Objects tool to open a selection box listing the users available in the model.
4. Select one or more users and click **OK** to insert these users into the group.

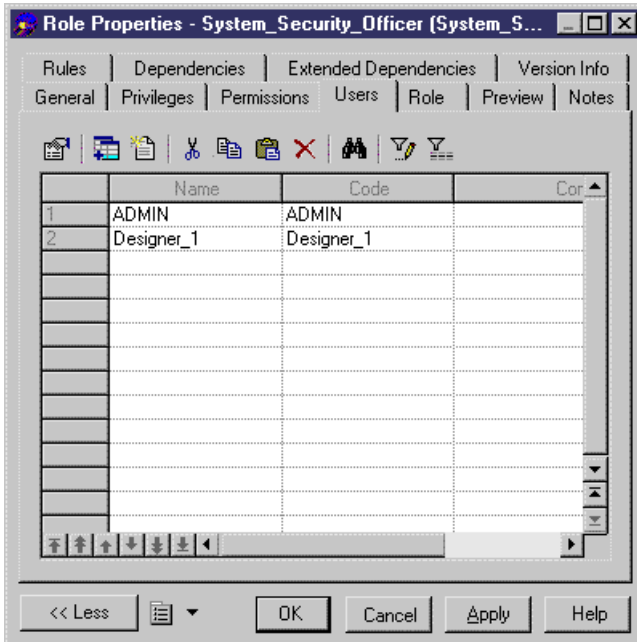


5. Click OK.

Assigning a User to a Role

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

1. Select **Model > Users and Roles > Roles** to open the List of Roles.
2. Select a role in the list, click the Properties tool to open its property sheet and then click the Users tab.
3. Click the Add Objects tool to open a selection box listing the users available in the model.
4. Select one or more users and click OK to assign these users to the role.



5. Click OK.

Synonyms (PDM)

A synonym is an alternative name for various types of objects (table, view, sequence, procedure, function, synonym or database package).

Synonyms are created to:

- Mask the name and owner of an object
- Provide location transparency for remote objects of a distributed database
- Simplify SQL statements for database users

For example, table SALES_DATA is owned by user JWARD. A standard select statement on this table would be:

```
SELECT * FROM jward.sales_data
```

The database administrator can create a synonym for this table and owner and call it SALES. In this case, the SQL statement is simplified in the following way:

```
SELECT * FROM sales
```

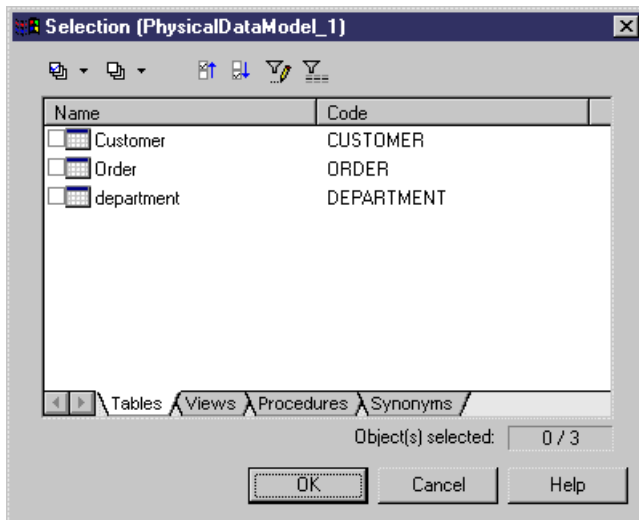
In PowerDesigner synonyms are created for *base objects*. Base objects support multiple synonyms while each synonym can have only one base object. You can view the synonyms depending on a base object in the Dependencies tab of the base object property sheet. If you delete the base object of a synonym, the synonym is deleted as well.

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.

Creating a Synonym

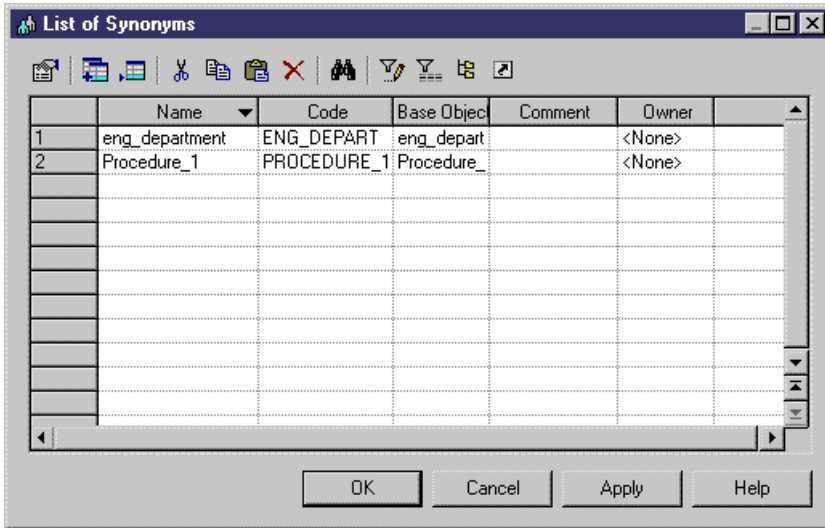
You can create a synonym as follows:

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.

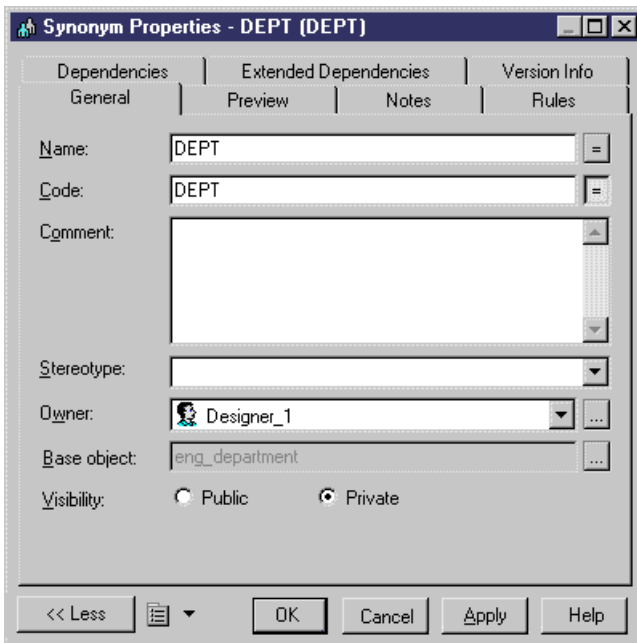


3. Select one or more objects and click OK.

Synonyms for each of the selected objects are created in the List of Synonyms. By default, a synonym has the same name as its base object. If the Base Object column is not shown in the list, click the Customize Columns and Filter tool, select Base Object in the list of available columns, and click OK.



- Click in the Name column of one of the rows and enter a new name for the synonym. Alternatively, you can click the Properties tool to open the property sheet of the synonym and edit its name and other properties there.



- Click OK in each of the dialog boxes.

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

Synonym Properties

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

The General tab contains the following properties:

Property	Description
Name	Name of the synonym (usually corresponding to a simplified name of the base object)
Code	Code of the synonym
Comment	Additional information about the synonym
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined
Owner	Name of the synonym owner. You choose an owner from a list of users. A synonym can only have one owner at a time
Base Object	Name of the object origin of the synonym. The Ellipsis button displays a selection dialog box that lets you select objects among all the models opened in the current workspace and belonging to the same DBMS family as the current DBMS
Visibility	Allows to define a synonym as public (accessible to all database users) or private (available to a specific user)
Type	For those DBMS that support it (for example DB2) you can create an alias instead of a synonym. In PowerDesigner synonyms and aliases are managed in the same way whereas their behavior in the database may be different

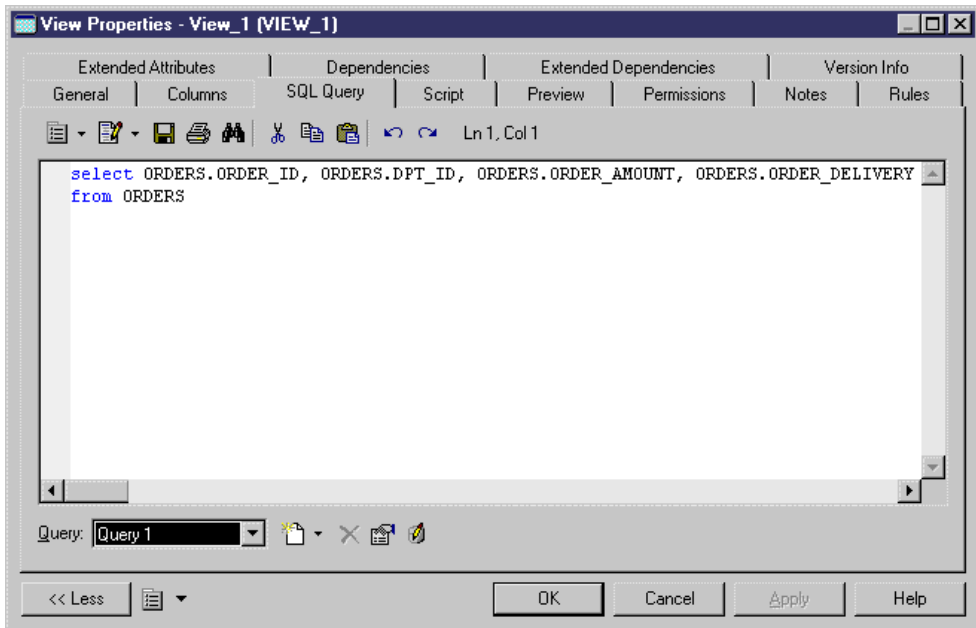
For more information on aliases, see the DB2 documentation.

Creating a View for a Synonym

You can create views for synonyms in the same way as you create views for tables. The view query displays the content of the object used for the synonym. For example, the ORDERS_PROD_DEPT table has a synonym ORDERS:

ORDERS_PROD_DEPT	
Order_ID	<pk>
Dpt_ID	
Order_Amount	
Order_Delivery	

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



1. Ensure that no objects are selected in the diagram and select **Tools > Create View** to open a selection box listing all the available objects in the model.
2. Click the Synonyms tab and select one or more synonyms to add to the view.
3. Click OK. The view is created in the diagram.

For more information about creating views, see *Views (PDM)* on page 179.

Defaults (PDM)

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

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

Example

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

Creating a Default

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

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

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

Default Properties

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

The General tab contains the following properties:

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

To view the default creation statement, click the **Preview** tab. For example:

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

Assigning a Default to a Column or a Domain

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

For more information on check parameters, see *Check Parameters (CDM/LDM/PDM)* on page 105.

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

For more information, see section *Rebuilding defaults* on page 148.

1. Open the property sheet of a column or a domain, and click the Standard Checks tab.
2. Select a default in the Default list in the Value groupbox.

Alternatively, you can type a default value in the listbox; this does not create a default object in the model, it only assigns a default value for the current column or domain. If you

type a name that already exists in the list, the default object is attached to the column or domain.

3. Click OK in each of the dialog boxes.

Rebuilding Defaults

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

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

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

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

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

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

1. Select **Tools > Rebuild Objects > Rebuild Defaults** to open the Default Rebuild dialog box.
2. Specify a default name template in the Domain and Column boxes.
3. [optional] Select the Reuse default with identical value check box – this option will reuse default objects with identical value among columns and domains. If you do not select this option, rebuild creates one default per object.
4. [optional] Select the Delete and rebuild check box – this option detaches the default objects attached to selected objects and deletes them if they are not used. If you select all objects, this option allows you to clean up the model from all existing defaults and recreate new default objects.
5. [optional] Click the Selection tab to specify domains and tables for default generation.
6. Click OK.

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

Domains (CDM/LDM/PDM)

Domains help you identify the types of information in your model. They define the set of values for which a column/entity attribute is valid. Applying domains to columns/entity

attributes makes it easier to standardize data characteristics for columns/entity attributes in different tables/entities.

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

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

Creating a Domain

You can create a domain as follows:

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

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

Domain Properties

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

The General tab contains the following properties:

Property	Description
Name	Specifies the name of the item, which should be clear and meaningful, and should convey the item's purpose to non-technical users
Code	Specifies the technical name of the object, which is used for generating code or scripts, which may be abbreviated, and should not generally include spaces
Comment	Specifies a descriptive label for the domain
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined
Owner	[PDM only] Specifies the name of domain owner. You choose an owner from a list of users. A domain can only have one owner at a time. This is normally the domain creator
Data type	Specifies the form of the data corresponding to the domain, such as numeric, alphanumeric, Boolean, or others. The <undefined> data type indicates a domain without a data type. If an <undefined> data type is present when you generate your database, it is replaced by the default data type for your database

Property	Description
Length	<p>[where appropriate] Specifies the maximum number of characters. In the Phys-Data Type list of available data types (select Database > Edit current database > Script > Data Type > PhysData Type), a variable indicates where you have to type a length or precision, as follows:</p> <ul style="list-style-type: none"> • %n - length • %s - length with precision • %p - decimal precision <p>For example, if you are using Sybase Adaptive Server Anywhere and you choose the data type</p> <pre>char (%n)</pre> <p>, you can choose a length of ten by typing</p> <pre>char (10)</pre> <p>.</p>
Precision	[where appropriate] Specifies the number of places after the decimal point, for data values that can take a decimal point
Mandatory	Specifies that domain values are mandatory for all columns/entity attributes using that domain
Identity	[where supported] When selected, indicates that the data is auto-incremented for columns using that domain
With default	[PDM only] (For those DBMS that support it). When selected, indicates if a default value is assigned to a column using the domain, when a Null value is inserted
Profile	[PDM only] Specifies the test Data profile assigned to the domain

The following tabs are also available:

- Standard Checks - contains checks which control the values permitted for the column/entity attribute (see *Setting Data Profiling Constraints* on page 105)
- Additional Checks - allows you to specify additional constraints (not defined by standard check parameters) for the column/entity attribute.
- Rules - lists the business rules associated with the column/entity attribute (see *Business Rules (CDM/LDM/PDM)* on page 193).

List of Standard Data Types

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

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:

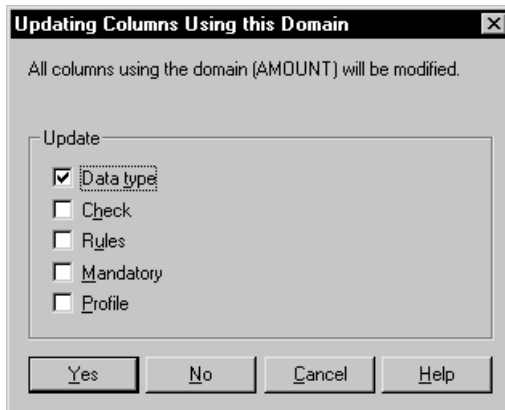
Standard data type	DBMS-specific physical data type	Content	Length
Binary	binary / RAW	Binary strings	Maximum
Long Binary	image / BLOB	Binary strings	Maximum
Bitmap	image / BLOB	Images in bitmap format (BMP)	Maximum
Image	image / BLOB	Images	Maximum
OLE	image / BLOB	OLE links	Maximum
Other	—	User-defined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

Cascading Updates to Columns/entity Attributes Associated with the Domain

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

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

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



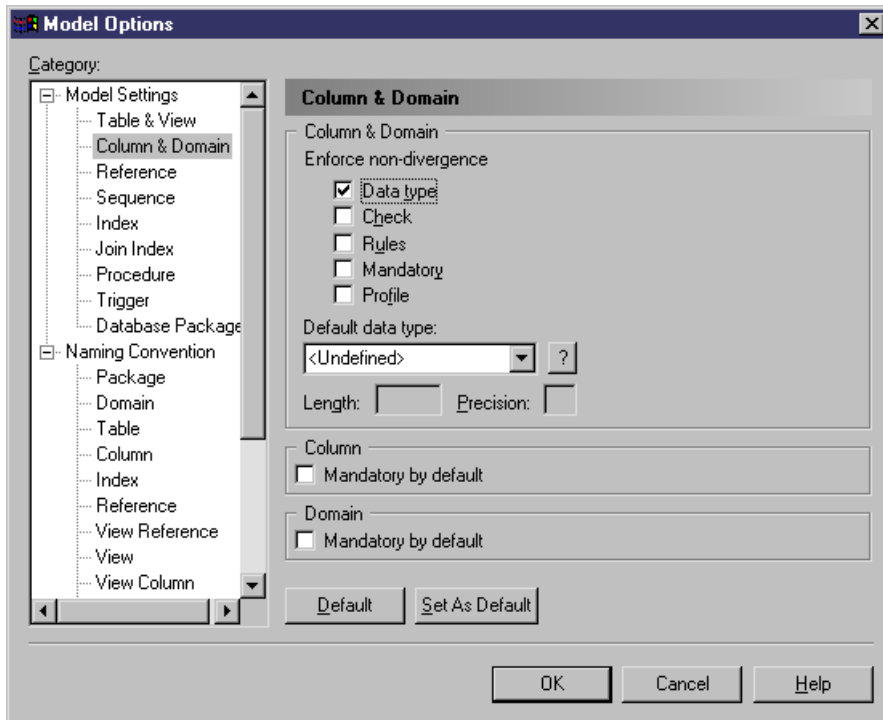
The Data Type check box is selected or not according to the options set to enforce non-divergence from a domain (see *Enforcing non-divergence from a domain in a data model* on page 154).

3. Select any other properties that you want to update (Check, Rules, Mandatory, Profile) for all columns/entity attributes using the domain.
4. Click one of the following buttons:
 - Yes - The columns/entity attributes currently using the domain are modified according to the update
 - No - The columns/entity attributes currently using the domain are not modified according to the update but the current modification is accepted if domain divergence is allowed in the model options (see *Enforcing non-divergence from a domain in a data model* on page 154).
 - Cancel - The update is cancelled and nothing is changed

Enforcing Non-divergence from a Domain in a Data Model

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

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



2. Select the check boxes of the column/entity attribute properties that are not permitted to diverge from the domain definition. You can specify any or all of:
 - Data type - data type, length, and precision
 - Check - check parameters such as minimum and maximum values
 - Rules – business rules
 - Mandatory – mandatory property of the column
 - [PDM only] Profile - test data profile

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

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

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

Sequences (PDM)

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

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

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

For more information on data types, see *Selecting a data type for a column* on page 115.

Example

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

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

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

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

Creating a Sequence

There are two steps to using sequences:

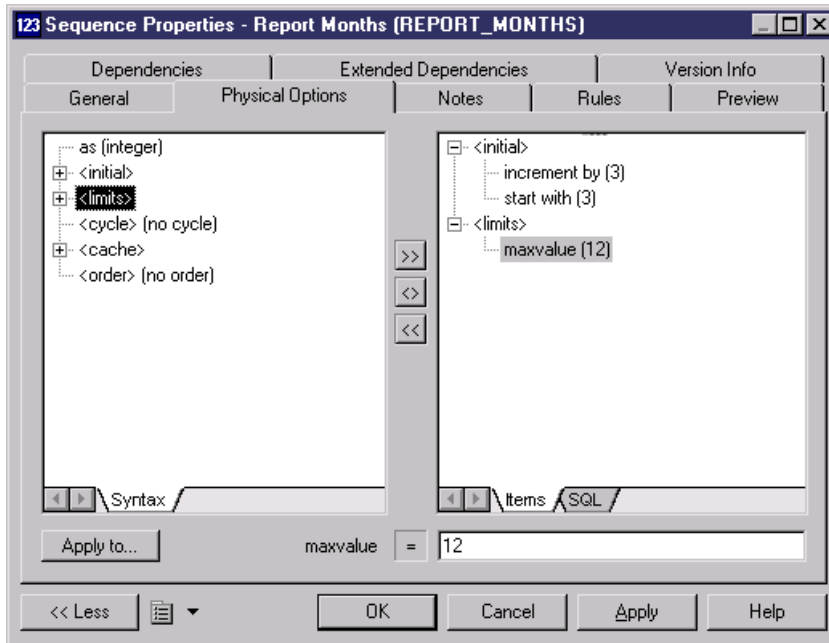
- Create a sequence (including defining sequence options)
- Apply and enable a sequence to a column

Creating a Sequence and Specifying its Physical Options

You create a sequence and specify its physical options as follows:

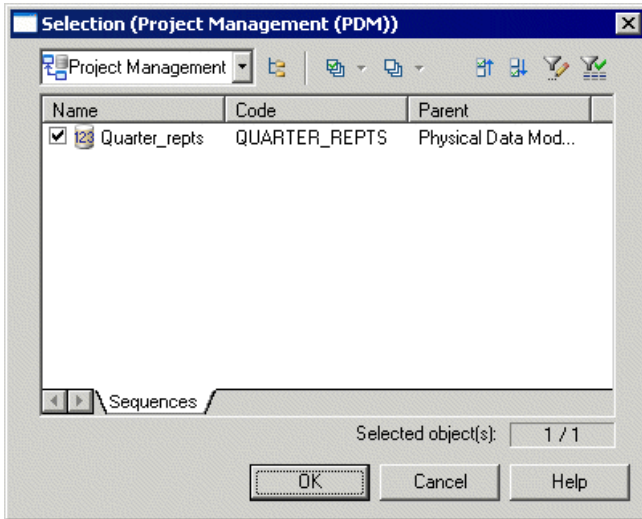
1. Select **Model > Sequences** to open the List of Sequences.

2. Click the Add a Row tool and type a name for the new sequence.
3. Double-click the arrow to the left of the new sequence to display its property sheet.
4. Click the Physical Options tab and enter any appropriate parameters. These options are DBMS-specific. For more information on using this tab, see *Physical Options* on page 301.



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

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



6. Click OK in each of the dialog boxes.

Applying and Enabling a Sequence on a Column

You apply and enable a sequence on a column as follows:

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

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

For more information on rebuilding triggers, see *Chapter 5, Building Triggers and Procedures* on page 221.

Sequence Properties

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

The General tab contains the following properties:

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

The following tabs are also available:

- Physical Options - lists the physical options associated with the sequence (see *Physical Options* on page 301). For information about these options, see your DBMS documentation.

Changing the DBMS of a Model which Contains Sequences and Auto-incremented Columns

If you create an auto-incremented column or a sequence attached to a column, and then decide to change the target DBMS various transformations will be effected.

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

Sequences and Intermodel Generation

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

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

Abstract Data Types (PDM)

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

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

Abstract data type is	Description
Created	You can create an abstract data type of any kind supported by your DBMS. If you create an abstract data type of type JAVA, you can link it to a Java class in an OOM to access the Java class properties (see <i>Linking an abstract data type to a Java class</i> on page 164).
Reverse engineered	An abstract data type in a database can be reverse engineered into a PDM. If you also reverse engineer the JAVA classes into an OOM, then the abstract data types of the type JAVA in the PDM are automatically linked to the Java classes in the OOM (see <i>Reverse-engineering a PDM linked to an OOM</i> on page 167)

For more information on reverse engineering a database into a PDM, see *Reverse Engineering a Database into a PDM* on page 373.

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

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

Type	Description	Example
Array	Fixed length collection of elements	VARRAY (Oracle 8 and higher)

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

Example

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

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

Creating an Abstract Data Type

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

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

See also *Creating object and SQLJ object abstract data types* on page 162.

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

Abstract Data Type Properties

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

The General tab contains the following properties:

Property	Description
Name	Specifies the name of the item, which should be clear and meaningful, and should convey the item's purpose to non-technical users

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

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

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

Creating Object and SQLJ Object Abstract Data Types

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

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

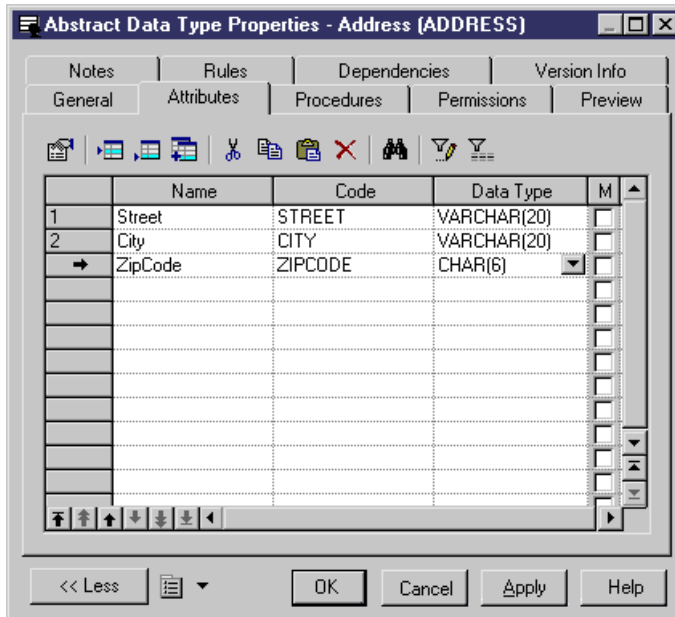
An object abstract data type with a supertype can inherit non-final procedures. You can use the Inherit Procedure tool in the Procedures tab of the abstract data type to select a non-final procedure from a parent abstract data type. Inheritance only applies to non-final procedures.

In the following example, you want to create an Address object with Street, City, and ZipCode attributes, and a Location procedure:

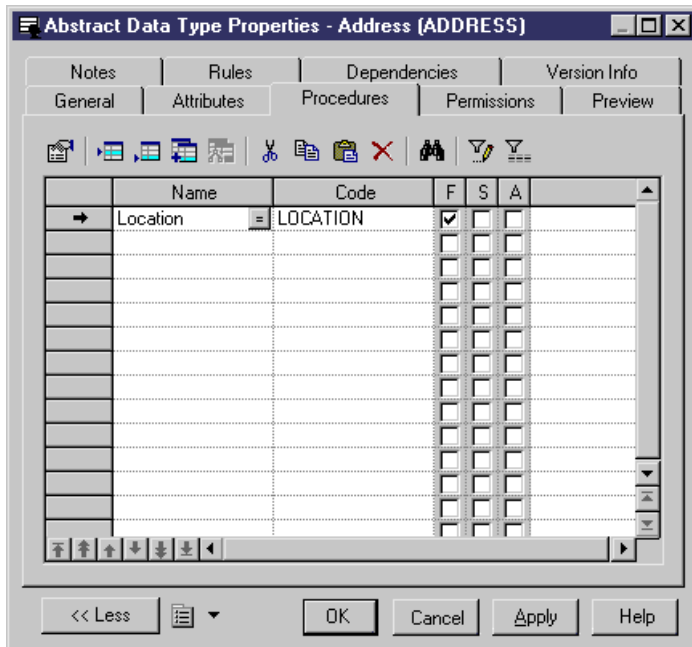
1. Open the property sheet of the abstract data type and select either OBJECT or SQLJ_OBJECT from the type list.

The Attributes and Procedures tabs are displayed.

2. Click the Attributes tab.



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



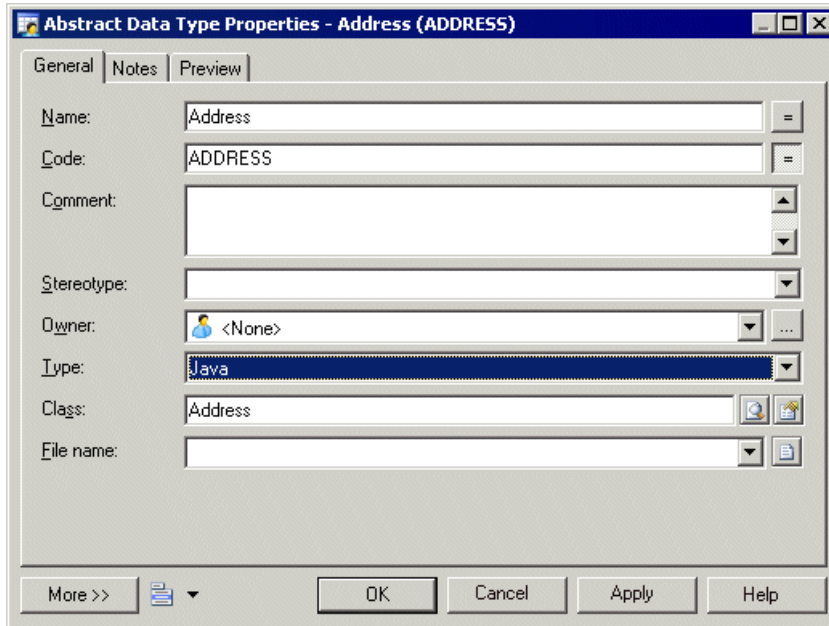
5. For each procedure, click the Add a Row tool, and:
 - enter a Name and Code
 - [optional] Select the Final (F), Static (S) and/or Abstract (A) columns
6. Click OK in each of the dialog boxes.

Linking an Abstract Data Type to a Java Class

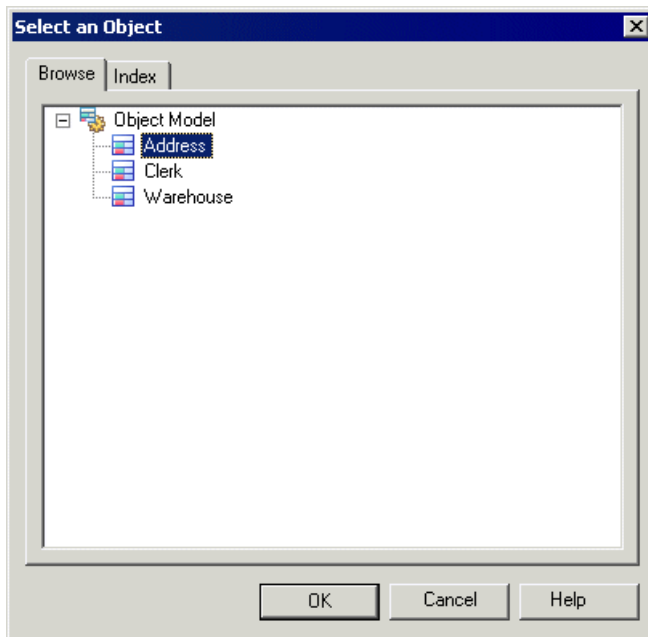
You can specify a Java class in the PDM, and then link it to a Java class in an OOM. The OOM must be open in the current Workspace to be available for linking.

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

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



2. Click the Ellipsis button to the right of the Class box to open the Select an Object window, which lists all the Java classes that are available in the OOMs currently open in the Workspace.

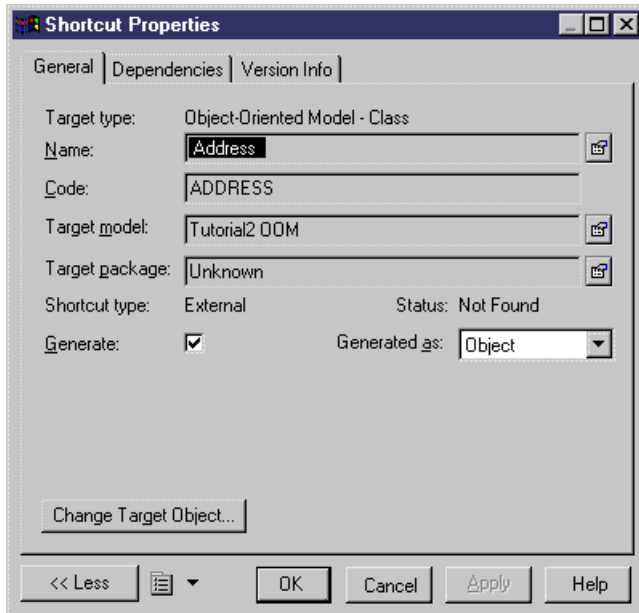


3. Select a Java class and click OK.

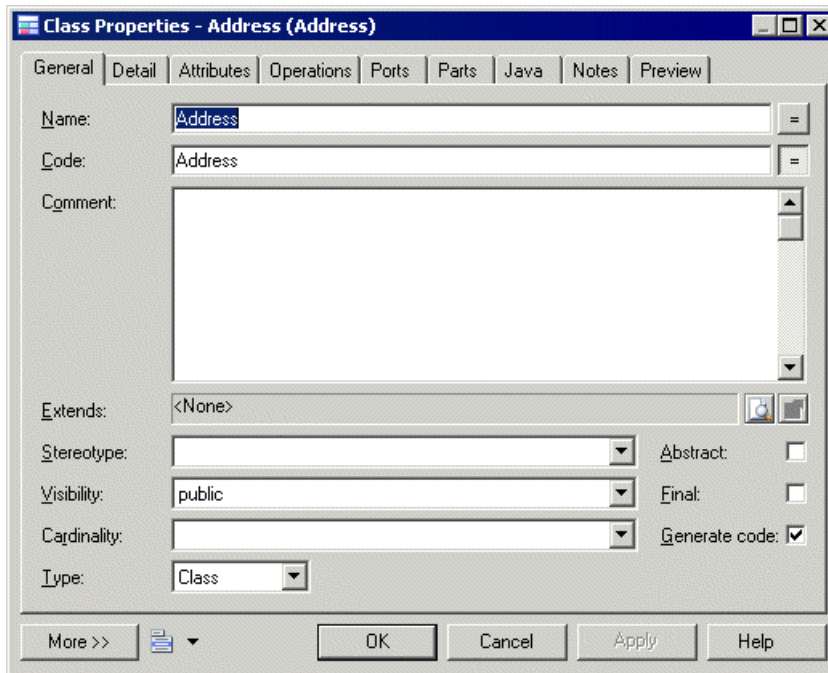
The abstract data type is now linked to the Java class, and the class name is displayed in the Class box.

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

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



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



5. Click OK in each of the dialog boxes.

Reverse-engineering a PDM Linked to an OOM

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

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

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

You can access the properties of these Java classes from the property sheets of the corresponding abstract data types in the PDM. For more information, see *Linking an abstract data type to a Java class* on page 164.

References (PDM)

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

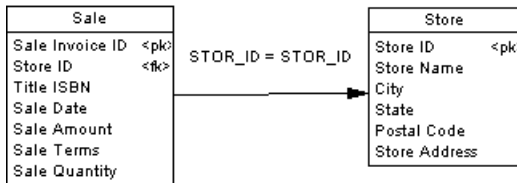
When column pairs are linked by a reference, each value in the child table column refers to an equivalent value in the parent table column.

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

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

Example

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



Creating a Reference

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

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

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

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

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

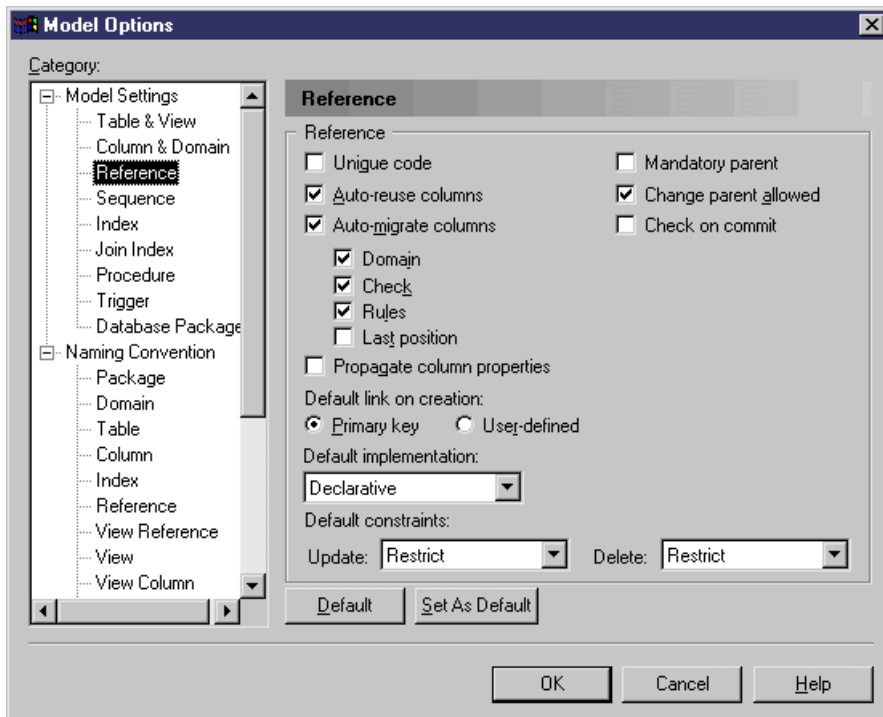
For general information about creating objects, see the Objects chapter in the *Core Features Guide*.

Automatic Reuse and Migration of Columns

When you create a reference, PowerDesigner can automatically:

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

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



3. Specify your choices for column reuse and migration as follows:
 - To auto-reuse existing columns in child tables as foreign key columns when creating references - select the Auto-reuse columns check box. Note that the column in the child table must have the same code as the migrating primary key column, and cannot

already be a foreign key column for it to be suitable for reuse. If you want to reuse a child table column that is already a foreign key column, you must do this manually from the Joins tab of the reference property sheet.

- To auto-migrate primary key columns in parent tables for use as foreign key columns in child tables - select the Auto-migrate columns check box. This will also enable the column properties check boxes, allowing you to specify which of the parent column properties to migrate.
 - Domains
 - Check (check parameters)
 - Rules (business rules)
 - Last position (migrated columns should be added at the end of the table column list. If the Last position option is not selected, migrated columns are inserted between key columns and other columns which implies that a child table must be dropped and recreated each time you add a reference and modify an existing database.)

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

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

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

Examples

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

Matching Child Table Column Exists

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

Table_1
Col_1 <pk>
Col_2 <pk>
Col_3

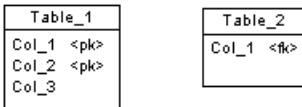
Table_2
Col_1

Auto-reuse	Auto-migrate	Result	Description of child table
Selected	Selected		Col_1 is reused and Col_2 is created

Auto-reuse	Auto-migrate	Result	Description of child table
Not selected	Selected		T1_Col_1 is created and Col_2 is created
Selected	Not selected		Col_1 is reused and Col_2 is not created
Not selected	Not selected		No column is reused and no column is created

Matching Child Table Column Is Already a FK Column

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



Auto-reuse	Auto-migrate	Result	Description of child table
Selected	Selected		T1_Col_1 is created and Col_2 is created
Not selected	Selected		T1_Col_1 is created and Col_2 is created
Selected	Not selected		No columns are reused or created
Not selected	Not selected		No columns are reused or created

Notes:

- By default, only the properties of the primary key column are migrated to the foreign key. If the primary key column is attached to a domain, the domain will not be migrated to the

new foreign key column unless the Enforce non-divergence option model option is selected (see *Enforcing non-divergence from a domain in a data model* on page 154).

- The following table shows the results of changing references when you have selected the auto-migrate columns option:

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

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

Delete unused foreign key columns

Modify reference join

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

For more information on other model options for references, see *Setting PDM Model Options* on page 16.

Reference Properties

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

The General tab contains the following properties:

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

Property	Description
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined
Parent table	Specifies the parent table of the reference. This table contains the primary key, or alternate key, linked by the reference. Use the tools to the right of the list to create, browse for, or view the properties of the currently selected table.
Parent role	Specifies the role of the parent table in the reference. The text is displayed in the diagram, near the parent table
Child table	Specifies the child table of the reference. This table contains the foreign key linked by the reference
Child role	Specifies the role of the child table in the reference. The text is displayed in the diagram, near the child table
Generate	When selected, indicates to generate the reference in the database

Reference Property Sheet Joins Tab

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

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




Join creation is determined by the following Model Options:

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

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

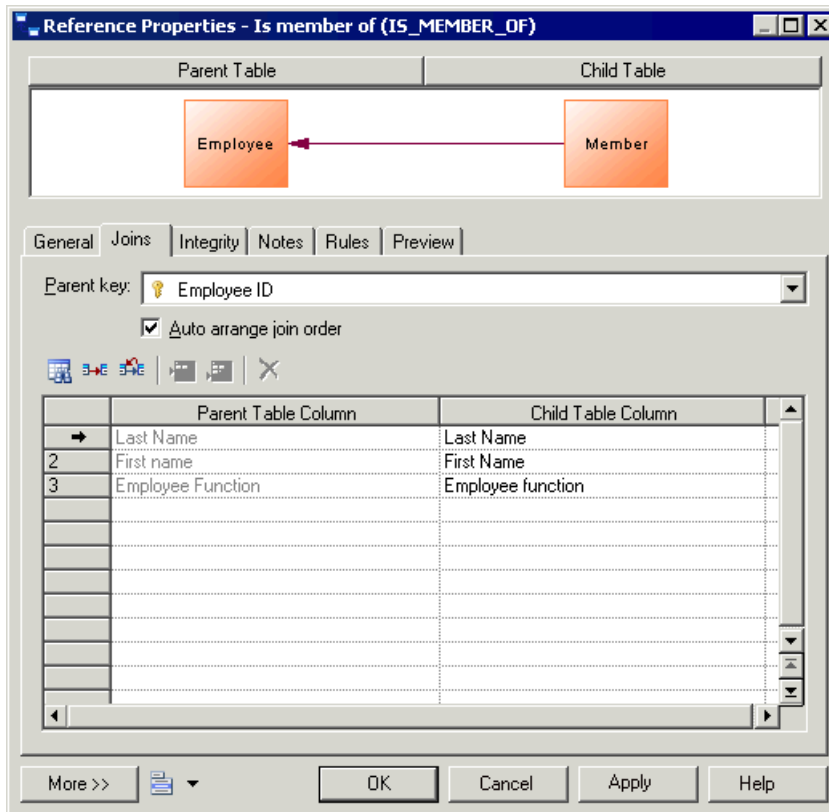
Note: A foreign key column can be changed to link to another parent table column, either within the key relationship, or independent of it.

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

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

1. Double-click a reference in the diagram to open its property sheet, and then click the Joins tab.
2. Select a key in the Parent Key list to create joins on its columns. If you select <NONE>, the column lists are empty and you must specify your own columns to join.

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



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

3. [optional] If you selected <NONE> from the Parent Key list, click the Parent Table Column and select a column from the list, then click the Child Table Column and select a child column.
4. [optional] Select the Auto arrange join order check box to sort the list by the key column order. If this option is not selected, you can re-arrange the columns using the arrow buttons.
5. Click OK.

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

Reference Property Sheet Integrity Tab

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

The Integrity tab contains the following properties:

Property	Description
Constraint name	Name of the referential integrity constraint. Maximum length is 254 characters
Implementation	Specifies how referential integrity will be implemented. You can choose between: <ul style="list-style-type: none"> • Declarative- Referential integrity constraints are defined for particular references. When the reference is generated the target DBMS evaluates the reference validity and generates appropriate error messages • Trigger - Referential integrity constraints are implemented by triggers based on the integrity constraints defined in the reference property sheet. The trigger evaluates reference validity and generates appropriate user-defined error messages

Property	Description
Cardinality	<p>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:</p> <ul style="list-style-type: none"> • 0..* - A parent can have zero or more children. There is no maximum. • 0..1 - A parent can have zero or one children. • 1..* - A parent can have one or more children. There is no maximum. • 1..1 - A parent must have exactly one child <p>Alternately, you can enter your own integer values in one of the following formats:</p> <ul style="list-style-type: none"> • x..y - A parent can have between x and y children. • x - A parent can have exactly x children. • x..y, a..b - A parent can have between x and y or between a and b children. <p>You can use * or n to represent no limit.</p> <p>Examples:</p> <ul style="list-style-type: none"> • 2..n - There must be at least 2 children. • 10 - There must be exactly 10 children. • 1..2, 4..n - There must be one, two, four or more children.
User-defined	Indicates a user-defined constraint name
Update constraint	<p>How updating a key value, in the parent table affects the foreign key value in the child table. Depending on the implementation and DBMS, you can choose between:</p> <ul style="list-style-type: none"> • None - Update or deletion of a value in the parent table has no effect on the child table. • Restrict - A value in the parent table cannot be updated or deleted if one or more matching child values exists • Cascade - Update or deletion of a value in the parent table causes an update or delete of matching values in the child table • Set null - Update or deletion of a value in the parent table sets matching values in the child table to NULL • Set default - Update or deletion of a value in the parent table sets matching values in the child table to the default value
Delete constraint	How deleting a row in the parent table affects the child table
Mandatory parent	Each foreign key value in the child table must have a corresponding key value, in the parent table
Change parent allowed	A foreign key value can change to select another value in the referenced key in the parent table

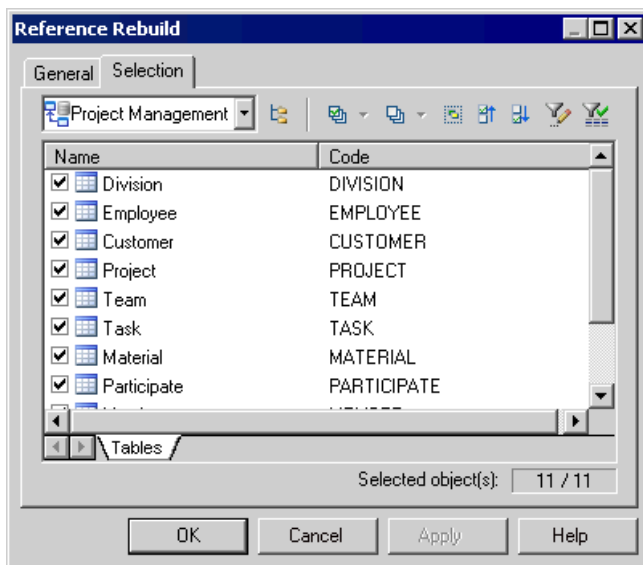
Property	Description
Check on commit	[SQL Anywhere® 5.0 and 5.5 only] Verifies referential integrity only on the commit, instead of verifying it after row insertion. You can use this feature to control circular dependencies
Cluster	Indicates whether the reference constraint is a clustered constraint (for those DBMSs that support clustered indexes)

Rebuilding References

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

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

1. Select **Tools > Rebuild Objects > Rebuild References** to open the Rebuild References dialog box.
2. Select a mode:
 - Delete and Rebuild - All existing references are deleted, and new references built based on matching key columns
 - Preserve - All existing references are kept, and new references are built based on new matching key columns
3. [optional] Click the Selection tab and specify the tables for which you want to rebuild references. By default, all tables are selected.



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

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

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

Displaying Referential Integrity and Cardinality on a Reference Symbol

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 on reference symbols is as follows:

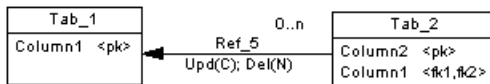
- upd() - Update
- del() - Delete
- cpa - Change Parent Allowed

A letter between the parentheses indicates the type of constraint:

- () - 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:

[*minimum*..*maximum*]



In the above example, the reference label displays:

- Cascade on update
- Set null on delete
- Cardinality is 0 . . n (any number of children is acceptable)

For information about changing the notation of references, see *Setting PDM Model Options* on page 16. For detailed information about working with display preferences, see "Display Preferences" in *Chapter 8, Customizing Your Modeling Environment* of the *Core Features Guide*.

Views (PDM)

A view is a subset of columns drawn from one or more tables defined by a SQL query, which may specify complex criteria for how the tables are joined.

Creating a View

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

- Use the View tool in the diagram Palette.
- Select **Model > Views** to access the List of Views, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select **New > View**.
- Select **Tools > Create View**. For more details, see *Creating a view from the Tools menu* on page 179. You can, optionally, pre-select one or more tables and views in the diagram to create a view automatically populated with their columns.

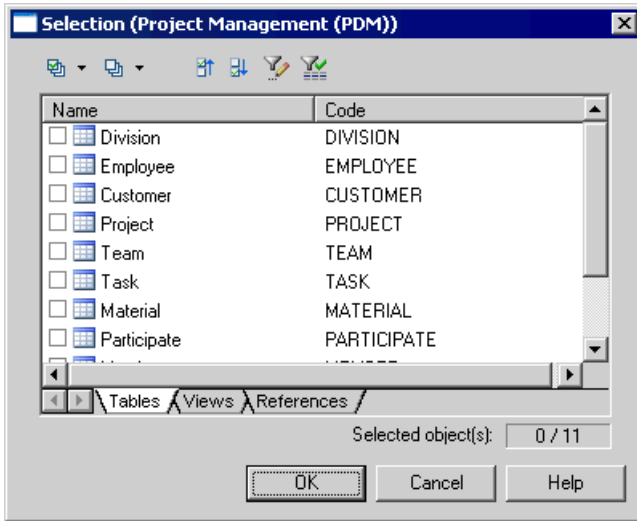
For general information about creating objects, see the Objects chapter in the *Core Features Guide*.

Creating a View from the Tools Menu

You can create a view from the Tools menu. This method allows you to automatically populate the view with columns from tables and other views

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.



3. Select the appropriate objects and then click OK.

A view symbol is displayed in the diagram. It displays all the columns in each of the tables and views selected for the view. The names for the tables and views appear at the bottom of the view symbol.



View Properties

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

The General tab contains the following properties:

Property	Description
Name	Specifies the name of the item, which should be clear and meaningful, and should convey the item's purpose to non-technical users.
Code	Specifies the technical name of the object, which is used for generating code or scripts, which may be abbreviated, and should not generally include spaces.
Comment	Specifies a descriptive label for the view.
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined.
Owner	Specifies the name of view owner. You choose an owner from a list of users. A view can only have one owner at a time. This is normally the view creator.
Usage	Specifies the use of the view: Query only defines a view for consultation only, view cannot update tables; Updatable defines a view for consultation and update, view can update tables; and With Check options implements controls on view insertions.
Dimensional type	Specifies the multidimensional type of the view, that is Dimension or Fact.
Type	Specifies the type of the view, where supported by your DBMS. For information about DB2 and SQL Anywhere materialized views, see <i>Materialized Views</i> on page 186. For information about XML views, see <i>Creating an XML Table or View</i> on page 85.
Generate	Includes view generation as part of database generation script.
User-defined	When selected, makes sure the view query is not parsed by PowerDesigner internal parser. This protects the view query from any update using model objects and keeps its syntax as defined by user. Otherwise, the view query is parsed and modified according to model values.

The following tabs are also available:

- **Columns** - lists the columns in the view based on the SELECT orders from the queries. To add or remove columns from this list, you must modify the appropriate query.

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

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

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

In this case, if you modify the view column code (for example, renaming Name to ClientName, the view creation script will reflect the change:

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

- **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 *Working with View Queries* on page 183).
- **Triggers** - [if your DBMS supports triggers on views] Lists the triggers associated with the view. You can define a trigger to fire when one or more attributes of a table view column are modified. For information about working with triggers, see *Chapter 5, Building Triggers and Procedures* on page 221).
- **Preview** - Displays the SQL code to be generated for the view (see *Previewing SQL Statements* on page 397).

View Column Properties

When you select a column in the list of view columns and click the Properties tool in the toolbar, the view column property sheet is displayed.

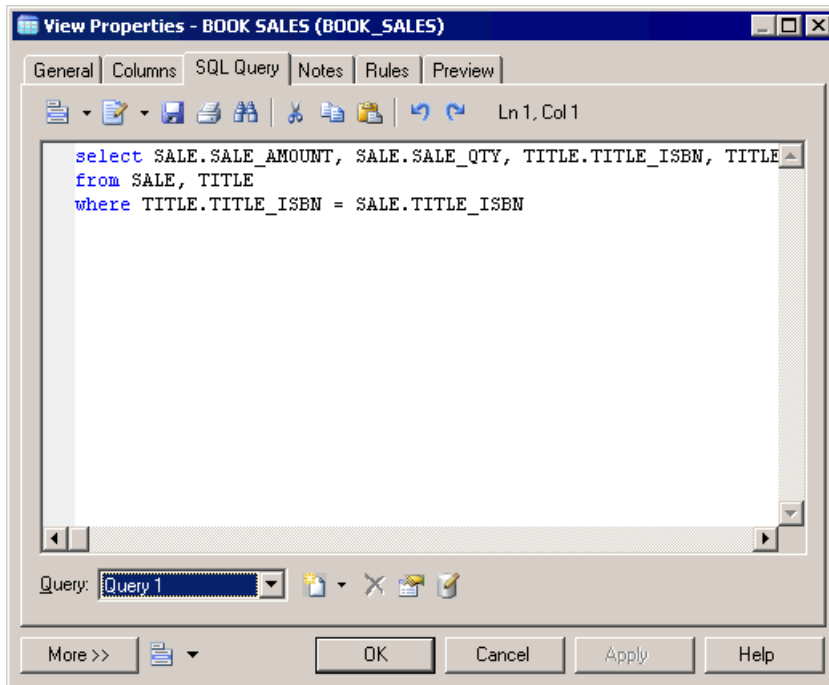
The following properties are available:

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

Property	Description
Precision	Maximum number of places after the decimal point

Working with 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 *Defining Queries with the SQL Editor* on page 399).
- 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

Query Property Sheet SQL Tab

This tab displays the SQL code for the query.

You can edit the code of an individual query in its SQL tab in any of the following ways:

- Edit the code directly in the tab
- Click the Edit with SQL Editor tool to open the code in PowerDesigner's built-in SQL Editor (see *Writing SQL Code in the PowerDesigner SQL Editor* on page 399).
- Click the Edit with tool (CTRL+E) to open the code in your favorite editor

Any edits you make in the SQL tab will propagate to the query's "clause" tabs and the SQL Query tab of the parent view.

Query Property Sheet Tables Tab

This tab lists the tables in the FROM clause, which specify where the query data will be drawn from.

You can add or delete tables as appropriate, and reorder the tables in the list using the arrows at the bottom of the tab. Any changes you make will propagate to the query's SQL tab and to the SQL Query tab of the parent view.

1. Click in the first empty row in the list.
2. Select a table from the list. To enter a more complex expression via the SQL Editor, click the ellipsis button to the right of the list.
3. [optional] Enter an alias for the table in the Alias column.
4. [optional] Select a join keyword and a second table, choose an appropriate join condition keyword, and then specify the join condition. Select additional tables and join them as necessary.

Query Property Sheet Columns Tab

This tab lists the columns in the `SELECT` clause, which specify what data will be displayed in the query.

You can add or delete columns as appropriate, and reorder the columns in the list using the arrows at the bottom of the tab. Any changes you make will propagate to the query's SQL tab and to the SQL Query tab of the parent view.

1. Click in the first empty row in the list.
2. Select a column from the list. You can add all the columns in a table by selecting a list entry with the table name followed by an asterisk. To enter a more complex expression via the SQL Editor, click the ellipsis button to the right of the list.
3. [optional] Enter an alias for the column in the Alias column.

Query Property Sheet Where Tab

This tab lists the expressions in the `WHERE` clause, which restrict the data retrieved by the query.

You can add or delete expressions as appropriate, and reorder the expressions in the list using the arrows at the bottom of the tab. Any changes you make will propagate to the query's SQL tab and to the SQL Query tab of the parent view.

1. Click in the first empty row in the list.
2. [optional] Enter a prefix in the Prefix column.
3. In the first Expression column, select a column from the list. To enter a more complex expression via the SQL Editor, click the ellipsis button to the right of the list.
4. In the Operator column, select an operator from the list.
5. In the second Expression column, select a column from the list. To enter a more complex expression via the SQL Editor, click the ellipsis button to the right of the list.
6. [optional] Enter a suffix in the Suffix column.

Query Property Sheet Group By Tab

This tab lists the columns in the `GROUP BY` clause, which control how the data retrieved by the query will be grouped.

You can add or delete columns as appropriate, and reorder the columns in the list using the arrows at the bottom of the tab. Any changes you make will propagate to the query's SQL tab and to the SQL Query tab of the parent view.

1. Click in the first empty row in the list.
2. Select a column from the list. To enter a more complex expression via the SQL Editor, click the ellipsis button to the right of the list.

Query Property Sheet Having Tab

This tab lists the expressions in the HAVING clause, which restrict the data returned by a query with a GROUP BY clause.

You can add or delete expressions as appropriate, and reorder the expressions in the list using the arrows at the bottom of the tab. Any changes you make will propagate to the query's SQL tab and to the SQL Query tab of the parent view.

1. Click in the first empty row in the list.
2. [optional] Enter a prefix in the Prefix column.
3. In the first Expression column, select a column from the list. To enter a more complex expression via the SQL Editor, click the ellipsis button to the right of the list.
4. In the Operator column, select an operator from the list.
5. In the second Expression column, select a column from the list. To enter a more complex expression via the SQL Editor, click the ellipsis button to the right of the list.
6. [optional] Enter a suffix in the Suffix column.

Query Property Sheet Order By Tab

This tab lists the columns in the ORDER BY clause, which control the way in which the data retrieved by the query will be sorted.

You can add or delete columns as appropriate, and reorder the columns in the list using the arrows at the bottom of the tab. Any changes you make will propagate to the query's SQL tab and to the SQL Query tab of the parent view.

1. Click in the first empty row in the list.
2. In the Column column, select a column from the list. To enter a more complex expression via the SQL Editor, click the ellipsis button to the right of the list.
3. In the Sort Direction column, select either ASC or DESC.

Materialized Views

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

Materialized views are supported in the following ways:

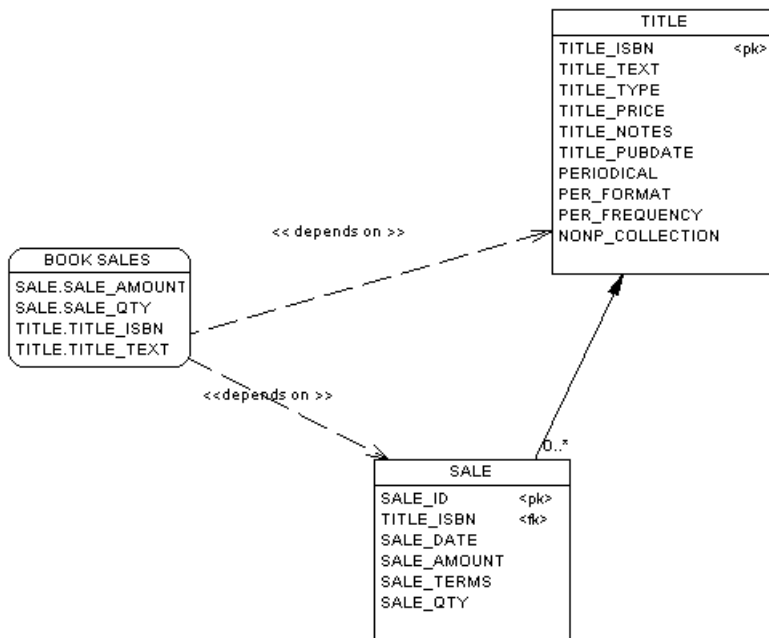
- DB2 - Select materialized query table (or for earlier versions, summary table) in the **Type** list on the **General** tab of a view property sheet.
- HP Neoview - Use the List of Materialized Views (available from **Model > Materialized Views**).
- Oracle - Use the List of Materialized Views (available from **Model > Materialized Views**).

- SQL Anywhere - Select Materialized View in the **Type** list on the **General** tab of a view property sheet.

Using Extended Dependencies for Views

You can create extended dependencies between a view and the tables it depends on. Such links help to make relationships between model objects clearer but are not interpreted and checked by PowerDesigner.

In the following example, a view identified as Book Sales can have two extended dependencies indicating that the view depends on the Title and Sale tables. The diagram displays their extended dependencies and stereotypes.



For more information on extended dependencies, see *Creating Links with Extended Dependencies* on page 28.

Defining a Generation Order for Views

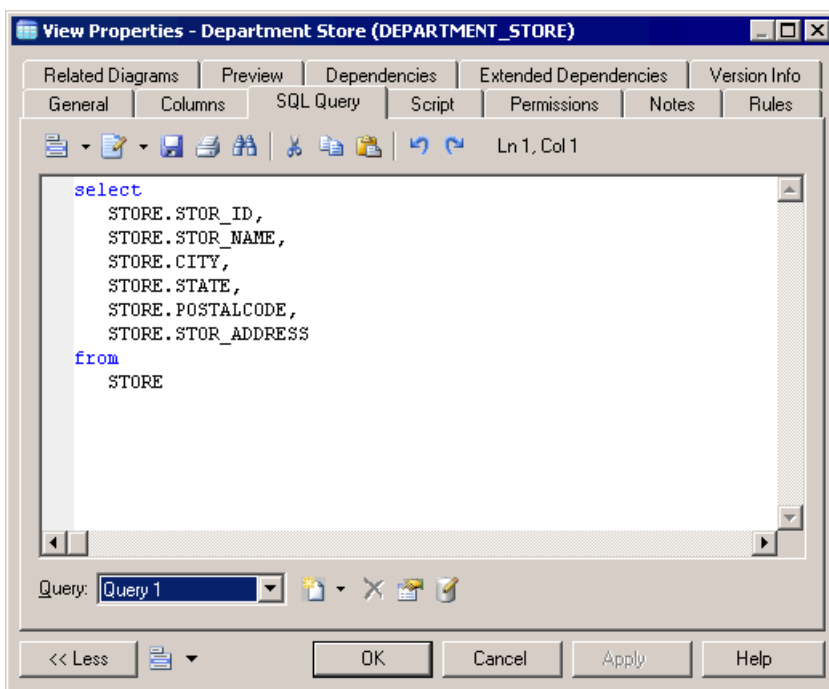
You can use extended dependencies to define an order in the generation of views.

Extended dependencies are free links between PDM objects. These links help to make object relationships clearer between model objects. Usually, these links are not interpreted and checked by PowerDesigner as they are meant to be used for documentation purposes only. However, if you assign the *<<DBCreateAfter>>* stereotype to an extended dependency between views, it will be analyzed during generation.

The view from which you start the extended dependency is the dependent view and the view at the other end of the link is the influent view. The influent view will be generated before the dependent view.

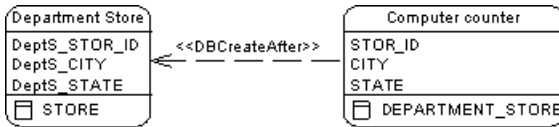
If you create a reflexive and/or circular set of extended dependencies with the `<<DBCcreateAfter>>` stereotype, an error message is displayed during the check model. If you choose to ignore this error, the views will be generated in alphabetical order, without taking into account the generation order, which could cause errors in the creation of views in the database.

For example you create the view DEPARTMENT STORE from the table STORE. The view retrieves information from the table as you can check in the SQL Query tab of the view property sheet:



You decide to create another view called COMPUTER COUNTER to show only part of the department store offer. This view is created from the view DEPARTMENT STORE, and retrieves information from it.

By default views are generated in alphabetical order, so the generation of COMPUTER COUNTER will fail since the view DEPARTMENT STORE from which it depends is not generated. To bypass this problem, you can create an extended dependency with the `<<DBCcreateAfter>>` stereotype from COMPUTER COUNTER to DEPARTMENT STORE.



This allows you to set an order in the generation of views: DEPARTMENT STORE will be generated before COMPUTER COUNTER.

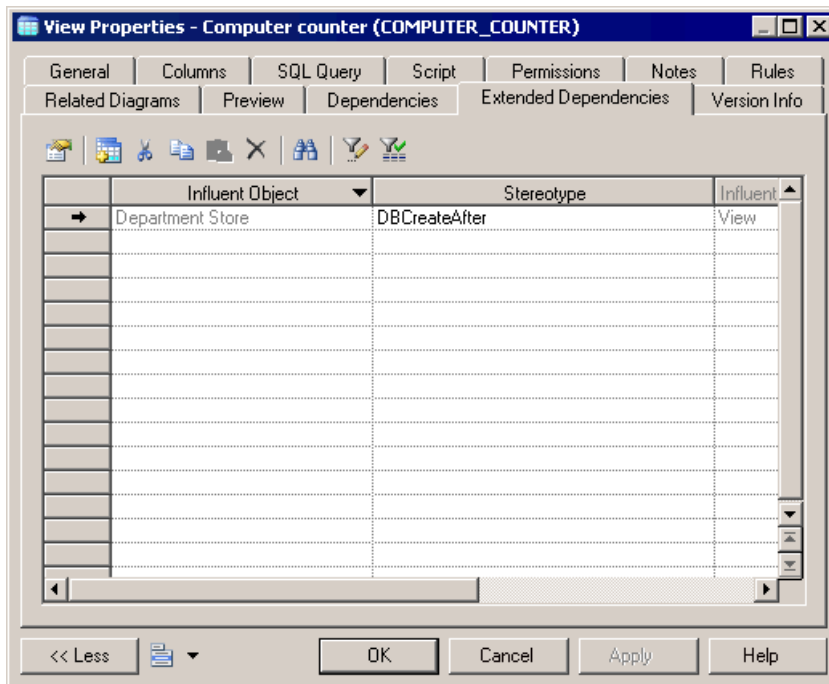
You can create an extended dependency between views from the list of views or from the diagram.

For more information on how to create views from a list, see *Defining a Generation Order for Stored Procedures* on page 267.

1. Select the Extended Dependencies tool in the palette.
2. Click inside the dependent view and while holding down the mouse button, drag the cursor into the influent view. Release the mouse button.
3. Double-click the extended dependency link.

The dependent view property sheet opens to the Extended Dependencies tab. you can check that the influent view is displayed in the Influent Object column of the list of extended dependencies.

4. Click inside the Stereotype column, click the down arrow and select <<DBCreateAfter>> in the list.



5. Click OK.

For more information on extended dependencies, see "Using extended dependencies", in the Objects chapter of the *Core Features Guide*.

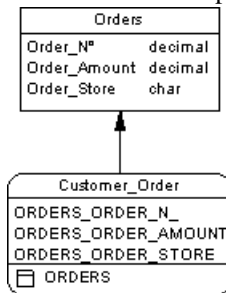
View References (PDM)

A *view reference* is a link between a parent table or view and a child table or view. It is used to define a set of predefined joins between the columns of the parent and the child table or view.

View references are not generated in the database.

Example

Table Orders is the parent of view Customer_Order.



Creating a View Reference

You can create a view reference between two views or between a table and a view. A view reference cannot link two tables.

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

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

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

View Reference Properties

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

The General tab contains the following properties:

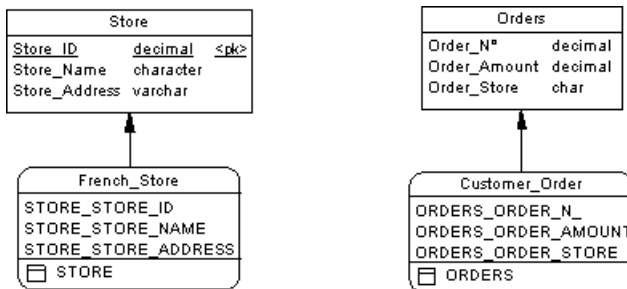
Property	Description
Name	Specifies the name of the item, which should be clear and meaningful, and should convey the item's purpose to non-technical users
Code	Specifies the technical name of the object, which is used for generating code or scripts, which may be abbreviated, and should not generally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

A view reference also includes joins, that are links between parent columns and child columns.

Defining View Reference Joins

A *join* is a link between a column in a parent table or view and a column in a child table or view that is defined within a view reference.

If you create a new view from existing views, the joins defined on these views influence the WHERE statement in the SQL query of the new view.

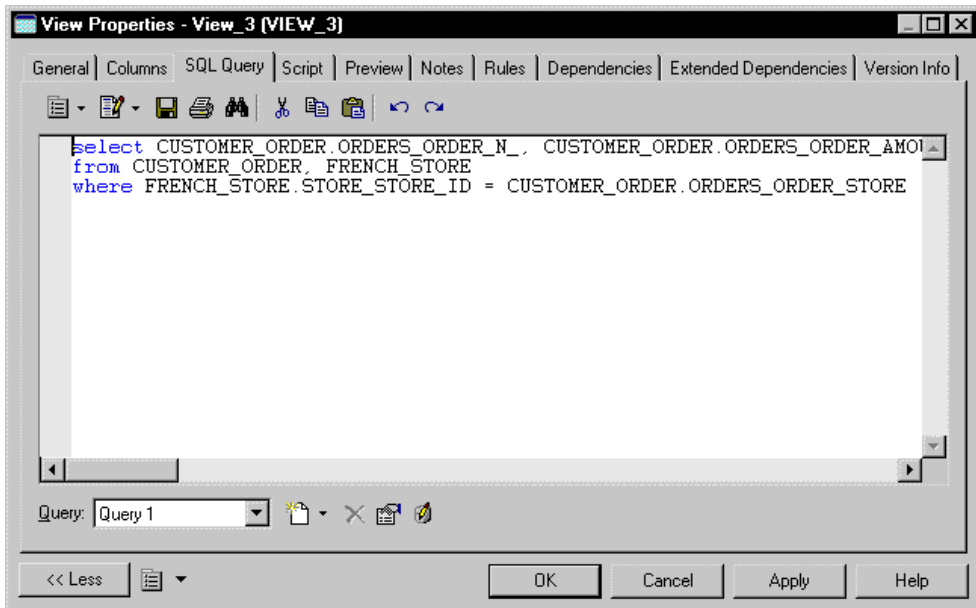


In the above example, French_Store is a view of table Store. You define a join between Store_ID in the table and STORE_STORE_ID in the view.

Customer_Orders is a view of table Orders. You define a join between Order_No in the table and ORDER_ORDER_N in the view.

You create a view reference between French_Store and Customer_Order in which you define a join between ORDER_ORDER_STORE and STORE_STORE_ID. This is to establish a correspondence between the store ID and the store where the order is sent.

If you create a view from French_Store and Customer_Orders, you can check in the SQL query tab of the view that the SELECT order takes into account the join defined between the views. The SELECT statement will retrieve orders sent to French stores only.



In the Joins tab of a view reference property sheet, you can use the Reuse Columns tool to reuse existing child columns with same code as parent columns.

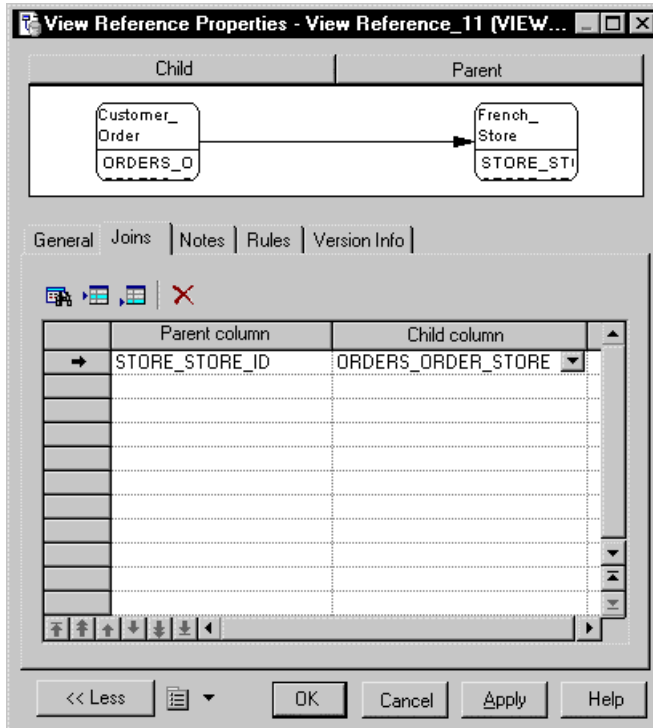
1. Double-click a view reference in the diagram to display the view reference property sheet.
2. Click the Joins tab to display the Joins tab.
3. Click the Reuse Columns tool to reuse existing child columns with same code as parent columns.

or

Click the Add a Row tool.

A join is created but you have to define the parent and child columns.

4. Click in the Parent Column column and select a column in the list.
5. Click in the Child Column column and select a column in the list.



6. Click OK.

Business Rules (CDM/LDM/PDM)

A business rule is a rule that your business follows. This can be a government-imposed law, a customer requirement, or an internal guideline.

Business rules often start as simple observations. For example, "customers call toll-free numbers to place orders". During the design process they develop into more detailed expressions. For example, what information a customer supplies when placing an order or how much a customer can spend based on a credit limit.

Business rules guide and document the creation of a model. For example, the rule "an employee belongs to only one division" can help you graphically build the link between an employee and a division.

Business rules complement model graphics with information that is not easily represented graphically. For example, some rules specify physical concerns in the form of formulas and validation rules. These technical expressions do not have a graphical representation.

During intermodel generation the business rules transfer directly into the generated model where you can further specify them.

There are three ways to use business rules in a PDM:

- You can apply a business rule to an object in the PDM
- You can create a server expression for a business rule which can be generated to a database
- A business rule expression can also be inserted in a trigger or stored procedure (see *Chapter 5, Building Triggers and Procedures* on page 221)

Before you create business rules, formulate your rules by asking yourself the following questions:

- What business problems do I want to address?
- Are there any mandatory procedures for my system?
- Do any specifications set the scope of my project?
- Do any constraints limit my options?
- How do I describe each of these procedures, specifications, and constraints?
- How do I classify these descriptions: as definitions, facts, formulas, or validation rules?

Creating a Business Rule

You can create a business rule in any of the following ways:

- Select **Model > Business Rules** to access the List of Business Rules, and click the Add a Row tool
- Right-click the model or package in the Browser, and select **New > Business Rule**
- Open the property sheet of the object to which you want to apply the rule, click the Rules tab, and click the Create an Object tool

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

Business Rule Properties

You can modify an object's properties from its property sheet. To open a business rule property sheet, double-click its Browser entry in the Business Rules folder.

The General tab contains the following properties:

Property	Description
Name	The name of the item which should be clear and meaningful, and should convey the item's purpose to non-technical users
Code	The technical name of the item used for generating code or scripts, which may be abbreviated, and should not generally include spaces
Comment	Descriptive label for the rule
Stereotype	Sub-classification used to extend the semantics of an object.

Property	Description
Type	<p>Specifies the nature of the business rule. You can choose between:</p> <ul style="list-style-type: none"> • 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.

Expression Tab

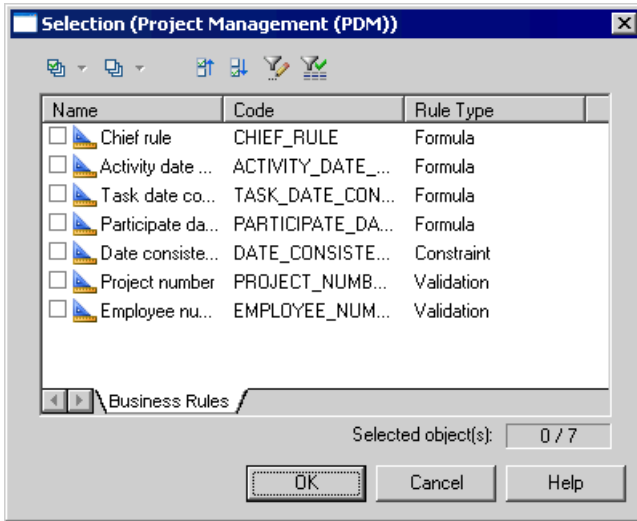
A business rule typically starts out as a description. As you develop your model and analyze your business problem, you can complete the rule by adding a technical expression. The syntax of expressions depends on the target database, and each rule can include two types of expression:

- **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

Applying a Business Rule to a Model Object

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

1. Open the model object's property sheet and click the Rules tab.
2. Click the Add Objects tool to open a list of available business rules.



3. Select one or more business rules and click OK.

The business rules are added to the object and appear in the list of business rules for the object.

4. Click OK to return to the model diagram.

Note: When you apply a business rule to an object, the U (Used) column beside this business rule is automatically checked in the List of business rules to indicate that the business rule is used by at least one object in the model. The U column allows you to visualize unused business rules, you can then delete them if necessary.

Example: Creating and Attaching a Constraint Rule

Validate and constraint business rules have their expressions generated as constraints for DBMSs that support them. 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.

The type of constraint generated depends on the **General > EnableCheck** and **General > EnableMultiCheck** items in the DBMS:

DBMS	Generation result
DBMS does not support check constraints	No constraint is generated.
Database does not support multiple check constraints	Constraint and validation rules and any check parameters defined on the table or column (see <i>Setting Data Profiling Constraints</i> on page 105) are concatenated into a single constraint expression.

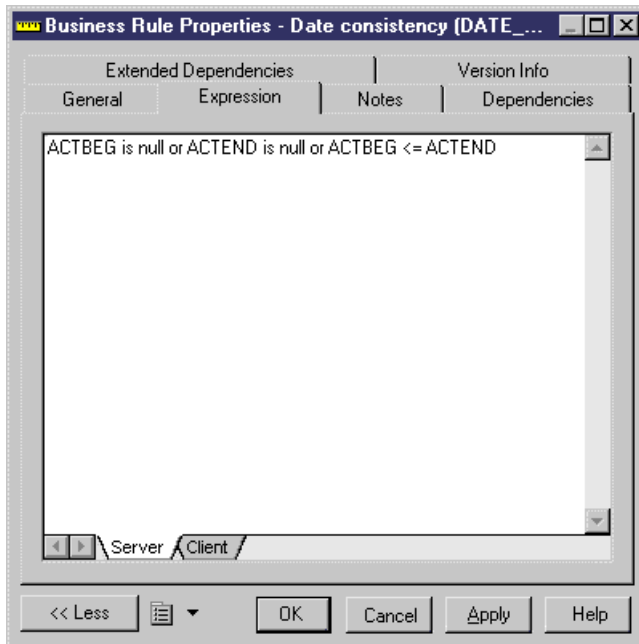
DBMS	Generation result
Database supports multiple check constraints	First, the check parameters and validation business rules are generated into a single constraint, then the constraint business rules are generated into separate constraints in the order in which they are attached. Constraints are thus ordered

If you want to enforce code uniqueness for both validate and constraint rules, you must set the **General > UniqueConstName** item to *Yes*.

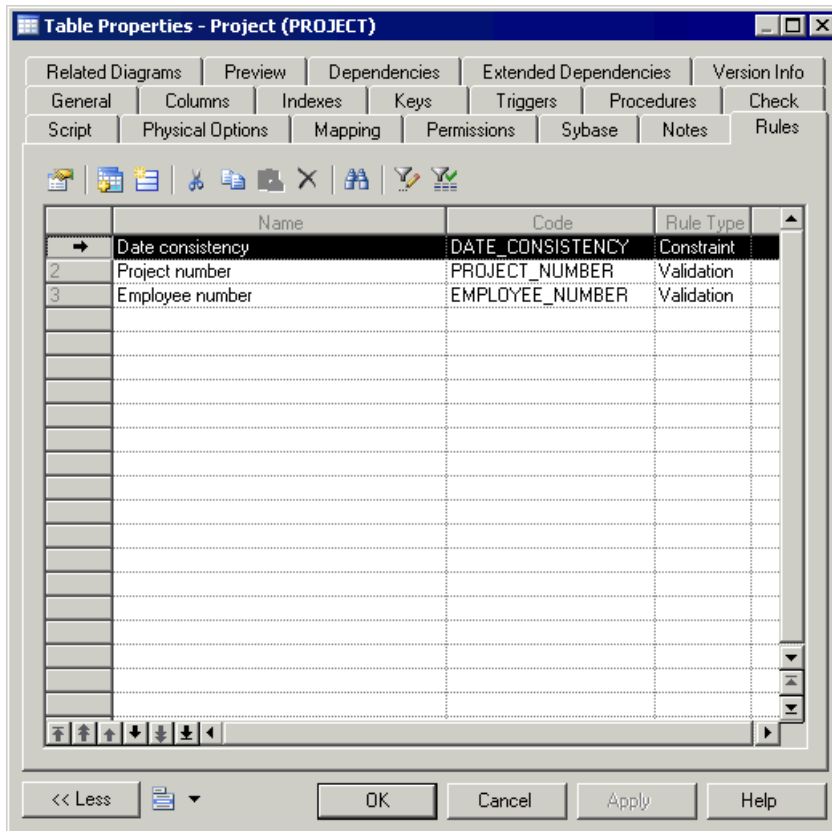
You can preview the constraints that will be generated on the **Preview** tab of the table property sheet.

When reverse engineering, the constraint order is respected:

- The first constraint is retrieved to the **Check** page of the table property sheet
 - Each constraint following the initial constraint is retrieved as a constraint business rule attached to the table
1. Select **Model > Business Rules** to open the List of Business Rules, and click the **Add a Row** tool.
 2. Enter a name and a code for the new rule, and click the **Properties** tool to open its property sheet:
 3. Select *Constraint* in the **Type** list, and then click the **Expression** tab and enter an expression on the Server sub-tab:



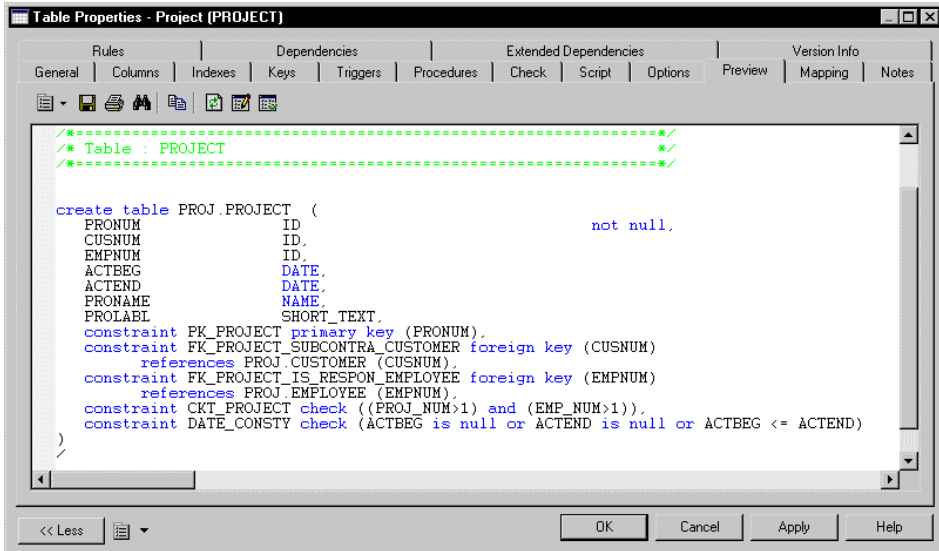
4. Click **OK** to save your changes and return to the model.
5. Open the table or column's property sheet and click the **Rules** tab.
6. Click the **Add Objects** tool to open a list of available business rules, select a constraint business rule from the selection list and click **OK** to attach it to the object.



7. [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	Description
Check parameter (in the Check page of the table)	This check verifies that the customer number is different from the employee number
Validation business rule	PROJ_NUM to check that the column project number is not null EMP_NUM to check that the employee number is not null
Constraint business rule	DATE_CONSTY to check that the start date of the project is inferior to the end date of the project



CHAPTER 4 Building Multidimensional Diagrams

A *multidimensional data diagram* provides a graphical view of your datamart or data warehouse database, and helps you identify its facts, cubes and dimensions.

Numeric values or measures such as sales total, budget limits, are the facts of the business. The area covered by a business, in terms of geography, time, or products are the dimensions of the business.

The multidimensional diagram is used to design the cubes in an OLAP engine, together with the different analysis dimensions.

Business analysts use OLAP databases to send queries and retrieve business information from the different dimensions existing in the database.

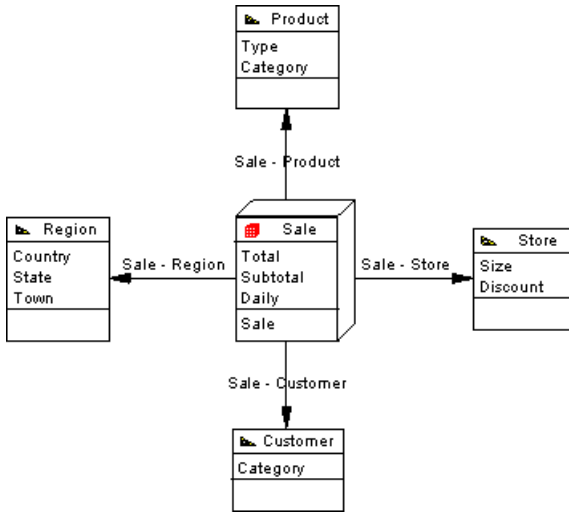
OLAP databases are populated with data from a data warehouse or data mart database. This data transfer is implemented via a relational to multidimensional mapping, the data warehouse or data mart database being the data source of the OLAP database. The OLAP cube is designed to support multidimensional analysis queries, it is organized according to user-defined dimensions.

Multidimensional analysis queries usually involve calculated lists, such as growth and decline rates and time-based comparisons. They aim at detecting trends and relationships. These types of queries are supported by an OLAP database, but not by an operational database.

For more information on the relational to multidimensional mapping, see "Relational to Multidimensional Mapping" in *Chapter 12, Object Mappings* in the *Core Features Guide*.







Example

Sales data can have the dimensions product, region, customer, and store. Facts, for example, the sales totals, are viewed through the user-defined dimensions. When you retrieve the sales total of a particular product for a particular region, you are viewing the sales total through the product and region dimensions. The most common dimension is time because the purpose of multidimensional analytical queries is to find trends.



Multidimensional Diagram Objects

PowerDesigner supports all the objects necessary to build multidimensional diagrams.

Object	Tool	Symbol	Description
Cube			Collection of measures related to aspects of the business and used to carry out a decision support investigation. See <i>Cubes (PDM)</i> on page 203.
Dimension			Axis of investigation of a cube (time, product, geography). See <i>Dimensions (PDM)</i> on page 212.
Attribute	[none]	[none]	Used to qualify a dimension. For example, attribute Year qualifies the Date dimension. See <i>Attributes (PDM)</i> on page 213.
Fact	[none]	[none]	Group of measures used among cubes. See <i>Facts (PDM)</i> on page 215.
Measure	[none]	[none]	Variable linked to a fact, used as the focus of a decision support investigation. See <i>Measures (PDM)</i> on page 216.
Hierarchy	[none]	[none]	Organizational structure that describes a traversal pattern though a dimension. See <i>Hierarchies (PDM)</i> on page 217.
Association			Association that relates a cube to a dimension. See <i>Associations (PDM)</i> on page 218.

Creating a Multidimensional Diagram

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

- Right-click the model in the Browser and select **New > Multidimensional Diagram** from the contextual menu
- Right-click the background of any diagram and select **Diagram > New Diagram > Multidimensional Diagram**.

To create a new PDM with a multidimensional diagram, select **File > New Model**, choose to create a physical data model from the Model type list, choose **Multidimensional Diagram** as the first diagram, and click **OK**.

Cubes (PDM)

A cube is a collection of measures corresponding to values stored into each of its data cells.

The measures (see *Measures (PDM)* on page 216) are organized into dimensions (see *Dimensions (PDM)* on page 212) to provide for faster retrieval and drill-down.

Usually a cube is associated with a fact that allows to define and share measures among cubes.

In a multidimensional diagram, the cube represents an OLAP cube. Cubes need to be created and populated via a text file in the OLAP engine. This text file contains a query used to extract data from a data warehouse or operational database to fill the cubes in the OLAP engine, and is defined in the Query tab of the cube's property sheet.



Creating a Cube

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

- Use the Cube tool in the diagram Palette.
- Select **Model > Cubes** to access the List of Cubes, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select **New > Cube**.
- Rebuild a cube from a fact table or view defined in a physical diagram (see *Retrieving Multidimensional Objects* on page 204).

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

Cube Properties

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

The General tab contains the following properties:

Property	Description
Name	Specifies the name of the item, which should be clear and meaningful, and should convey the item's purpose to non-technical users.
Code	Specifies the technical name of the object, which is used for generating code or scripts, which may be abbreviated, and should not generally include spaces.
Comment	Specifies a descriptive label for the cube.
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined.
Fact	Specifies the fact used by the cube. Use the tools to the right of the list to create, browse for, or view the properties of the currently selected fact.

The following tabs are also available:

- Fact Measures - lists the Measures linked to the fact used by the cube (see *Measures (PDM)* on page 216).
- Queries - displays the SQL statement required to generate the cube data text file that is used to populate the OLAP cube.

Retrieving Multidimensional Objects



During the design of a data warehouse, you will need to identify which of your tables and views will represent facts, and which dimensions. In this section, we will refer to both tables and views as being either:

- A *fact table* – which stores variable numerical values related to aspects of a business (for example, sales, revenue, budget). These are usually the values you want to obtain when you carry out a decision support investigation.
- A *dimension table* – which stores data related to the axis of investigation of a fact (for example, geography, time, product). A dimension table should be connected to a central fact table.

You can specify your tables and views individually as facts or dimensions, or you can use Multidimensional Objects Retrieval Wizard to perform this task automatically, based on the references connecting them:

- Child tables or views become Fact tables or views
- Parent tables or views become Dimension tables or views

The new type is indicated in the Dimensional Type field in the object's property sheet, and a type icon is displayed in the upper left corner of its symbol:

Fact table	Dimension table
	

1. Select **Tools > Multidimension > Retrieve Multidimensional Objects** to open the Multidimensional Objects Retrieval Wizard.
2. Specify the objects to be retrieved. By default both Facts and Dimensions will be retrieved.

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 *IQ Join Indexes* on page 569.

3. [optional] Click the Selection tab to specify which tables to retrieve multidimensional objects from.
4. Click OK to retrieve the multidimensional objects..

The selected tables are assigned a multidimensional type.

Rebuilding Cubes

Once the fact and dimension tables of the data warehouse schema are designed, you can use this information to build the multidimensional cubes. The Rebuild Cubes Wizard transforms fact tables or views into cubes, and dimension tables or views into dimensions. You can then design the cubes taking into account the different analysis axes of the dimensions. These cubes will serve to generate the text files used to create and populate the OLAP engine.

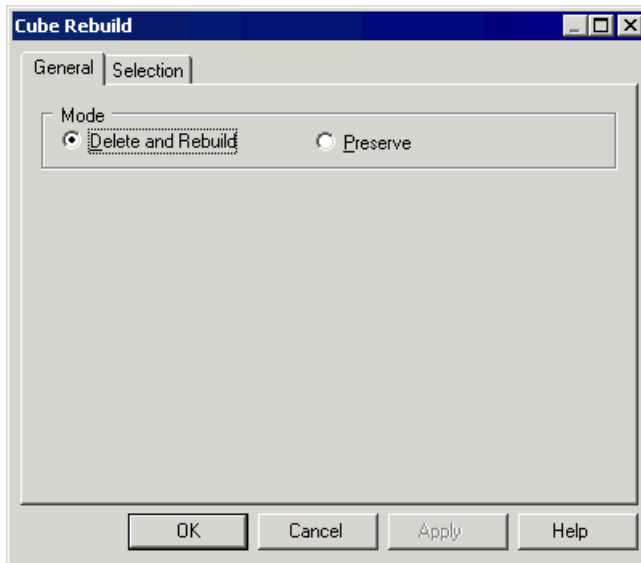
The Rebuild Cubes feature works only if there are tables in the physical diagram with a multidimensional type (Fact or Dimension). You can assign types either manually (see *Table Properties* on page 82) or via the Multidimensional Objects Retrieval Wizard (see *Retrieving Multidimensional Objects* on page 204).

The Rebuild Cubes Wizard creates multidimensional objects in a new or existing multidimensional diagram as follows:

Physical object	After rebuild cubes, creates
Fact table	A fact with the name of the fact table. A cube with the name of the fact table.
Column in a Fact table (except foreign keys)	A measure with the name of the column.

Physical object	After rebuild cubes, creates
Dimensions tables attached to the fact table	A dimension, named through the concatenation of the dimension tables along the path to the child table, from the furthest to the closest. A hierarchy that becomes the default hierarchy, and which contains attributes corresponding to the primary key columns of the tables converted into a dimension
Column in a Dimension table (except foreign keys)	An attribute, named through the concatenation of the dimension table name and column name if column names are ambiguous. Otherwise the name is identical to the name of the column.
Reference between a fact and a dimension table	A cube dimension association

1. Select **Tools > Multidimension > Rebuild Cubes** to open the Cube Rebuild Wizard:

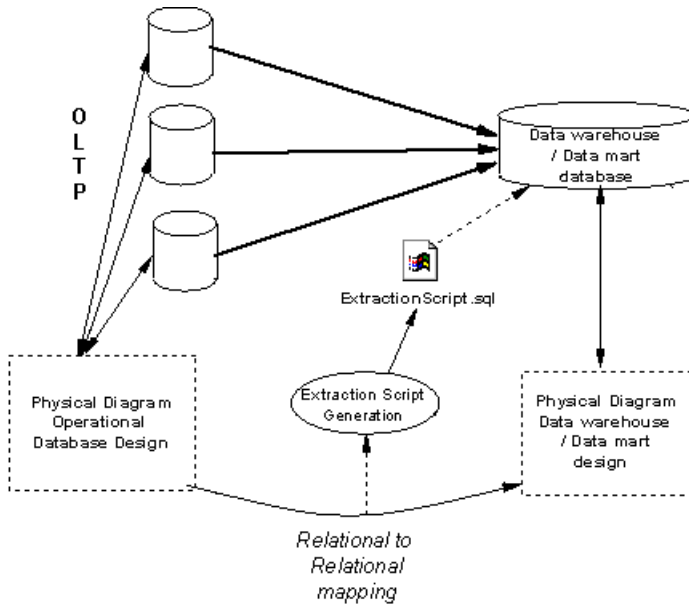


2. Specify a rebuild mode. You can choose between the following options:
 - Delete and Rebuild – all cubes are deleted and rebuilt, including those to which you have made modifications.
 - Preserve – only those cubes that have not been modified are deleted and rebuilt. Any cubes that you have modified are preserved.
3. [optional] Click the Selection tab to specify which tables or views will be used to rebuild cubes. Only those tables and views that have a multidimensional type are available for rebuilding.
4. Click OK to rebuild cubes.

A message in the Output window informs you that the rebuild is successful. The cube and dimension are created and displayed in a multidimensional diagram.

Generating Extraction Scripts

The Generate Extraction Script feature allows you to generate script files that will be used to fill and update tables in a data warehouse or data mart database.

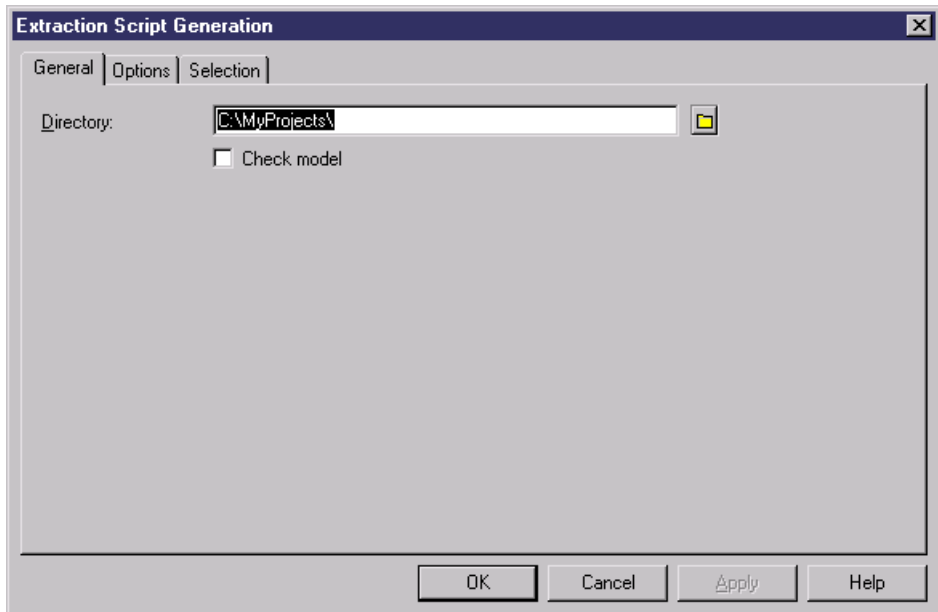


The link between the operational database and the data warehouse or data mart database is a relational to relational mapping.

For more information about relational to relational mapping, see "Relational to relational mapping" in *Chapter 12, Object Mappings* in the *Core Features Guide*.

You can generate a script file for each data source, you can also select the tables in the data source which `select` orders will be generated in the script file. The extraction scripts list all the `select` orders defined in the table mappings.

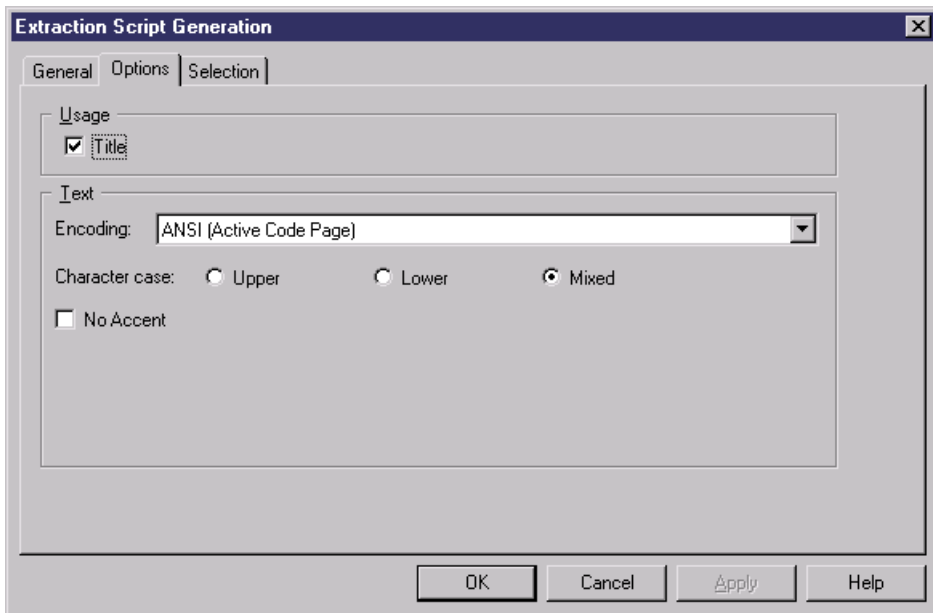
1. In the Physical Diagram, select **Database > Generate** Extraction Script to open the Extraction Script Generation dialog box.



2. Specify a destination directory for the generated file, and select the Check Model check box if you want to verify the PDM syntax before generation. The name of the script is identical to the name of the data source.
3. [optional] Click the Options tab and specify any appropriate options. For more information, see *Extraction Script Generation Options tab* on page 209.
4. [optional] Click the Selection tab, and select the tables that you want to use in the script generation.
5. Click OK to generate the script files in the specified directory.

Extraction Script Generation Options Tab

The Options tab allows you to specify the format for the script.

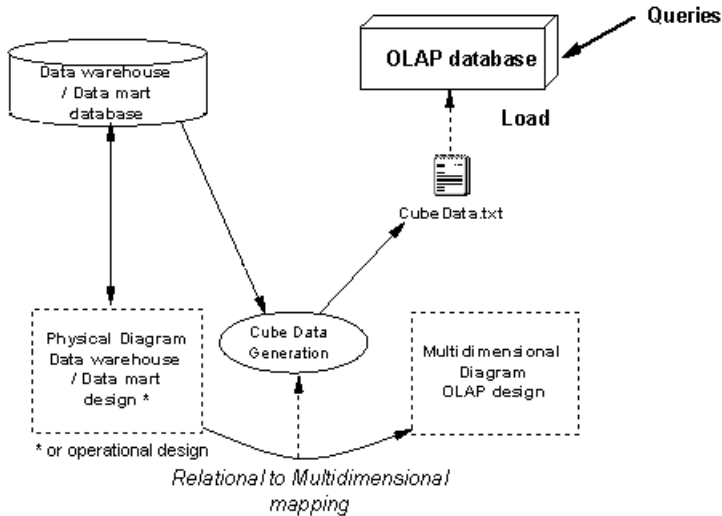


The following options are available:

Option	Description
Title	Inserts the database header and the name of the tables before each select query.
Encoding	Encoding format to use for generation. You should select the encoding format that supports the language used in your model and the database encoding format.
Character Case	Defines the character case in the generated text file.
No Accent	When selected, disallows the use of accents.

Generating Cube Data

Cubes in an OLAP database have to be filled with data from a data warehouse, data mart or operational database. PowerDesigner allows you to generate a text file that will be used by an OLAP tool to create and populate cubes using data from operational sources.



In a PDM multidimensional diagram, each cube is associated with a query. There is one cube per mapping and per data source. The query defined on a cube is used to extract data from a data warehouse or operational database to populate the cubes in the OLAP database. The link between the data warehouse database and the OLAP database is a relational to multidimensional mapping.

For more information about relational to multidimensional mapping, see "Relational to multidimensional mapping" section in *Chapter 12, Object Mappings* in the *Core Features Guide*.

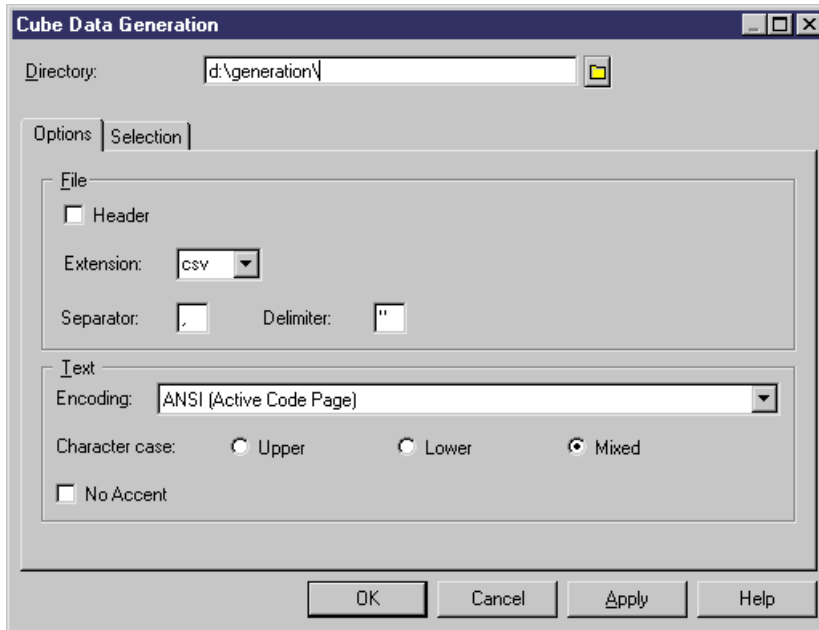
When you generate cube data, PowerDesigner produces one text file for each selected cube and each selected data source. The name of the generated file is a concatenation of the name of the cube and the name of the data source.

Each file contains the following fields:

Field	Details
Dimension	Lists the attributes of the cube
Member	Lists the attribute values
Data fields	Contains the values stored in the fact measures

1. In the multidimensional diagram, select **Tools > Generate Cube Data**.

The Generate Cube Data dialog box is displayed.



2. Define a destination directory for the generated file in the Directory box.
3. Select the generation options in the Options tab.
4. Select the cubes and data sources for which you want to generate a file from the sub- tabs in the Selection tab.
5. Click OK.

The generated files are stored in the destination directory you have defined.

Cube Data Generation Options

You can customize the format of the generated text files from the Generate Cube Data dialog box.

Option	Description
Header	When selected, includes the name of the attribute at the beginning of the generated text file
Extension	Extension of the generated text file, you can choose between .txt and .csv
Separator	Separator used between columns
Delimiter	String delimiter

Option	Description
Encoding	Encoding format to use for generation. You should select the encoding format that supports the language used in your model and the database encoding format
Character Case	Defines the character case in the generated text file
No Accent	When selected, disallows the use of accents

Dimensions (PDM)

A dimension is an axis of analysis in a multidimensional structure.

The dimension is made of an ordered list of attributes that share a common semantic meaning in the domain being modeled. Each attribute designates a unique position along the axis.

Time
Year
Quarter
Month
Week
Year_time <Default> <h>

The dimension can be mapped to tables or views: this mapping allows to transfer operational data to the dimension.

For more information on object mapping, see "Mapping a Physical Object to Multidimensional Objects" in *Chapter 12, Object Mappings* in the *Core Features Guide*.

A dimension may have one or more hierarchies representing attribute sets.

Creating a Dimension

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

- Use the Dimension tool in the diagram Palette.
- Select **Model > Dimensions** to access the List of Dimensions, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select **New > Dimension**.

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

Dimension Properties

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

following sections detail the property sheet tabs that contain the properties most commonly entered for dimensions.

The General tab contains the following properties:

Property	Description
Name	Specifies the name of the item, which should be clear and meaningful, and should convey the item's purpose to non-technical users
Code	Specifies the technical name of the object, which is used for generating code or scripts, which may be abbreviated, and should not generally include spaces
Comment	Specifies a descriptive label for the dimension
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined
Default Hierarchy	Specifies the dimension hierarchy used by default for a cube to perform its consolidation calculations. The hierarchy used by the cube is defined on the cube dimension association

The following tabs are also available:

- Attributes - lists the attributes that qualify the dimension (see *Attributes (PDM)* on page 213).
- Hierarchies - lists the hierarchies used to organize the dimension attributes (see *Hierarchies (PDM)* on page 217).
- Mapping - defines the mapping between the current dimension and a table or a view in a data source.

Attributes (PDM)

An *attribute* is used to qualify dimensions used in queries. For example, the Time dimension can contain attributes Year, Quarter, Month, and Week.

Time
Year
Quarter
Month
Week
Year_time <Default> <h>

Attributes can be organized in hierarchies (see *Hierarchies (PDM)* on page 217).

Creating an Attribute

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

- Open the Attributes tab in the property sheet of a dimension, and click the Add a Row tool.
- Right-click a dimension in the Browser, and select **New > Attribute**.

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

Attribute Properties

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

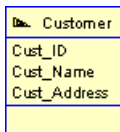
The General tab contains the following properties:

Property	Description
Name	Specifies the name of the item, which should be clear and meaningful, and should convey the item's purpose to non-technical users
Code	Specifies the technical name of the object, which is used for generating code or scripts, which may be abbreviated, and should not generally include spaces
Comment	Specifies a descriptive label for the attribute
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined
Dimension	Parent dimension of the attribute

Attribute Property Sheet Detail Attributes Tab

A detail attribute participates in the definition of an attribute.

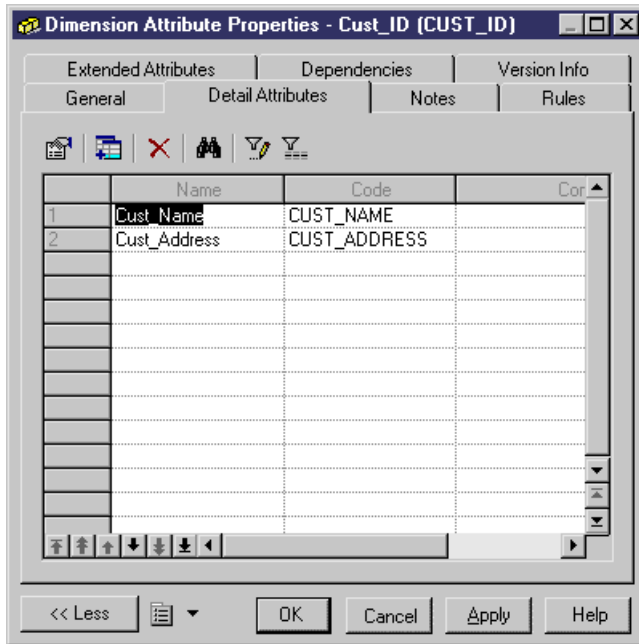
Detail attributes appear in the list of dimension attributes, but you can use them to further define a given attribute. For example, attributes Cust_Name and Cust_Address appear in the list of dimension attributes, however they are used as detail attributes of attribute Cust_ID.



Customer
Cust_ID
Cust_Name
Cust_Address

1. Open the property sheet of a dimension, and click the Attributes tab.
2. Select an attribute in the list, click the Properties tool to open its property sheet, and then click the Detail Attributes tab.

- Click the Add Detail Attributes tool to open a Selection dialog listing the attributes available in the model, select one or more, and then click OK.



- Click Apply.

Facts (PDM)

A fact corresponds to the focus of a decision support investigation.

It is a set of measures (see *Measures (PDM)* on page 216) manipulated by a cube (see *Cubes (PDM)* on page 203). For example, Sale, Revenue, Budget could be facts.



Facts can be reused among different cubes.

Creating a Fact

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

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

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

Fact Properties

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

The General tab contains the following properties:

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

The following tabs are also available:

- Measures - lists the measures manipulated by the cube with which the fact is associated (see *Measures (PDM)* on page 216).
- Mapping - contains the mapping between the fact and a table or a view in a data source.

Measures (PDM)

A measure is a variable that corresponds to the focus of an investigation.

Measures describe the meaning of the analytical values stored in each data cell of a cube.

Measures are most of the time numeric values like for example Price or Total.

Measures can also be the result of an operation or calculation as indicated in the formula box of the measure property sheet.

Creating a Measure

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

- Open the Measures tab in the property sheet of a fact, and click the Add a Row tool.
- Right-click a fact in the Browser, and select **New > Measure**.

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

Measure Properties

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

The General tab contains the following properties:

Property	Description
Name	Specifies the name of the item, which should be clear and meaningful, and should convey the item's purpose to non-technical users
Code	Specifies the technical name of the object, which is used for generating code or scripts, which may be abbreviated, and should not generally include spaces
Comment	Specifies a descriptive label for the measure
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined
Fact	Specifies the parent fact of the measure
Formula	Indicates if the measure is a computed expression and allows to define this expression

Hierarchies (PDM)

A *hierarchy* defines one of two paths through a dimension.

For more information about dimensions, see *Dimensions (PDM)* on page 212.

The path are the following:

- an *organizational* path - describes a traversal pattern through a dimension, from the most general to the most specific attribute of the dimension. It is an ordered subset of the attributes.
- a *consolidation* path - represents a consolidation of attributes. For example, a Time dimension with a base periodicity of days might have a hierarchy specifying the consolidation of days into weeks, weeks into months, months into quarters, and quarters into years.

Creating a Hierarchy

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

- Open the Hierarchies tab in the property sheet of a dimension, and click the Add a Row tool.

- Open the Attributes tab in the property sheet of a dimension, and click the Create Hierarchy tool.
- Right-click a dimension in the Browser, and select **New > Hierarchy**.

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

Hierarchy Properties

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

The General tab contains the following properties:

Property	Description
Name	Specifies the name of the item, which should be clear and meaningful, and should convey the item's purpose to non-technical users
Code	Specifies the technical name of the object, which is used for generating code or scripts, which may be abbreviated, and should not generally include spaces
Comment	Specifies a descriptive label for the hierarchy
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined
Dimension	Specifies the parent dimension of the hierarchy

The following tabs are also available:

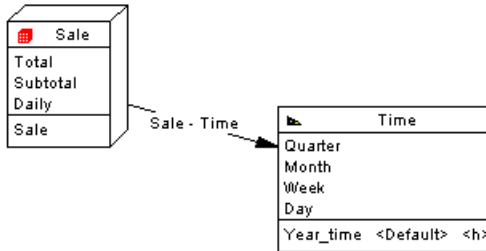
- Attributes - lists the attributes associated with the hierarchy (see *Attributes (PDM)* on page 213).

Associations (PDM)

An association relates a cube to the dimension that defines it.

It shows the axis of investigation of the dimension (see *Dimensions (PDM)* on page 212) in the cube (see *Cubes (PDM)* on page 203).

For example, the Sale cube is linked to the Time dimension by the Sale - Time association to analyze sales through the time dimension.



There can be only one association between a cube and a dimension.

Creating an Association

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

- Use the Association tool in the diagram Palette.
- Select **Model > Associations** to access the List of Associations, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select **New > Association**.

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

Association Properties

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

The General tab contains the following properties:

Property	Description
Cube	Specifies the cube origin of the association. Click the Properties tool to view the properties of the currently selected object.
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	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined.
Hierarchy	Specifies the hierarchy used by the cube for the consolidation calculation. Click the Properties tool to view the properties of the currently selected object.

CHAPTER 5 Building Triggers and Procedures

PowerDesigner provides support for modeling triggers and stored procedures.

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.

You can use triggers to enforce referential integrity (where declarative constraints are not sufficient) and to implement sequences for columns.

Trigger Templates and Template Items

A trigger template is a pre-defined form for creating triggers. PowerDesigner ships templates for each supported DBMS. Depending on the current DBMS, there are pre-defined templates for insert, update, and delete trigger types.

A template item is a reusable block of SQL script that can implement referential integrity, or do any other work on database tables. PowerDesigner ships template items for each supported DBMS. A template item is inserted into a trigger template script, or a trigger script. The template item calls a corresponding SQL macro which can implement an insert, update, delete, or error message constraint on one or more tables in the database.

You can use the PowerDesigner templates and template items, copy and edit them, or create your own from scratch. For more information, see *Trigger Templates (PDM)* on page 235.

Creating Triggers

You can create triggers for referential integrity individually or by default, and create your own triggers from the property sheet of a table (or view, if supported by your DBMS).

You can write a trigger from scratch directly in its property sheet, but we recommend that you use a trigger template and/or trigger template items to define the trigger code. These allow you to create triggers in a modular fashion, to make re-use of your trigger code easier and give your triggers more portability (see *Trigger Templates (PDM)* on page 235).

Implementing Referential Integrity with Triggers

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

Note: To instruct PowerDesigner to implement referential integrity between tables using triggers by default, select **Tools > Model Options**, click **Model Settings > Reference** in the **Category** list, select **Trigger** in the **Default implementation** list, and then click **OK**.

You can implement referential integrity between two tables by a trigger manually as follows:

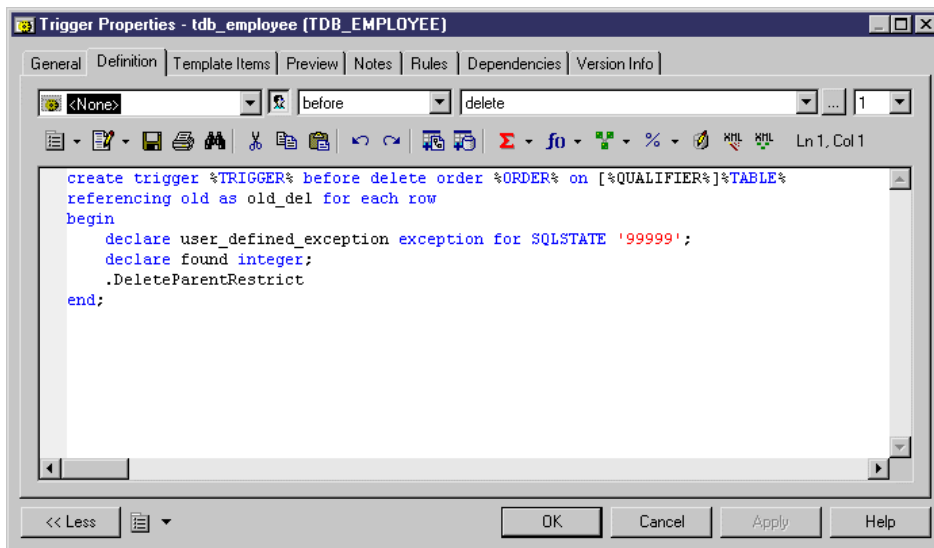
1. Create a reference between two tables, and then double click the reference symbol to open its property sheet.
2. Click the **Integrity** tab, and then select **Trigger** from the **Implementation** list.
3. Specify the form of Update and Delete constraints using the radio buttons (see *Reference Property Sheet Integrity Tab* on page 175), and then click *OK* to return to the diagram.
4. If you have set the **Automatically rebuild triggers** model option, then triggers will have been created automatically in the parent and child tables. To verify this, double-click the table symbol to open its property sheet, and then click the **Triggers** tab. If the triggers are not present, you will need to rebuild your triggers manually (see *Rebuilding Triggers* on page 226).

Creating a Trigger from a Template

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

1. Double-click a table symbol to open its property sheet, and then click the **Triggers** tab.
2. Click the **Add a Row** tool to create a new trigger, and type a name and code.
3. Click **Apply** to commit the creation of the new trigger, and then click the **Properties** tool to open its property sheet.
4. Click the **Definition** tab, and select a trigger template from the **Template** list.

The time and event fields will be set and the template code copied into the definition box.



5. [optional] Modify the trigger definition code. You can add trigger template items, use PDM variables and macros and various other tools available from the toolbar (see *SQL Editor Tools* on page 401).

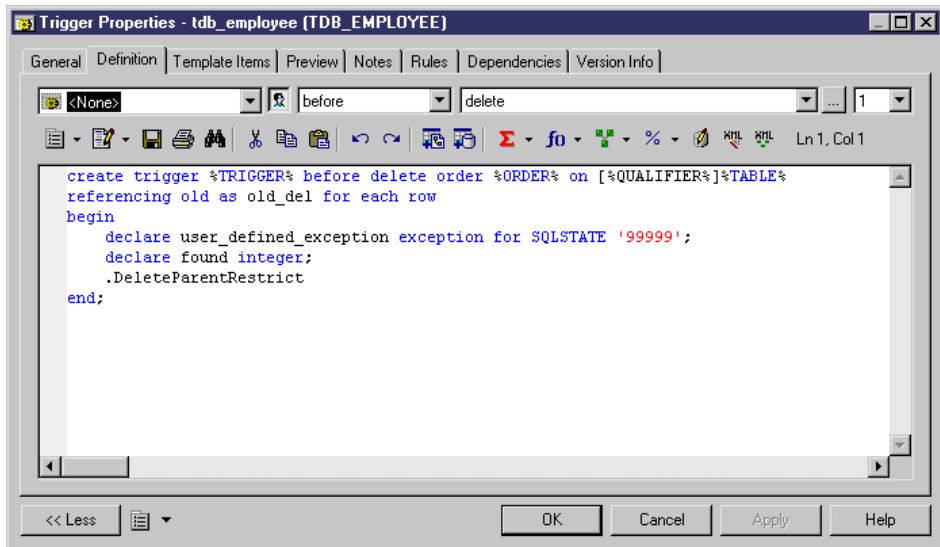
If you edit the code, then the trigger will be marked as user-defined and will be excluded from most forms of rebuilding (see *Rebuilding Triggers* on page 226).

6. You can also modify the trigger's other properties. For a full list of the properties available, see *Trigger Properties* on page 224.
7. Click OK in each of the dialog boxes.

Creating a Trigger from Scratch

You can create a trigger without basing it on a template. However, we recommend that you use a template as this will simplify reuse of your code and make your triggers more portable.

1. Double-click a table symbol to open its property sheet, and then click the Triggers tab.
2. Click the Add a Row tool to create a new trigger, and type a name and code.
3. Click Apply to commit the creation of the new trigger, and then click the Properties tool to open its property sheet.
4. Click the Definition tab.



5. Enter the trigger definition code. You can add trigger template items, use PDM variables and macros and various other tools available from the toolbar (see *SQL Editor Tools* on page 401).

The trigger will be marked as user-defined and will be excluded from most forms of rebuilding (see *Rebuilding Triggers* on page 226).

6. You can also modify the trigger's other properties. For a full list of the properties available, see *Trigger Properties* on page 224.
7. Click OK in each of the dialog boxes.

Note: When using the PowerDesigner Eclipse plug-in, you can right-click a trigger in the Browser and select Edit in SQL Editor from the contextual menu to open it in the Eclipse SQL Editor. You can optionally connect to your database in order to obtain auto-completion for table names. The trigger definition is added as a .SQL file to the Generated SQL Files list in the Workspace Navigator.

Trigger and DBMS Trigger Properties

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

The General tab contains the following properties:

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

Definition Tab

This tab allows you to enter code for the trigger. The following properties are available:

Property	Description
Template	Specifies the template on which the trigger is based.
User-defined	This button is automatically depressed when you modify the definition of a trigger. You can click the button to restore the template trigger definition.
Time	Specifies the time attribute of the template. 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.

Property	Description
Event	<p>Specifies the event attribute of the template. Click the ellipsis tool to the right of this field to select multiple events (see <i>Defining Triggers with Multiple Events</i> on page 234)</p> <p>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.</p> <p>For DBMS triggers, this field allows you to enter any text.</p>
Order	[table and view triggers only] Specifies the firing order of trigger.

For information about the tools available on the toolbar, see *SQL Editor Tools* on page 401.

Template Items Tab

This tab lists the trigger template items available for use in the trigger definition (see *Trigger Template Items (PDM)* on page 241).

Preview Tab

This tab displays the SQL code that will be generated for the trigger (see *Previewing SQL Statements* on page 397).

Rebuilding Triggers

PowerDesigner can rebuild triggers to ensure that they are attached to all tables joined by references to ensure referential integrity.

PowerDesigner can rebuild triggers either:

- Automatically, whenever a relevant change is made, if you have enabled the Automatically rebuild triggers model option
- Manually, when you select **Tools > Rebuild Objects > Rebuild Triggers**

The Rebuild Triggers function creates new triggers based on template items that correspond to trigger referential integrity defined for references and sequence implementation for columns.

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 250.

Rebuilding Triggers Automatically

PowerDesigner can automatically rebuild your triggers whenever a relevant change is made.

1. Select **Tools > Model Options** to open the Model Options window.
2. In the Category pane, click on Trigger under the Model Settings node to go to the Trigger model options
3. Select the Automatically rebuild triggers checkbox, and click OK.

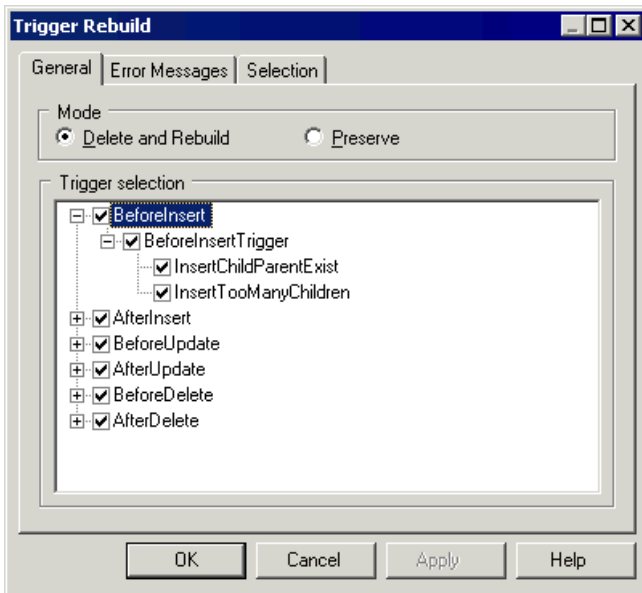
PowerDesigner will rebuild all triggers and will, from now on, rebuild triggers whenever you make a relevant change in the model.

Rebuilding Triggers Manually

You can rebuild triggers manually at any time.

1. Select **Tools > Rebuild Objects > Rebuild Triggers** to open the Trigger Rebuild window.
2. Specify a rebuild mode. You can choose between the following options:
 - Delete and Rebuild – all triggers attached to templates are deleted and rebuilt, including those to which you have made modifications
 - Preserve – only those triggers attached to templates that have not been modified are deleted and rebuilt. Any triggers that you have modified are preserved.
3. The Trigger selection box shows an expandable tree view of trigger types. There are three levels in this tree:
 - All trigger types supported by the current DBMS
 - All trigger templates corresponding to the trigger types
 - All template items defined for each trigger template

For example, in the list below, the two template items `InsertChildParentExist` and `InsertTooManyChildren` are used in the `BeforeInsertTrigger` template that is, in turn, used in all triggers with a time of `Before` and an event type of `Insert`:



You can select which trigger types, trigger templates, and template items will be rebuilt by expanding the appropriate nodes and selecting or clearing the check boxes as required.

4. [optional] Click the Error Messages tab to define the types of error messages to generate. For more information about this tab, see the *Creating and generating user-defined error messages* on page 269 section.
5. [optional] Click the Selection tab. to specify which tables to rebuild the triggers for.
6. Click OK to begin the rebuild process.

Progress is shown in the Output window. You can view the triggers that have been created from the Triggers tab of the table property sheet, or from the List of Triggers.

Modifying Triggers

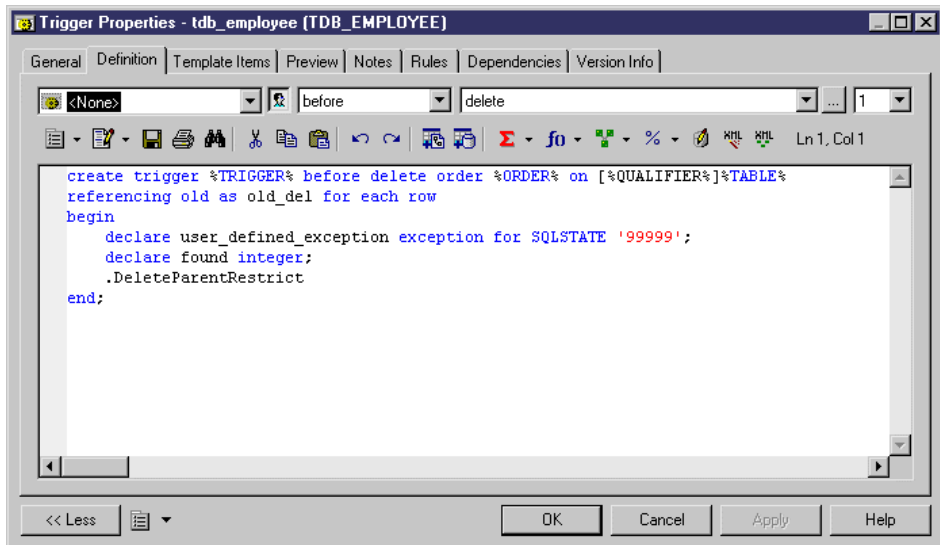
PowerDesigner provides various methods for editing a trigger.

You can:

- Edit the code directly in the Definition tab of its property sheet.
- Attach a predefined trigger template or create and attach your own reusable trigger templates
- Insert predefined trigger template item code or create your own reusable trigger template items

Note: If you modify the definition of a DBMS trigger template or template item, you are modifying the DBMS definition file. We recommend that you only ever work on a copy of the original DBMS definition file.

1. Open the trigger property sheet in one of the following ways:
 - Open the relevant table property sheet click the Triggers tab, select the trigger from the list, and then click the Properties tool
 - Select **Model > Triggers > Triggers** to open the List of Triggers, select the trigger from the list, and then click the Properties tool
 - In the Browser, find the entry for the trigger, right-click it, and select Properties from the contextual menu
2. Click the Definition tab to display the trigger code.



3. Enter the trigger definition code. You can attach a trigger template, add template items, use PDM variables and macros and various other tools available from the toolbar (see *SQL Editor Tools* on page 401).

The trigger will be marked as user-defined and will be excluded from most forms of rebuilding (see *Rebuilding Triggers* on page 226), if you select the "Preserve" mode.

4. You can also modify the trigger's other properties. For a full list of the properties available, see *Trigger Properties* on page 224.
5. Click OK in each of the dialog boxes.

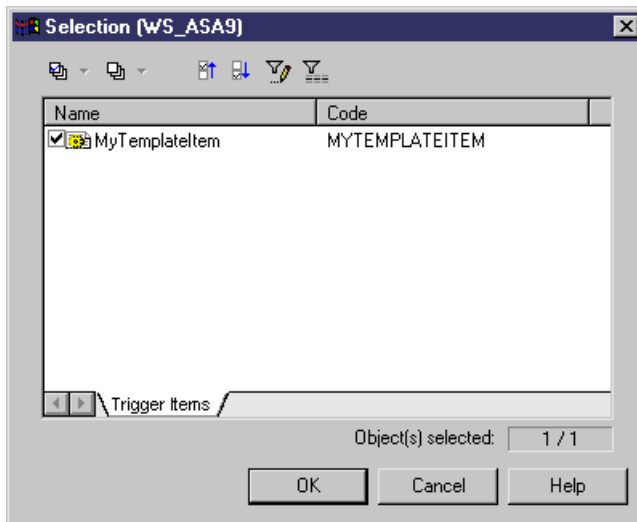
Inserting a Template Item into a Trigger or Trigger Template

Template items are inserted in a trigger or trigger template definition using a dot followed by the template item name. For example, the following script contains two template items `InsertChildParentExist` and `InsertTooManyChildren`:

```
/* Before insert trigger "%TRIGGER%" for table "[%QUALIFIER%]%TABLE%" */
create trigger %TRIGGER% before insert order %ORDER% on [%QUALIFIER%]%TABLE%
```

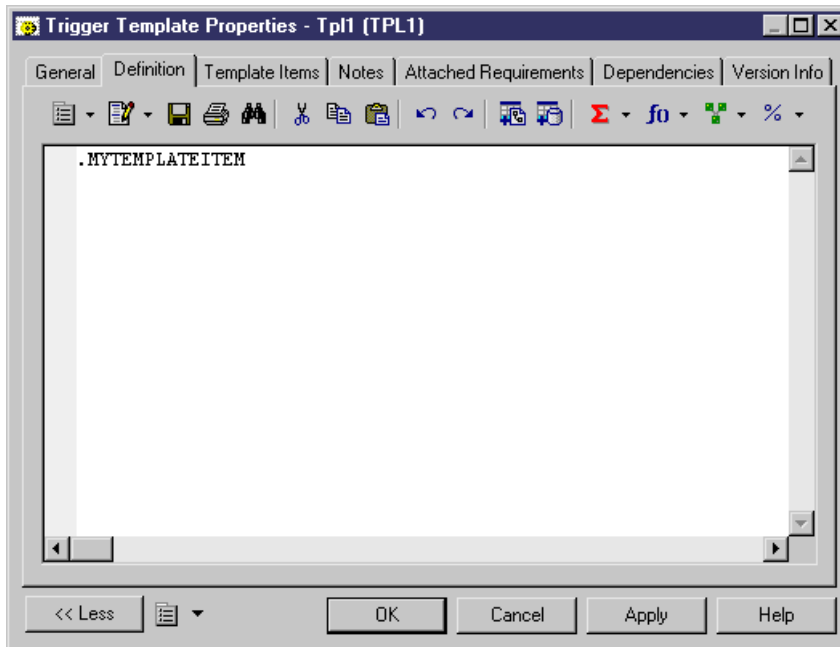
```
referencing new as new_ins for each row
begin
  declare user_defined_exception exception for SQLSTATE '99999';
  declare found integer;
  .InsertChildParentExist
  .InsertTooManyChildren
end
/
```

1. Open the property sheet of the trigger or trigger template that you want to modify, and then click the Definition tab.
2. Click at the point in the code where you want to insert the trigger template item, and then click one of the following tools:
 - Add Trigger Item From DBMS – to open a selection box containing a list of trigger template items defined in the DBMS definition file
 - Add Trigger Item From Model – to open a selection box containing a list of trigger template items defined in the model



3. Select the item to insert and then click OK to return to the definition tab.

The trigger template item will be inserted in your code. It will also appear in the list on the Template Items tab.



Declaring a Template Item in a Trigger Definition

Certain DBMS require that a cursor and variables are declared for each template item before the template item name is used in the script. This can be a statement that calls a corresponding procedure

You can use the following format to declare a template item:

```
Decl
```

template item name.

For example, the trigger definition for Oracle 8 contains the `.DeclInsertChildParentExist` statement which declares the following `.InsertChildParentExist` template item:

```
-- Before insert trigger "[%QUALIFIER%]%TRIGGER%" for table
 "[%QUALIFIER%]%TABLE%"
create trigger [%QUALIFIER%]%TRIGGER% before insert
on [%QUALIFIER%]%TABLE% for each row
declare
    integrity_error exception;
    errno integer;
    errmsg char(200);
    dummy integer;
    found boolean;
    .DeclInsertChildParentExist
begin
    .InsertChildParentExist
```

```

-- Errors handling
exception
  when integrity_error then
    raise_application_error(errno, errmsg);
end;
/

```

In a generated trigger script, `.DeclInsertChildExist` corresponds to the following definition:

```

.FOREACH_PARENT()
-- Declaration of InsertChildParentExist constraint for the parent
"[%PQUALIFIER%]%PARENT%"
.DEFINE "CURSOR" "cpk%REFNO%_%.25L:TABLE%"
cursor %CURSOR%( .JOIN("var_%.L26:FK% %L:COLTYPE%", "", "", "", " is")
select 1
from [%PQUALIFIER%]%PARENT%
where .JOIN("%PK% = var_%.L26:FK%", "and ")
and .JOIN("var_%.L26:FK% is not null", "and ", "", ";")
.ENDFOR

```

Trigger Naming Conventions

The pre-defined trigger templates that ship with PowerDesigner indicate naming conventions for the trigger scripts that it generates. The naming convention consists of a prefix indicating the trigger type followed by the table code.

The default naming conventions include a variable (`%L:TABLE`). The name of the resulting trigger script replaces this variable with a lower-case table code. For example, a resulting trigger script may have the name `ti_employee`.

You can change the trigger naming convention in PowerDesigner pre-defined DBMS trigger templates from the Trigger Templates tab of the DBMS property sheet.

1. Select **Database > Edit Current DBMS** to open the DBMS definition file in the Resource Editor, and then click the Trigger Template tab.
2. Click a trigger template in the list, and then click the Properties tool to open its property sheet .
3. Type a new trigger name in the Trigger Name text box at the bottom of the tab.
For example, `mytempl_%TABLE%`
4. Click OK in each of the dialog boxes.

Calling a Related Procedure in a Trigger Template

Some target databases do not accept code within a trigger statement. For these databases, a trigger template can call a related procedure as a parameter, which is defined in a procedure template. In these cases, procedure templates are listed in the list of trigger templates.

Example

Informix does not accept code in trigger templates. The template `InsertTrigger` calls the procedure in the form of the variable `%PROC%`, as follows:

```
-- Insert trigger "[%QUALIFIER%]%TRIGGER%" for table "[%QUALIFIER%]
%TABLE%"
create trigger [%QUALIFIER%]%TRIGGER% insert on [%QUALIFIER%]%TABLE%
referencing new as new_ins
  for each row (execute procedure %PROC%(.FKCOLN("new_ins.%COLUMN%",
", ", ", ", "));")
/
```

The template `InsertProc` defines the procedure, as follows:

```
-- Insert procedure "%PROC%" for table "[%QUALIFIER%]%TABLE%"
create procedure %PROC%(.FKCOLN("new_%.14L: COLUMN% %COLTYPE%", " ",
", ", " "))
  .DeclInsertChildParentExist
  .DeclInsertTooManyChildren
  define errno integer;
  define errmsg char(255);
  define numRows integer;

  .InsertChildParentExist
  .InsertTooManyChildren

end procedure;
/
```

Multiple Triggers

Some DBMSs allow you to have multiple triggers of the same type (time and event) defined for any given table. Triggers of the same type are triggers that are invoked for the same insert, update, or delete event.

Example

A company is considering large numbers of candidates for new positions in various posts. You want to ensure that all new employees will have a salary that is within the range of others working in the same field, and less than his or her prospective manager.

On an `EMPLOYEE` table, you create two *BeforeInsert* triggers, `tibTestSalry1_EMPLOYEE` to verify that a proposed salary falls within the correct range, and `tibTestSalry2_EMPLOYEE` to verify that the proposed salary is less than that of the prospective manager.

```

create trigger tibTestSalry1 before insert order 1 on EMPLOYEE
referencing new as new_ins for each row
begin

    [Trigger code]

end

create trigger tibTestSalry2 before insert order 2 on EMPLOYEE
begin

    [Trigger code]

end

```

For a specified table, you can indicate the order that a trigger executes, or fires, within a group of triggers of the same type.

Indicating Trigger Order

You indicate trigger order as follows:

1. Click the Definition tab from the trigger property sheet.
2. Select a number from the Order dropdown list box to indicates the position in the firing order that the trigger fires.
3. Click OK in each of the dialog boxes.

Defining Triggers with Multiple Events

Some DBMSs support multiple events on triggers. If such is the case, the Ellipsis button to the right of the Event box on the trigger definition tab is available.

You can click the Ellipsis button to open the Multiple Events Selection box. If you select several events and click OK, the different events will be displayed in the Event box, separated by the appropriate delimiter.



DBMS Triggers (PDM)

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.

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

DBMS triggers can use trigger templates and trigger template items, just like table and view triggers see *Trigger Templates (PDM)* on page 235.

For information about DBMS trigger properties, see *Trigger and DBMS Trigger Properties* on page 224.

Creating DBMS Triggers

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 package in the Browser, and select **New > DBMS Trigger**

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

Trigger Templates (PDM)

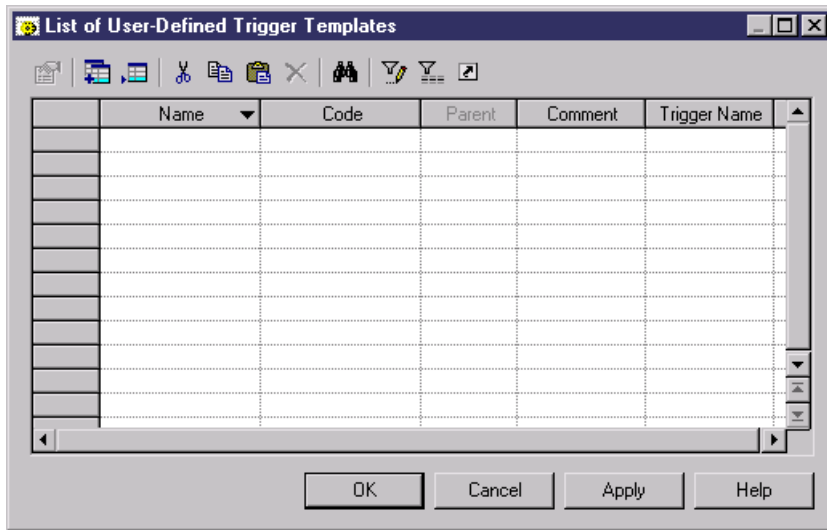
PowerDesigner trigger templates and template items allow you to create triggers in a modular reusable fashion.

PowerDesigner uses pre-defined trigger templates to create triggers to implement referential integrity between tables. Trigger templates exist for each type of trigger supported by the DBMS. Each template specifies:

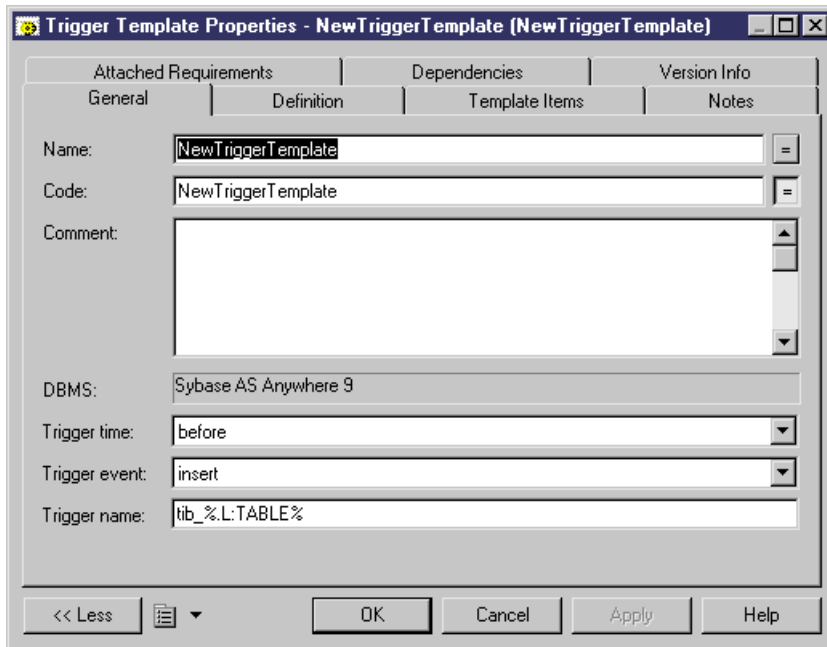
- a *Time* – relevant to an event. One of:
 - After
 - Before
- an *Event* – that can occur to a table row. One of:
 - Delete
 - Insert
 - Update
- *Code* – that performs the trigger action. The code may contain references to trigger template items, which are re-usable blocks of script.

You can create trigger templates and trigger template items in your DBMS definition file or in your model. These elements will be available as follows:

Trigger template or template item created in:	Available to
DBMS (shared definition file)	All models sharing the DBMS
DBMS (copied definition file)	Current model only
Model	Current model only



2. Click on one of the following tools:
 - *Create from DBMS trigger template* – opens a selection box listing all the trigger templates available in the current DBMS. Select a check box for the type of trigger template that you want to use as the basis for your new template and click OK to return to the trigger template list. The duplicate DBMS template has been added to the list.
 - *Add a Row* – adds a new blank template to the list.
3. Type a new name and code for the new template and click Apply to commit its creation.
4. Click the Properties tool to open the property sheet of the new trigger template:



5. Click the Definition tab and enter or modify the definition code. You can add trigger template items, use PDM variables and macros and various other tools available from the toolbar (see *SQL Editor Tools* on page 401).
6. You can also modify other of the trigger template's properties. For a full list of the properties available, see *Trigger Template Properties* on page 240.
7. Click OK in each of the dialog boxes.

If you have created DBMS trigger template, a confirmation box will appear, asking if you want to save the changes to the DBMS.

Click Yes to confirm the template creation.

PowerDesigner Pre-defined Trigger Templates

The pre-defined DBMS templates provided with PowerDesigner control referential integrity constraints for insert, update, and delete events. Depending on the current DBMS, there is a before and after event template for each trigger type.

You can modify the code of these pre-defined trigger templates, but they cannot be deleted or renamed.

The following templates types exist, but may vary by DBMS:

Insert Templates

Template type	Generates trigger/procedure executing...
InsertTrigger	With insert
BeforeInsertTrigger	Before insert
AfterInsertTrigger	After insert
InsertProc	When called by InsertTrigger
BeforeInsertProc	When called by BeforeInsertTrigger
AfterInsertProc	When called by AfterInsertTrigger

Update Templates

Template type	Generates trigger/procedure executing...
UpdateTrigger	With update
BeforeUpdateTrigger	Before update
AfterUpdateTrigger	After update
UpdateProc	When called by UpdateTrigger
BeforeUpdateProc	When called by BeforeUpdateTrigger
AfterUpdateProc	When called by AfterUpdateTrigger

Delete Templates

Template type	Generates trigger/procedure executing...
DeleteTrigger	With delete
BeforeDeleteTrigger	Before delete
AfterDeleteTrigger	After delete
DeleteProc	When called by DeleteTrigger
BeforeDeleteProc	When called by BeforeDeleteTrigger
AfterDeleteProc	When called by AfterDeleteTrigger

Modifying a Trigger Template

You can modify both your templates and those that are provided with PowerDesigner.

1. Open the trigger template property sheet in one of the following ways:
2. To modify a DBMS trigger template: select **Database > Edit Current DBMS** to open the DBMS definition file in the resource editor, and then click the Trigger Templates tab.
3. To modify a model trigger template: select **Model > Triggers > Trigger Templates** to open the List of User-Defined Trigger Templates.
4. Click a trigger template in the list, and then click the Properties tool to open its property sheet.
5. Click the Definition tab and modify the trigger definition code. You can add trigger template items, use PDM variables and macros and various other tools available from the toolbar. For more information, see *SQL Editor Tools* on page 401.
6. You can also modify other of the trigger template's properties. For a full list of the properties available, see *Trigger Template Properties* on page 240.
7. Click OK in each of the dialog boxes.

If you have created DBMS trigger template, a confirmation box will appear, asking if you want to save the changes to the DBMS.

Click Yes to confirm the template creation.

Trigger Template Properties

You can modify a trigger templates properties from its property sheet. The following sections detail the property sheet tabs that contain the properties most commonly entered for trigger templates.

The General tab contains the following properties:

Property	Description
Name	The name of the item which should be clear and meaningful, and should convey the item's purpose to non-technical users
Code	The technical name of the item used for generating code or scripts, which may be abbreviated, and should not generally include spaces
Comment	Descriptive label for the trigger template
DBMS	Current DBMS
Trigger time	Time attribute of the trigger template. The list displays the values defined in the trigger templates and template items of the current DBMS

Property	Description
Trigger event	Event attribute of the trigger template. The list displays the values defined in the trigger templates and template items of the current DBMS
Trigger name	Name of trigger associated with template
Applies to table triggers or view triggers	For those DBMS that support view triggers, it allows you to define if the trigger template applies to table or view triggers

Definition Tab

This tab contains a field for entering its definition code (see *Modifying Triggers* on page 228).

Template Items Tab

This tab list the template items that are defined in the trigger template and that will be generated when a trigger is generated from the template.

You can add any template item from the model or DBMS to the Trigger template definition by clicking an **Add Trigger Item** tool on the **Definition** tab, and selecting a trigger item. It is automatically added to this tab.

A template item that is deleted from this tab is not deleted from the trigger template definition. You can therefore limit the template items available for generation by removing template items from this tab, without having to remove them from the trigger template definition.

When you use Rebuild Triggers to automatically create triggers for selected tables, the template items that are listed on this tab are those available for generation. Whether they are generated or not depends on the following:

- Template items are generated in a trigger if they match the trigger implemented referential integrity defined for a reference attached to the table
- Template items are generated in a trigger if they are user-defined, regardless of trigger referential integrity constraints

Trigger Template Items (PDM)

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

In a generated trigger script, a template item calls a macro that implements a trigger referential integrity constraint or does any other updating work on tables in the database.

Example

A trigger template for Sybase Adaptive Server Anywhere 6 contains the `.InsertChildParentExist` template item, which corresponds to the following definition:

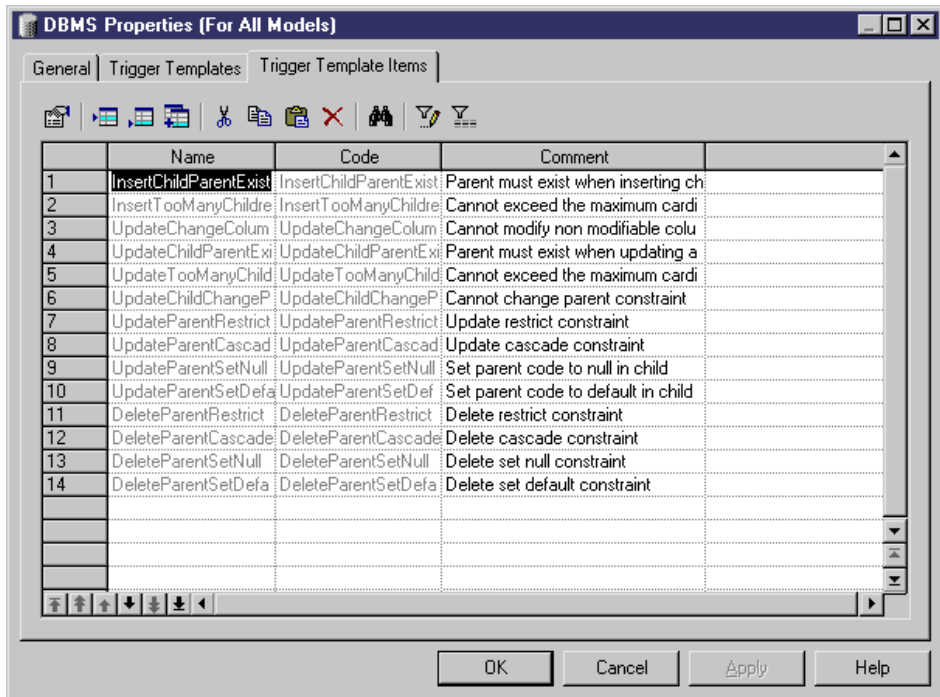
```
.FOREACH_PARENT()  
/* Parent "[%PQUALIFIER%]%PARENT%" must exist when inserting a child  
in "[%CQUALIFIER%]%CHILD%" */  
if (.JOIN("new_ins.%FK% is not null", "", " and", ") then")  
begin  
    set found = 0;  
    select 1  
        into found  
        from dummy  
    where exists (select 1  
                  from [%PQUALIFIER%]%PARENT%  
                  where .JOIN("%PK% = new_ins.%FK%", "and ", "", ");")  
    if found <> 1 then  
        message 'Error: Trigger(%TRIGGER%) of table [%QUALIFIER%]%TABLE%';  
        message '    Parent code must exist when inserting a child!';  
        signal user_defined_exception;  
    end if;  
end  
end if;  
.ENDFOR
```

Creating a Trigger Template Item

You usually create a template item when an existing template item is not suitable, or to create a repeatable block of code to do updating work on tables in the database.

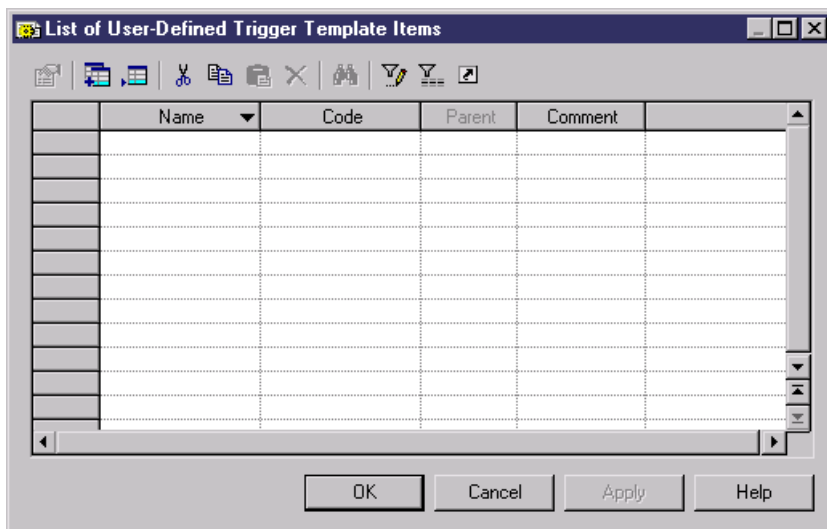
You can create a new trigger template in your DBMS definition file or as part of your model. You can begin by copying an existing template or write one from scratch.

1. To create a DBMS trigger template item: select **Database > Edit Current DBMS** to open the DBMS definition file in the resource editor, and then click the Trigger Template Items tab:



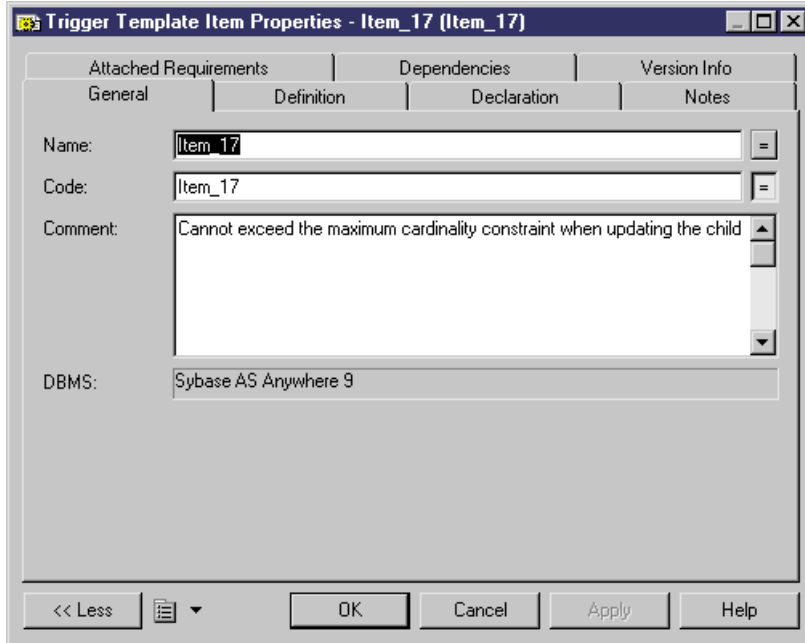
or

To create a model trigger template item: select **Model > Triggers > Trigger Template Items** to open the List of User-Defined Trigger Template Items:



- Click on one of the following tools:

- **Create from DBMS Trigger Item** – opens a selection box listing all the trigger template items available in the current DBMS. Select a check box for the type of item that you want to use as the basis for your new item and click OK to return to the trigger template item list. The duplicate DBMS template item has been added to the list.
 - **Add a Row** – adds a new blank template item to the list.
3. Type a new name and code for the new template item and click Apply to commit its creation.
 4. Click the Properties tool to open the property sheet of the new template item:



5. Click the Definition tab and enter or modify the definition code. You can use PDM variables and macros and various other tools available from the toolbar. For more information, see *SQL Editor Tools* on page 401.
6. You can also modify other of the trigger template item's properties. For a full list of the properties available, see *Trigger Template Properties* on page 240.
7. Click OK in each of the dialog boxes.

If you have created DBMS trigger template item, a confirmation box will appear, asking if you want to save the changes to the DBMS.

Click Yes to confirm the template creation.

PowerDesigner Pre-defined Trigger Template Items

PowerDesigner ships pre-defined template items for each pre-defined trigger template defined in the supported DBMS. The Rebuild Triggers function uses both pre-defined and user-defined trigger templates to automatically create triggers for selected tables.

In the pre-defined trigger templates, each pre-defined template item corresponds to a referential integrity constraint. Although a pre-defined template item is defined in a trigger template, it is only generated in a trigger script if it implements the trigger referential integrity defined for a reference.

Template items have the following generation conditions:

Template item is listed in...	Template item is...
Template Items tab of trigger property sheet	Available for generation
Template Items tab of trigger template property sheet	Generated

You can modify the code for these pre-defined template items, but they cannot be deleted or renamed.

The PowerDesigner pre-defined template items that are available depend on the current DBMS.

Insert Constraints

The template items below implement referential integrity in insert trigger templates.

Template item	Integrity constraint	Description
DeclInsertChildParentExist InsertChildParentExist	Mandatory parent	Parent must exist when inserting a child
DeclInsertTooManyChildren InsertTooManyChildren	Cannot exceed maximum cardinality constraint	Cannot insert a child if maximum cardinality has been reached
DeclInsertSequenceColumn InsertSequenceColumn	Select value in sequence list for column	Select a value for the column from a list of sequences

Update Constraints

The template items below implement referential integrity in update trigger templates.

Template item	Integrity constraint	Description
DeclUpdateChildParentExist UpdateChildParentExist	Mandatory parent	Parent must exist when updating a child
DeclUpdateChildChangeParent UpdateChildChangeParent	Change parent not allowed	Cannot modify parent code in child
DeclUpdateParentRestrict UpdateParentRestrict	Restrict on update	Cannot modify parent if child exists
DeclUpdateParentCascade UpdateParentCascade	Cascade on update	Modify parent code in all children
DeclUpdateChangeColumn UpdateChangeColumn	Non-modifiable column	Cannot modify column
DeclUpdateParentSetNull UpdateParentSetNull	Set null on update	Set parent code to null in all children
DeclUpdateParentSetDefault UpdateParentSetDefault	Set default on update	Set parent code to default in all children
DeclUpdateTooManyChildren UpdateTooManyChildren	Cannot exceed maximum cardinality constraint	Cannot update a child if maximum cardinality has been reached

Delete Constraints

The template items below implement referential integrity in delete trigger templates.

Template item	Integrity constraint	Description
DeclDeleteParentRestrict DeleteParentRestrict	Restrict on delete	Cannot delete parent if child exists
DeclDeleteParentCascade DeleteParentCascade	Cascade on delete	Delete parent code in all children
DeclDeleteParentSetNull DeleteParentSetNull	Set null on delete	Delete in parent sets child to null

Template item	Integrity constraint	Description
DeclDeleteParentSetDefault DeleteParentSetDefault	Set default on delete	Delete in parent sets child to default

Constraint Messages

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

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

Modifying a Trigger Template Item

You can modify both your template items and those that are provided with PowerDesigner.

- Open the trigger template item property sheet in one of the following ways:
 - To modify a DBMS trigger template item: select **Database > Edit Current DBMS** to open the DBMS definition file in the resource editor, and then click the Trigger Template Items tab.
 - To modify a model trigger template item: select **Model > Triggers > Trigger Templates Items** to open the List of User-Defined Trigger Template Items.
- Click a trigger template item in the list, and then click the Properties tool to open its property sheet.
- Click the Definition tab and modify the trigger definition code. You can use PDM variables and macros and various other tools available from the toolbar (see *SQL Editor Tools* on page 401).
- You can also modify other of the trigger template item's properties. For a full list of the properties available, see *Trigger Template Item Properties* on page 247.
- Click OK in each of the dialog boxes.

If you have created DBMS trigger template item, a confirmation box will appear, asking if you want to save the changes to the DBMS.

Click Yes to confirm the template creation.

Trigger Template Item Properties

You can modify an object's properties from its property sheet. To open a trigger template item property sheet, access it via the List of User-Defined Trigger Template Items List or the

DBMS Properties window. The following sections detail the property sheet tabs that contain the properties most commonly entered for trigger template items.

The General tab contains the following properties:

Property	Description
Name	The name of the item which should be clear and meaningful, and should convey the item's purpose to non-technical users
Code	The technical name of the item used for generating code or scripts, which may be abbreviated, and should not generally include spaces
Comment	Descriptive label for the template item
DBMS	Current DBMS

The following tabs are also available:

- Definition - allows you to enter the SQL code for the template item. For information about the tools available, see *SQL Editor Tools* on page 401.
- Declaration - contains the declaration for the template item in trigger scripts. For information about the tools available, see *SQL Editor Tools* on page 401.

Stored Procedures and Functions (PDM)

You can define stored procedures and functions for any DBMS that supports them.

A stored procedure is a precompiled collection of SQL statements stored under a name and processed as a unit. Stored procedures are stored within a database; can be executed with one call from an application; and allow user-declared variables, conditional execution, and other programming features.

The use of stored procedures can be helpful in controlling access to data (end-users may enter or change data but do not write procedures), preserving data integrity (information is entered in a consistent manner), and improving productivity (statements in a stored procedure only need to be written one time).

A user-defined function is a form of procedure that returns a value to the calling environment for use in queries and other SQL statements.

Creating a Stored Procedure or Function

You can create a stored procedure or function in various ways.

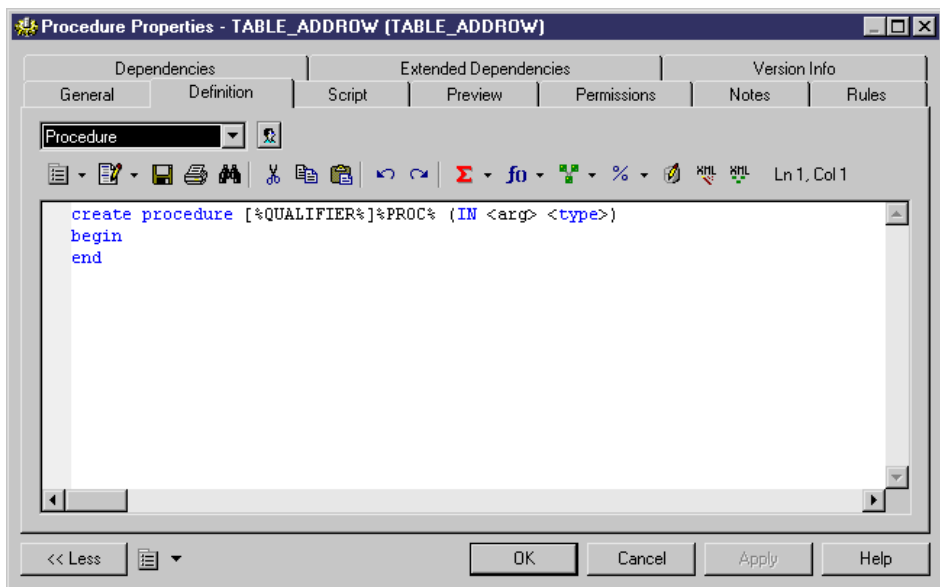
- Use the Procedure tool in the diagram Palette
- Open the Procedures tab in the property sheet of a table, and click the Add a Row tool
- Select **Model > Procedures** to access the List of Procedures, and click the Add a Row tool

- Right-click the model or package in the Browser, and select **New > Procedure**

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

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

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 258).
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 401).
7. You can also modify the procedure's other properties. For a full list of the properties available, see *Procedure Properties* on page 250.
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

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

The General tab contains the following properties:

Property	Description
Name	Specifies the name of the item which should be clear and meaningful, and should convey the item's purpose to non-technical users.
Code	Specifies the technical name of the item used for generating code or scripts, which may be abbreviated, and should not generally include spaces.
Comment	Specifies a descriptive label for the procedure.
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined.
Owner	Specifies the name of the procedure owner.
Table	Specifies the table to which the procedure is attached. Use the tools to the right of the list to create, browse for, or view the properties of the currently selected table.

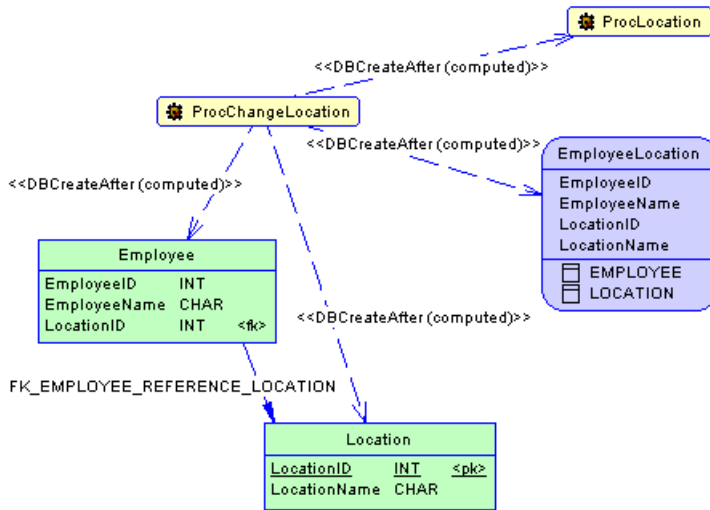
The following tabs are also available:

- Definition - allows you to enter the SQL code for the procedure. For information about the tools available, see *SQL Editor Tools* on page 401.

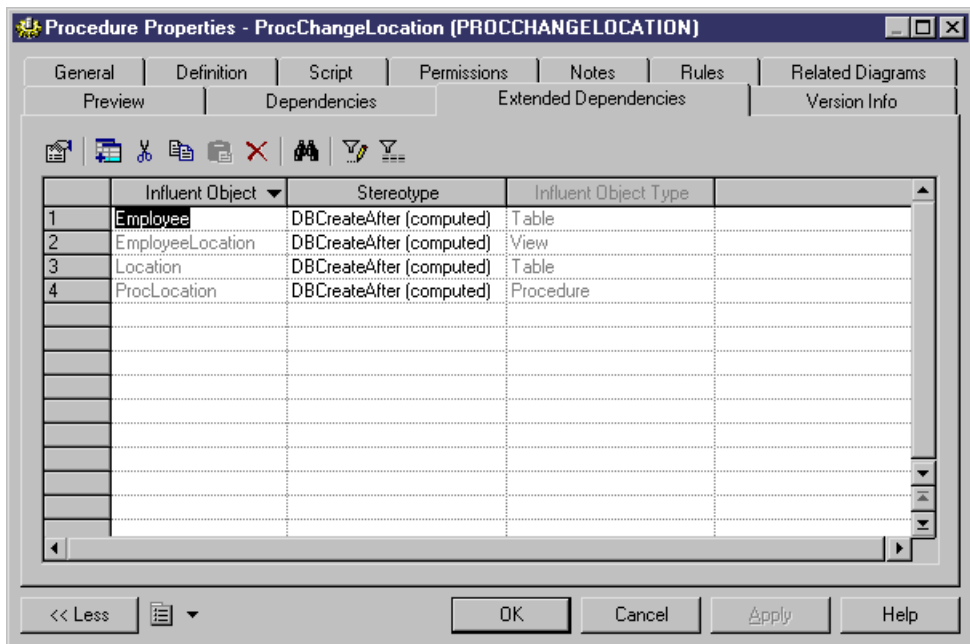
Tracing Trigger and Procedure Dependencies

When you write a trigger or procedure, PowerDesigner automatically creates dependencies to any table, view, procedure, or database package referenced in the code. These dependencies are taken into account when performing an impact analysis prior to deleting the trigger or procedure or objects on which they depend. For procedures, if the procedure has a symbol in your diagram, then any dependencies will be shown graphically by way of arrows linking the procedure to these objects.

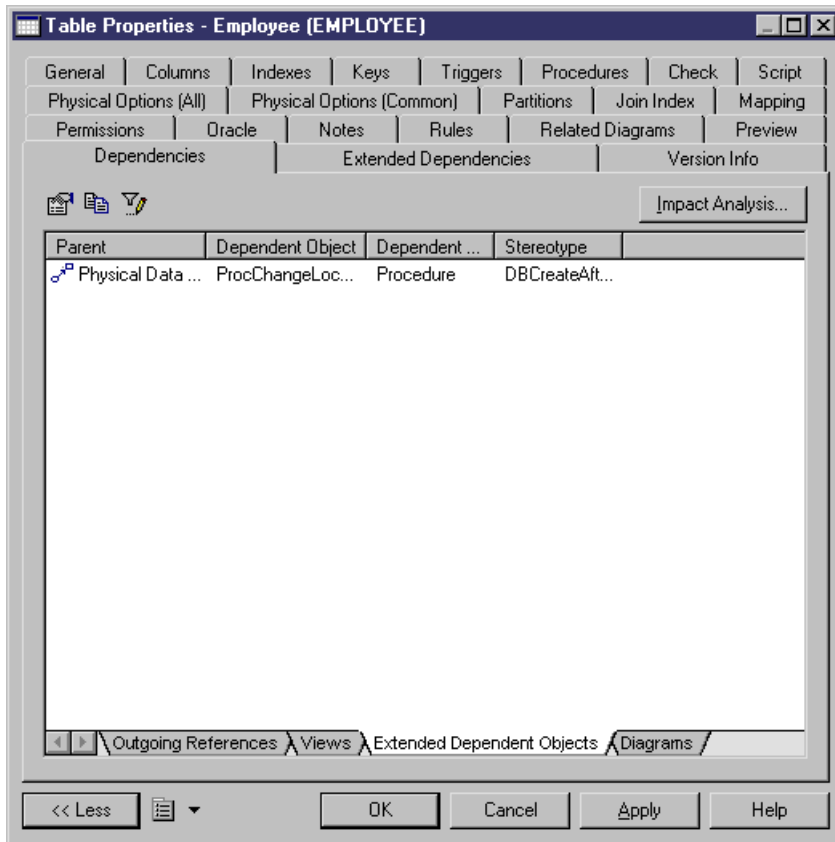
The diagram below shows a procedure, ProcChangeLocation, which is dependent on a number of other objects:



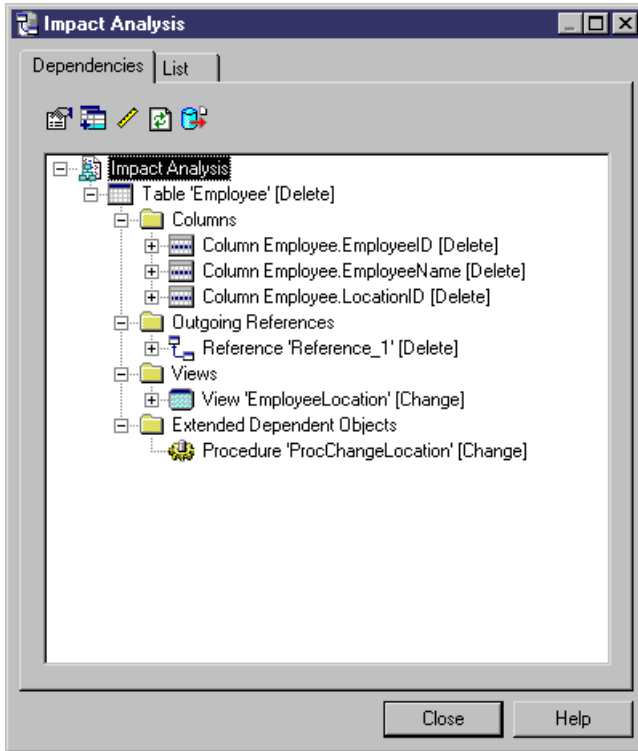
The Extended Dependencies tab of the trigger or procedure property sheet lists the objects upon which it depends. The stereotype <<DBCreateAfter (computed)>> specifies that PowerDesigner has determined that the trigger or procedure can only be created after these objects.



The Dependencies tab of the Employee table property sheet shows that ProcChangeLocation is dependent upon it.

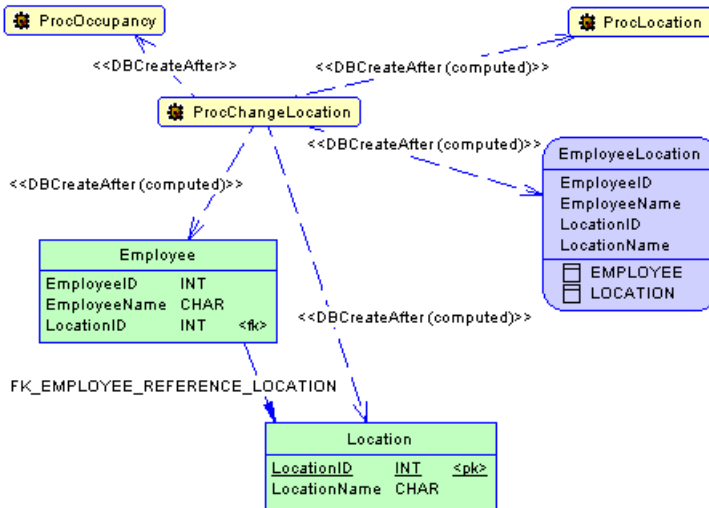


If you were to perform an impact analysis prior to deleting the Employee table, you would be warned of the dependency of the procedure upon it.

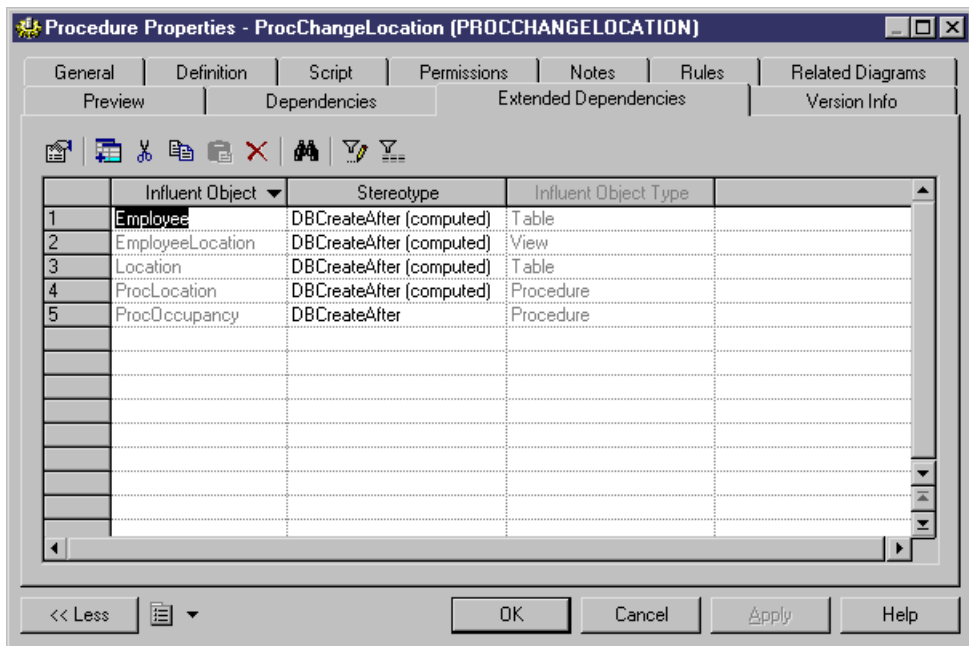


Creating Procedure Dependencies Manually

Since procedures have diagram symbols, you can manually add dependencies for them. To do this, use the Link/Extended Dependency tool in the palette. In the diagram below, I have added a dependency from ProcChangeLocation to a new procedure, ProcOccupancy:



Since ProcOccupancy is not directly referenced in ProcChangeLocation, I needed to manually add the stereotype <<DBCreateAfter>> to the dependency in the ProcChangeLocation property sheet Extended Dependencies tab:



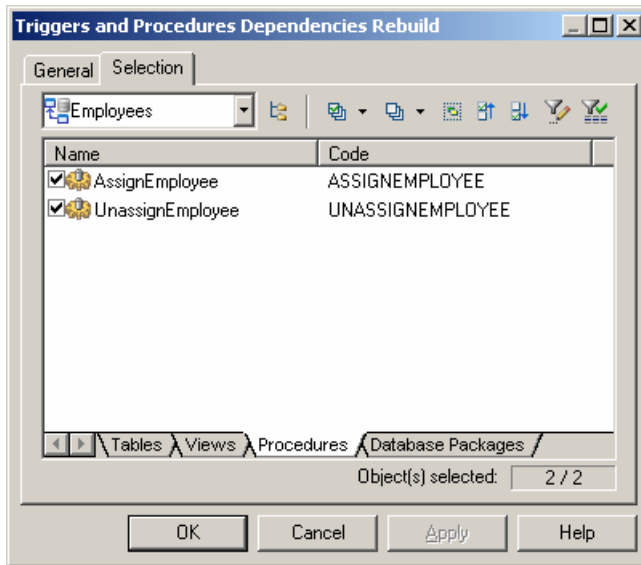
Rebuilding Trigger and Procedure Dependencies

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.

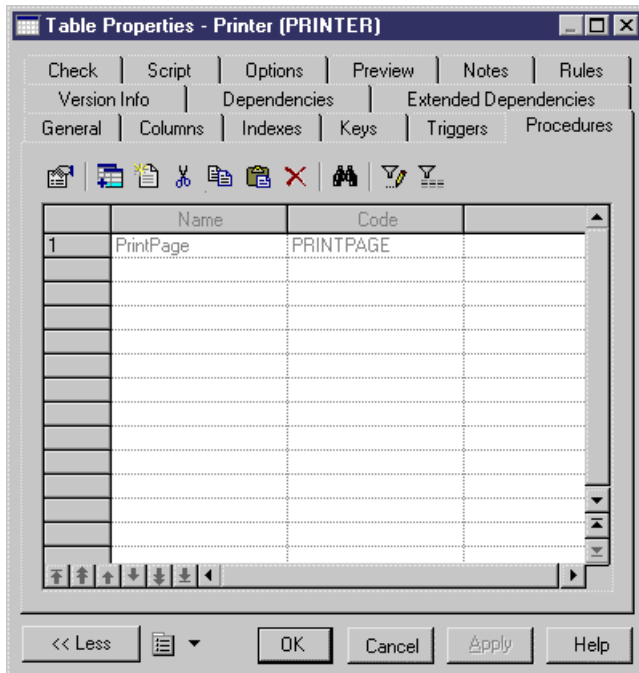
When you generate an OOM from a PDM, the procedures attached to a table become operations with the <<procedure>> stereotype in the generated class. By attaching procedures to tables, you are able to define class operations in the generated OOM.

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.

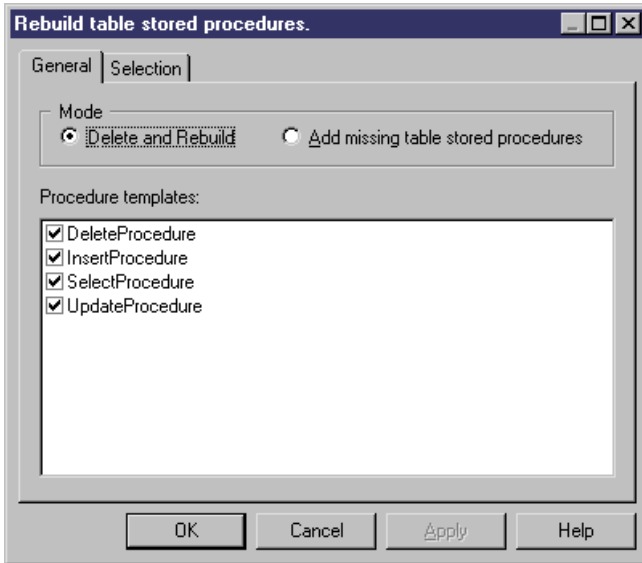


3. Click OK.

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 create procedures in a modular reusable fashion.

PowerDesigner provides certain basic procedure templates and also allows you to write your own. You can create procedure templates in your DBMS definition file or in your model. These elements will be available as follows:

Procedure template created in:	Available to
DBMS (shared definition file)	All models sharing the DBMS
DBMS (copied definition file)	Current model only

For more information on the DBMS share and copied definition files, see *Generating a Database from a PDM* on page 345.

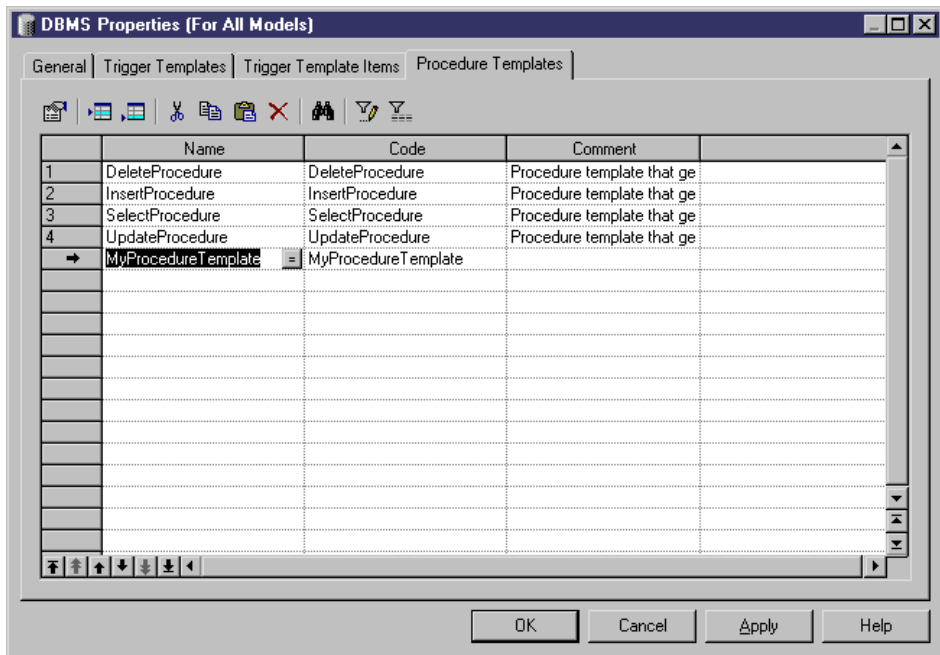
Note: By modifying a DBMS template, you modify its definition in the current DBMS. You should only modify a DBMS if you have already made a copy of the DBMS definition file provided with PowerDesigner.

For more information on working with DBMS resource files, see the DBMS Resource File Reference chapter of the *Customizing and Extending PowerDesigner* manual.

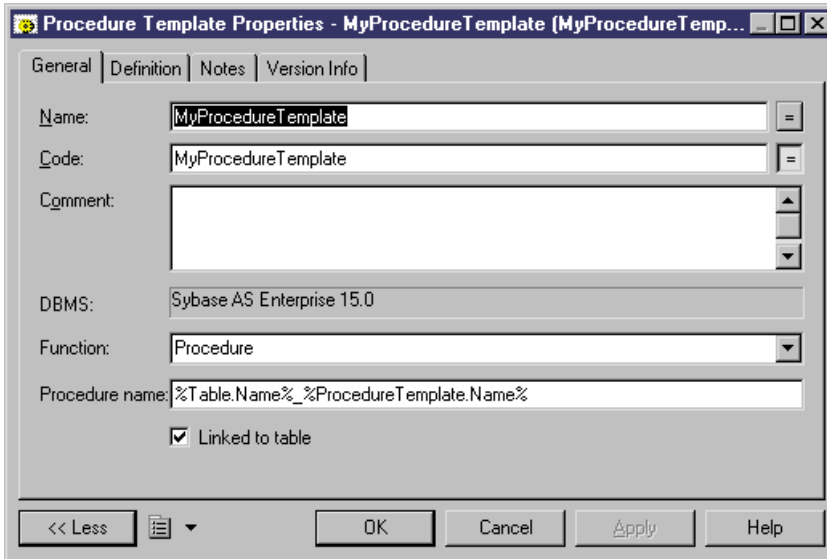
Creating a Procedure Template

You can create a new procedure template in your DBMS definition file or as part of your model. You can begin by copying an existing template or write one from scratch.

1. Select **Database > Edit Current DBMS** to open the DBMS definition file in the resource editor, and then click the Procedure Templates tab:



2. Click the Add a Row tool to add a new blank template to the list.
3. Type a new name and code for the new template and click Apply to commit its creation.
4. Click the Properties tool to open the property sheet of the new procedure template:



5. Click the Definition tab and enter or modify the definition code. You can use PDM variables and macros and various other tools available from the toolbar (see *SQL Editor Tools* on page 401).
6. You can also modify other of the procedure template's properties. For a full list of the properties available, see *Procedure Template Properties* on page 261.
7. Click OK in each of the dialog boxes.

A confirmation box will appear, asking if you want to save the changes to the DBMS.

Click Yes to confirm the template creation.

PowerDesigner Pre-defined Procedure Templates

The pre-defined DBMS templates provided with PowerDesigner generate SQL insert, delete, update, and select statement procedures.

You can modify the code of these pre-defined procedure templates, but they cannot be deleted or renamed.

The following templates, which create procedures linked to tables, are provided for those databases that support them:

Template type	Generates procedure executing...
DeleteProcedure	a SQL Delete statement
InsertProcedure	a SQL Insert statement
SelectProcedure	a SQL Select statement

Template type	Generates procedure executing...
UpdateProcedure	a SQL Update statement

Modifying a Procedure Template

You can modify both your templates and those that are provided with PowerDesigner.

1. Select **Database > Edit Current DBMS** to open the DBMS definition file in the resource editor, and then click the Procedure Templates tab.
2. Click a procedure template in the list, and then click the Properties tool to open its property sheet.
3. Click the Definition tab and modify the procedure definition code. You can use PDM variables and macros and various other tools available from the toolbar. For more information, see *SQL Editor Tools* on page 401.
4. You can also modify other of the procedure template's properties. For a full list of the properties available, see *Procedure Template Properties* on page 261.
5. Click OK in each of the dialog boxes.

A confirmation box will appear, asking if you want to save the changes to the DBMS.

Click Yes to confirm the template creation.

Procedure Template Properties

You can modify a procedure template's properties from its property sheet. The following sections detail the property sheet tabs that contain the properties most commonly entered for procedure templates.

The General tab contains the following properties:

Property	Description
Name	The name of the item which should be clear and meaningful, and should convey the item's purpose to non-technical users.
Code	The technical name of the item used for generating code or scripts, which may be abbreviated, and should not generally include spaces.
Comment	Descriptive label for the procedure template.
DBMS	Current DBMS.
Function	Specifies whether the template defines procedures or functions.
Procedure Name	Specifies the format of the resulting procedure names.
Linked to table	Specifies whether the resulting procedure will be linked to a table.

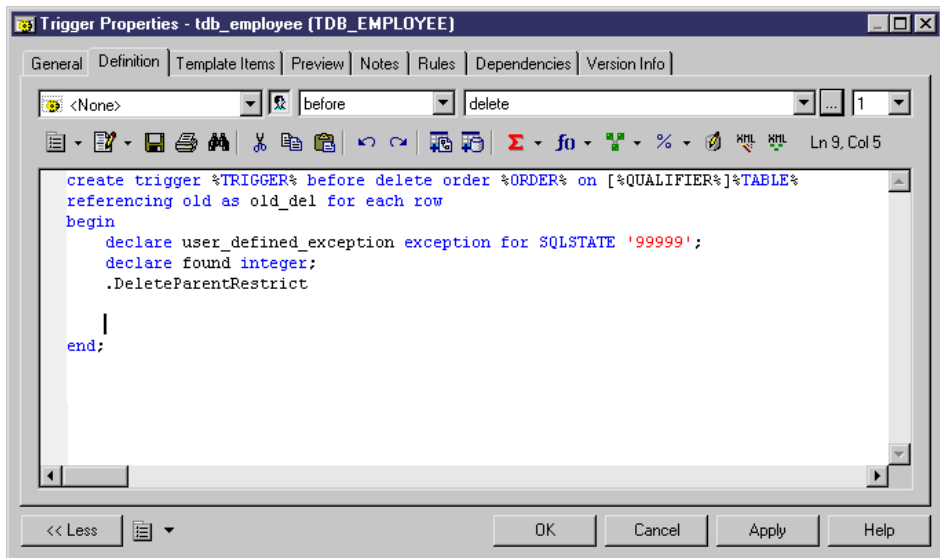
The following tabs are also available:

- Definition - contains a field for entering its definition code. For information about editing this code, see *Modifying a procedure template* on page 261.

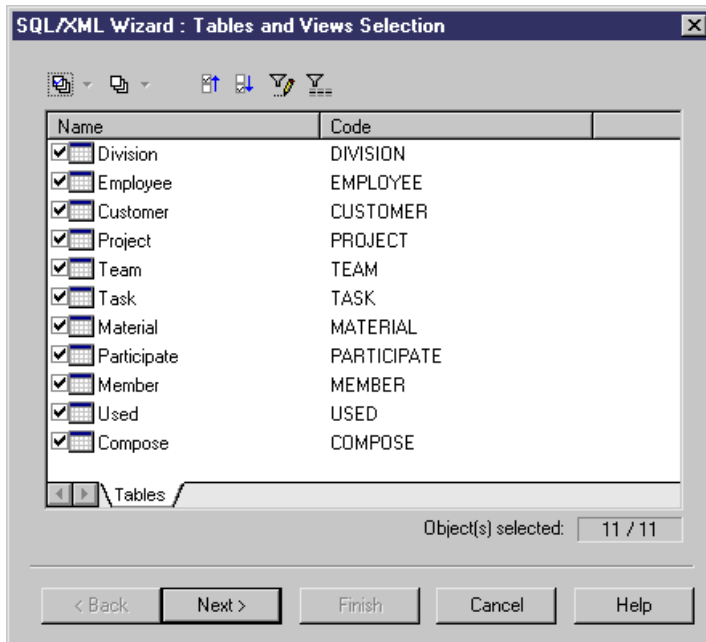
Creating SQL/XML Queries with the Wizard

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.

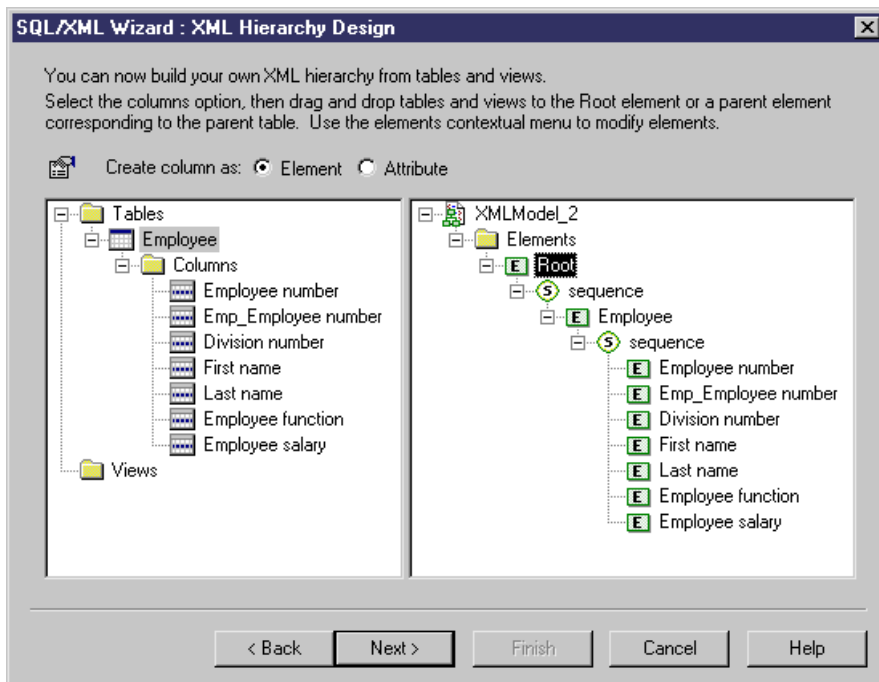
1. Open the trigger property sheet, click the *Definition* tab and position the cursor in the trigger definition where you want to insert the SQL/XML query:



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:



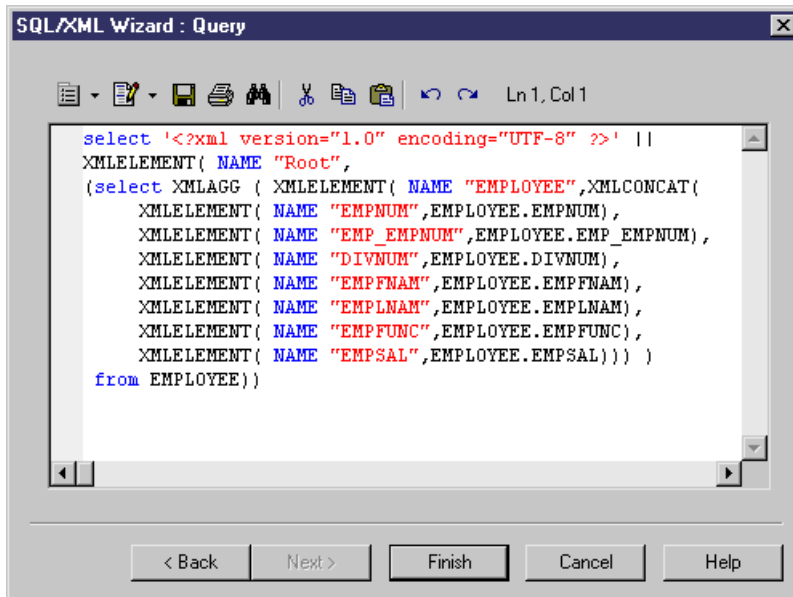
On this tab, you construct the XML hierarchy that you want to generate:

- 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.

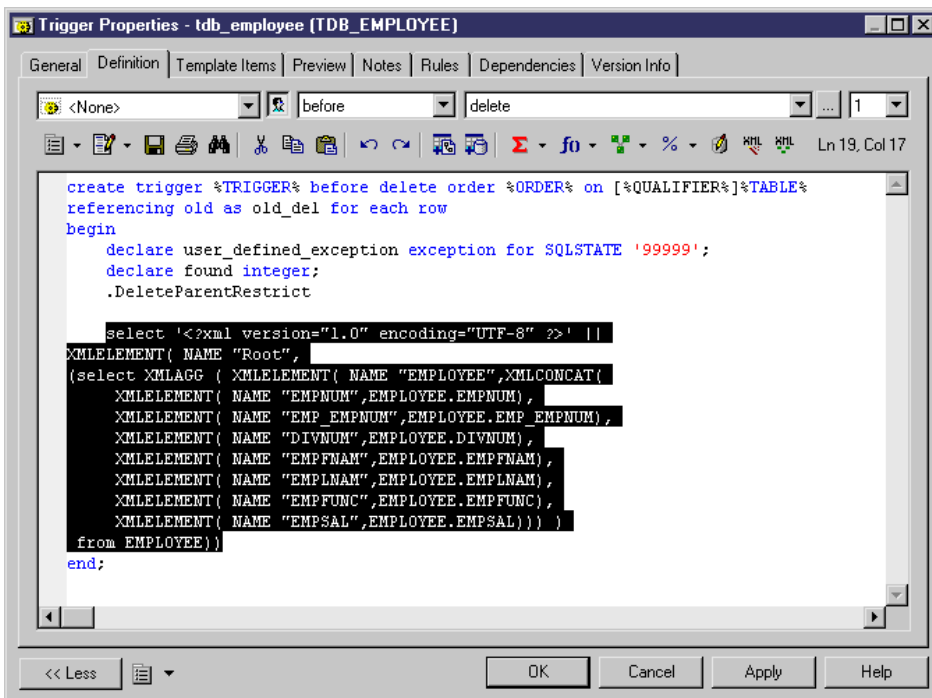
4. You can build your XML hierarchy using the following techniques:

- 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 *Newobject* from the contextual menu.
- Delete an XML node by right-clicking it and selecting Delete from the contextual menu.

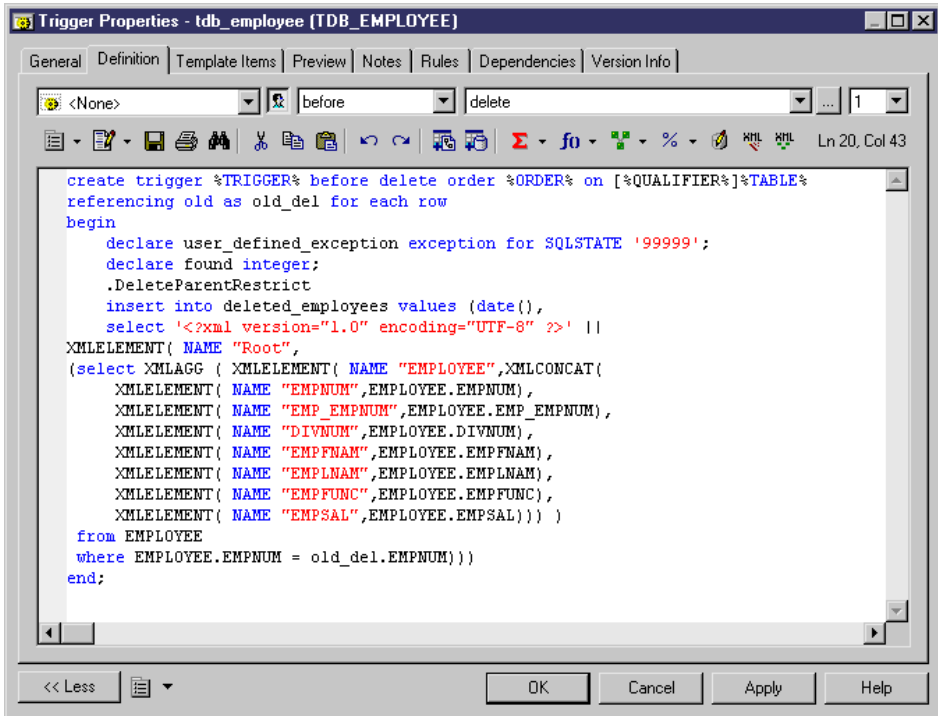
5. When you have finished building your hierarchy, click Next to go to the Query tab:



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 346.

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 350.

3. [optional] Click the Selection tab and select the Table or Procedure subtab at the bottom of the tab. Select the tables or procedures that you want to generate for. Note that if you want to generate a trigger script for tables owned by a particular owner, you can select an owner from the Owner list.
4. Click OK to begin the generation.

Defining a Generation Order for Stored Procedures

You can use extended dependencies to define an order in the generation of stored procedures.

Extended dependencies are free links between PDM objects. These links help to make object relationships clearer between model objects. Usually, these links are not interpreted and checked by PowerDesigner as they are meant to be used for documentation purposes only.

However, if you assign the <<*DBCreateAfter*>> stereotype to an extended dependency between stored procedures, it will be analyzed during generation.

The procedure from which you start the extended dependency is the dependent procedure and the procedure at the other end of the link is the influent procedure. The influent procedure will be generated before the dependent procedure.

If you create a reflexive and/or circular set of extended dependencies with the <<*DBCreateAfter*>> stereotype, an error message is displayed during the check model. If you choose to ignore this error, the stored procedures will be generated in alphabetical order, without taking into account the generation order, which could cause errors in the creation of stored procedures in the database.

For example, A publisher may decide to sell certain books at a reduced rate (15%) when a customer's order is above 10 000\$.

In this model, one stored procedure GENERAL CHECK globally verifies orders: check books availability, check the order amount, check if discount rate is calculated, and applies discount rate to order bill. During the execution of this stored procedure, the procedure DISCOUNT CALC is called to calculate the 15% discount rate. It is important to generate GENERAL CHECK before DISCOUNT CALC; you can define an extended dependency to set the generation order between these two objects. To do so, you have to open the property sheet of the dependent stored procedure, click the Extended Dependencies tab and create an extended dependency with the <<*DBCreateAfter*>> stereotype with the influent stored procedure.

Creating and Generating User-defined Error Messages

You can create user-defined error messages. The error messages are stored in a message table which you need to create in your database. When you select trigger generation parameters, you can choose to generate an error message from this table.

In the generated trigger script, the message table is called in a SELECT command. If an error number in the script corresponds to an error number value in the table column, then the standard error message is replaced by the message defined in the table.

Creating a Message Table

You create a message table which stores error message information.

1. Create a table with columns to store the following information:

Column to store...	Description
Error number	Number of the error message that is referenced in the trigger script
Message text	Text of message

2. Generate the table in your database.

3. Select **Database > Execute SQL**.

A dialog box asks you to identify a data source and connection parameters.

4. Select a data source and fill in connection parameters.

5. Click Connect.

An SQL query editor box is displayed.

6. Type an SQL statement to insert a message number and text in the appropriate columns. You can use the following format for example:

```
insert into table values (error number, 'error message')
```

```
insert into ERR_MSG values (1004, 'The value that you are trying to
insert does not exist in the referenced table')
```

7. Click Execute.

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

8. Click OK.

You return to the SQL query box.

9. Click Close.

Generating a User-defined Error Message

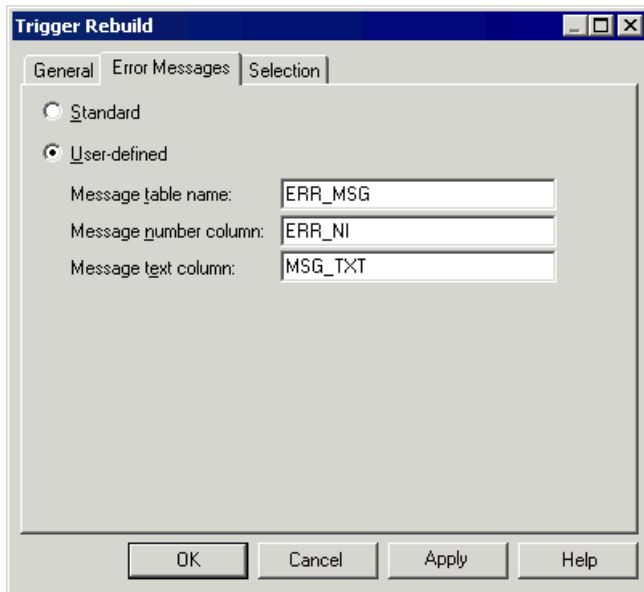
You can choose to generate a user-defined error message from the trigger generation parameters box.

1. Select **Tools > Rebuild Objects > Rebuild Triggers**.

The Rebuild Triggers dialog box is displayed.

2. Click the Error Messages tab, and select the User-defined radio button.
3. Type the name of the table that contains the error message in the Message Table Name box.
4. Type the name of the column that contains the error number in the Message Number box.
5. Type the name of the column that contains the error message text in the Message Text column.

Below is an example of the details for a table called ERR_MSG.



6. Click the General tab and select the mode and triggers to create.
7. Click the Selection tab and select the tables for which you want to create triggers.

For more information on rebuilding triggers, see section *Rebuilding Triggers* on page 226.

8. Click OK.

The trigger rebuilding process is shown in the Output window.

Select **Database > Generate Database** to open the Database Generation window.

9. Select generation parameters as required.
10. Click OK.

For information on selecting trigger generation parameters, see section *Generating Triggers and Procedures* on page 266.

CHAPTER 6 Building Web Services

Web services are applications stored on web servers that you can access remotely through standard web protocols (HTTP, SOAP) and data formats (HTML, XML...), whatever the systems and programming languages.

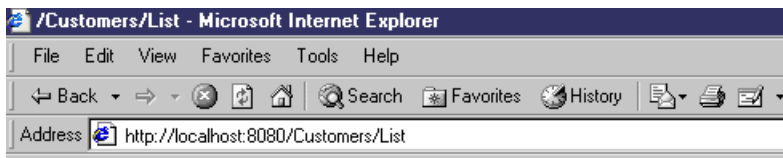
In SOAP requests, queries are encapsulated into services, whereas in HTTP requests, operations are invoked directly. In PowerDesigner, you can design web services for both protocols.

PowerDesigner supports web services for the following databases:

- Sybase Adaptive Server Anywhere 9 and over
- Sybase Adaptive Server Enterprise 15 and over
- Sybase IQ12.6 and over
- IBM DB2 v8.1 and over

Web services can be used to simplify access to databases. If you use web services to query databases, you no longer need drivers to communicate with those databases.

In the following example, you can see the result of an HTTP request for a database web service:



/Customers/List

Customer custid	Customer custname	Customer custaddr
1	Finley	Chicago
2	Takashi	Tokyo
3	Smith	London
4	Dupont	Paris

AdaptiveServerAnywhere/9.0.0.1108

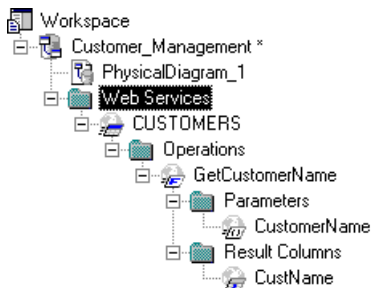
Web services are made of a set of operations. Each operation contains a SQL query for retrieving data from a database.

Web Services (PDM)

In PowerDesigner, web services are made of web operations which themselves contain web parameters and result columns:

- *Web operations* specify the SQL statements used to retrieve data from databases
- *Web parameters* are the parameters which appear in the SQL statements
- *Result columns* are the columns in which the results are displayed

These objects have no symbols, but they appear in the Browser tree view.



This structure is compatible with the definition of web services in the supported databases.

Import Web Service as Service Provider

You can import a web service as a service provider into a Business Process Model (BPM) to define the links between a concrete implementation of service interfaces and operations and their abstract definition.

For more information, see "Importing a service provider from an OOM or a PDM" in the Building Business Process Diagrams chapter of the *Business Process Modeling* guide.

Web Services in Sybase ASA, ASE, and IQ

PowerDesigner supports web services for ASA 9 and over, ASE 15 and over and IQ 12.6 and over.

You must specify the type of the web service in the Service type list on the General tab of its property sheet (see *Web service properties* on page 275).

Web services can be invoked by either of two protocols:

- A web service invoked via an HTTP request can have a RAW, HTML or XML type. When several web services concern the same table in a database, their name usually starts with the name of the table, followed by a slash and a specific name identifying the query (e.g. Customer/List, Customer/Name). In that case, the name of the table is called the *local path* (which is defined on the General tab of the web service property sheet).

PowerDesigner treats HTTP web operations which share a local path as belonging to the web service with that local path name.

- [ASA and IQ only] A web service invoked in a SOAP request can have a SOAP or a DISH type.

PowerDesigner treats SOAP web services for these databases as web operations belonging to a DISH web service.

Implementation (SQL Statement)

When you create a web service, you must type a SQL statement to select which data you want to retrieve from the database in the Implementation tab of the property sheet of its web operation(s). For DISH web services, SQL statements are defined in the SOAP web services bearing their prefix name.

Web Services in IBM DB2

PowerDesigner supports web services for IBM DB2 v8.1 and over.

In IBM DB2, web services are defined by Document Access Definition Extension (DADX) files.

For more information about generating DADX files, see *Generating web services for IBM DB2 v8.1* on page 282.

A DADX file specifies a web service through a set of *operations* defined by *SQL statements* or Document Access Definition (DAD) files, which specify the mapping between XML elements and DB2 tables.

For more information on DAD files, see "Generating a DAD file for IBM DB2", in the Exchanging Data with Databases Supporting XML" chapter of the *XML Modeling guide*.

Creating a Web Service

You can create a web service in any of the following ways:

- Select **Model > Web Services** to access the List of Web Services, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select **New > Web Service**.

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

Web Service Properties

You can modify an object's properties from its property sheet. To open a web service property sheet, double-click its Browser entry in the Web Services folder. The following sections detail the property sheet tabs that contain the properties most commonly entered for web services.

The General tab contains the following properties:

Property	Description
Name	Name of the web service. Used in URIs to access the web service. It can neither start with a slash nor contain two consecutive slashes
Code	Code of the web service
Comment	Descriptive label for the web service
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined
Local path	Name prefixing the web service. If you type a path, the <i>User-Defined</i> tool (beside the Local path box) is pushed-in. Click the User-Defined tool to recover the original path. The default value is the name of the web service
Service type	<p>[ASA, ASE, and IQ only] Specifies the type of web service. You can choose from:</p> <ul style="list-style-type: none"> • <i>DISH</i> - [ASA and IQ only] acts as a proxy for a group of SOAP services and generates a <i>WSDL</i> file for each of its SOAP services. When you create a DISH service, you must specify a <i>prefix</i> name (on the Extended Attributes tab) for all the SOAP services to which the DISH service applies. PowerDesigner treats SOAP web services as <i>Web operations</i> (see <i>Web Operations (PDM)</i> on page 277) of DISH web services. • <i>HTML</i> - [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). • <i>RAW</i> - the result of the SQL statement or procedure is sent without any additional formatting. • <i>SOAP</i> - [ASE only] generates a <i>WSDL</i> (Web Services Description Language) file. • <i>XML</i> - the result of the SQL statement or procedure is sent in XML. By default, the result is converted into XML RAW format.

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 277).
- Sybase - [ASA/SQL Anywhere, ASE, and IQ] Includes Sybase-specific properties (see *Chapter 20, Sybase SQL Anywhere* on page 577)
- 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 is a child object of a web service. It allows you to define the SQL statement of a web service and to display its parameters and result columns.

Creating a Web Operation

You can create a web operation as follows:

- Open the Operations tab in the property sheet of a web service, and click the Add a Row tool.

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

Web Operation Properties

You can modify an object's properties from its property sheet. To open a web operation property sheet, double-click its Browser entry in the Operations folder beneath the parent web service. The following sections detail the property sheet tabs that contain the properties most commonly entered for web operations.

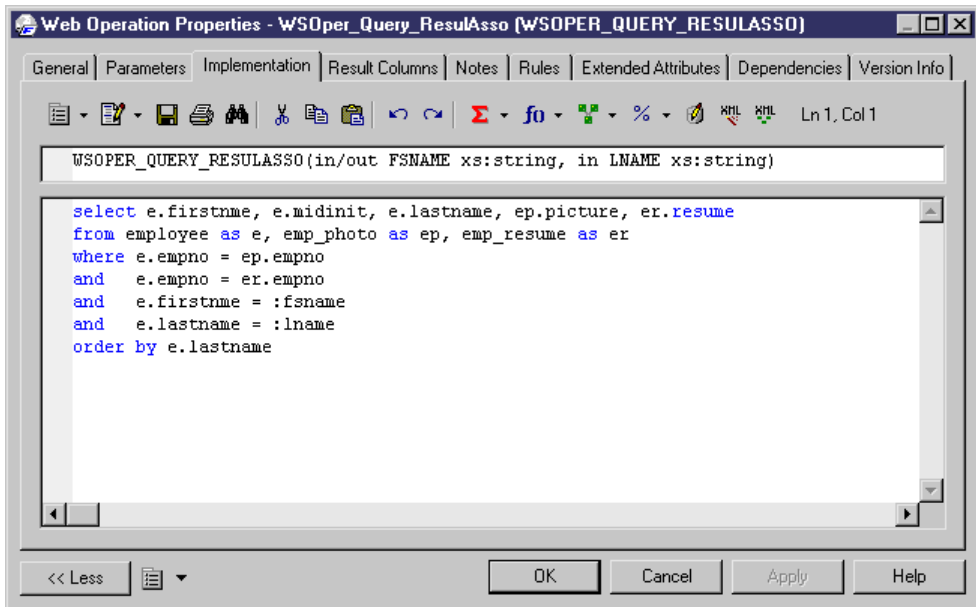
The General tab contains the following properties:

Property	Description
Name	Name of the web operation. In URIs, it comes after the name of the web service followed by a slash. It can neither start with a slash nor contain two consecutive slashes
Code	Code of the web operation

Property	Description
Comment	Descriptive label for the web operation
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined
Web Service	Code of the web service containing the web operation. Click the Properties tool to open the web service property sheet
Owner	[ASE 15 only] Specifies the owner of the operation.
Operation Type	<p>[IBM DB2 only] Specifies the type of operation. You can choose from the following:</p> <ul style="list-style-type: none"> • <i>call</i> - invokes a stored procedure with parameters and result columns for the web operation • <i>query</i> - retrieves relational data using the SQL select statement in the Implementation tab • <i>retrieveXML</i> - 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 • <i>storeXML</i> - 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 • <i>update</i> - executes the SQL update statement with optional parameters. Parameters can be created from the Parameters tab in the web operation property sheet

Implementation Tab

The **Implementation** tab contains the SQL statement of the Web operation. For information about the tools on this tab, see *SQL Editor Tools* on page 401.



Security Tab

This tab is available for ASA/SQL Anywhere and IQ, 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
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:

- **Parameters** - Lists the parameters associated with the Web operation (see *Web Parameters (PDM)* on page 280), which are part of the SQL statement defined on the Implementation tab. You can create parameters on this tab, or reverse engineer them from a web service (ASA, ASE, and IQ only). In addition to the standard list tools, you can 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.
- **Result Columns** - Lists the result columns associated with the Web operation (see *Web Operation Result Columns* on page 280). In addition to the standard list tools, you can 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.

- Sybase - [ASE only] Displays Sybase-specific options (see *Chapter 18, Sybase ASE* on page 549).

Web Operation Result Columns

Result columns are sub-objects of web operations. They are part of the SQL statement defined in the Implementation tab of a web operation property sheet, and belong to a table in the target database. They are listed in the Result Columns tab of a web operation property sheet.

The General tab of a result column property sheet displays the following properties:

Property	Description
Name	Name of the result column
Code	Code of the result column
Comment	Descriptive label for the result column
Data Type	[IBM DB2 only] Select an XML schema data type from the list, or click the <i>Select Object</i> tool to open a selection dialog box where you select a global element in an XML model open in the workspace
Is element	[IBM DB2 only] Checked and greyed when a global element is attached to a result column

Web Parameters (PDM)

Web parameters are child objects of web operations. They are part of the SQL statement defined in the Implementation tab of a web operation property sheet. They are listed in the Parameters tab of a web operation property sheet.

Creating a Web Parameter

You can create a web parameter in any of the following ways:

- Open the Parameters tab in the property sheet of a web operation, and click the Add a Row tool.
- Right-click a web operation in the Browser, and select **New > Web Parameter**.

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

Web Parameter Properties

You can modify an object's properties from its property sheet. To open a web parameter property sheet, double-click its Browser entry in the Parameters folder beneath its parent web operation.

The General tab contains the following properties:

Property	Description
Name	Name of the web parameter.
Code	Code of the web parameter.
Comment	Descriptive label for the web parameter.
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined.
Web Operation	Name of the web operation containing the web parameter.
Parameter Type	Select <i>in</i> if you want the web parameter to be an input parameter. Select <i>in/out</i> if you want the web parameter to be both an input and output parameter. Select <i>out</i> 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 <i>Select Object</i> tool to open a selection dialog box where you select a global element in an XML model open in the workspace. [For ASE] Select a datatype from the list.
Is element	[IBM DB2 only] Checked and greyed when a global element is attached to a web parameter.

Testing Web Services

PowerDesigner provides a method for testing web services within the model environment. You must be connected to the appropriate database.

1. Right-click the browser entry for a web service of type DISH or SOAP and select Show WSDL

or:

Right-click the browser entry for a web service operation belonging to a web service of another type and select Test Web Service Operation from the contextual menu.

2. Review the generated URL and then click OK.

For a web service of type SOAP, the WSDL file will be displayed in your browser

or

For a web service of type RAW, the result will be displayed in your browser

Generating Web Services

You generate web services in order to implement them on target databases.

Generating Web Services for Sybase ASA, ASE, and IQ

You can generate database web services to a script or to a live database connection.

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 346.

2. [optional] Click the Options tab and click on Web Service in the left-hand pane to display the web service generation options. Change the default options as appropriate.
3. [optional] Click the Selection tab and select the Web Services subtab at the bottom of the tab. Select the web services that you want to generate.
4. Click OK to begin the generation.

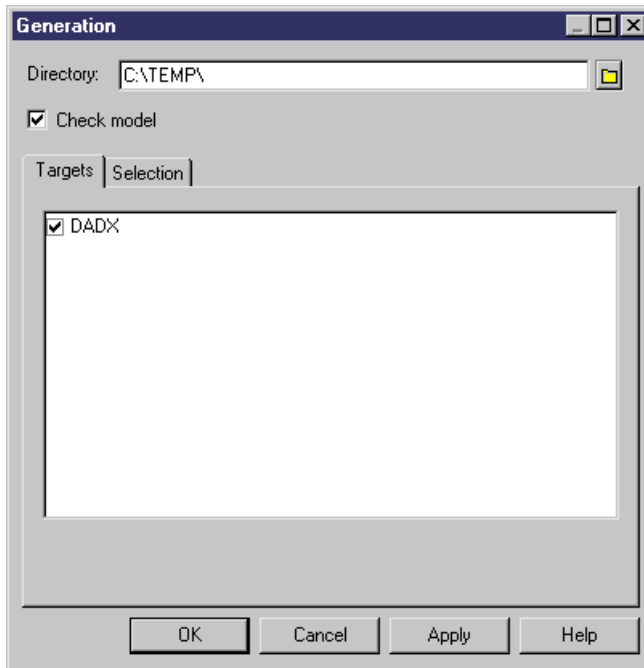
Note that for web services generated to a live database connection, you may have to refresh the Web Services folder before they appear.

Generating Web Services for IBM DB2

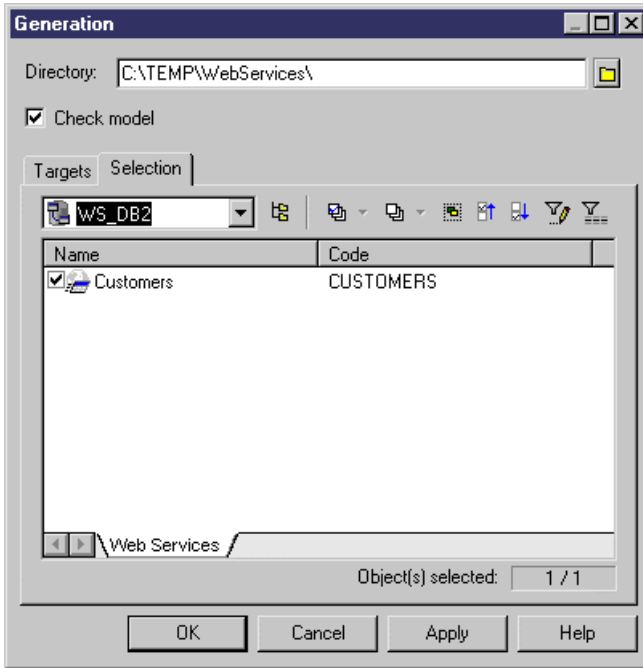
In IBM DB2, web services are defined by Document Access Definition Extension (DADX) files. PowerDesigner can generate these DADX files.

To enable the DADX generation extensions in your model, select **Model > Extensions**, click the **Import** 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 box with DADX selected in the Targets tab.

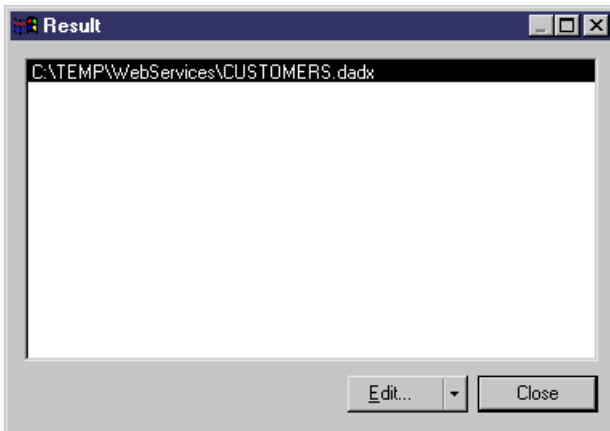


2. Click the **Select a Path** tool to the right of the **Directory** field, and specify a path for the DADX files.
3. Click the **Selection** tab, and select the web services for which you want to generate a DADX file.



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 reverse engineer web services from a database to a PDM, when you want to reuse these web services in the PDM. Once reverse engineered, you can modify and generate them in the database.

You can only reverse engineer web services from Sybase ASA, ASE, and IQ.

You can reverse engineer web services into a new or existing PDM from a script or live database connection via the Database Reverse Engineering dialog box.

For information about using the Database Reverse Engineering dialog box, see *Reverse Engineering a Database into a PDM* on page 373.

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 7 Building a Physical Implementation

In addition to modeling the logical structure of your data systems, you can use PowerDesigner to specify the physical environment to which the database will be deployed.

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.

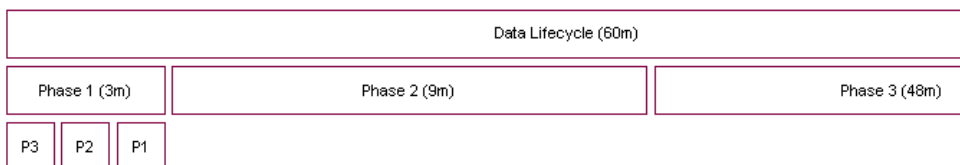
You can attach any number of tables to a lifecycle, and create multiple lifecycles to provide different speeds and/or methods for data aging. Each table can only be associated with one lifecycle. A lifecycle can be:

- **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 *Phase Properties* on page 297). 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 296).

6. Open the property sheet for each of your tablespaces (see *Tablespace and Storage Properties* on page 299) 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 **Lifecycle** tab of the table property sheet (see *Table Properties* on page 82).

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 290.

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 lifecycles, 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 archiving data stored in an external database.
- One or more folders named `yyyyymmdd` 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 lifecycles:

- `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

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

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

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

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: <ul style="list-style-type: none"> • 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.

Property	Description
Total retention	<p>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.</p> <p>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.</p> <p>For access-based lifecycles, the total retention time is used as the minimum total time that the data must remain in the lifecycle.</p>
Phases	<p>Lists the phases (see <i>Phases (PDM)</i> on page 297) associated with the lifecycle. You can create phases using the Create a New Phase tool. Click on a phase to open its property sheet.</p> <hr/> <p>Note: Your phase will display a yellow warning overlay until it is completely defined.</p>
Partition range	<p>[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.</p>

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 291).

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.

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 295)
- 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 296)
- 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 296)

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 10, Generating Other Models from a Data Model* on page 387. For information about using the Merge Models dialog, see *Chapter 10, Generating Models and Model Objects in the Core Features Guide*.

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 *Chapter 12, Object Mappings* in the *Core Features Guide*.

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 290), 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.

- 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.

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 Phases tab of a lifecycle using the Insert Row and Add a Row tools.

Phase Properties

You can modify an object's properties from its property sheet. To open a phase property sheet, double-click its Browser entry, or select it on the Phases tab of a lifecycle property sheet and click the Properties tool.

Property	Description
Name	Specifies the name of the item which should be clear and meaningful, and should convey the item's purpose to non-technical users.
Code	Specifies the technical name of the item used for generating code or scripts, which may be abbreviated, and should not generally include spaces.
Comment	Specifies a descriptive label for the item.
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined.
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.
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 <i>Archiving Data From External Databases</i> on page 295), 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.

Property	Description
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 <i>Linking an External Database via the Data Source Wizard</i> on page 296) to create a data source and apply the appropriate tables to the lifecycle.

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 <code>create tablespace</code>	buffer pool <code>create bufferpool</code>
IBM DB2 UDB for OS/390	table space <code>create tablespace</code>	storage group <code>create stogroup</code>
Informix	NA	NA
Ingres	NA	NA
InterBase	NA	NA
Microsoft Access	NA	NA
Microsoft SQL Server	NA	filegroup <code>alter database add filegroup...</code>
MySQL	NA	NA
Oracle	tablespace <code>create tablespace</code>	storage structure (not physical storage)
PostgreSQL	NA	NA

DBMS	Tablespace represents...	Storage represents...
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 model menu or in the Browser.

- Select **Model > Tablespaces** (or **Storages**) to access the appropriate list, and click the Add a Row tool
- Right-click the model or package in the Browser, and select **New > Tablespace** (or **Storage**)

Tablespace and Storage Properties

You can modify an object's properties from its property sheet. To open a tablespace or storage property sheet, double-click its Browser entry.

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

Cost tab

The Cost tab is available if data lifecycle modeling (see *Lifecycles (PDM)* on page 289) 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 301).
- 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.

Partitions (PDM)

A partition is a physical division of the contents of a database table, created to improve manageability and performance.

You can create as many partitions as necessary in a table.

Note: If you associate a table with a lifecycle (see *Lifecycles (PDM)* on page 289), PowerDesigner will delete all existing table partitions in order to create the necessary partitions to move data between lifecycle phases.

Creating a Partition

You create partitions on the Partitions tab of a table using the Insert Row and Add a Row tools.

Note: Some PowerDesigner features automate the creation of partitions for you. See *Denormalizing Tables and Columns* on page 87 and *Modeling a Lifecycle* on page 290.

Partition Properties

You can modify an object's properties from its property sheet. To open a partition property sheet, double-click its Browser entry, or select it on the Partitions tab of a table property sheet and click the Properties tool.

Property	Description
Parent object	[read only] Specifies the table of which the partition forms a part.
Name	Specifies the name of the item which should be clear and meaningful, and should convey the item's purpose to non-technical users.
Code	Specifies the technical name of the item used for generating code or scripts, which may be abbreviated, and should not generally include spaces.
Comment	Specifies a descriptive label for the item.
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined.
Values	Specifies the upper bound of the partition
Generated	Specifies whether the partition is generated automatically or whether it should be generated during a generate or update database command.
DBSpace	Specifies the tablespace with which the partition is associated. Select a tablespace from the list or click the tools to the right of this field to create a new tablespace, delete or search for an existing tablespace, or open the property sheet of the currently selected one.

Physical Options

A physical option is a parameter (included at the end of a Create statement) that defines how an object is optimized or stored in a database. Physical options are not supported by all databases, and vary by DBMS. In ASA 6, for example, you can define physical options for tables, columns, indexes, tablespaces, and databases. Other DBMSs provide options for keys, storages, and sequences.

The syntax for a physical option depends on the DBMS. For example:

DBMS	Tablespace option syntax
Oracle 6.0 or higher	Tablespace
Sybase Adaptive Server Anywhere	In

For more information about the syntax of physical options and how they are controlled in the DBMS resource file, see "Physical Options" in the DBMS Resource File Reference chapter of the *Customizing and Extending PowerDesigner* manual.

When you change DBMS, the physical option settings are applied as far as possible to the new DBMS. If a specific physical option was selected in the model, the default value is preserved for the option in the new DBMS. Unselected physical options are reset with the new DBMS default values.

PowerDesigner allows you to set:

- Default physical options for all the objects in the model (see *Defining default physical options* on page 302)
- Physical options for a specific object, which override the default physical option values (see *Defining physical options for individual objects* on page 304)

Defining Default Physical Options

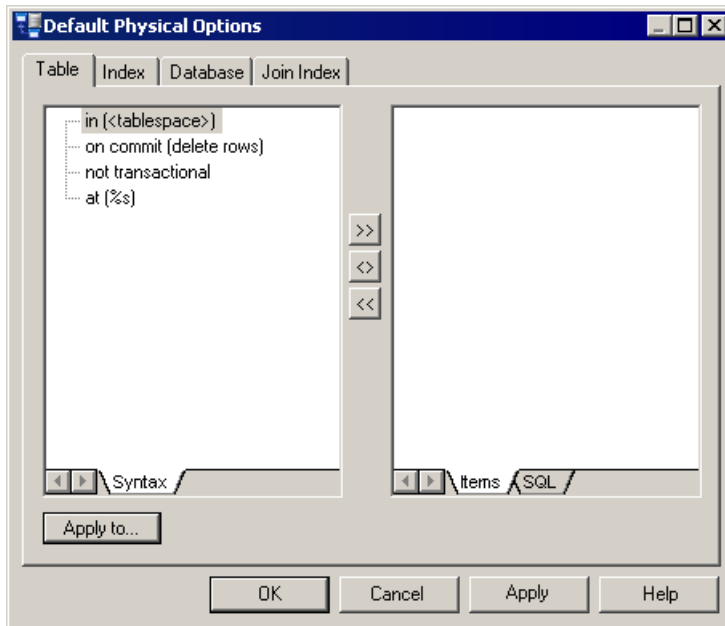
Default physical options define physical options for all the objects of a particular type in the model.

The default physical options are stored in the DBMS definition file (see *Specifying the PDM DBMS* on page 25).

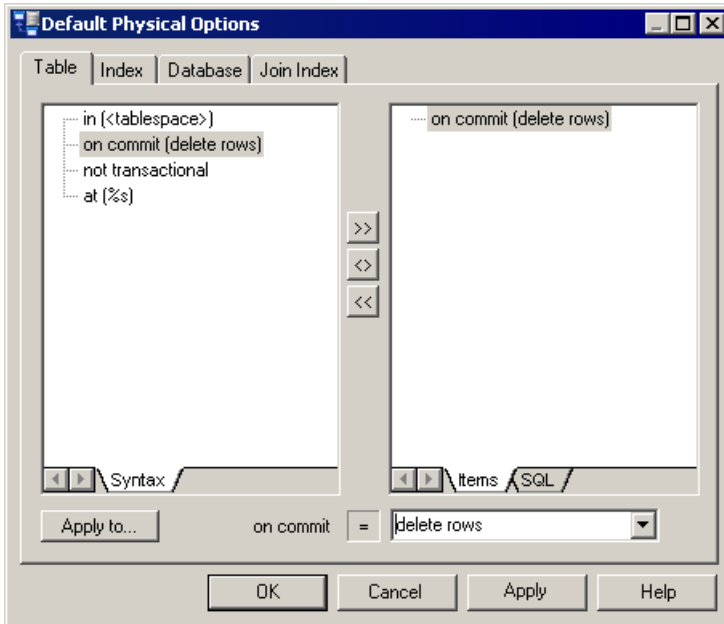
The values you assign for default physical options can be included in a database creation script that you generate from a PDM, or can be directly generated in the target database. You select the appropriate default physical options from the Selection tab of the Database Generation dialog box. Physical option default values can also be reverse engineered from a database into a PDM.

You can view the script for a physical option in the Preview tab for the object for which it is defined.

1. Select **Database > Default Physical Options** to display the Default Physical Options dialog box. There is a tab for each kind of object that supports physical options. The Table tab opens by default:



2. The Syntax tab in the left pane lists the physical options available in the DBMS, and the right pane contains the physical options that have been selected for the object.
3. To add an option for the object, select it in the list in the Syntax pane and click the Add tool between the panes to copy it to the Items pane. To add only a sub-parameter for the option, expand the option in the Syntax pane, select the required parameter and then click the Add tool.
4. To set a value for a physical option parameter, select the parameter in the list in Items pane and enter or select the appropriate value in the field that is displayed below the pane. The entered value will then be displayed against the parameter in the Items list.



5. Repeat the above steps as many times as necessary to specified all your required physical options. By default, these options will be applied to all tables in the model. To specify that the options should apply to only certain of the existing tables, click the **Apply to** button to display a selection dialog. Select the tables that you want to apply the options to from the list and then click **OK**.
6. Select the other tabs to specify physical options for other option types. (Note that the **Apply to** button is not available on the **Database** tab).
7. Click **OK** to close the **Default Physical Options** dialog box.

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

Tool	Action when clicked
>>	Adds physical option selected in Syntax tab (left pane) to Items tab (right pane)
<>	Aligns a selected physical option in the Items tab with the corresponding physical option in the Syntax tab
<<	Removes physical option selected in Items tab

Defining Physical Options for Individual Objects

You can set physical options for selected objects to override the default physical options.

For more information about default physical options (see *Defining default physical options* on page 302).

There are two different interfaces for specifying physical options for individual objects, both of which are accessible through tabs on the object's property sheet:

- *Physical Options (Common)*—this tab is displayed by default (along with the *Partition* tab, if applicable), and lists the most commonly-used physical options in the format of a standard property sheet tab. You can enter values for the necessary option parameters and click OK
- *Physical Options (All)*—this tab is hidden by default, and lists all the available physical options for the object in a tree format. To display this tab, click the Property Sheet Menu button and select **Customize Favorite Tabs > Physical Options (All)**. You should follow the procedure in *Defining default physical options* on page 302, to modify the appropriate options.

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

Note: You can choose a tablespace or storage already defined in the model as a value for a tablespace or storage that you define for a table. These are listed in a list below the right pane in the Options tab. In DB2 OS/390, tablespaces are, by default, prefixed by the name of the database attached to the model (see *DBMS-Specific Features* on page 5). For more information on defining tablespaces and storages, see *Generating a Database from a PDM* on page 345.

PART II

Working with Data Models

The chapters in this part provide information about PowerDesigner features that allow you to customize, check, generate from, and reverse-engineer to your data models .

CHAPTER 8 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 window opens, allowing you to specify the kinds of checks to perform, and the objects to apply them to. For detailed information about this window and correcting problems reported, see "Checking a Model" in the Models chapter of the *Core Features Guide*.

The following sections document the Data Model -specific checks available by default. For information about checks made on generic objects available in all model types, see "Checking a Model" in the Models chapter of the *Core Features Guide*.

Abstract Data Type Checks (PDM)

PowerDesigner provides default model checks to verify the validity of abstract data types.

Check	Description and Correction
Abstract data type name and code uniqueness	Abstract data type names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code

Check	Description and Correction
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
Abstract data type procedure name and code uniqueness	Abstract data type procedure names and codes must be unique in the abstract data type. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
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

Check	Description and Correction
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
Association name and code uniqueness	Association names and codes must be unique in the namespace. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code.
Number of links ≥ 2	An association is isolated and therefore does not define a relationship between entities. Manual correction: Define at least two links between the isolated association and one or several entities. Automatic correction: None.
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.

Check	Description and Correction
Absence of properties with identifier links	An association with an identifier link cannot have any properties. Manual correction: Move the association properties into the dependent entity (the one linked to the association with an identifier link). Automatic correction: None.
Bijective association between two entities	There are bijective associations between two entities when a two-way one to one association between the entities exist. This is equivalent to a merge of two entities. Manual correction: Merge the entities or modify the cardinality links. Automatic correction: None.
Maximal cardinality links	An association with more than two links can only have links with a maximum cardinality greater than one. Manual correction: Change the maximum cardinality of such links to be greater than 1. Automatic correction: None.
Reflexive identifier links	An identifier link introduces a dependency between two entities. An association with this type of link cannot therefore be reflexive. Manual correction: Change the relationship between the entities or clear the Identifier check box for a link. Automatic correction: None.
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	Association names and codes must be unique in the model. Manual correction: Select a hierarchy in the Hierarchy list in the association property sheet Automatic correction: None

Column Checks (PDM)

PowerDesigner provides default model checks to verify the validity of columns.

Check	Description and Correction
Column name and code uniqueness	<p>Column names and codes must be unique in a table.</p> <p>Manual correction: Modify the duplicate name/code</p> <p>Automatic correction: Appends a number to the duplicate name/code</p>
Column code maximum length	<p>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.</p> <p>Manual correction: Modify the column code length to meet this requirement</p> <p>Automatic correction: Truncates the code length to the maximum length specified in the DBMS definition</p>
Domain divergence	<p>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.</p> <p>Manual correction: Select one or more of the Enforce non divergence model options to enforce non divergence</p> <p>Automatic correction: Restores divergent attributes from domain to column (domain values overwrite column values)</p>
Column mandatory	<p>In some DBMS, the columns included in a key or a unique index should be mandatory.</p> <p>Manual correction: Select the Mandatory check box in the column property sheet</p> <p>Automatic correction: Makes the column mandatory</p>
Detect inconsistencies between check parameters	<p>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.</p> <p>Manual correction: Modify default, minimum, maximum or list of values in the check parameters tab</p> <p>Automatic correction: None</p>

Check	Description and Correction
Precision > Maximum length	<p>The data type precision should not be greater than the length. Note that some DBMS accept a precision higher than the length.</p> <p>Manual correction: Make the data type length greater than the precision</p> <p>Automatic correction: None</p>
Undefined data type	<p>A model should not contain columns with undefined data type, all columns should have a defined data type.</p> <p>Manual correction: Select a data type in the column property sheet</p> <p>Automatic correction: None</p>
Foreign key column data type and constraint parameters divergence	<p>Primary/alternate and foreign key columns involved in a join should have consistent data types and constraint parameters.</p> <p>Manual correction: Modify foreign key data types and constraint parameters to make them consistent</p> <p>Automatic correction: Parent column overwrites existing data type and constraint parameters in the foreign key column</p>
Column with sequence not in a key	<p>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.</p> <p>Manual correction: Attach the sequence to a column that is part of a key</p> <p>Automatic correction: None</p>
Auto-incremented column with data type not numeric	<p>An auto-incremented column must have a numeric data type.</p> <p>Manual correction: Change the column data type</p> <p>Automatic correction: Changes data type to numeric data type</p>
Auto-incremented column is foreign key	<p>A foreign key column should not be auto-incremented.</p> <p>Manual correction: Deselect the Identity check box in the column property sheet</p> <p>Automatic correction: None</p>
Missing computed column expression	<p>A computed column should have a computed expression defined.</p> <p>Manual correction: Add a computed expression to the column in the Details tab of the column property sheet</p> <p>Automatic correction: None</p>
Invalid mapping from source column	<p>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.</p> <p>Manual correction: Remove the additional mappings.</p> <p>Automatic correction: None</p>

Check	Description and Correction
Data type compatibility of mapped columns	<p>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.</p> <p>Manual correction: Harmonize the data types in the source and target columns.</p> <p>Automatic correction: None</p>
Existence of mapping for mandatory columns	<p>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.</p> <p>Manual correction: Map the mandatory column to a column in the external database.</p> <p>Automatic correction: None</p>

Cube Checks (PDM)

PowerDesigner provides default model checks to verify the validity of cubes.

Check	Description and Correction
Cube name and code uniqueness	<p>Cubes names and codes must be unique in the model.</p> <p>Manual correction: Modify the duplicate name/code</p> <p>Automatic correction: Appends a number to the duplicate name/code</p>
Existence of association	<p>A cube must have at least one association with a dimension.</p> <p>Manual correction: Create an association between the cube and a dimension</p> <p>Automatic correction: None</p>
Existence of fact	<p>A cube must be associated to a fact.</p> <p>Manual correction: Click the Ellipsis button beside the Fact box in the cube property sheet, and select a fact from the List of Facts</p> <p>Automatic correction: None</p>
Duplicated association with the same dimension	<p>A cube cannot have more than one association with the same dimension.</p> <p>Manual correction: Delete one of the associations</p> <p>Automatic correction: None</p>

Database Checks (PDM)

PowerDesigner provides default model checks to verify the validity of databases.

Check	Description and Correction
Database name and code uniqueness	Database names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Database code maximum length	The code of the database is longer than the maximum allowed by the DBMS. Manual correction: Reduce the length of the code Automatic correction: Reduces the code to a permissible length
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
Database package name and code uniqueness	Database package names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Database package name and code maximum length	The database package name and code length is limited by the maximum length specified in the DBMS definition (MaxLen entry, in the Objects > DB Package category) and in the naming conventions of the model options. Manual correction: Modify the name/code length to meet this requirement Automatic correction: Truncates the name/code length to the maximum length specified in the DBMS definition

Check	Description and Correction
Existence of package <i>subobject</i>	<p>A database package must have a number of subobjects defined in order to be correctly modeled.</p> <p>Manual correction: Create one or more of the relevant object on the appropriate tab of the database package property sheet:</p> <ul style="list-style-type: none"> • Procedures (or use existing stored procedures and duplicate them in the database package) • Cursors • Variables • Types • Exceptions <p>Automatic correction: None</p>

Database Package Subobject Checks (PDM)

PowerDesigner provides default model checks to verify the validity of database package cursors, exceptions, procedures, types, and variables.

Check	Description and Correction
Package cursor name and code uniqueness	<p>Subobject names and codes must be unique in the database package.</p> <p>Manual correction: Modify the duplicate name/code</p> <p>Automatic correction: Appends a number to the duplicate name/code</p>
Package <i>subobject</i> definition empty	<p>[cursors, procedures, types] These subobjects must have a definition.</p> <p>Manual correction: Create the definition in the Definition tab of the subobject's property sheet</p> <p>Automatic correction: None</p>
Check for undefined return types	<p>[cursors, procedures] These subobjects must have a return data type.</p> <p>Manual correction: Select a return data type in the subobject's property sheet</p> <p>Automatic correction: None</p>
Existence of parameter	<p>[cursors, procedures] These subobjects must contain parameters for input values.</p> <p>Manual correction: Create one or several parameters in the Parameters tab of the subobject's property sheet</p> <p>Automatic correction: None</p>

Check	Description and Correction
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
Data item name and code uniqueness	Data item names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code.
Data item not used	There are unused data items. These are useless for PDM generation. Manual correction: To use a data item, add it to an entity. If you do not need an unused data item, delete it to allow PDM generation. Automatic correction: None.
Data item used multiple times	There are entities using the same data items. This can be tolerated if you defined this check as a warning. Manual correction: Take care to ensure consistency when defining data item properties. Automatic correction: None.

Check	Description and Correction
Detect differences between data item and associated domain	<p>There is a divergence between data items and associated domains. This can be tolerated if you defined this check as a warning.</p> <p>Manual correction: Ensure consistency when defining data item properties</p> <p>Automatic correction: Restores divergent attributes from domain to data items (domain values overwrite data item values).</p>
Detect inconsistencies between check parameters	<p>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.</p> <p>Manual correction: Modify default, minimum, maximum or list of values in the check parameters page</p> <p>Automatic correction: None.</p>
Precision > maximum length	<p>The data type precision should not be greater than or equal to the length.</p> <p>Manual correction: Make the data type length greater than or equal to the precision.</p> <p>Automatic correction: None.</p>
Undefined data type	<p>Undefined data types for data items exist. To be complete, a model should have all its data items data types defined.</p> <p>Manual correction: While undefined data types are tolerated, you must select data types for currently undefined data types before you can generate a PDM.</p> <p>Automatic correction: None.</p>
Invalid data type	<p>Invalid data types for data items exist. To be complete, a model should have all its data types for data items correctly defined.</p> <p>Manual correction: While tolerated, you must select valid data types for currently non-valid data types to generate the PDM.</p> <p>Automatic correction: None.</p>

Data Source Checks (PDM)

PowerDesigner provides default model checks to verify the validity of data sources.

Check	Description and Correction
Data source name and code uniqueness	Data source names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Existence of physical data model	A data source must contain at least one physical data model in its definition. Manual correction: Add a physical data model from the Models tab of the property sheet of the data source. Automatic correction: Deletes data source without physical data model.
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
Default name and code uniqueness	Default names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code

Check	Description and Correction
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
Dimension name and code uniqueness	Dimension names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Existence of attribute	A dimension must have at least one attribute. Manual correction: Create an attribute in the Attributes tab of the dimension property sheet Automatic correction: None
Existence of hierarchy	A dimension must use at least one hierarchy. Manual correction: Create a hierarchy in the Hierarchies tab of the dimension property sheet Automatic correction: None
Dimension have duplicated hierarchies	Dimensions should not have duplicated hierarchies, that is to say hierarchies organizing identical attributes. Manual correction: Remove one of the duplicated hierarchies Automatic correction: None

Check	Description and Correction
Dimension without a default hierarchy	<p>A dimension should have a default hierarchy.</p> <p>Manual correction: Select a hierarchy in the Default Hierarchy list of the dimension property sheet</p> <p>Automatic correction: None</p>
Dimension mapping not defined	<p>A dimension should be mapped to tables or views in an operational model in order to be populated by data from this model.</p> <p>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</p> <p>Automatic correction: Destroys the mapping for the dimension. This removes the data source from the Mapping list in the dimension Mapping tab</p>
Attribute mapping not defined	<p>Attributes must be mapped to columns in the data source tables or views.</p> <p>Manual correction: Map the attributes to columns in the data source</p> <p>Automatic correction: None</p>
Incomplete dimension mapping for multidimensional generation	<p>All attributes, detail attributes and hierarchies of the dimension must be mapped to tables and columns. You must map the dimension objects before generation.</p> <p>Manual correction: Map dimension objects to tables and columns</p> <p>Automatic correction: None</p>

Domain Checks (CDM/LDM/PDM)

PowerDesigner provides default model checks to verify the validity of domains.

Check	Description and Correction
Domain name and code uniqueness	<p>Domain names and codes must be unique in the model.</p> <p>Manual correction: Modify the duplicate name/code</p> <p>Automatic correction: Appends a number to the duplicate name/code</p>
Domain code maximum length	<p>[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.</p> <p>Manual correction: Modify the domain code length to meet this requirement</p> <p>Automatic correction: Truncates the domain code length to the maximum length specified in the DBMS definition</p>

Check	Description and Correction
Detect Inconsistencies between check parameters	<p>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.</p> <p>Manual correction: Modify default, minimum, maximum or list of values in the check parameters tab</p> <p>Automatic correction: None</p>
Precision > maximum length	<p>The data type precision should not be greater than the length.</p> <p>Manual correction: Make the data type length greater than the precision</p> <p>Automatic correction: None</p>
Undefined data type	<p>A model should not contain domains with undefined data type, all domains should have a defined data type.</p> <p>Manual correction: Select a data type from the domain property sheet</p> <p>Automatic correction: None</p>
Invalid data type	<p>[CDM/LDM only] Invalid data types for domains exist. To be complete, a model should have all its domain data types correctly defined.</p> <p>Manual correction: While tolerated, you must select valid data types for currently non-valid data types to generate the PDM.</p> <p>Automatic correction: None.</p>

Entity Attribute Checks (CDM/LDM)

PowerDesigner provides default model checks to verify the validity of entity attributes.

Check	Description and Correction
Entity attribute name and code uniqueness	<p>Attribute names and codes must be unique in the entity.</p> <p>Manual correction: Modify the duplicate name/code</p> <p>Automatic correction: Appends a number to the duplicate name/code.</p>
Detect differences between attribute and associated domain	<p>[LDM only] There is a divergence between attributes and associated domains. This can be tolerated if you defined this check as a warning.</p> <p>Manual correction: Ensure consistency when defining attribute properties</p> <p>Automatic correction: Restores divergent attributes from domain to attributes (domain values overwrite attribute values).</p>

Check	Description and Correction
Detect inconsistencies between check parameters	<p>[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.</p> <p>Manual correction: Modify default, minimum, maximum or list of values in the check parameters page</p> <p>Automatic correction: None.</p>
Precision > maximum length	<p>[LDM only] The data type precision should not be greater than or equal to the length.</p> <p>Manual correction: Make the data type length greater than or equal to the precision.</p> <p>Automatic correction: None.</p>
Undefined data type	<p>[LDM only] Undefined data types for attributes exist. To be complete, a model should have all its attributes data types defined.</p> <p>Manual correction: While undefined data types are tolerated, you must select data types for currently undefined data types before you can generate a PDM.</p> <p>Automatic correction: None.</p>
Invalid data type	<p>[LDM only] Invalid data types for attributes exist. To be complete, a model should have all its data types for attributes correctly defined.</p> <p>Manual correction: While tolerated, you must select valid data types for currently non-valid data types to generate the PDM.</p> <p>Automatic correction: None.</p>

Entity Identifier Checks (CDM/LDM)

PowerDesigner provides default model checks to verify the validity of entity identifiers.

Check	Description and Correction
Entity identifier name and code uniqueness	<p>Entity identifier names and codes must be unique in the namespace.</p> <p>Manual correction: Modify the duplicate name/code</p> <p>Automatic correction: Appends a number to the duplicate name/code.</p>

Check	Description and Correction
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
Entity name and code uniqueness	Entity names and codes must be unique in the namespace. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code.
Entity name and code maximum length	The entity name and code length is limited to a maximum length of 254 characters specified in the naming conventions of the model options. Manual correction: Modify the entity name/code length to meet this requirement. Automatic correction: Truncates the entity name/code length to the maximum length specified in the naming conventions.
Existence of attributes	An entity must always contain at least one attribute. Manual correction: Add an attribute to the entity or delete the entity. Automatic correction: None.
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.

Check	Description and Correction
Existence of identifiers	An entity must contain at least one identifier. Manual correction: Add an identifier to the entity or delete the entity. Automatic correction: None.
Existence of relationship or association link	An entity must have at least one relationship or association link. Manual correction: Add a relationship or an association link to the entity or delete the entity. Automatic correction: None.
Redundant inheritance	An entity inherits from another entity more than once. This is redundant and adds nothing to the model. Manual correction: Delete redundant inheritances Automatic correction: None.
Multiple inheritance	An entity has multiple inheritance. This is unusual but can be tolerated if you defined this check as a warning. Manual correction: Make sure that the multiple inheritance is necessary in your model. Automatic correction: None.
Parent of several inheritances	An entity is the parent of multiple inheritances. This is unusual but can be tolerated if you defined this check as a warning. Manual correction: Verify if the multiple inheritances could not be merged. Automatic correction: None.
Redefined primary identifier	Primary identifiers in child entities must be the same as those in their parents. Manual correction: Delete those primary identifiers in the child entities that are not in the parent entity. Automatic correction: None.

Fact Checks (PDM)

PowerDesigner provides default model checks to verify the validity of facts.

Check	Description and Correction
Fact name and code uniqueness	Fact names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code

Check	Description and Correction
Existence of measure	<p>A fact must have at least one measure.</p> <p>Manual correction: Create a measure in the Measures tab of the fact property sheet</p> <p>Automatic correction: None</p>
Fact mapping not defined	<p>A fact must be mapped to tables or views in an operational model in order to be populated by data from this model.</p> <p>Manual correction: Map the fact to tables or views. You may need to create a data source before you can create the mapping</p> <p>Automatic correction: Destroys the mapping for the fact. This removes the data source from the Mapping list in the fact Mapping tab</p>
Measure mapping not defined	<p>Fact measures must be mapped to columns in the data source tables or views.</p> <p>Manual correction: Map the fact measure to columns in the data source</p> <p>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</p>

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 and code uniqueness	<p>Names and codes must be unique in the model.</p> <p>Manual correction: Modify the duplicate name/code</p> <p>Automatic correction: Appends a number to the duplicate name/code</p>
Existence of attribute	<p>[hierarchies only] A dimension hierarchy must have at least one attribute.</p> <p>Manual correction: Add an attribute to the hierarchy from the Attributes tab of the hierarchy property sheet</p> <p>Automatic correction: None</p>

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 and code uniqueness	Names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Existence of partition	[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 and code uniqueness	Depending on the DBMS, a model or a table cannot contain two indexes with identical name and/or code. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code

Check	Description and Correction
Index code maximum length	<p>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.</p> <p>Manual correction: Modify the index code length to meet this requirement</p> <p>Automatic correction: Truncates the index code length to the maximum length specified in the DBMS definition</p>
Existence of index column	<p>An index must have at least one index column.</p> <p>Manual correction: Add an index column from the Column tab of the index property sheet or delete the index</p> <p>Automatic correction: Deletes the index without column</p>
Undefined index type	<p>[indexes] An index type must be specified.</p> <p>Manual correction: Specify a type in the index property sheet or delete the index with no type</p> <p>Automatic correction: None</p>
Index column count	<p>The current DBMS does not support more than the number of index columns specified in the MaxColIndex entry of the current DBMS.</p> <p>Manual correction: Delete one or more columns in the index property sheet. You can create additional indexes for these columns</p> <p>Automatic correction: None</p>
Uniqueness forbidden for HNG index type	<p>[indexes] An index of HNG (HighNonGroup) type cannot be unique.</p> <p>Manual correction: Change the index type or set the index as non unique</p> <p>Automatic correction: None</p>
Index inclusion	<p>An index should not include another index.</p> <p>Manual correction: Delete the index that includes an existing index</p> <p>Automatic correction: None</p>

Inheritance Checks (CDM/LDM)

PowerDesigner provides default model checks to verify the validity of inheritances.

Check	Description and Correction
Inheritance name and code uniqueness	<p>Inheritance names and codes must be unique in the model.</p> <p>Manual correction: Modify the duplicate name/code</p> <p>Automatic correction: Appends a number to the duplicate name/code.</p>

Check	Description and Correction
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
Join index name and code uniqueness	Join index names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Existence of base table	Join index must have a base table. Manual correction: Select a base table in the join index property sheet Automatic correction: None
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
Key name and code uniqueness	Key names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Key code length	The key code length is limited by the maximum length specified in the DBMS definition (MaxConstLen entry, in the Object > Key category). Manual correction: Modify the key code length to meet this requirement Automatic correction: Truncates the key code length to the maximum length specified in the DBMS definition
Key column exists	Each key must have at least one column. Manual correction: Add a column to the key from the Column tab of the key property sheet Automatic correction: Deletes key without column
Key inclusion	A key cannot include another key (on some columns, regardless of their order). Manual correction: Delete the key that includes an existing key Automatic correction: None
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 and code uniqueness	Lifecycle and phase names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Existence of 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.
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 table-space	[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

Check	Description and Correction
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 lifecycle 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 lifecycle 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
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
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	<p>[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:</p> <p>Check tab of the table property sheet</p> <p>Additional Check tab of the column property sheet</p> <p>General tab of the key property sheet</p> <p>A constraint name must be unique in a model.</p> <p>Manual correction: Modify the duplicated constraint name in the corresponding tab</p> <p>Automatic correction: Modifies the duplicated constraint name of a selected object by appending a number to its current name</p>
Constraint name maximum length	<p>[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.</p> <p>Manual correction: Modify the constraint name to meet this requirement</p> <p>Automatic correction: Truncates the constraint name to the maximum length specified in the DBMS definition</p>
Circular dependencies	<p>[PDM only] Extended dependencies with the stereotype <<DBCreateAfter>> can be used between stored procedures to define a generation order for stored procedures. An extended dependency with the stereotype <<DBCreateAfter>> should not introduce a circular dependency in the model.</p> <p>Manual correction: Remove the <<DBCreateAfter>> extended dependency</p> <p>Automatic correction: None</p>
Circular dependency	<p>[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.</p> <p>Manual correction: Clear the Dependent check box for the link or delete an inheritance link.</p> <p>Automatic correction: None.</p>
Circularity with mandatory links	<p>[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.</p> <p>Manual correction: Clear the Mandatory parent check box or delete a dependency on a relationship.</p> <p>Automatic correction: None.</p>

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
Procedure name and code uniqueness	Procedure names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Procedure code maximum length	The procedure code length is limited by the maximum length specified in the DBMS definition (MaxLen entry, in the Objects > Procedure category). Manual correction: Modify the procedure code length to meet this requirement Automatic correction: Truncates the procedure code length to the maximum length specified in the DBMS definition
Procedure definition body empty	A procedure definition should have a body to specify its functionality. Manual correction: Specify a procedure body from the Definition tab of the procedure property sheet Automatic correction: None
Existence of permission	Permissions are usage restrictions set on a procedure for a particular user, group or role. Manual correction: Define permissions on the procedure for users, groups and roles Automatic correction: None

Reference and View Reference Checks (PDM)

PowerDesigner provides default model checks to verify the validity of references and view references.

Check	Description and Correction
Name and code uniqueness	Reference names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Reflexive and mandatory reference	[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
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
Relationship name and code uniqueness	Relationship names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code.
Reflexive dependency	A dependency means that one entity is defined through a relationship with another. A dependent relationship cannot therefore be reflexive. Manual correction: Change or delete the reflexive dependency. Automatic correction: None.
Reflexive mandatory	A reflexive mandatory relationship exists. Manual correction: Deselect the Mandatory check boxes for the relationship to be non-mandatory. Automatic correction: None.
Bijective relationship between two entities	There is a bijective relationship between two entities when there is a two-way one to one relationship between the entities. This is equivalent to a merge of two entities. Manual correction: Merge the entities or modify the relationship. Automatic correction: None.
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
Sequence name and code uniqueness	Default names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Sequence code maximum length	The code of the sequence is longer than the maximum allowed by the DBMS. Manual correction: Reduce the length of the code Automatic correction: Reduces the code to a permissible length

Synonym Checks (PDM)

PowerDesigner provides default model checks to verify the validity of synonyms.

Check	Description and Correction
Synonym name and code uniqueness	Synonym names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code

Check	Description and Correction
Synonym name and code maximum length	<p>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.</p> <p>Manual correction: Modify the name/code length to meet this requirement</p> <p>Automatic correction: Truncates the name/code length to the maximum length specified in the DBMS definition</p>
Existence of the base object	<p>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.</p> <p>Manual correction: Select a base object from the synonym property sheet</p> <p>Automatic correction: Deletes the synonym</p>

Table and View Checks (PDM)

PowerDesigner provides default model checks to verify the validity of tables and views.

Check	Description and Correction
Name and code uniqueness	<p>Table and view names and codes must be unique in the model.</p> <p>Manual correction: Modify the duplicate name/code</p> <p>Automatic correction: Appends a number to the duplicate name/code</p>
Name and code length	<p>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.</p> <p>Manual correction: Modify the name/code length to meet this requirement</p> <p>Automatic correction: Truncates the name/code length to the maximum length specified in the DBMS definition</p>
Constraint name conflicts with index name	<p>[tables only] A constraint name of the table cannot be the same as an index name.</p> <p>Manual correction: Change the name of the table constraint</p> <p>Automatic correction: None</p>
Existence of column, reference, index, key	<p>[tables only] A table should contain at least one column, one index, one key, and one reference.</p> <p>Manual correction: Add missing item to the definition of the table</p> <p>Automatic correction: None</p>

Check	Description and Correction
Number of auto-incremented columns	<p>[tables only] Auto-incremented columns contain automatically calculated values. A table cannot contain more than one auto-incremented column.</p> <p>Manual correction: Delete all but one auto-incremented column</p> <p>Automatic correction: None</p>
Table index definition uniqueness	<p>[tables only] Identical indexes are indexes with the same columns, order and type. A table cannot have identical indexes.</p> <p>Manual correction: Delete index or change its properties</p> <p>Automatic correction: None</p>
Table mapping not defined	<p>[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.</p> <p>Manual correction: Map the current table to one or several tables or views in the model belonging to the data source</p> <p>Automatic correction: Destroys the mapping for the table. This removes the data source from the Mapping list in the table Mapping tab</p>
Column mapping not defined	<p>[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.</p> <p>Manual correction: Map the current column to one or several columns in the models belonging to the data source</p> <p>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</p>
Existence of permission	<p>Permissions are usage restrictions set on a table or view for a particular user, group or role.</p> <p>Manual correction: Define permissions on the table or view for users, groups and roles</p> <p>Automatic correction: None</p>
Existence of partition key	<p>[tables only] A table managed by an age-based lifecycle policy must have a column specified as its partition key.</p> <p>Manual correction: Specify a column as the partition key.</p> <p>Automatic correction: None</p>

Check	Description and Correction
Invalid start date setting	<p>[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.</p> <p>Manual correction: Change one or other date so that the table start date is equal to or later than the lifecycle start date.</p> <p>Automatic correction: Changes the table start date to the lifecycle start date.</p>
Missing lifecycle policy	<p>[tables only] A table managed by a lifecycle must not reference tables not managed by a lifecycle.</p> <p>Manual correction: Add the referenced tables to the lifecycle.</p> <p>Automatic correction: None</p>
Invalid mapping from source table	<p>[tables only] In a lifecycle where the first phase references an external database, each archive table must be mapped to exactly one external table.</p> <p>Manual correction: Remove the additional mappings.</p> <p>Automatic correction: None</p>
Partial column mapping of source table	<p>[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.</p> <p>Manual correction: Create the missing mappings.</p> <p>Automatic correction: None</p>
Existence of partition key mapping	<p>[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.</p> <p>Manual correction: Create the missing mapping.</p> <p>Automatic correction: None</p>
Tablespace outside lifecycle	<p>[tables only] A table managed by a lifecycle must be assigned to a tablespace associated with the lifecycle.</p> <p>Manual correction: Assign the table to a tablespace associated with the lifecycle.</p> <p>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.</p>

Tablespace and Storage Checks (PDM)

PowerDesigner provides default model checks to verify the validity of tablespaces and storages.

Check	Description and Correction
Name and code uniqueness	Names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
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
Trigger name and code uniqueness	Trigger names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
Trigger code 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	<p>The event specified in the DBMS trigger definition must be available in its chosen scope.</p> <p>Manual correction: Modify the trigger code to reference an event in the chosen scope.</p> <p>Automatic correction: None</p>

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 and code uniqueness	<p>Names and codes must be unique in the model.</p> <p>Manual correction: Modify the duplicate name/code</p> <p>Automatic correction: Appends a number to the duplicate name/code</p>
Code maximum length	<p>The code length is limited by the maximum length specified in the DBMS definition (MaxLen entry, in the Objects > User and Group categories).</p> <p>Manual correction: Modify the code length to meet this requirement</p> <p>Automatic correction: Truncates the code length to the maximum length specified in the DBMS definition</p>
Existence of user	<p>[groups, roles] A group is created to factorize privilege and permission granting to users. A group without user members is useless.</p> <p>Manual correction: Add users to group or delete group</p> <p>Automatic correction: Deletes unassigned group</p>
Password empty	<p>[users, groups] Users and groups must have a password to be able to connect to the database.</p> <p>Manual correction: Define a password for the user or group</p> <p>Automatic correction: None</p>

View Checks (PDM)

PowerDesigner provides default model checks to verify the validity of views.

Check	Description and Correction
View name and code uniqueness	View names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
View code maximum length	The view code length is limited by the maximum length specified for the table code length. Manual correction: Modify the view code length to meet this requirement Automatic correction: Truncates the view code length to the maximum length specified in the DBMS definition
Existence of permission	Permissions are usage restrictions set on a view for a particular user, group or role. Manual correction: Define permissions on the view for users, groups and roles Automatic correction: None

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 and code uniqueness	Web service and Web operation names and codes must be unique in the model. Manual correction: Modify the duplicate name/code Automatic correction: Appends a number to the duplicate name/code
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 9 **Generating and Reverse-Engineering Databases**

PowerDesigner provides full support for round trip generation and reverse-engineering between a PDM and a database.

Generating a Database from a PDM

PowerDesigner is capable of generating sophisticated SQL scripts as files or for execution via a live database connection.

Connecting to a Database

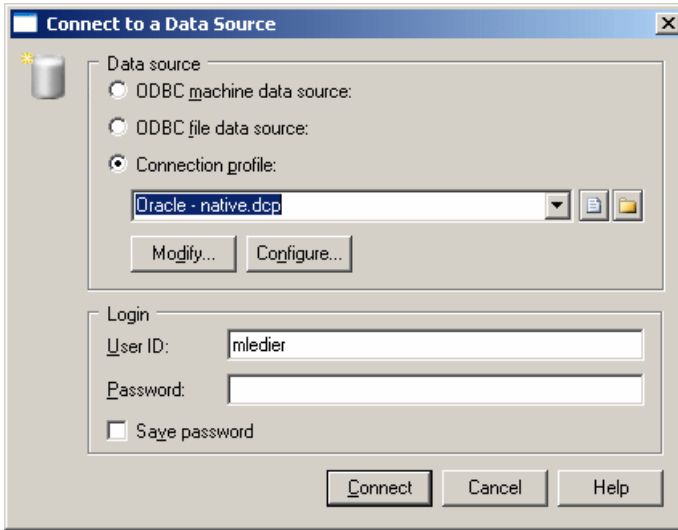
PowerDesigner provides various methods for connecting to your database.

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 "Connecting to a Database" in *Chapter 1, Getting Started with PowerDesigner* of the *Core Features Guide*.

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.

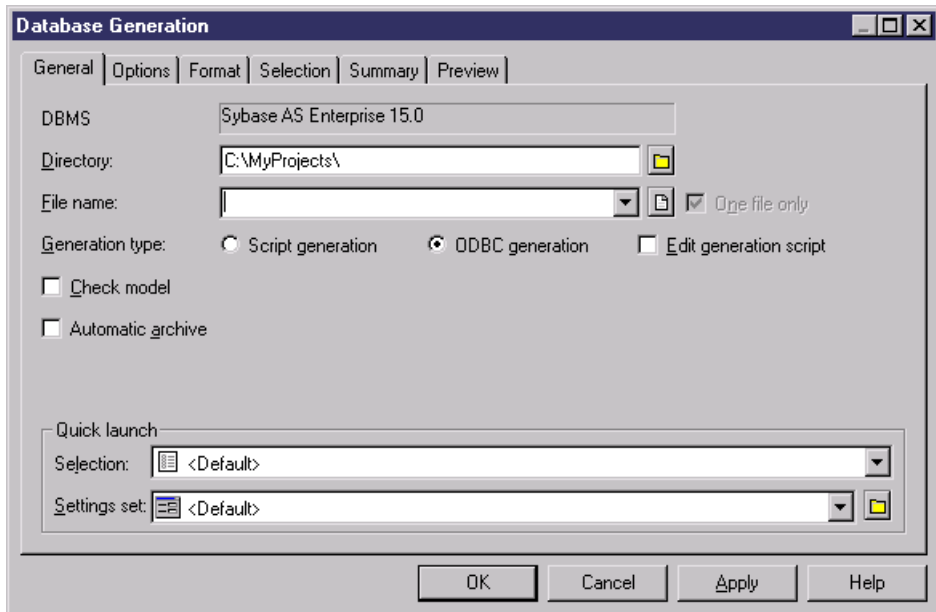
You can display information about your connection at any time by selecting **Database > Connection Information**. The amount of information available depends on your DBMS and your connection profile.

To disconnect from a database, select **Database > Disconnect**.

Generating a Database

PowerDesigner can generate a database creation script that you can run in your DBMS environment or generate a database structure directly to a live database connection.

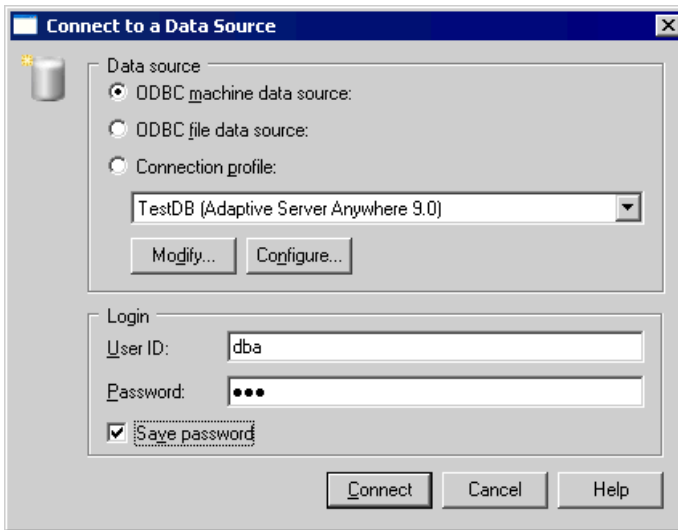
1. Select **Database > Generate Database** to open the Database Generation dialog box (for more information, see *Database Generation dialog General tab* on page 349).



2. Type a destination directory and a filename for the script file in the Directory and File Name boxes.
3. Select the Script generation or Direct Generation radio button.
4. [optional] Click the Options tab, and specify creation options for your database objects (for more information, see *Database Generation dialog Options tab* on page 351).
5. [optional] Click the Format tab, and specify format options for your database objects (for more information, see *Database Generation dialog Format tab* on page 354).
6. [optional] Click the Selection tab, and specify the database objects to be created (for more information, see *Database Generation dialog Selection tab* on page 356).
7. [optional] Click the Summary tab to view the summary of your settings and selections (for more information, see *Database Generation dialog Summary tab* on page 357).
8. [optional] Click the Preview tab to preview the database script to be used (for more information, see *Database Generation dialog Preview tab* on page 358).
9. Click OK to begin the generation.

If you are creating a database script: The output window shows the progress of the generation process, and indicates the syntax for running the script. At the end of script generation, a Result box is displayed. It lists the file path of the generated script file. Click Edit to open the script in a text editor or Close to close the Result box.

If you are generating a database directly: If you are not currently connected to a database, a dialog box asks you to identify a data source and connection parameters.

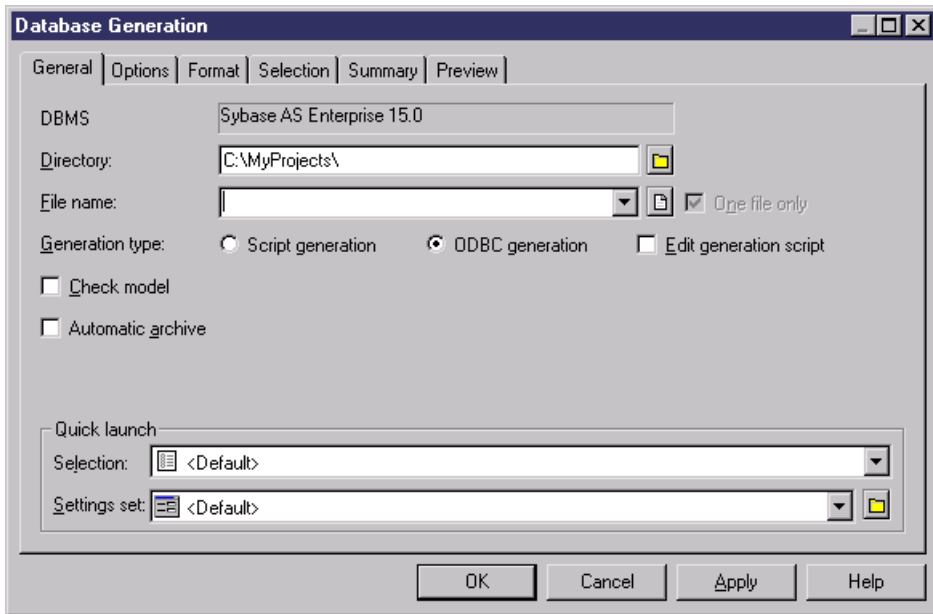


Select a machine data source or file data source, type your user ID and password and then click Connect. You may be prompted for additional connection parameters.

Note: Advanced users may want to further customize database generation by, for example, customizing the order in which objects are generated, adding scripts to run before or after generation, and generating their own extended objects. For information on these and other advanced topics, the DBMS Resource File Reference chapter in the *Customizing and Extending PowerDesigner* manual.

Database Generation Dialog General Tab

The General tab is the main tab for controlling database generation.



You can set the following options

Parameter	Description
Directory	[required] Specifies the destination directory for the script file.
File name	[required] Specifies the destination filename for the script file.
One file only	Specifies that the generation script is created as a single file. By default, a separate script file is created for each table.
Generation type	Specifies the type of generation to perform. You can choose between: <ul style="list-style-type: none"> Script generation - generate a script to be executed on a DBMS at a later time Direct generation – generate a script and execute it on a live database connection
Edit generation script	[available only when Direct generation is selected] Opens the generation script in a text editor for review or editing before execution on the database.
Check model	Specifies that a model check is performed before script generation.
Automatic archive	Creates an archive version of the PDM after generation.

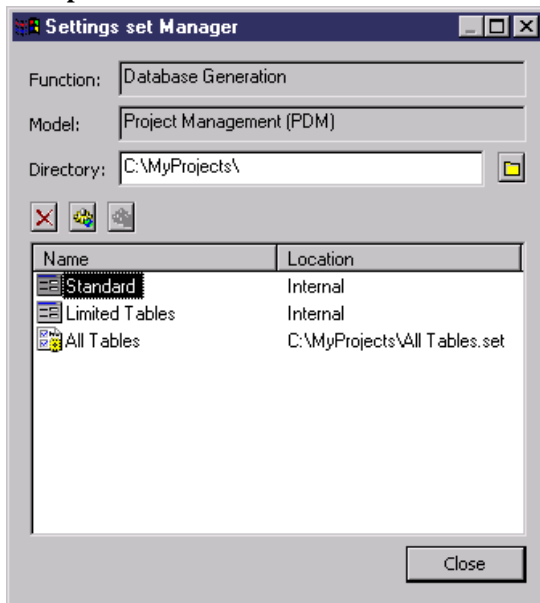
Quick Launch Selection and Settings Sets

The Quick Launch groupbox at the bottom of the General tab allows you to load pre-configured selections and settings sets for use when generating the database.

- Selection - the ensemble of selections of database objects made on the **Selection** tab (see *Database Generation dialog Selection tab* on page 356). 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 351) and format options (see *Database Generation dialog Format tab* on page 354)



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.

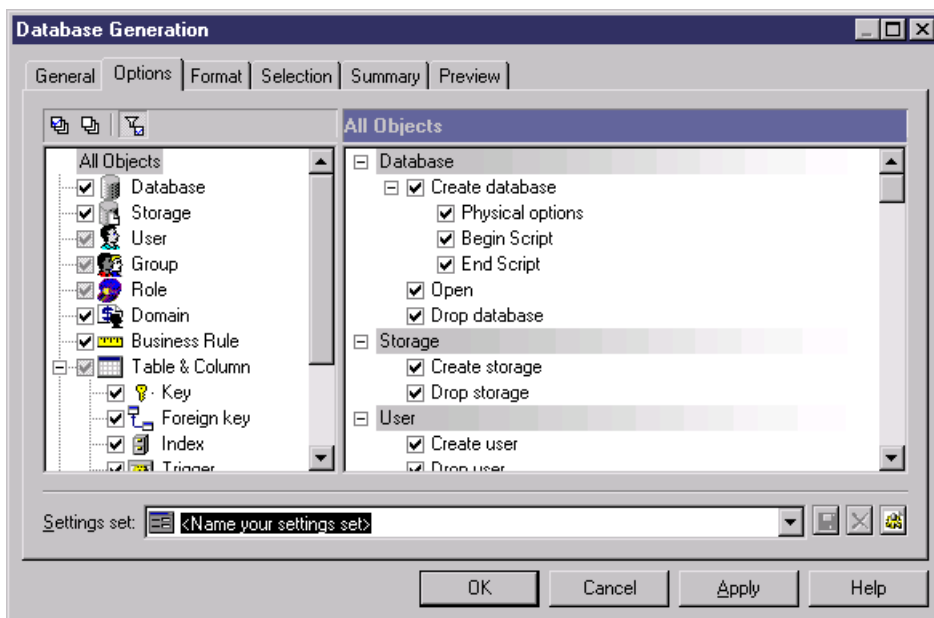
Icon	Use
	Export the selected settings sets to an external file. Only available when an internally-saved settings set is selected.
	Import the selected settings sets to inside the model. Only available when an externally-saved settings set is selected.

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.

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>	<p>Generates the object.</p> <p>When generating primary, alternate, or foreign keys or indexes, you can choose between:</p> <ul style="list-style-type: none"> • 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 <p>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.</p>
Drop <object>	<p>Deletes an existing object, before recreating it.</p> <p>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 section Default in the DBMS Resource File Reference chapter in the <i>Customizing and Extending PowerDesigner</i> manual.</p>
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	<p>[domains, tables, and columns] Generates check parameters and validation rules for domains, tables, and columns.</p> <p>For table and columns, if this option is selected you can choose between:</p> <ul style="list-style-type: none"> • 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 <p>The generation of checks outside the table is possible only if the AddTableCheck entry exists in the Table category of your DBMS.</p>
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.

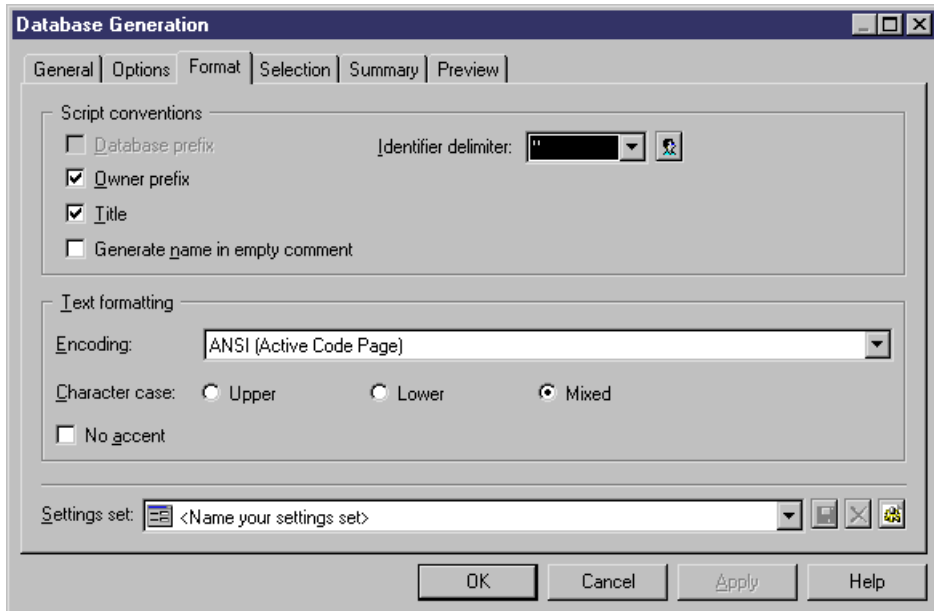
Parameter	Description
User-defined type	[columns] Generates a user-defined data type for the column.
Decl. Integrity	[foreign keys] Generates declarative referential integrity for references specified to be Declarative in their property sheets. You can specify any or all of the following: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • For insert • For update • For delete
Synonym Filter	[synonyms] You can specify from none to all of: <ul style="list-style-type: none"> • 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	<p>[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:</p> <pre>select a, b from Table1</pre> <p>columns a and b are view columns by default. The default generation statement is:</p> <pre>create view V1 as select a, b from Table1</pre> <p>If you select the Force column list option, the generation statement will become:</p> <pre>create view V1(a,b) as select a, b from Table1</pre>

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 350.

Database Generation Dialog Format Tab

The options on the Format tab allow you to control the format of database generation scripts.



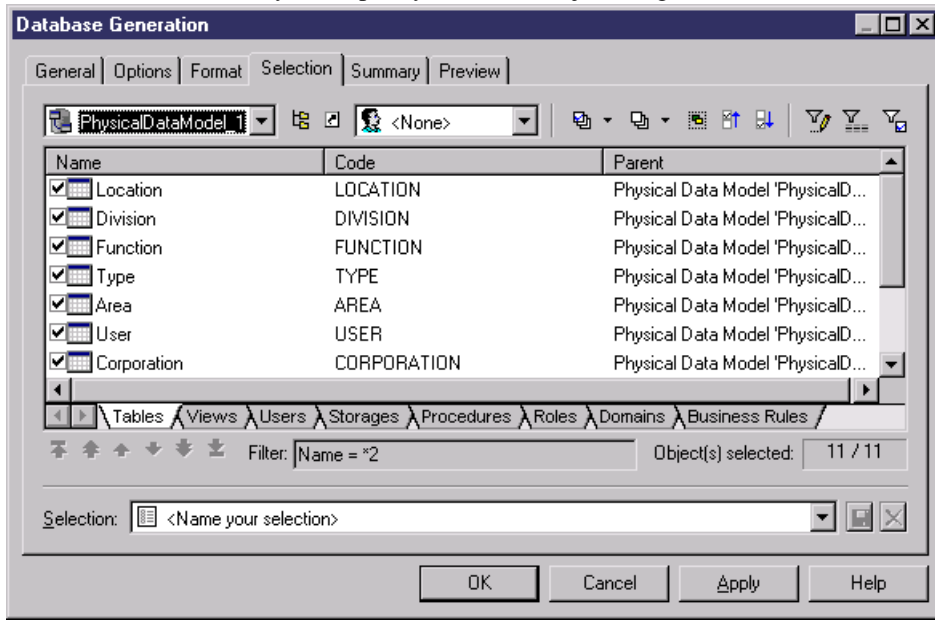
Some of the following options may not be available, depending on your target database.

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 350.

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: <ul style="list-style-type: none"> • Upper - all uppercase characters • Lower - all lowercase characters • Mixed - lowercase and uppercase characters
No accent	Non-accented characters replace accented characters in script

Database Generation Dialog Selection Tab

The Selection tab allows you to specify individual objects to generate.



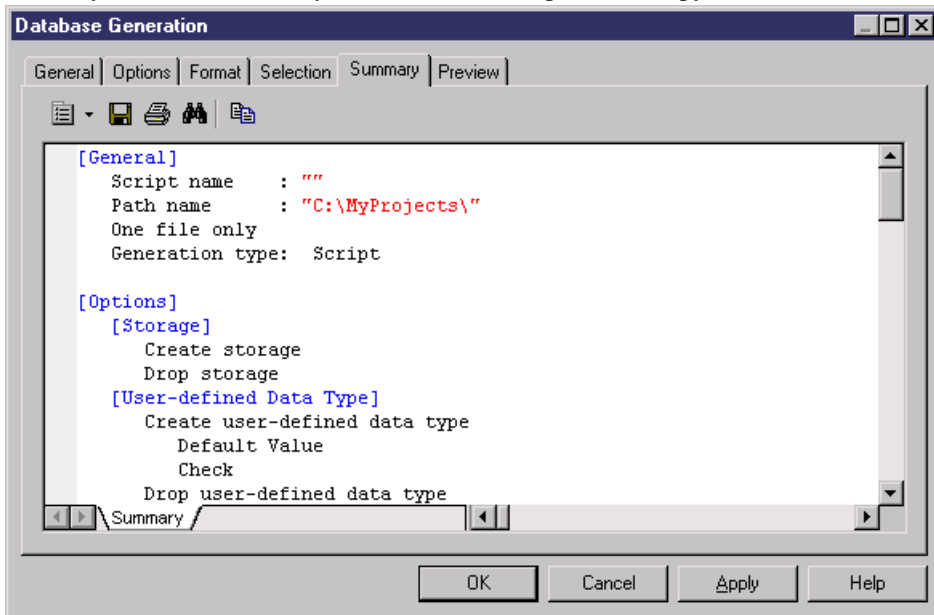
By default, all the objects in the model except for those that belong to a package, or that are shortcuts from another model are listed and selected for generation.

You can save your selection via the Selection bar at the bottom of the tab. For more information, see *Quick launch selection and settings sets* on page 350.

For more information about Selection windows, see "Adding an item from a selection list" section in the Objects chapter of the *Core Features Guide*.

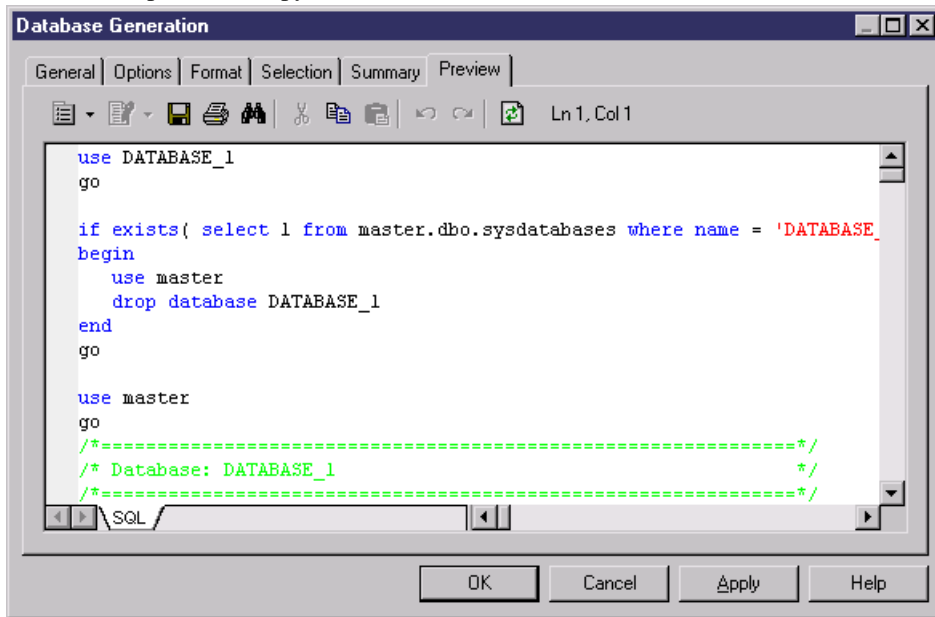
Database Generation Dialog Summary Tab

The Summary tab allows you to view a summary of your generation options. The option summary is not editable, but you can search, save, print, and copy its contents.



Database Generation Dialog Preview Tab

The Preview tab allows you to view the SQL script. The script is not editable, but you can search, save, print, and copy its contents.



Generating a Microsoft Access 97 Database

PowerDesigner and MS Access 97 use .DAT files to exchange information. These files are created from the PDM files via the script generation. The access.mdb database uses or creates .DAT files to generate or reverse Access databases.

You can define the database generation parameters from the access.mdb database window.

1. Generate the database script from PowerDesigner.
2. Double-click access.mdb in the PowerDesigner \tools directory.
3. Select Generate Access Database from PowerDesigner Script File.
4. Type the destination database in the Select Database box.
5. Type the file created by PowerDesigner in the PowerDesigner File.
6. Click on the Create button.



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 "Specifying Text Editors" in <i>Chapter 8, Customizing Your Modeling Environment of the Core Features Guide</i>)	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

Variable	Description
%PATHSCRIPT%	Filename and path of script file
%STARTCMD%	Command that runs the script
%AUTHOR%	Author of the current model

For a complete list of the variables available and how to format them, see *PDM Variables* on page 404 and *Formatting Variables* on page 416.

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%	<i>Column list</i>
%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

Variable	Description
%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 *PDM Variables* on page 404 and *Formatting Variables* on page 416.

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

Generating Test Data to a Database

Test data is sample data that PowerDesigner generates as rows in your database tables to verify database 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: As triggers can considerably increase the time required to generate the database and can block insertions, and as they are not needed in this context, we recommend that you do not implement triggers or remove them if you are using an existing test database.

Test data generation uses test data profiles (see *Test Data Profiles* on page 109) to define the data to generate. We recommend that you create test data profiles to accurately model your data and associate them with the appropriate columns or domains, but if you have not done so, then default profiles are used.

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. Select **Database > Generate Test Data** to open the Test Data Generation dialog box.
2. On the **General** tab (see *Test Data Generation General tab* on page 362), specify whether you want to perform a Script, Direct, or Data file generation and complete the appropriate parameters.
3. [optional] Click the **Number of Rows** tab (see *Test Data Generation Number of Rows tab* on page 364) to change the number of rows to be generated for each table.
4. [optional] Click the **Format** tab (see *Test Data Generation Format tab* on page 365) and modify the script formatting options.
5. [optional] Click the **Selection** tab and select which tables you want to generate test data for. By default all tables are selected.

6. Click **OK** to start the generation.

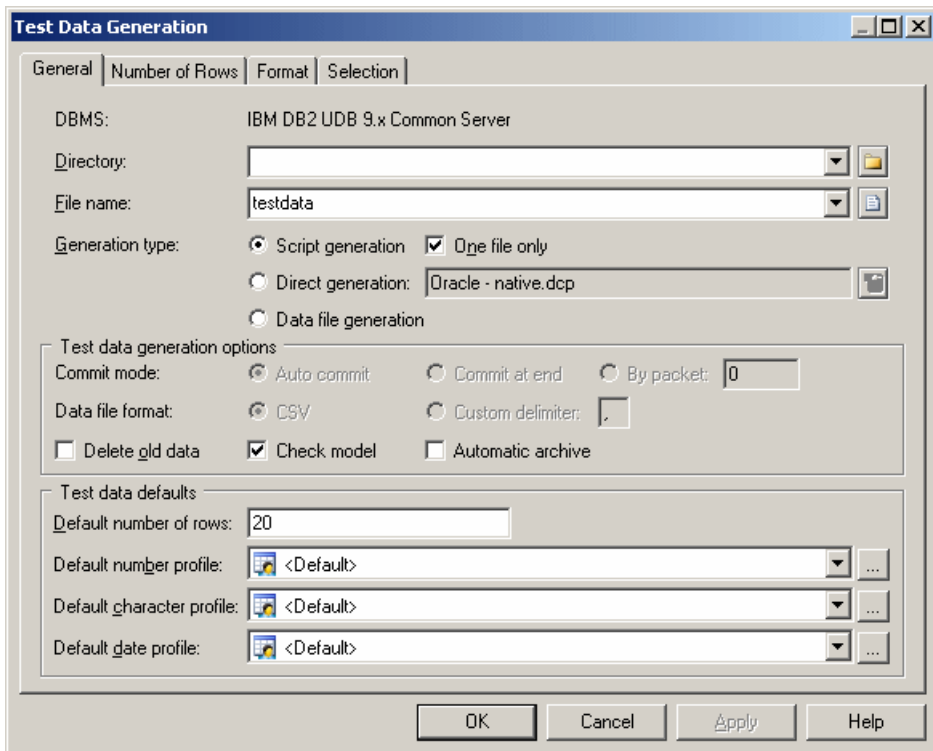
If you are generating a test data script, then a Result dialog box asks you if you want to Edit or Close the newly generated file.

If you are generating test data to a live database connection, then the Connect to a Data Source dialog box opens. Select a data source, and then click **Connect**.

A message in the Output window indicates that the test data generation is completed.

Test Data Generation General Tab

The General tab allows you to specify where you will generate your test data and to specify default test data profiles.



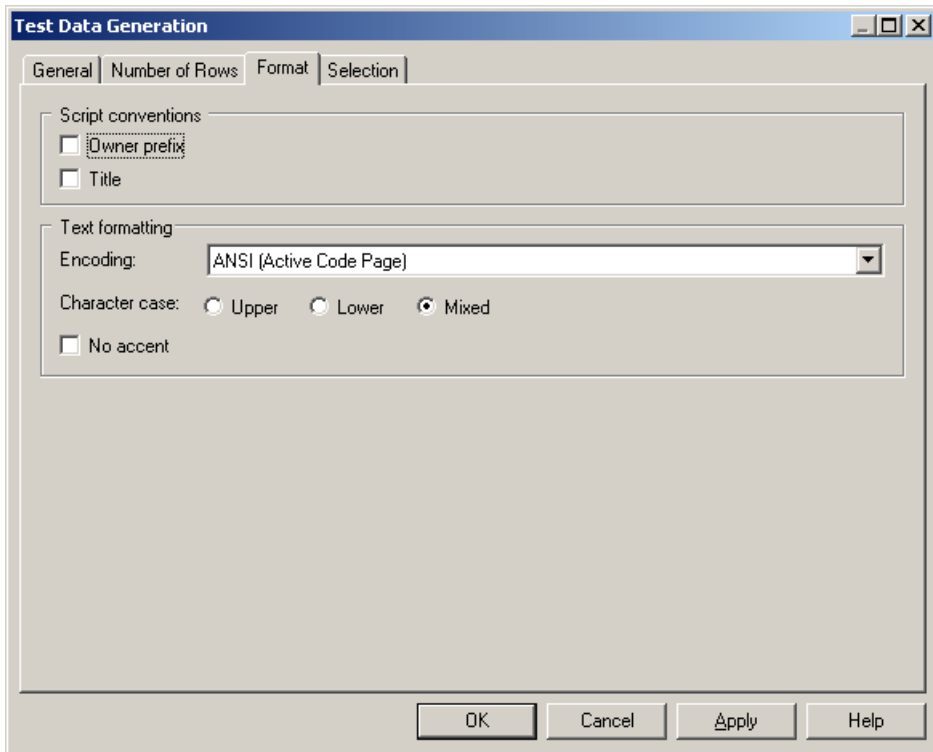
The following options are available on this tab:

Option	Result of selection
Directory	Specifies the directory in which the file will be saved.
File name	Specifies the name of the file. Select the One file only checkbox to specify that only a single file can be generated.

Option	Result of selection
Generation type	<p>Specifies how the test data will be generated. The following settings are available:</p> <ul style="list-style-type: none"> • Script generation - in DBMS-specific syntax • Direct generation – to a live database connection • Data file – as a set of values in a file
Commit mode	<p>Specifies when the data will be committed. The following settings are available:</p> <ul style="list-style-type: none"> • Auto - automatically during script generation • At end - at end of script generation • By packet - at defined intervals during script generation
Data file format	<p>For use with the data file option. The following settings are available:</p> <ul style="list-style-type: none"> • CSV – comma-separated values • Custom delimiter – specify a custom delimiter
Delete old data	Deletes existing data before generating new data..
Check model	Checks the PDM before generating the test database or script, and stops generation if an error is found.
Automatic archive	Creates an archive of any previous test data.
Default number of rows	Specifies the default number of rows for table
Default number profile	Specifies the default number profile for table
Default character profile	Specifies the default character profile for table
Default date profile	Specifies the default date profile for table

Test Data Generation Format Tab

The Format tab allows you to set script format options.



The following options are available on this tab:

Option	Result of selection
Owner prefix	Specifies that an owner prefix is added.
Titles	Specifies that each section of the script includes commentary in the form of titles.
Encoding	Specifies the encoding format to use for test data generation. You should select the encoding format that supports the language used in your model and the database encoding format.
Character case	Specifies the character case to use. The following settings are available: <ul style="list-style-type: none"> • Upper - all uppercase characters • Lower - all lowercase characters • Mixed - both uppercase and lowercase characters

Option	Result of selection
No accent	Non-accented characters replace accented characters in script.

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:

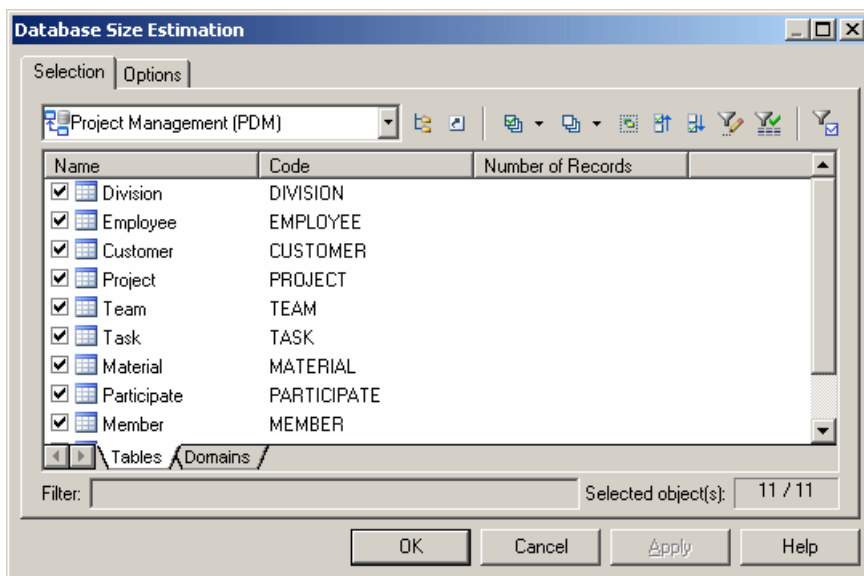
- Estimated number of records in tables - Specify the number of rows (and their annual projected growth rate) in a table in the **Number** and **Row growth rate** fields on the **General** tab of its property sheet (see *Table Properties* on page 82).
- Table columns and their sizes - Specify the average size for variable length columns in the **Average length** field on the **Detail** tab of its property sheet (see *Column Properties* on page 103). 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 *Modifying the Estimate Database Size Mechanism* in *Chapter 2, DBMS Resource File Reference* in *Customizing and Extending PowerDesigner*).

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**.

The output list displays the estimated database size in KB, and provides a warning if any tables in the model have not had a number of a records defined:

```

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).

```

Modifying a Database

You can modify an existing database schema by synchronizing it with your model.

The existing schema can be in the form of:

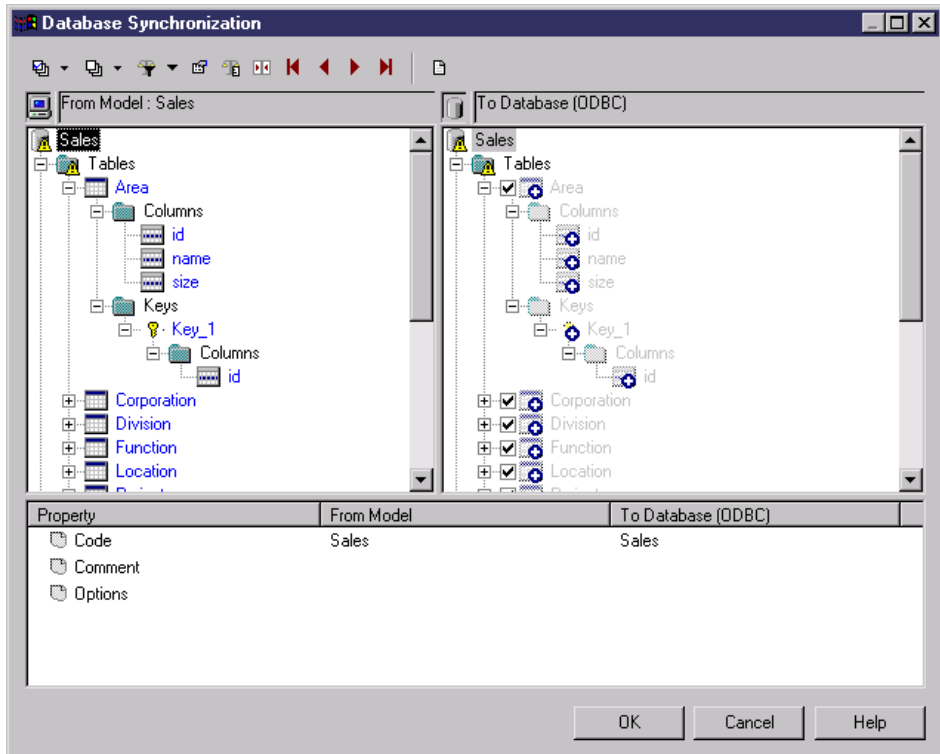
- an archive model
- a live database connection
- a script file
- a model from the repository

The PDM (source model) and the existing database schema (target model) are merged using a database synchronization window, which allows you to choose which objects are added, deleted, or updated in the target.

1. Select **Database > Apply Model Changes to Database** to open the Apply Model Changes to Database dialog box.
2. Type a destination directory and filename for the script file in the Directory and File Name boxes.
3. Specify the type of generation to perform. You can choose between a script and a live database connection.
4. Specify how PowerDesigner will obtain the database schema to modify. You can choose between:
 - Using an archive model – Click the button to the right to browse to the archived model.
 - Using a data source – Click the button to the right to connect to your data source.
 - Using a script file – Select a script from the list or click the button to the right to browse to the script.
 - Using a model from repository – Click the Change Model Version tool to the right to browse to a version of the currently selected model.
5. If you want to retain your existing data, select the Backup Tables option. If this option is not selected, then all existing data will be erased. For details of this and other options on this tab, see *Apply Model Changes to Database Dialog Options* on page 370.
6. [optional] If you want to change the default generation options, then click the Options tab. For more information about these options, see *Apply Model Changes to Database Dialog Options* on page 370.
7. [optional] If you want to change the format of your script, then click the Format tab. This tab has the same functionality as in the Database Generation window (see *Database Generation dialog Format tab* on page 354).
8. [optional] If you want to control which database objects will be modified, then click the Selection tab. This tab has the same functionality as in the Database Generation window (see *Database Generation dialog Selection tab* on page 356)
9. Click OK. If you are using a live database connection, then the Reverse Engineering window will open, allowing you to select or clear check boxes in the target model for

objects that you want to include or remove from the source model. Make your selections and then click OK to continue.

10. The Database Synchronization window will open. Select or clear check boxes in the target model for objects that you want to include or remove from the model.



For more information on comparing and merging models, see *Chapter 7, Comparing and Merging Models* in the *Core Features Guide*.

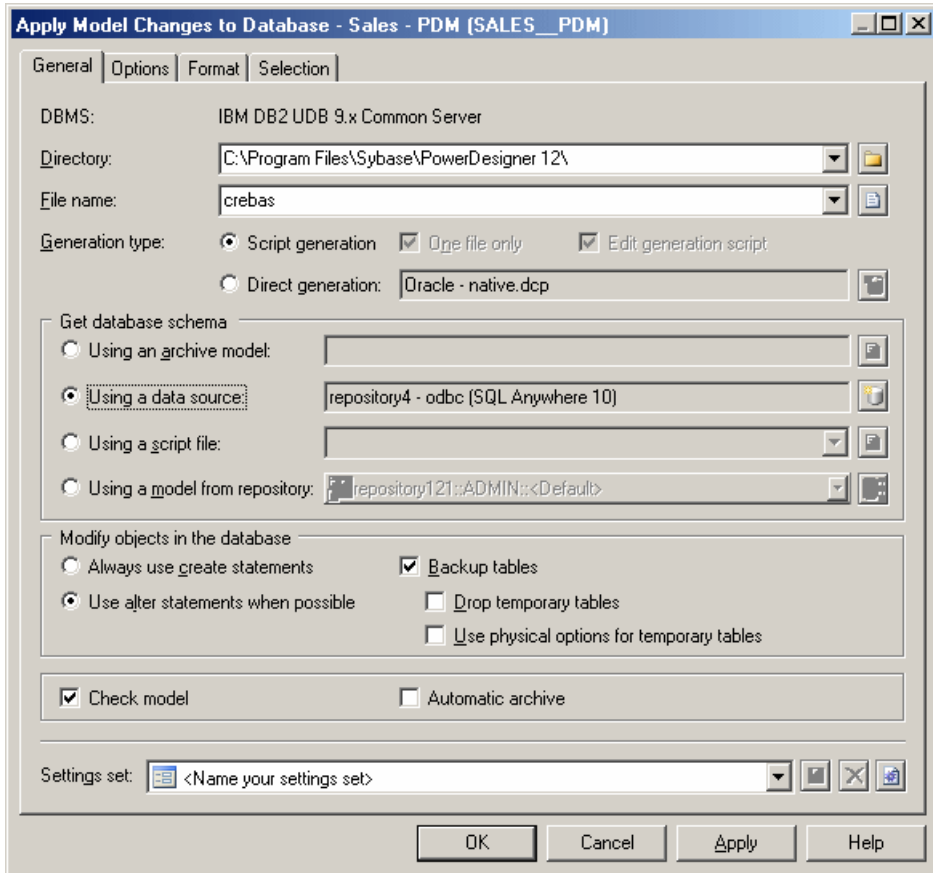
11. Click OK.

- If you are generating a script, at the end of generation a result box opens listing the file path of the generated file. To open the script in a text editor, click the file in the result box and click the Edit button. To close the Result box, click the Close button.
- If you are generating a database directly, a Data Source connection box is displayed. Type your connection details and click the Connect button. A message box shows the progress of the generation process. At the end of generation click OK to close the box.

Apply Model Changes to Database Dialog Options

There are four tabs of options on this dialog tab controls the main options for database modification.

General tab



The following options are available on this tab:

Option	Description
Directory	[required] Specifies the destination directory for the script file.
File name	[required] Specifies the destination filename for the script file.
One file only	Specifies that the generation script is created as a single file. By default, a separate script file is created for each table.

Option	Description
Generation Type	Specifies the type of generation to perform. You can choose between: <ul style="list-style-type: none"> • Script generation - generate a script to be executed on a DBMS at a later time • Direct generation – generate a script and execute it on a live database connection
Edit generation script	[available only when direct generation is selected] Opens the generation script in a text editor for review or editing before execution on a live database connection.
Obtains database schema	Select the kind of schema that the model will modify. You can choose between: <ul style="list-style-type: none"> • Using an archive model - Modified PDM is merged with an archived PDM. • Using a data source - Modified PDM is merged with a reverse engineered database schema for a live database connection. • Using a script file - Modified PDM script file is merged with an existing database script file. • Using a model from repository - Modified PDM is merged with a selected version of a PDM consolidated in the repository.
Backup tables	Specifies that any existing table will be copied to temporary backup tables during the modification, and then restored to the updated tables. If this option is not selected, then all existing data will be erased.
Always use create statements/ Use alter statements when possible	Select a radio button to specify whether create statements should always be used to modify database tables, or whether alter statements should be used where possible
Drop temporary tables	[available only when Backup Tables is selected] Specifies that the temporary backup tables are removed after script execution.
Use physical options for temporary tables	[available only when Backup Tables is selected] Specifies that the temporary backup tables are generated with their physical options.
Check model	Specifies that a model check is performed before script generation.
Automatic archive	Creates an archive version of the PDM after generation.

Options Tab

The availability of these options is dependent on the selected DBMS:

Option	Result of selection
Inside/ Outside	Specifies whether and where various constraints will be generated. Note that if alter statements are used to modify tables, then constraints may be generated outside the table even if the inside radio button is selected here.
Index Filter	Specifies which kinds of indexes to generate.
Comment	Specifies whether comments are generated.

You can load option settings previously used for database generation via the Settings set bar at the bottom of the tab. For more information, see *Quick Launch Selection and Settings Sets* on page 350.

Format Tab

This tab controls the format of your modification script, and has the same functionality as the equivalent tab in Database Generation (see *Database Generation dialog Format tab* on page 354).

Selection Tab

The Selection tab allows you to specify individual objects to generate, and has the same functionality as the equivalent tab in Database Generation (see *Database Generation dialog Selection tab* on page 356).

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.

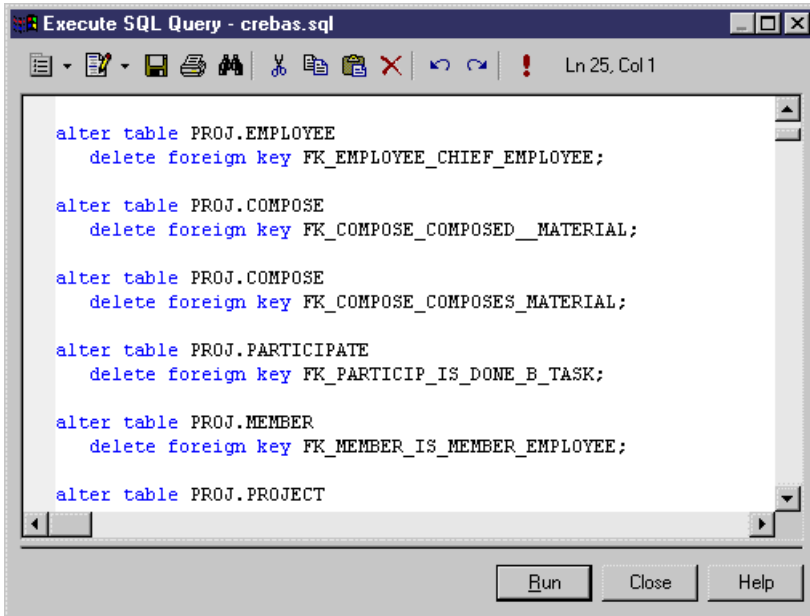
Executing SQL Queries

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.



```

alter table PROJ.EMPLOYEE
    delete foreign key FK_EMPLOYEE_CHIEF_EMPLOYEE;

alter table PROJ.COMPOSE
    delete foreign key FK_COMPOSE_COMPOSED__MATERIAL;

alter table PROJ.COMPOSE
    delete foreign key FK_COMPOSE_COMPOSES_MATERIAL;

alter table PROJ.PARTICIPATE
    delete foreign key FK_PARTICIP_IS_DONE_B_TASK;

alter table PROJ.MEMBER
    delete foreign key FK_MEMBER_IS_MEMBER_EMPLOYEE;

alter table PROJ.PROJECT

```

The query results are displayed in the Results window.

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 existing PDM from one or more script files or from a live database.

Note: When you reverse engineer a database, whether from a script or a data source, make sure that you set the rebuild options appropriately. Click the Options tab and select or clear the checkboxes to rebuild references and/or primary keys according to your needs. By default, no rebuild options are selected.

Reverse Engineering from Scripts

PowerDesigner can reverse engineer a PDM for one or more SQL script files. 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.

If you use more than one script files, make sure that the order of the files respects dependencies among objects (for example, trigger creation scripts must come after table creation scripts; and grant permission scripts must come after both table and user creation scripts).

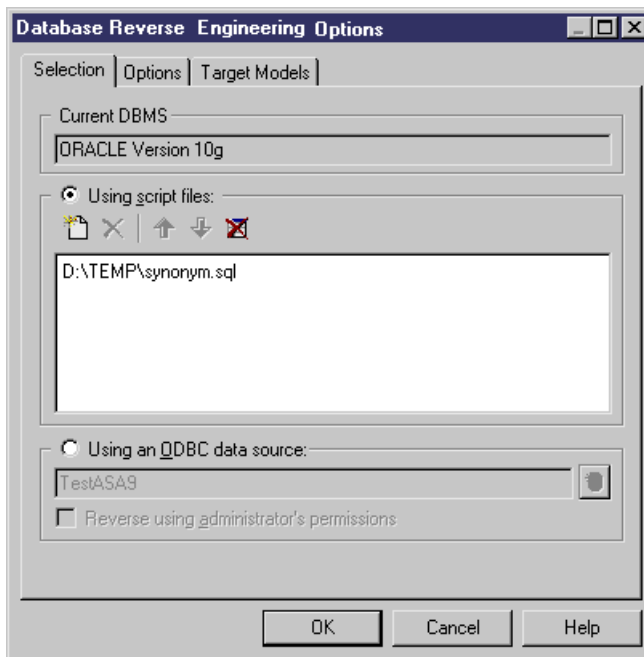
Note: When working with the PowerDesigner Eclipse plug-in you can, in addition to the procedure below, select any SQL file in the Navigator, right-click it and select Reverse Engineer from SQL File. You will be given the option to reverse into an existing or new PDM.

1. To reverse engineer a script into an existing PDM, select **Database > Update Model from Database**.



or



To reverse engineer a script and create a new PDM, select **File > Reverse Engineer > Database** to open the New Physical Data Model dialog box. Specify a model name, choose a DBMS from the list, and then click **OK**.

2. When the Database Reverse Engineering Options dialog opens, click the **Using script files** radio button.



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.

Tool	Description
	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. The reverse engineering process handles files sequentially. Trigger scripts must always be executed after table scripts. This is the only constraint for ordering your files in the list, but it is essential for a successful reverse engineering of triggers.

3. Click the **Options** tab to specify any reverse engineering options (see *Reverse Engineering Options Tab* on page 378).
4. Click the Target Models tab to specify any external shortcuts (see *Reverse Engineering Target Models tab* on page 382).
5. Click **OK** to begin the process of reverse engineering. When the process is complete, a confirmation message is given in the Output window. If you are reverse engineering to an existing PDM, then the Merge Models dialog box opens to help you merge the new objects into your PDM.

For more information on comparing and merging two models, see *Chapter 7, Comparing and Merging Models* in the *Core Features Guide*.

Reverse Engineering from a Live Database

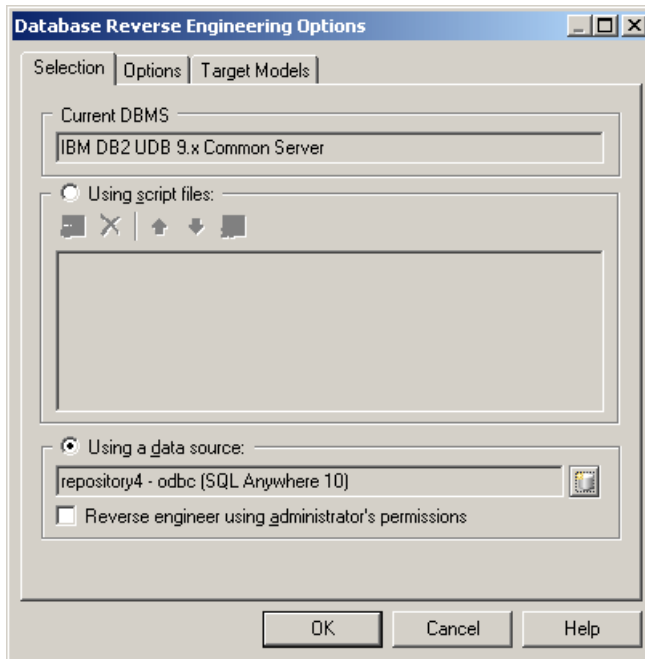
PowerDesigner can reverse engineer a PDM from a live database connection. You must specify a data source and connection information. You can select to use administrator permissions in order to be able to select the system tables that are reserved to a database admin.

1. To reverse engineer from a live database connection into an existing PDM, select **Database > Update Model from Database** to open the Database Reverse Engineering Options dialog box.

or

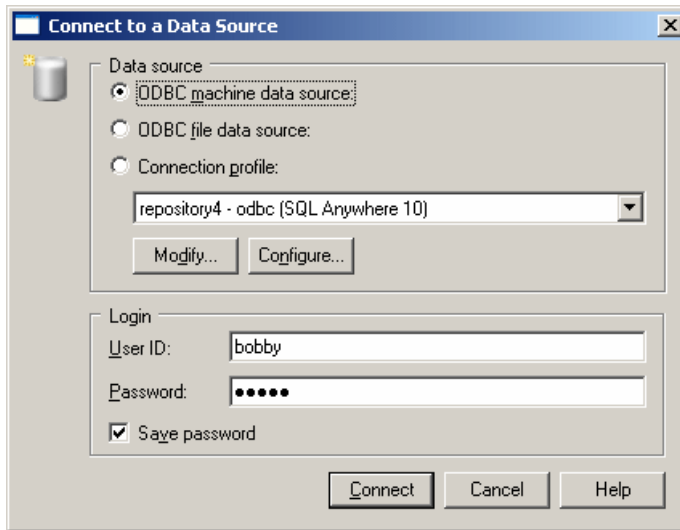
To reverse engineer from a live database connection and create a new PDM, select **File > Reverse Engineer > Database** to open the New Physical Data Model dialog box. Specify a model name, choose a DBMS from the list, and then click OK.

2. When the Database Reverse Engineering Options dialog box opens, click the Using a data source radio button.

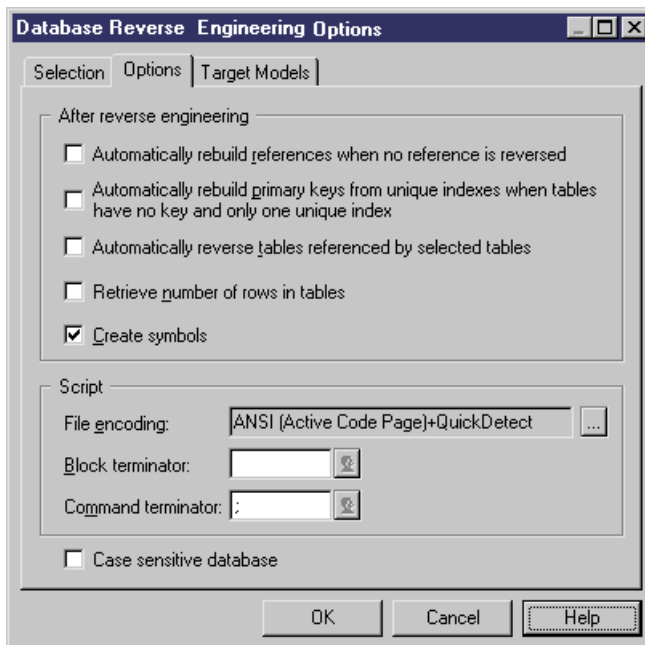


Note: A data source might be predefined, or you can type the name of an existing data source. In both cases, when you click OK, a database connection dialog box opens, if you need to specify additional connection parameters. Click Connect and the Database Reverse Engineering dialog box is displayed. (Go to step 9)

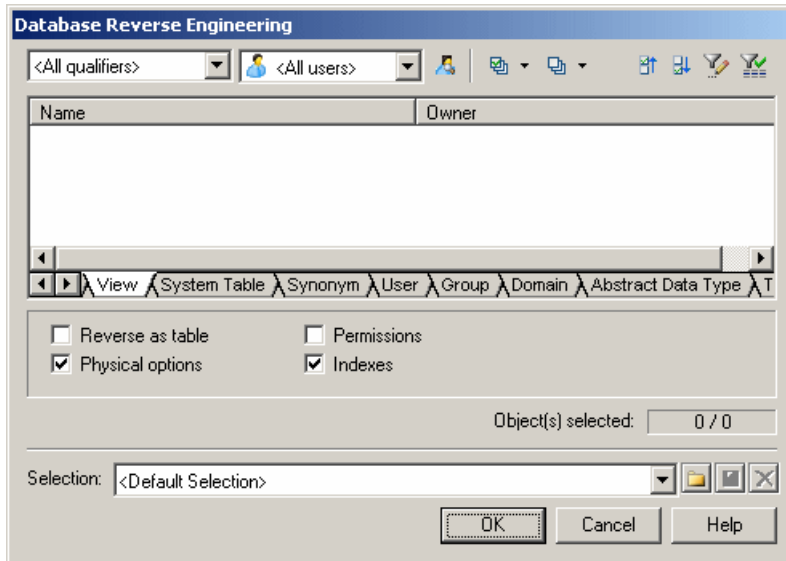
3. Click the *Connect to a Data Source* tool to open the Connect to an ODBC Data Source dialog box.



4. Select the appropriate source, type a user ID and a password, and then click **Connect** to return to the Database Reverse Engineering Options dialog box.
5. If you want to select tables reserved to the database administrator, then you must select the *Reverse using administrator's permissions* check box.
6. Click the **Options** tab to specify any reverse engineering options. For more details, see *Reverse engineering Options tab* on page 378.



7. Click the Target Models tab to specify any external shortcuts. For more details, see *Reverse engineering Target Models tab* on page 382.
8. Click OK to open the ODBC Reverse Engineering dialog box. This box allows you to specify a selection of objects to reverse engineer. Only tables and triggers are selected by default.



For more information about selecting objects, see *Database Reverse Engineering Selection window* on page 381.

9. Click OK to begin the process of reverse engineering. When the process is complete, a confirmation message is given in the Output window. If you are reverse engineering to an existing PDM, then the Merge Models dialog box opens to help you merge the new objects into your PDM.

For more information on comparing and merging two models, see the Comparing and Merging Models chapter in the *Core Features Guide*.

Reverse Engineering Options Tab

When you reverse engineer a database schema using script files or a data source, you can define rebuild options after reverse engineering.

The rebuild options automatically perform the following tasks after reverse engineering:

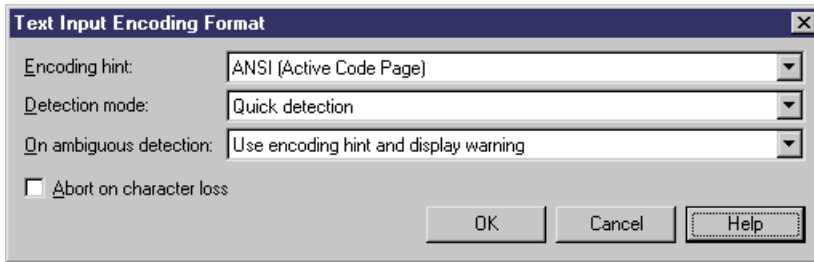
Reverse options	Description
Automatically rebuild references when no reference is reversed	Rebuilds references when no references are reverse engineered. The rebuild references feature starts by detecting columns with identical name and data type in different tables. A reference is created between each column belonging to a primary key and a column, with identical name and data type, that does not belong to a primary or a foreign key in another table.
Automatically rebuild primary keys from unique indexes when tables have no key and only one unique index	Rebuilds primary keys using unique indexes when tables have no key and only one unique index.
Automatically reverse tables referenced by selected tables	Reverse engineers the parents of the selected child tables in order to complement the definition of these child tables.
Create symbols	<p>Creates a symbol for each reversed object in the diagram. Otherwise, reversed objects are visible only in the browser.</p> <p>The layout of the symbols in the diagram will be automatically arranged. In cases where there are a large number of objects with complex interactions, the auto-layout feature is likely to create synonyms of objects to improve the diagram readability. For example, if a table has a large number of references, the auto-layout feature will create a synonym of this table in another location of the diagram in order to improve the diagram presentation.</p>
Apply code to name conversion to reversed objects	Applies the code to name conversion script specified in the model options (see "Naming Conventions" in <i>Chapter 8, Customizing Your Modeling Environment of the Core Features Guide</i>).
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 <i>Reverse engineering encoding format</i> on page 380).
Block terminator	Specifies the end of block character for the reversed script. By default, displays the value defined in the DBMS, under Script\SQL\Syntax. You can modify this value, in which case it will be saved in the Registry for reuse in other models. You can restore the DBMS value using the Restore from DBMS tool.
Command terminator	Specifies the end of command character for the reversed script. By default, displays the value defined in the DBMS, under Script\SQL\Syntax. You can modify this value, in which case it will be saved in the Registry for reuse in other models. You can restore the DBMS value using the Restore from DBMS tool.
Case sensitive database	Specifies that the database is case sensitive and enables the case sensitive option in the model.

For more information on indexes, rebuilding references and rebuilding primary keys, see *Chapter 3, Building Physical Diagrams* on page 79.

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	<p>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.</p> <p>The following settings are available:</p> <ul style="list-style-type: none"> • 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	<p>Specifies what action should be taken in case of ambiguity. The following settings are available:</p> <ul style="list-style-type: none"> • Use encoding hint and display warning - the encoding hint format is used and a warning message is displayed. • Use encoding hint - the encoding hint format is used but no warning message is displayed. • Use detected encoding - the encoding format detected by PowerDesigner is used
Abort on character loss	Allows you to stop reverse engineering if characters cannot be identified and are to be lost in current encoding

Here is an example on how to read encoding formats from the list:

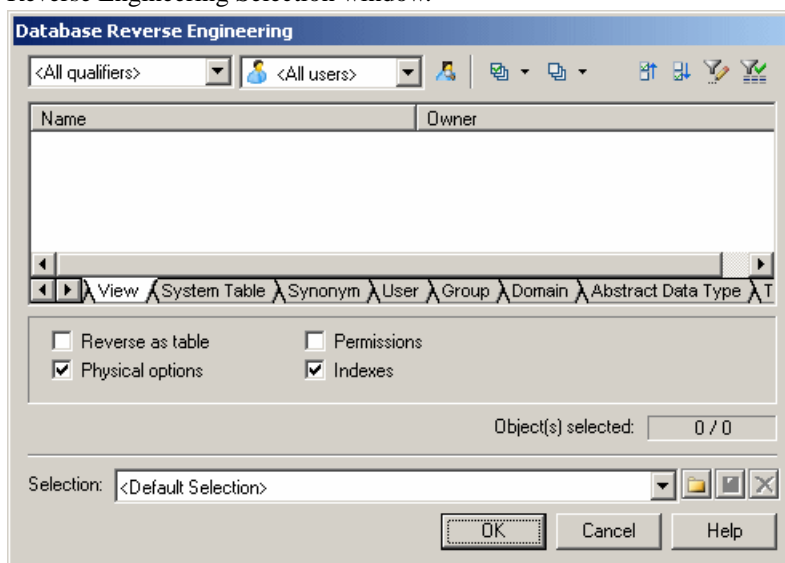
```

ASCII
OEM
UTF-8 -----> No Byte-Order-Mark in the header
UTF-8 (with signature)
Unicode
Unicode (with signature) -----> There must be a Byte-Order-Mark
Unicode big endian                in the header for the file to be valid
Unicode big endian (with signature)
ANSI (Active Code Page)

```

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.

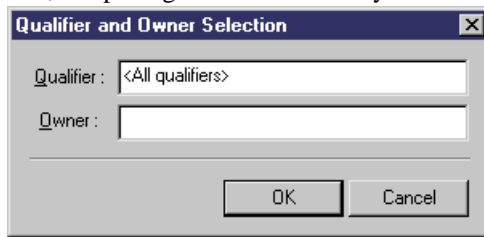


Click the subtabs to view the different types of objects. Certain object types have attributes, or options, that appear below the object lists. Options that are not available for the selected object type or DBMS are grayed. When you select tables containing triggers on the **Table** tab, the triggers 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.

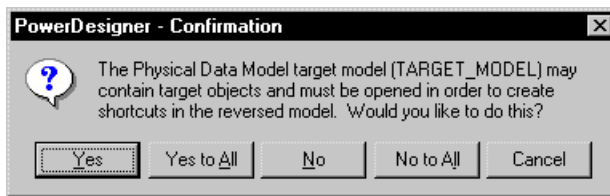
This tab is always visible, even if the model does not contain shortcuts, so that you can add target models and create shortcuts instead of duplicating objects.

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.

Reverse Engineering a Microsoft Access 97 Database

PowerDesigner and MS Access 97 use .DAT files to exchange information. These files are reversed into the PDM. The access.mdb database uses or creates .DAT files to reverse Access databases.

You can define the database reverse parameters from the access.mdb database window.

1. Double-click ACCESS.MDB in the PowerDesigner \tools directory.
2. Select Reverse Engineer Access Database File to PowerDesigner Script.
3. Type the Access database name in the Select Database box.
4. Type the .DAT file to create in the PowerDesigner File.
5. Click the Create button.
6. Select **DBMS > Reverse Engineering Databases** in PowerDesigner.

7. Select the newly generated script file to reverse.
8. Click OK.

Optimizing Live Database Reverse Engineering Queries

Live database reverse engineering has been optimized in order to improve performance. All queries run according to an optimization process rule.

This process uses the following registry keys:

- RevOdbcMinCount defines a number of selected objects for reverse engineering. The default number is 100
- RevOdbcMinPerct defines a percentage of selected objects for reverse engineering. The default percentage is 10

These keys do not exist by default, you have to create and edit them in the Registry under:

```
Current User \Software\Sybase\PowerDesigner <version>\FolderOptions  
\Physical Objects
```

During reverse engineering, PowerDesigner compares the total number of current objects for reverse engineering to the value of RevOdbcMinCount, and if the total number of listed items is:

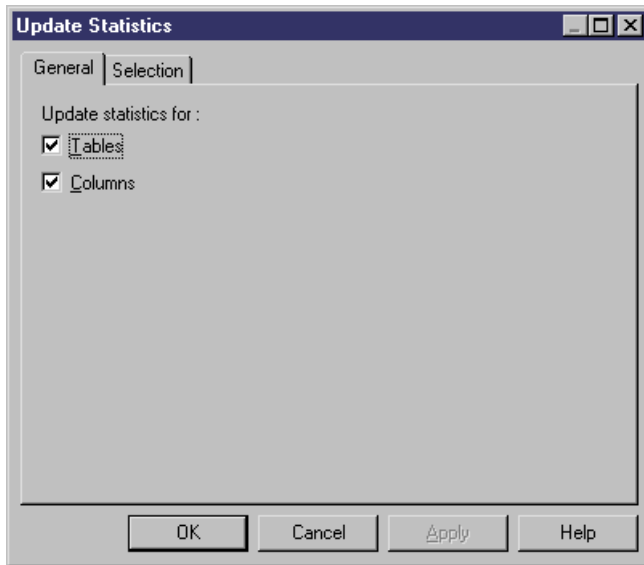
- lower than RevOdbcMinCount - then a global reverse query is executed.
- higher than RevOdbcMinCount - then the process uses key RevOdbcMinPerct, and if the percentage of reversed items is :
 - lower than RevOdbcMinPerct - then the same query is executed for each object.
 - higher than RevOdbcMinPerct - then a global query is executed.

Reverse Engineering Database Statistics

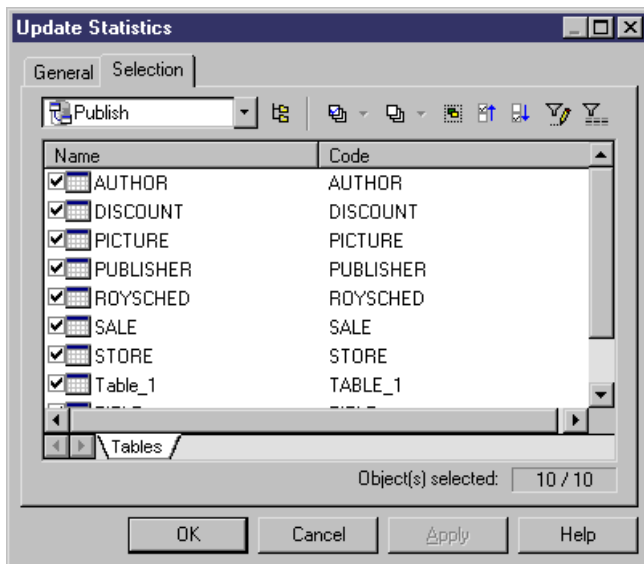
You can reverse engineer statistics for an existing database, such as the number of distinct or null values in a column or the average length of a character field. These can provide helpful information when optimizing a design.

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 375), 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.

CHAPTER 10 Generating Other Models from a Data Model

You can generate the following types of models from CDMs, LDMs, and PDMs:

Data Model	CDM	LDM	PDM	OOM	XSM
CDM	X	X	X	X	
LDM	X	X	X		
PDM	X	X	X	X	X

1. Select Tools, and then one of the following to open the appropriate Model Generation Options Window:
 - Generate Conceptual Data Model... Ctrl+Shift+C
 - Generate Logical Data Model... Ctrl+Shift+L
 - Generate Physical Data Model... Ctrl+Shift+P
 - Generate Object-Oriented Model... Ctrl+Shift+O
 - Generate XML Model... Ctrl+Shift+M
2. On the General tab, select a radio button to generate a new or update an existing model, and complete the appropriate options.
3. [optional – PDM-PDM generation only] Click the DBMS Preserve Options tab and set any appropriate options.
4. [optional] Click the Detail tab and set any appropriate options. We recommend that you select the Check model checkbox to check the model for errors and warnings before generation.
5. [optional] Click the Target Models tab and specify the target models for any generated shortcuts.
6. [optional] Click the Selection tab and select or deselect objects to generate.
7. Click OK to begin generation.

Note: For detailed information about the options available on the various tabs of the Generation window, see *Chapter 10, Generating Models and Model Objects* in the *Core Features Guide*.

Generating Other Models from a CDM

You can generate CDM objects to other model objects.

CDM	OOM	PDM
Entity	Class	Table
Entity attribute	Attribute	Table column
Primary identifier	-	Primary or foreign key depending on independent or dependent relationship
Identifier	-	Alternate key
Association	Relationship or association	-
Binary association with attributes	Association class	-
Inheritance	Generalization	-
Relationship	-	Reference

Persistent Entities (OOM)

All entities are generated as persistent classes with the "Generate table" persistence mode.

When the Generate check box of an entity is not selected, the generated class has the "Migrate columns" persistence mode.

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. The new name is composed of the first three letters of the original entity name followed by the code of the attribute.

Generating PDM Table Keys from CDM Entity Identifiers

The type of key that is generated in the PDM depends on the cardinality and type of dependency defined for a relationship in the CDM. Primary identifiers generate primary and foreign keys. Other identifiers that are not primary identifiers generate alternate keys:

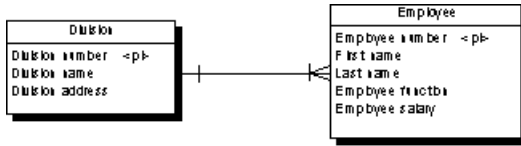
- A *primary key* is a column or columns whose values uniquely identify a row in a table.
- A *foreign key* is a column or columns that depend on and migrate from a primary key column in another table.
- An *alternate key* is a column or columns whose values uniquely identify a row in a table, and is not a primary key.

Independent One-to-many Relationships

In independent one-to-many relationships, the primary identifier of the entity on the one side of the relationship is generated as a:

- Primary key in the table generated by the entity on the one side of the relationship
- Foreign key in the table generated by the entity on the many side of the relationship

The following CDM shows an independent relationship. Each division contains one or more employees:



The following PDM will be generated:

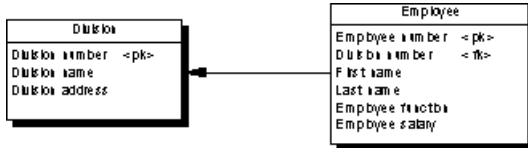
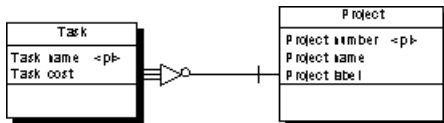


Table	Primary key	Foreign key
Division	Division number	—
Employee	Employee number	Division number

Dependent One-to-many Relationships

In dependent relationships, the primary identifier of the nondependent entity is generated as a primary/foreign key in the table generated by the dependent entity. The migrated column is integrated into the primary key if it already exists.

The following CDM shows a dependent relationship. Each task must have a project number.



The following PDM will be generated:

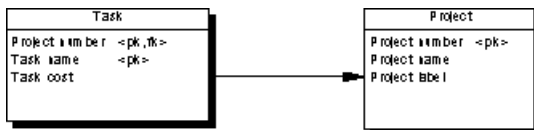
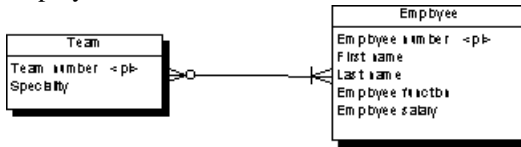


Table	Primary key	Foreign key
Project	Project number	—
Task	Project number/Task number	Project number

Independent Many-to-many Relationships

In independent many-to-many relationships, the primary identifiers of both entities migrate to a join table as primary/foreign keys. The CDM below shows an independent relationship. Each employee can be a member of one or more teams, and each team can have one or more employees as members.



The following PDM will be generated:

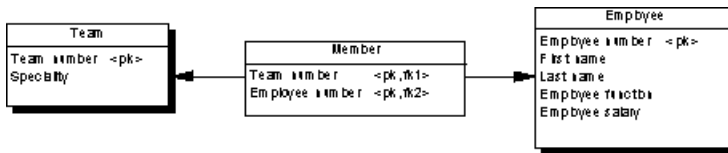


Table	Primary key	Foreign key
Team	Team number	—
Employee	Employee number	—
Member	Team number/Employee number	Team number/Employee number

Independent One-to-one Relationships

In independent one-to-one relationships, the primary identifier of one entity migrates to the other generated table as a foreign key.

Generating Tables from Entities with Inheritance Links

Two properties influence the generation of tables from entities with inheritance links.

Object	Property	When selected generates
Entity	Generate table	Table for the entity (parent or child)
Inheritance	Generation mode	Parent and/or children as indicated

Generating Other Models from an LDM

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 the ".convert_code and .convert_name Macros" section in the Customizing Generation with GTL chapter of the *Customizing and Extending PowerDesigner* manual.

XML Specifics

Generation of column as attribute or element is controlled by generation option

Foreign keys - When a foreign key is not a composition, it is generated as a KeyRef constraint

Oracle 8 and Interbase Sequence Translation

When a CDM is generated from a PDM, the data type of a table column attached to a sequence is translated to a serial data type in the new model.

The resulting CDM serial data type for an entity property has the format NO%n, where %n is a number indicating the length of the data type.

For more information on sequences, see sections on Oracle and Interbase in *DBMS-Specific Features* on page 5.

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 on conversion scripts, see the ".convert_code and .convert_name Macros" section in the Customizing Generation with GTL chapter of the *Customizing and Extending PowerDesigner* manual.

Configuring the Generated Model Options

When you configure the options of a CDM to generate, you may define options diverging from the PDM options.

To avoid conflicts, PowerDesigner applies the following rule for default values of CDM options: an option defined for the generated CDM should respect the equivalent option of the PDM.

Equivalent Enforce non-divergence model options are available in both the PDM and CDM.

PDM option	CDM option	Result in generated CDM
Enforce non-divergence	—	Enforce non-divergence in model according to PDM options. Data items and attributes attached to the domain cannot have divergent definitions
—	Enforce non-divergence	Enforce non-divergence in model according to CDM options defined using the Configure Model Options feature

Relationships Unique Code

(CDM) Unique Code for relationships is not selected by default in the CDM options.

However, if you select Unique Code for relationships in the CDM options, relationships are renamed during the generation of a PDM to a CDM.

Options with no equivalent, like Enforce Profile in the PDM without any corresponding option in a CDM, are generated using default values found in the registry.

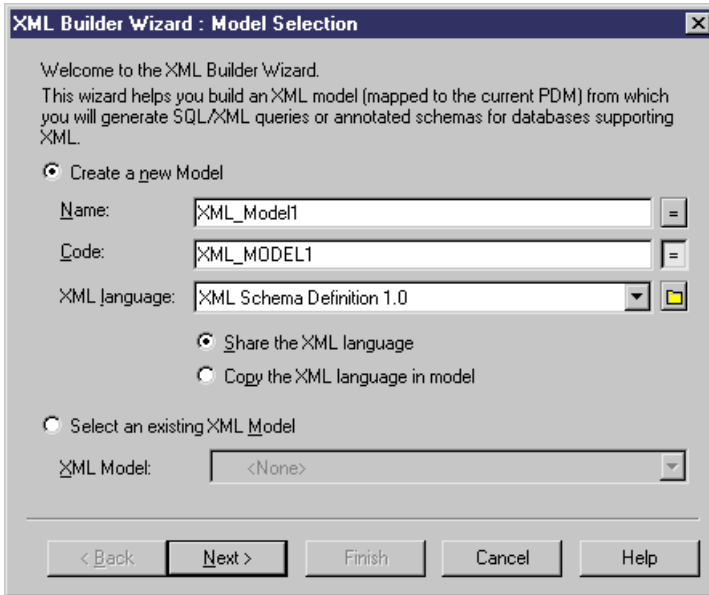
Options with No Equivalent in the Models

(OOM) Options with no equivalent, like Enforce Profile in the PDM without any corresponding option in an OOM, are generated using default values found in the registry.

Generating an XSM from a PDM Via the XML Builder Wizard

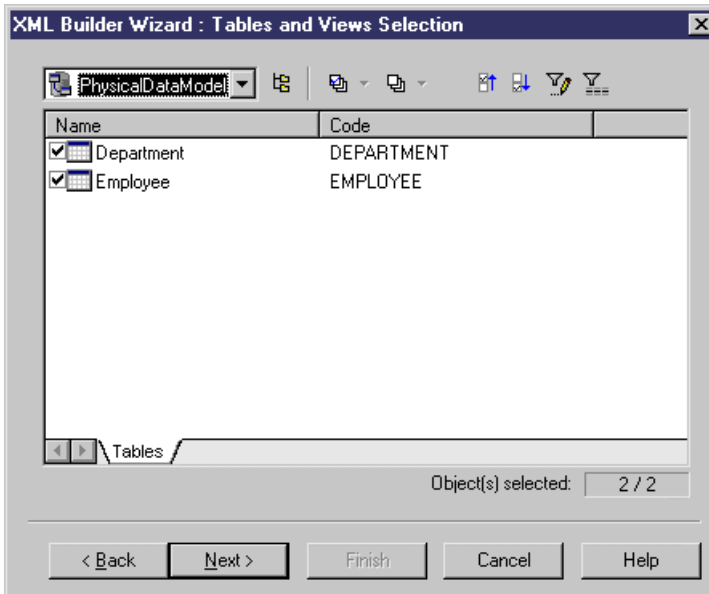
The XML Builder Wizard helps you build an XML model (XSM) that will be used to generate SQL/XML queries for retrieving data from databases. It is more powerful than the standard PDM-XSM generation, as it helps you to customize the XML hierarchy to be built, and sets up the XSM to retrieve data from relational databases supporting SQL/XML, and for generation of an annotated schema.

1. In your PDM, select **Tools > XML Builder Wizard** to open the XML Builder Wizard to the Model Selection page:



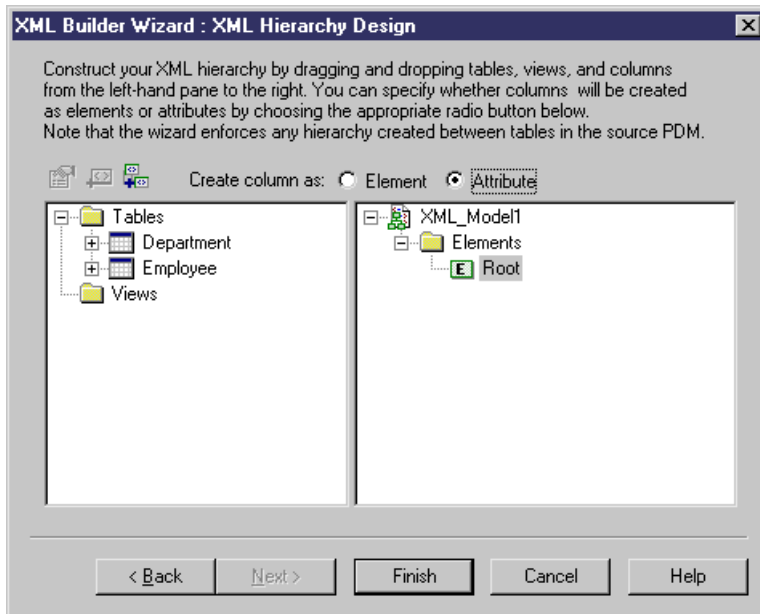
Select whether you want to create a new XML model or update an existing XML model currently open in your workspace.

2. Click Next to go to the Tables and Views Selection tab:



Select the tables and views you want to generate. By default, all tables and views are selected.




3. Click Next to go to the XML Hierarchy Design tab:



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.

The following tools are available to help you build your hierarchy:

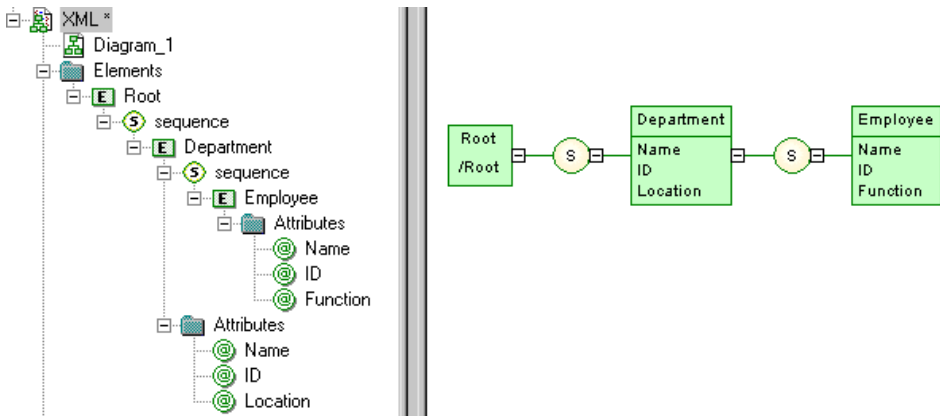
Tool	Description
	Properties – Opens the property sheet for the selected table, view, or column.
	Add Object - Adds the selected PDM object to the XML hierarchy.
	Create Default Hierarchy for Selected Objects - Adds the selected PDM objects to the XML hierarchy.

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** 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**.
- Delete an XML node by right-clicking it and selecting **Delete**.

5. Click Finish to generate the XSM.



In the case of an update to an existing XSM, your hierarchy will be created as a new root in the model.

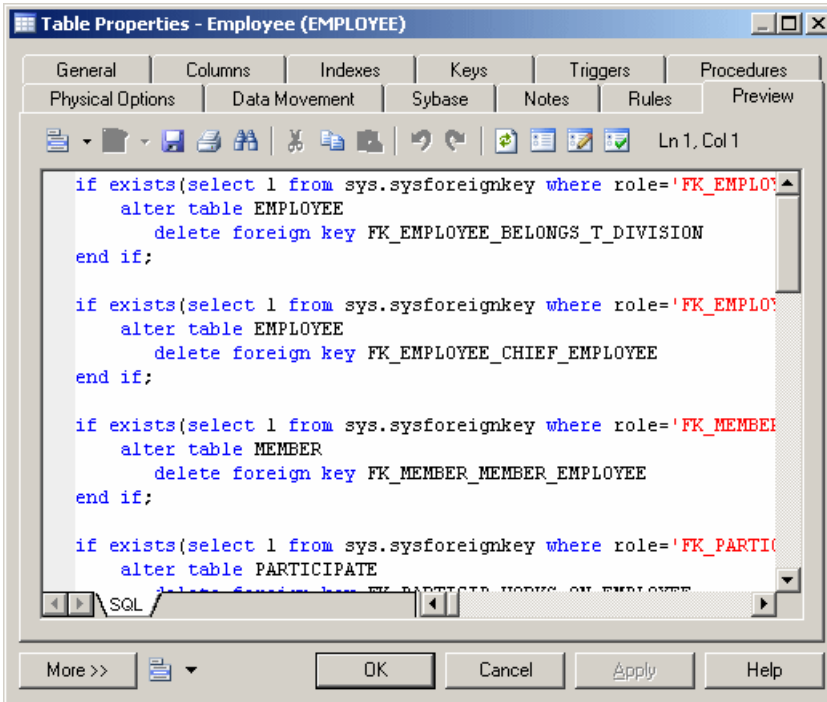
The *SQL/XML* extension file is automatically attached to the XML model to enable you to generate SQL/XML queries from global elements. For more information, see the Working with XML and Databases chapter in the *XML Modeling* guide.

CHAPTER 11 Working with SQL Statements in PowerDesigner

Each object that you create in your model is associated with SQL code that can be used to create or modify it in your database schema, and which is displayed on the Preview tab of its property sheet. Certain objects also require that you write your own custom SQL statements.

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

Text color	Represents
Red	Variable
Green	Comment

You can use the following tools and keyboard shortcuts from the Preview toolbar:

Tool	Description
	<p>Editor Menu [Shift+F11] Provides access to various editor features, including the insertion of and navigation through bookmarks in the Preview tab:</p> <ul style="list-style-type: none"> • Toggle Bookmark [Ctrl+F2] Inserts or removes a bookmark (a blue box) at the cursor position. • Next Bookmark [F2] • Previous Bookmark [Shift+F2] <p>Note that bookmarks are not printable and are lost if you refresh the tab, or use the Show Generation Options tool.</p>
	<p>Refresh [F5] - Refreshes the Preview tab.</p> <p>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.</p>
	<p>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 "Specifying Additional Generation Targets" in <i>Chapter 1, Resource Files and the Public Metamodel</i> in <i>Customizing and Extending PowerDesigner</i>.</p>
	<p>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.</p>
	<p>Ignore Generation Options [Ctrl+D] - Ignores changes to the generation options made with the Show Generation Options tool.</p>

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

Selected tool	Effect on generation options	Effect on preview
Ignore generation options	Generation options currently selected are overridden by pre-defined 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

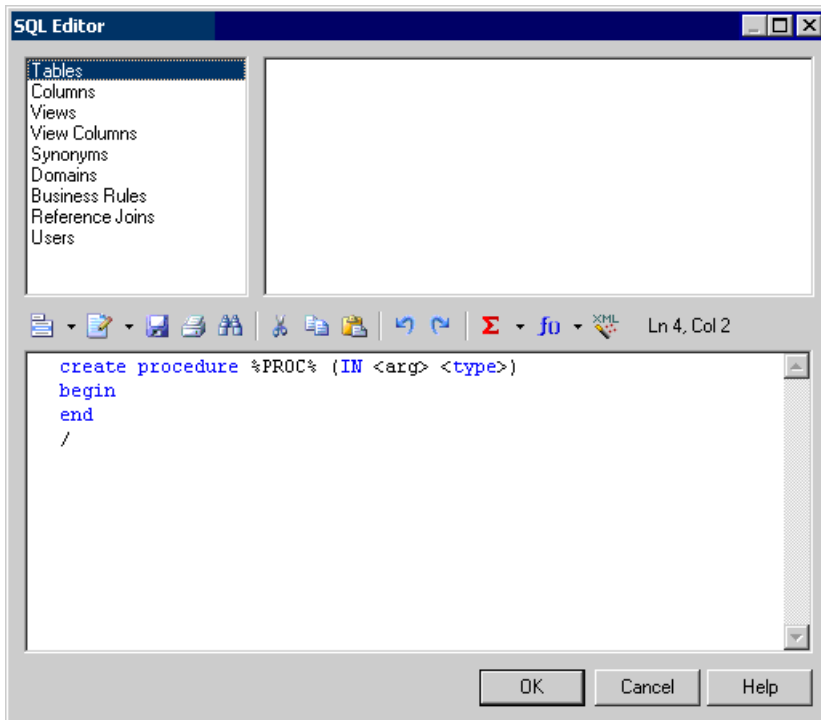
Writing SQL Code in the PowerDesigner SQL Editor

The PowerDesigner SQL Editor provides syntax coloring and tools to help you write SQL code.

You may need to write SQL code in order to:

- Specifying a view query (see *Creating, Editing, and Deleting Queries Associated with Views* on page 183)
- Writing a procedure or trigger (see *Chapter 5, Building Triggers and Procedures* on page 221)
- Define a computed column (see *Creating a Computed Column* on page 113)
- Insert scripts at the beginning and/or end of database or table creation (see *Customizing Scripts* on page 358)

The SQL Editor dialog box is divided into three panes with a toolbar across the middle:



- Upper left pane - lists available object types
- Upper right pane - lists the available objects of the selected type. Double-click an object to insert it into your code
- Toolbar - provides editor tools including lists proposing functions, operators, variables, and macros to insert into your code (see *SQL Editor Tools* on page 401).
- Lower pane - contains the query code










Note: Instead of hard coding the names of tables, columns, and other objects in your SQL statements, you can use the PowerDesigner Generation Template Language (GTL) or the PDM variables and macros to obtain these values from the model.

While you can perform many tasks using the PDM variables and macros, the GTL is more powerful, as it allows you to access any information about any object in the model.

For detailed information about GTL, see *Chapter 4, Customizing Generation with GTL in Customizing and Extending PowerDesigner*.

SQL Editor Tools

The following tools are available in the SQL Editor and in dialogs containing the editor, such as trigger and procedure property sheet **Definition** tabs.

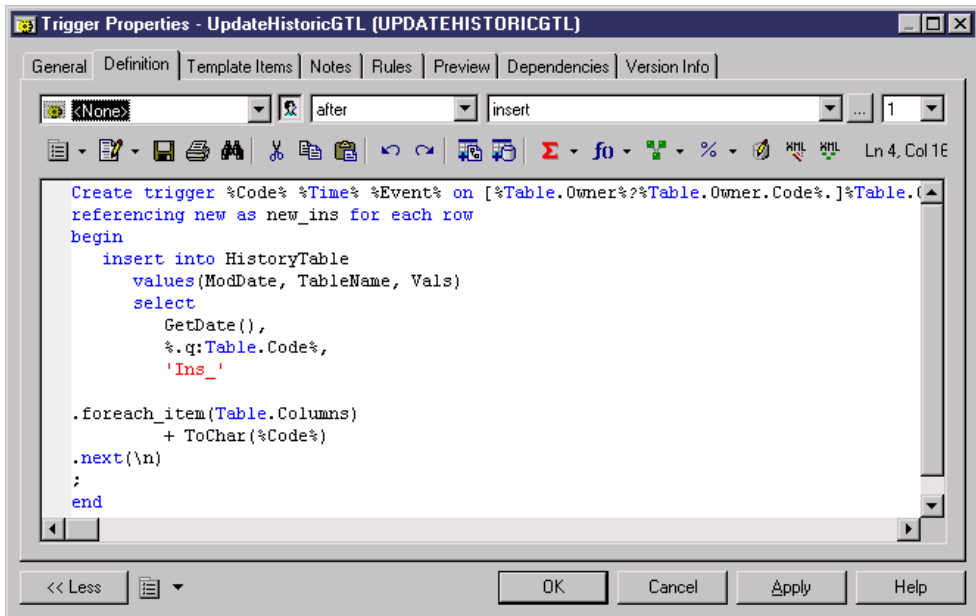
Tool	Description
	Add Trigger Item From Model - [triggers and trigger templates only] Opens a dialog box to select a trigger template item defined in the model for insertion in the trigger definition.
	Add Trigger Item From DBMS - [triggers and trigger templates only] Opens a dialog box to select a trigger template item defined in the DBMS definition file for insertion in the trigger definition.
	Operators - Lists logical operators for insertion in the trigger definition.
	Functions - Lists group, number, string, date, conversion and other functions for insertion in the trigger definition.
	Macros - Lists macros for insertion in the trigger definition (see <i>Chapter 11, Working with SQL Statements in PowerDesigner</i> on page 397).
	Variables - Lists variables for use with operators and functions for insertion in the trigger definition (see <i>PDM Variables</i> on page 404). You can also use formatting variables to force values to lower-case or upper-case or to truncate the length of values characters, as follows (see <i>Formatting Variables</i> on page 416).
	Edit with SQL Editor - [object property sheet tabs only] Opens the standalone SQL Editor dialog which provides object types and available objects for insertion in the trigger definition.
	SQL/XML Wizard - Opens the SQL/XML Wizard to build a SQL/XML query from a table or a view and insert it in the trigger definition (see <i>Using the SQL/XML Wizard</i> on page 262).
	Insert SQL/XML Macro - Opens a dialog box to select a global element in an XML model. The XML model must be open in the workspace and have the SQL/XML extension file attached. Inserts a SQL/XML macro referencing the selected element in the trigger definition.

Writing SQL using GTL

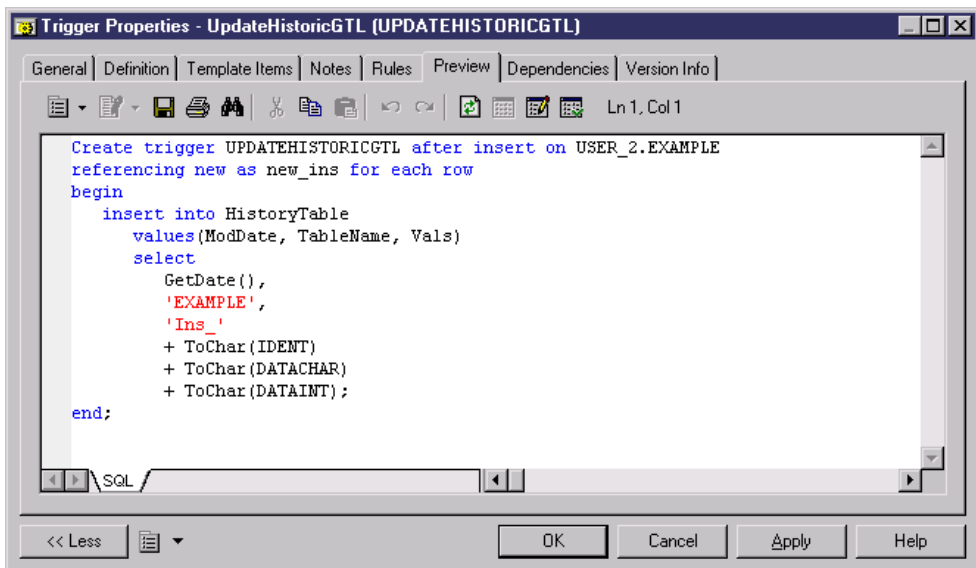
You can use the PowerDesigner Generation Template Language (GTL) to write SQL statements.

In the following example, a trigger written using GTL is attached to the Example table, and writes the contents of any insertion to HistoryTable.

See *Writing SQL using PDM Variables and Macros* on page 403 for an example of the same trigger written using the PowerDesigner macros and variables:



The actual trigger code can be viewed on the Preview tab:



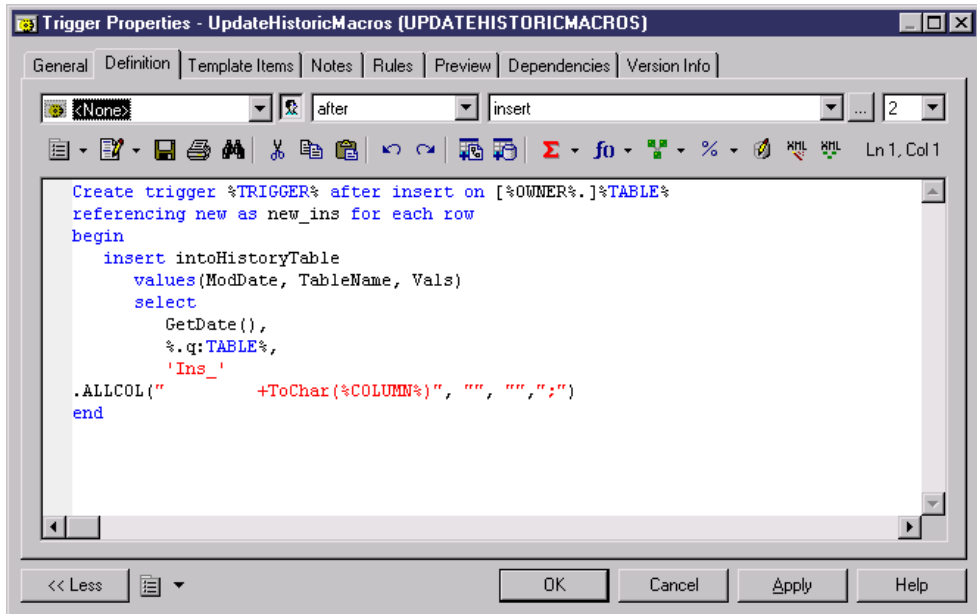
For more information, see the Customizing Generation with GTL chapter of the *Customizing and Extending PowerDesigner* manual.

Writing SQL using PDM Variables and Macros

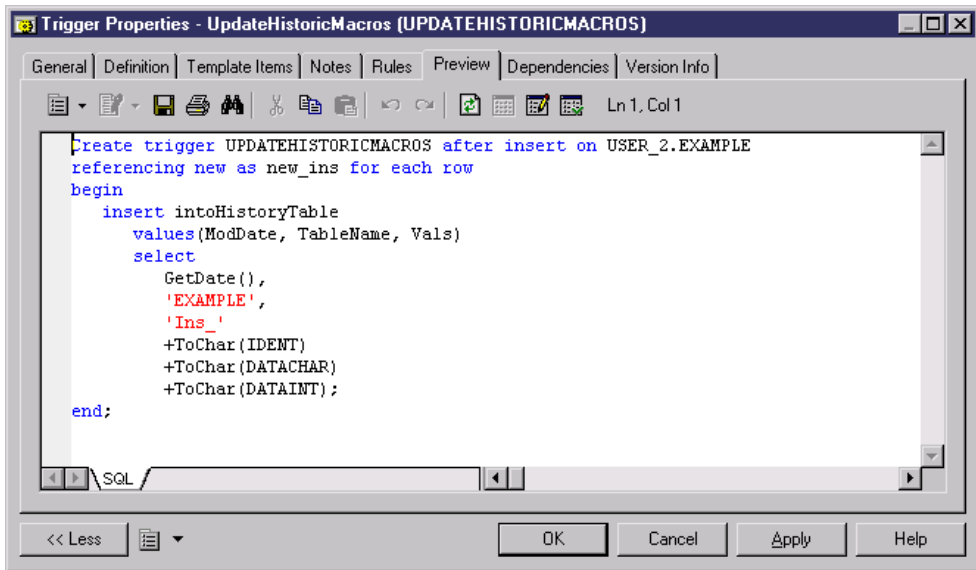
You can use PDM variables and macros to write SQL statements.

In the following example, a trigger written using the PDM variables and macros is attached to the Example table, and writes the contents of any insertion to HistoryTable.

See *Writing SQL using GTL* on page 401 for an example of the same trigger written using the PowerDesigner GTL:



The actual trigger code can be viewed on the Preview tab:



For lists of the available variables and macros, see *PDM Macros* on page 417, *PDM Variables* on page 404, and *PowerDesigner Formatting Variables* on page 416.

PDM Variables

Many objects have both a code variable and a generated code variable.

The code variable is the attribute code that is defined in the object property sheet.

The generated code variable is computed from the attribute code according to available generation options. The values for the code and generated code variables can be different in the following situations:

- When the Code is a reserved word or it contains invalid characters, the generated code is quoted
- When the code is longer than authorized by the database, the generated code is truncated

Note:

To access variables of sub-objects (columns in a table for example) you have to use loop macros or GTL macros to browse the list of sub-objects. For example, in a trigger the following macro loops on table columns and for each column with the CannotModify attribute outputs the code of the column followed by "cannot be modified":

```
.foreach_item(Table.Columns)
.if(%CannotModify%)
-- "%Code%" cannot be modified
.endif
.next
```

For more information, see *PDM Macros* on page 417.

Common Variables for All Named Objects

The following variables can be used for all named object definitions:

Variable name	Comment
@OBJTNAME	Object name
@OBJTCODE	Object code
@OBJTLABL	Object comment
@OBJTDESC	Object description

Common Variables for Objects

The following variables can be used in all object definitions:

Variable name	Comment
COMMENT	Object comment. If no comment is defined the object name is used
OWNER	Generated code of the object owner, or the object parent
DBPREFIX	Database prefix of object (name of database + '.' if database defined)
QUALIFIER	Whole object qualifier (database prefix + owner prefix)
OPTIONS	SQL text defining physical options for object
CONSTNAME	Object constraint name
CONSTRAINT	Object constraint SQL body. For example(A <= 0) AND (A >= 10)
RULES	Concatenation of server expression for business rule associated with object

Table Variables

The following variables can be used in a table definition:

Variable name	Comment
TABLE	Generated code of the table
TNAME	Table name
TCODE	Table code
TLABL	Table comment
PKEYCOLUMNS	List of primary key columns. For example: A, B
TABLDEFN	Complete body of the table definition. It contains definition of columns, checks and keys

Variable name	Comment
CLASS	Abstract data type name
CLUSTERCOLUMNS	List of columns used for the cluster

Variables for Domain and Column Checks

The following variables can be used in domain check parameter and column check parameter definitions:

Variable name	Comment
UNIT	Standard check unit attribute
FORMAT	Standard check format attribute
DATATYPE	Data type. For example int, char(10) or numeric(8, 2)
DTPCODE	Data type code. For example int, char or numeric
LENGTH	Data type length. For example 0, 10 or 8
PREC	Data type precision. For example 0, 0 or 2
ISRONLY	TRUE if the read-only attribute of standard check is selected
DEFAULT	Default value
MINVAL	Minimum value
MAXVAL	Maximum value
VALUES	List of values. For example(0, 1, 2, 3, 4, 5)
LISTVAL	SQL constraint associated with a list of values. For example C1 in (0, 1, 2, 3, 4, 5)
MINMAX	SQL constraint associated with minimum and maximum values. For example(C1 <= 0) AND (C1 >= 5)
ISMAND	TRUE if the domain or column is mandatory
MAND	Contains the keywords "null" or "not null" depending on if the attribute is mandatory or not mandatory
NULL	Contains keyword "null" if the domain or column is not mandatory
NOTNULL	Contains Keyword "not null" if the domain or column is mandatory
IDENTITY	Keyword "identity" if the domain or column is identity (Sybase specific)
WITHDEFAULT	Keyword "with default" if the domain or column is with default

Variable name	Comment
ISUPPERVAL	TRUE if the upper-case attribute of standard check is selected
ISLOWerval	TRUE if the lower-case attribute of standard check is selected

Column Variables

The following variables can be used in a column definition:

Variable name	Comment
COLUMN	Generated code of the column
COLNNO	Position of Column in List of columns of Table
COLNNAME	Column name
COLNCODE	Column code
PRIMARY	Contains Keyword "primary" if the column is a primary key column
ISPKEY	TRUE if the column is part of a primary key
FOREIGN	TRUE if the column is part of a foreign key
COMPUTE	Compute constraint text

Abstract Data Type Variables

The following variables can be used in an abstract data type definition:

Variable name	Comment
ADT	Generated code of the abstract data type
TYPE	Type of Abstract data type. For example "array", or "list"
SIZE	Abstract data type size
ISARRAY	TRUE if the abstract data type is of type array
ISLIST	TRUE if the abstract data type is of type list
ISSTRUCT	TRUE if the abstract data type is of type structure
ISOBJECT	TRUE if the abstract data type is of type object
ISJAVA	TRUE if the abstract data type is of type JAVA class
ADTDEF	Contains definition of the abstract data type

Abstract Data Type Attribute Variables

The following variable can be used in an abstract data type attribute definition:

Variable name	Comment
ADTATTR	Generated code of an abstract data type attribute

Domain Variables

The following variable can be used in a domain definition:

Variable name	Comment
DOMAIN	Generated code of a domain (also available for columns)

Business Rule Variables

The following variables can be used in a business rule definition:

Variable name	Comment
RULE	Generated code of a business rule
RULENAME	Rule name
RULECODE	Rule code
RULECEXP	Rule client expression
RULESEXP	Rule server expression

Index Variables

The following variables can be used in an index definition:

Variable name	Comment
TABLE	Generated code of the parent of an index, can be a table or a query table (view)
INDEX	Generated code of the index
INDEXNAME	Index name
INDEXCODE	Index code
UNIQUE	Contains keyword "unique" when an index is unique
INDEXTYPE	Contains the index type (DBMS dependant)
CIDXLIST	List of index columns. For example A asc, B desc, C asc
INDEXKEY	Contains keywords "primary", "unique" or "foreign" depending on an index origin
CLUSTER	Contains keyword "cluster" when an index is a clustered index

Variable name	Comment
INDXDEFN	Used for defining an index within a table definition

Index Column Variables

The following variables can be used in index column definitions:

Variable name	Comment
ASC	Contains keywords "ASC" or "DESC" depending on the sort order
ISASC	TRUE if the index column sort is ascending

Reference Variables

The following variables can be used in a reference definition:

Variable name	Comment
REFR	Generated code of the reference
PARENT	Generated code of the parent table
PNAME	Parent table name
PCODE	Parent table name
PQUALIFIER	Parent table qualifier. See also QUALIFIER
CHILD	Generated code of a child table
CNAME	Child table name
CCODE	Child table code
CQUALIFIER	Child table qualifier. See also QUALIFIER
REFRNAME	Reference name
REFRCODE	Reference code
FKCONSTRAINT	Foreign key (reference) constraint name
PKCONSTRAINT	Constraint name of the primary key used to reference the object
CKEYCOLUMNS	List of parent key columns. For example: C1, C2, C3
FKEYCOLUMNS	List of child foreign key columns. For example: C1, C2, C3

Variable name	Comment
UPDCONST	Update declarative constraint. This can be any of the following keywords Restrict Cascade Set null Set default
DELCONST	Delete declarative constraint. This can be any of the following keywords Restrict Cascade Set null Set default
MINCARD	Minimum cardinality
MAXCARD	Maximum cardinality
POWNER	Parent table owner name
COWNER	Child table owner name
CHCKONCMMT	TRUE when check on commit is selected on a reference (ASA 6.0 specific)
JOINS	Reference joins
REFRNO	Reference number in the child table collection of references

Reference Column Variables

The following variables can be used in reference column definitions:

Variable name	Comment
CKEYCOLUMN	Generated code of the parent table column (primary key)
FKEYCOLUMN	Generated code of a child table column (foreign key)
PK	Generated code of a primary key column
PKNAME	Primary key column name
FK	Generated code of a foreign key column
FKNAME	Foreign key column name
AK	Alternate key column code (same as PK)

Variable name	Comment
AKNAME	Alternate key column name (same as PKNAME)
COLTYPE	Primary key column data type
DEFAULT	Foreign key column default value
HOSTCOLTYPE	Primary key column data type used in procedure declaration. For example: without length

Key Variables

The following variables can be used in a key definition:

Variable name	Comment
COLUMNS COLN-LIST	List of key columns. For example "A, B, C"
ISPKEY	TRUE when a key is the primary key for the table
PKEY	Primary key constraint name
AKEY	Alternate key constraint name
KEY	Key constraint name
ISMULTICOLN	True if the key has more than one column
CLUSTER	Cluster keyword

Variables for Views

The following variables can be used in a view definition:

Variable name	Comment
VIEW	Generated code of the view
VIEWNAME	View name
VIEWCODE	View code
VIEWCOLN	List of columns of a view. For example: "A, B, C"
SQL	SQL text of a view. For example Select * from T1
VIEWCHECK	Contains the keyword "with check option" if this option is selected in the view property sheet
SCRIPT	Complete view creation order. For example create view V1 as select * from T1

Trigger Variables

The variables listed below can be used in a trigger definition. You can also use owning table variables in a trigger definition.

Variable name	Comment
ORDER	Order number of Trigger (where the current DBMS supports more than one trigger of one type)
TRIGGER	Generated code of the trigger
REFNO	Reference order number in the list of references for the table
ERRNO	Error number for standard error
ERRMSG	Error message for standard error
MSGTAB	Name of a table containing user-defined error messages
MSGNO	Name of a column containing error numbers in a user-defined error table
MSGTXT	Name of a column containing error messages in a user-defined error table
SCRIPT	SQL script of trigger or procedure.
TRGBODY	Trigger body (only for Oracle live database reverse engineering)
TRGDESC	Trigger description (only for Oracle live database reverse engineering)
TRGDEFN	Trigger definition

Database, Trigger, and Procedure Generation Variables

The following variables can be used for database, procedure, and trigger generation:

Variable name	Comment
DATE	Generation date & time
USER	Login name of the user executing the generation
PATHSCRIPT	Path where the file script will be generated
NAMESCRIPT	Name of the file script where SQL orders will be written
STARTCMD	Description explaining how to execute a generated script
ISUPPER	TRUE if upper case generation option is set
ISLOWER	TRUE if lower case generation option is set
DBMSNAME	Name of the DBMS associated with the generated model

Variable name	Comment
DATABASE	Code of the database associated with the generated model

Reverse Engineering Variables

The following variables can be used when reverse engineering a database into a PDM:

Variable name	Comment
R	Set to TRUE during reverse engineering
S	Allows to skip a word. The string is parsed for reverse engineering but is not generated
D	Allows to skip a numeric value. The numeric value is parsed for reverse engineering but is not generated
A	Allows to skip all text. The text is parsed for reverse engineering but is not generated
ISODBCUSER	True if the current user is the connected user
CATALOG	Catalog name that will be used in live database reverse engineering queries
SCHEMA	Schema that will be used in live database reverse engineering queries
SIZE	Data type size of a column or a domain. Used for live database reverse engineering, when a data type length is not defined in the system tables
VALUE	One value from the list of values in a column or domain
TRGTYPE	Variable used in the Create order of a trigger. Trigger type uses keywords for each trigger type, for example "BeforeInsert", or "AfterUpdate"
TRGEVENT	Variable used in the Create order of a trigger. Trigger event uses keywords for each trigger event, for example "Insert", "Update", and "Delete"
TRGTIME	Variable used in the Create order of a trigger. Timing of trigger uses the keywords "Null", "Before", and "After"

Database Synchronization Variables

The following variables can be used for database generation when synchronizing a modified PDM with an existing database:

Variable name	Comment
OLDOWNER	Old owner name of the object. See also OWNER
NEWOWNER	New owner name of the object. See also OWNER

Variable name	Comment
OLDQUALIFIER	Old qualifier of the object. See also QUALIFIER
NEWQUALIFIER	New qualifier for the object. See also QUALIFIER
OLDTABL	Old code of the table
NEWTABL	New code of the table
OLDCOLN	Old code of the column
NEWCOLN	New code of the column

Database Security Variables

The following database security variables are available:

Variable name	Comment
PRIVLIST	List of privileges for a grant/revoke order
PERMLIST	List of permissions for a grant/revoke order
GRANTEE	Name of the user, group, or role for a grant/revoke order
ID	Name of the user
GROUP	Name of the group
ROLE	Name of the role
OBJECT	Database objects (table, view, column, and so on)
PERMISSION	SQL grant/revoke order for a database object
PRIVILEGE	SQL grant/revoke order for an ID (user, group, or role)
GRANTOPTION	Option for grant: with grant option / with admin option
REVOKEOPTION	Option for revoke: with cascade

Metadata Variables

The following metadata variables are available:

Variable name	Comment
@CLSSNAME	Localized name for an object class. For example: Table, View, Column, Index
@CLSSCODE	Object class code. For example: TABL, VIEW, COLN, INDX

DBMS, Database Options Variables

The following DBMS and database options variables are available:

Variable name	Comment
TABLESPACE	Generated code of a tablespace
STORAGE	Generated code of a storage

Variables for ASE & SQL Server

The following DBMS specific variables are available for Sybase Adaptive Server Anywhere and Microsoft SQL Server:

Variable name	Comment
RULENAME	Name of a business rule object associated with a domain
DEFAULTNAME	Name of a default object associated with a domain
USE_SP_PKEY	Use sp_primary key to create primary keys (SQL Server)
USE_SP_FKEY	Use sp_foreign key to create foreign keys (SQL Server)

Sequence Variable

The following variable can be used in a sequence definition:

Variable name	Comment
SQNC	Name of sequence

Procedure Variables

The following variables can be used in a procedure definition:

Variable name	Comment
PROC	Generated code of a procedure (also available for trigger when a trigger is implemented with a procedure)
FUNC	Generated code of a procedure where the procedure is a function (with a return value)

Join Index Variables (IQ)

The following variables can be used in a join index definition:

Variable name	Comment
JIDX	Generated code for join index

Variable name	Comment
JIDXDEFN	Complete body of join index definition
REFRLIST	List of references (for live database connection)
RFJNLIST	List of reference joins (for live database connection)

Formatting Variables

You can control the case, length, quoting, and other aspects of the format of values resolved from variables

The variable formatting options are the following:

Option	Description
.L	Lower-case characters
.U	Upper-case characters
.F	Combined with L or U, applies conversion to first character
.q	Enclose the variable between single quotes if it is not already.
.Q	Enclose the variable between double quotes if it is not already.
.T	Removes blank spaces
.H	Converts number to hexadecimal
.c	Upper-case first letter and lower-case next letters
.n	Maximum length where n is the number of characters
nJ	Justifies to fixed length where n is the number of characters
-nJ	Right justify variable text to fixed length where n is the number of characters
width	Copies the specified minimum number of characters to the output buffer
[-]precision	Copies the specified maximum number of characters to the output buffer
M	Extracts a portion of the variable name, this option uses the width and precision parameters to identify the portion to extract

You embed formatting options in variable syntax as follows:

```
%[[?][-][width][.[-]precision][c][H][F][U|L][T][M][q][Q]:]<varname>
%
```

For example:

```
%.L:TABLE%
```

The table below shows formatted variables and their results in a script for the table EMPLOYEE.

Template statement with variable	Resulting script statement
create trigger %TABLE%	create trigger EMPLOYEE
create trigger %.L:TABLE%	create trigger employee
create trigger %.U:TABLE%	create trigger EMPLOYEE
create trigger %.4:TABLE%	create trigger EMPL
create trigger %.4L:TABLE%	create trigger empl

PDM Macros

You can use predefined macros in trigger templates, template items, triggers, and procedures. Macros perform specific functions.

AKCOLN

Repeats a statement for each alternate key in a table

Syntax

```
.AKCOLN("statement","prefix","suffix","last_suffix", "condition")
```

Argument	Description
statement	Statement to repeat for each column
prefix	Prefix for each new line
suffix	Suffix for each new line
last suffix	Suffix for the last line
condition	Alternate key code (if condition argument is left empty the macro returns a statement for each alternate key in the table)

Example

In a trigger for the table TITLEAUTHOR, the following macro:

```
message .AKCOLN("%COLUMN% is an alternate key column"," ", " ", " ", " ", "AKEY1")
```

generates the following trigger script:

```
message 'TA_ORDER is an alternate key column',
```

Note: For columns, the macro AKCOLN only accepts the variable %COLUMN%.

ALLCOL

Repeats a statement for each column in a table

Syntax

```
.ALLCOL("statement","prefix","suffix","last_suffix")
```

Argument	Description
statement	Statement to repeat for each column
prefix	Prefix for each new line
suffix	Suffix for each new line
last suffix	Suffix for the last line

Example

In a trigger for the table AUTHOR, the following macro:

```
.ALLCOL("%COLUMN% %COLTYPE%", " ", " ", " ", ";")
```

generates the following trigger script:

```
AU_ID char(12),  
AU_LNAME varchar(40),  
AU_FNAME varchar(40),  
AU_BIOGRAPH long varchar,  
AU_ADVANCE numeric(8,2),  
AU_ADDRESS varchar(80),  
CITY varchar(20),  
STATE char(2),  
POSTALCODE char(5),  
AU_PHONE char(12);
```

DEFINE

Defines a variable and initializes its value

Syntax

```
.DEFINE "variable" "value"
```

Argument	Description
variable	Variable name (without % signs)
value	Variable value (may include another variable surrounded by % signs)

Example

In a trigger for the table AUTHOR, the following macro:


```
.DEFINE "TRIGGER" "T_%"TABLE%"
message 'Error: Trigger(%TRIGGER%) of table %TABLE%'
```

generates the following trigger script:

```
message 'Error: Trigger(T_AUTHOR) of table AUTHOR';
```

DEFINEIF

Defines a variable and initializes its value if the test value is not null

Syntax

```
.DEFINEIF "test_value" "variable" "value"
```

Argument	Description
test_value	Value to test
variable	Variable name (without % signs)
value	Variable value (may include another variable surrounded by % signs)

Example

For example, to define a variable for a default data type:

```
%DEFAULT%
.DEFINEIF "%DEFAULT%" "_DEFLT" "%DEFAULT%"
Add %COLUMN% %DATATYPE% %_DEFLT%
```

ERROR

Handles errors.

Syntax

```
.ERROR (errno, "errmsg")
```

Argument	Description
errno	Error number
errmsg	Error message

Example

```
.ERROR(-20001, "Parent does not exist, cannot insert child")
```

FKCOLN

Repeats a statement for each foreign key column in a table.

Syntax

```
.FKCOLN("statement", "prefix", "suffix", "last_suffix")
```

Argument	Description
statement	Statement to repeat for each column
prefix	Prefix for each new line
suffix	Suffix for each new line
last suffix	Suffix for the last line

Example

In a trigger for the table TITLEAUTHOR, the following macro:

```
message .FKCOLN("%COLUMN% is a foreign key column","","","",";")
```

generates the following trigger script:

```
message 'AU_ID is a foreign key column,
TITLE_ISBN is a foreign key column;'
```

Note: For columns, the macro FKCOLN only accepts the variable %COLUMN%.

FOREACH_CHILD

Repeats a statement for each parent-to-child reference in the current table fulfilling a condition.

Syntax

```
.FOREACH_CHILD ("condition")
```

```
"statement"
```

```
.ENDFOR
```

Argument	Description
condition	Reference condition (see below)
statement	Statement to repeat

Condition	Selects
UPDATE RESTRICT	Restrict on update
UPDATE CASCADE	Cascade on update
UPDATE SETNULL	Set null on update
UPDATE SETDEFAULT	Set default on update
DELETE RESTRICT	Restrict on delete

Condition	Selects
DELETE CASCADE	Cascade on delete
DELETE SETNULL	Set null on delete
DELETE SETDEFAULT	Set default on delete

Example

In a trigger for the table TITLE, the following macro:

```
.FOREACH_CHILD("DELETE RESTRICT")
-- Cannot delete parent "%PARENT%" if children still exist in
"%CHILD%"
.ENDFOR
```

generates the following trigger script:

```
-- Cannot delete parent "TITLE" if children still exist in
"ROYSCHED"
-- Cannot delete parent "TITLE" if children still exist in "SALE"
-- Cannot delete parent "TITLE" if children still exist in
"TITLEAUTHOR"
```

FOREACH_COLUMN

Repeats a statement for each column in the current table fulfilling a condition.

Syntax

```
.FOREACH_COLUMN ("condition")
```

```
"statement"
```

```
.ENDFOR
```

Argument	Description
condition	Column condition (see below)
statement	Statement to repeat

Condition	Selects
empty	All columns
PKCOLN	Primary key columns
FKCOLN	Foreign key columns
AKCOLN	Alternate key columns

Condition	Selects
NMFCOL	Non-modifiable columns (columns that have Cannot Modify selected as a check parameter)
INCOLN	Triggering columns (primary key columns, foreign key columns; and non-modifiable columns)

Example

In a trigger for the table TITLE, the following macro:

```
.FOREACH_COLUMN("NMFCOL")
-- "%COLUMN%" cannot be modified
.ENDFOR
```

generates the following trigger script:

```
-- "TITLE_ISBN" cannot be modified
-- "PUB_ID" cannot be modified
```

FOREACH_PARENT

Repeats a statement for each child-to-parent reference in the current table fulfilling a condition.

Syntax

```
.FOREACH_PARENT ("condition")
"statement"
.ENDFOR
```

Argument	Description
condition	Reference condition (see below)
statement	Statement to repeat

Condition	Selects references defined with ...
empty	All references
FKNULL	Non-mandatory foreign keys
FKNOTNULL	Mandatory foreign keys
FKCANTCHG	Non-modifiable foreign keys

Example

In a trigger for the table SALE, the following macro:

```
.FOREACH_PARENT("FKCANTCHG")
-- Cannot modify parent code of "%PARENT%" in child "%CHILD%"
.ENDFOR
```

generates the following trigger script:

```
-- Cannot modify parent code of "STORE" in child "SALE"
-- Cannot modify parent code of "TITLE" in child "SALE"
```

INCOLN

Repeats a statement for each primary key column, foreign key column, alternate key column, or non-modifiable column in a table.

Syntax

```
.INCOLN("statement", "prefix", "suffix", "last_suffix")
```

Argument	Description
statement	Statement to repeat for each column
prefix	Prefix for each new line
suffix	Suffix for each new line
last suffix	Suffix for the last line

Example

In a trigger for the table TITLE, the following macro:

```
.INCOLN("%COLUMN% %COLTYPE%", "", "", ";")
```

generates the following trigger script:

```
TITLE_ISBN char(12),
PUB_ID char(12);
```

JOIN

Repeats a statement for column couple in a join.

Syntax

```
.JOIN("statement", "prefix", "suffix", "last_suffix")
```

Argument	Description
statement	Statement to repeat for each column
prefix	Prefix for each new line
suffix	Suffix for each new line
last suffix	Suffix for the last line

Example

In a trigger for the table TITLE, the following macro:

```
.FOREACH_PARENT()  
where .JOIN("%PK%=%FK%", " and", "", ";")  
message 'Reference %REFR% links table %PARENT% to %CHILD%'  
.ENDFOR
```

generates the following trigger script:

```
message 'Reference TITLE_PUB links table PUBLISHER to TITLE'
```

Note: For columns, the macro JOIN only accepts the variables %PK%, %AK%, and %FK%.

NMFCOL

Repeats a statement for each non-modifiable column in a table. Non-modifiable columns have Cannot Modify selected as a check parameter.

Syntax

```
.NMFCOL("statement", "prefix", "suffix", "last_suffix")
```

Argument	Description
statement	Statement to repeat for each column
prefix	Prefix for each new line
suffix	Suffix for each new line
last suffix	Suffix for the last line

Example

In a trigger for the table TITLE, the following macro:

```
.NMFCOL("%COLUMN% %COLTYPE%", "", "", "", ";")
```

generates the following trigger script:

```
TITLE_ISBN char(12),  
PUB_ID char(12);
```

PKCOLN

Repeats a statement for each primary key column in a table.

Syntax

```
.PKCOLN("statement", "prefix", "suffix", "last_suffix")
```

Argument	Description
statement	Statement to repeat for each column
prefix	Prefix for each new line
suffix	Suffix for each new line
last suffix	Suffix for the last line

Example

In a trigger for the table TITLEAUTHOR, the following macro:

```
message .PKCOLN("%COLUMN% is a primary key column'", "", "", "", ";")
```

generates the following trigger script:

```
message 'AU_ID is a primary key column',
        'TITLE_ISBN is a primary key column';
```

Note: For columns, the macro PKCOLN only accepts the variable %COLUMN%.

CLIENTEXPRESSION and SERVEREXPRESSION

Uses the client and/or server expression of a business rule in the trigger template, template item, trigger, and procedure script.

Syntax

```
.CLIENTEXPRESSION(code of the business rule)
```

```
.SERVEREXPRESSION(code of the business rule)
```

Example

The business rule ACTIVITY_DATE_CONTROL has the following server expression:

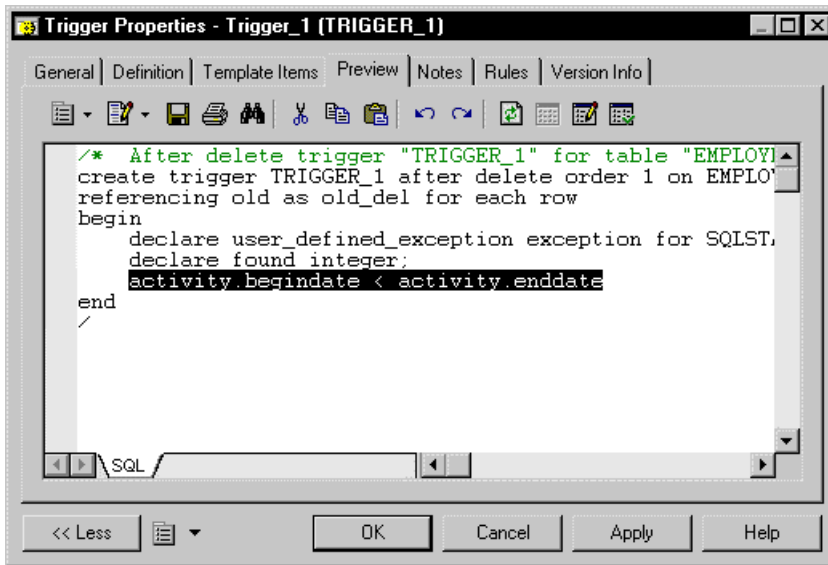
```
activity.begindate < activity.enddate
```

In a trigger based on template AfterDeleteTrigger, you type the following macro in the Definition tab of the trigger:

```
.SERVEREXPRESSION(ACTIVITY_DATE_CONTROL)
```

This generates the following trigger script:

```
activity.begindate < activity.enddate
end
```



SQLXML

Represents a SQL/XML query in the definition of a trigger, a procedure or a function.

Use one of the following tools:

- The *Insert SQL/XML Macro* tool opens a selection dialog box where you choose a global element from an XML model. The XML model must be open in the workspace, mapped to a PDM, and have the SQL/XML extension file attached. Click OK in the dialog box and the SQLXML macro is displayed in the definition code, with the code of the XML model (optional) and the code of the global element.
- The *Macros* tool, where you select *.SQLXML()* in the list. The SQLXML macro is displayed empty in the definition code. You must fill the parentheses with the code of an XML model (optional), followed by *::* and the code of a global element. The XML model, from which you choose a global element, must be open in the workspace, mapped to a PDM, and have the SQL/XML extension file attached.

After generation, the SQLXML macro is replaced by the SQL/XML query of the global element.

Syntax

.SQLXML(code of an XML model::code of a global element)

Note: the code of an XML model is optional.

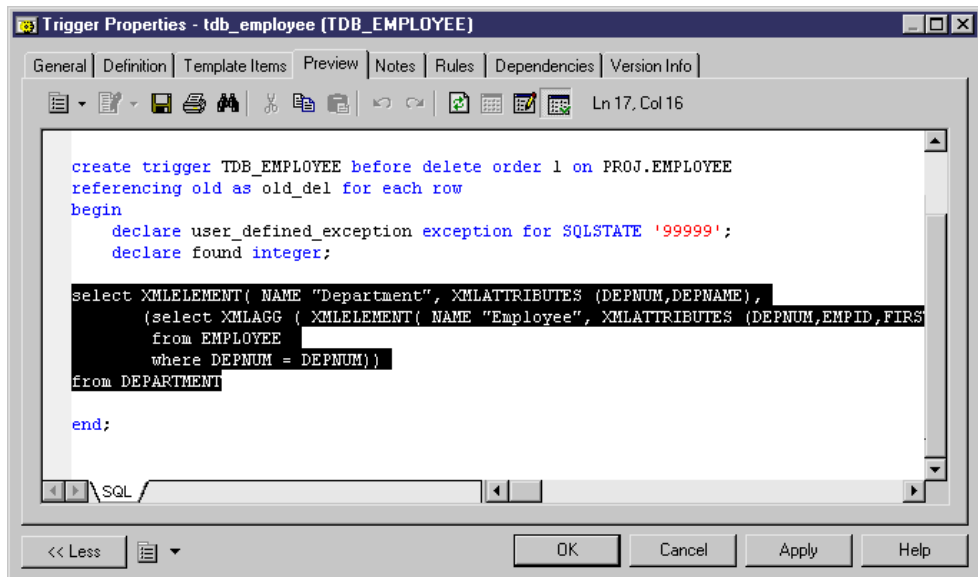
Example

In a trigger for the table EMPLOYEE, the following macro:

.SQLXML(CorporateMembership::DEPARTMENT)

generates the following trigger script:

```
select XMLELEMENT( NAME "Department", XMLATTRIBUTES
(DEPNUM,DEPNAME),
  (select XMLAGG ( XMLELEMENT( NAME "Employee", XMLATTRIBUTES
(DEPNUM,EMPID,FIRSTNAME,LASTNAME)) )
    from EMPLOYEE
    where DEPNUM = DEPNUM))
from DEPARTMENT
```



CHAPTER 12 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.

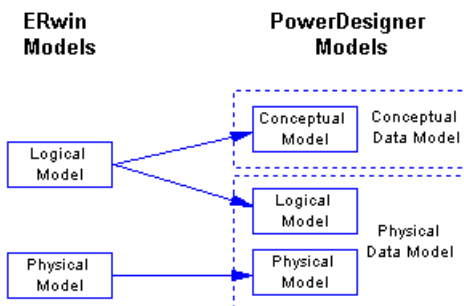
Supported ERwin File Types

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 the "Only save minimum amount of information" check box in the ERwin Save as XML File dialog box.

ERwin and PowerDesigner implement database design in different ways.

- An ERwin logical model can be imported into either:
 - a PowerDesigner conceptual data model (CDM), or
 - a PowerDesigner logical data model
- An ERwin Physical Model is imported into a PowerDesigner physical data model (PDM)



In the PowerDesigner Physical Data Model you can use two design levels: the logical model allows you to design the database structure and perform some database denormalization actions without taking into account any DBMS physical requirement. The physical model

allows you to design the database taking into account the physical requirements of a given DBMS

ERwin Import Limitations

PowerDesigner cannot import certain ERwin objects.

- ERwin triggers and stored procedures (not directly possible, but see the process in *Post-import checklist* on page 434)
- ERwin reports
- ER1 files
- ERwin data sources
- ERwin target clients

While PowerDesigner can import all your object display preferences and will retain color and font information, it does not support multiple colors for columns in a single table. The default column color will be used during the import.

Preparing to Import Your ERwin Models

This section provides a list of suggestions for how to prepare your ERwin files and set up your PowerDesigner environment:

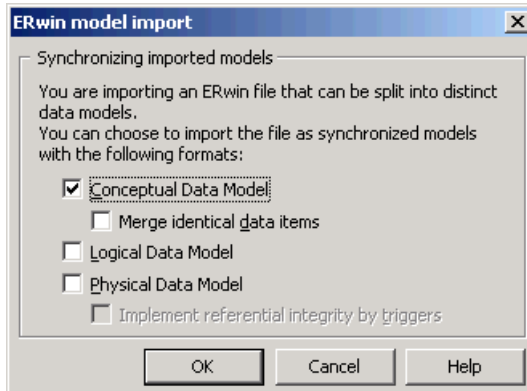
- We recommend that you import from an ERwin 4.x file (.xml) rather than a 3.x file (.erx), due to the greater amount of metadata available in the newer ERwin files.
- Review your ERwin model to see if any model object names are duplicated. It is good practice to avoid using duplicate names, and PowerDesigner will automatically attach a suffix to any duplicate objects that it encounters during the import process.

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.

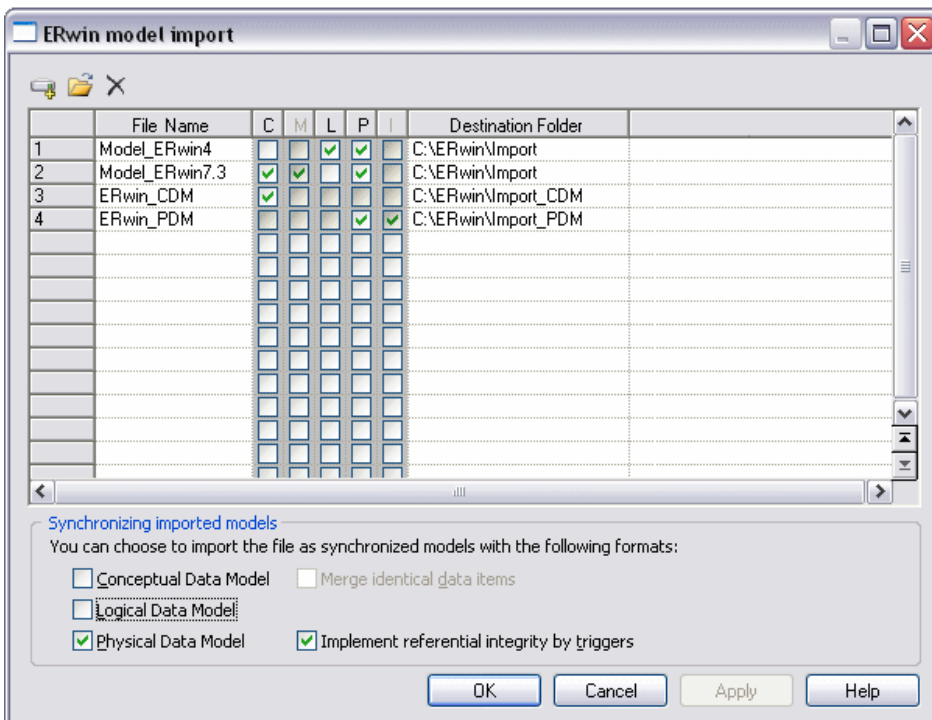
For example, your ERwin logical model may contain multiple entities that contain an attribute "address". By default, PowerDesigner will create a separate data item for each of these entity attributes. However if you select the **Merge identical data items** checkbox, then a single data item will be created, 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.

After Importing

This section describes what you should expect in your newly-imported models.

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 <code>Imported Attributes</code> and embedded in the model. For information about working with extension files, see <i>Chapter 1, Resource Files and the Public Metamodel in Customizing and Extending PowerDesigner</i> .

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

Post-import Checklist

This section lists a series of recommended post-import checks and processes.

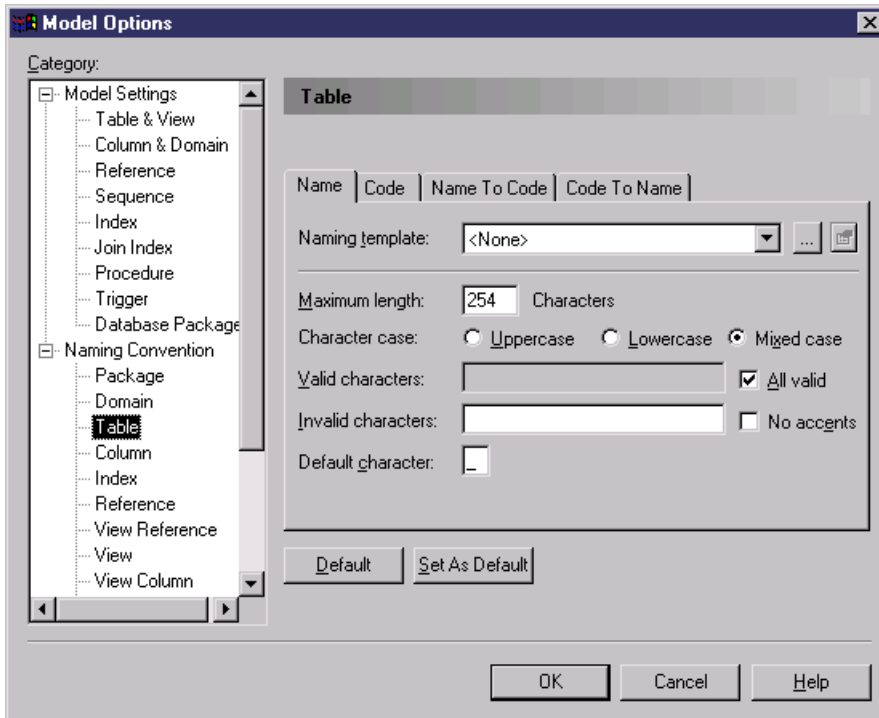
Import triggers: Triggers cannot be directly imported from ERwin. There are, however, two methods for transferring your constraint trigger information to PowerDesigner:

- *Automatically generate triggers:* To do this, select **Tools > Rebuild Objects > Rebuild Triggers**. Creating triggers in this way ensures that they will be synchronized automatically by PowerDesigner, but the actual code may be different from that which you are used to in ERwin.
- *Reverse engineer triggers:* To do this, generate the triggers from ERwin, and then reverse engineer them into PowerDesigner. Creating triggers in this way ensures that they use

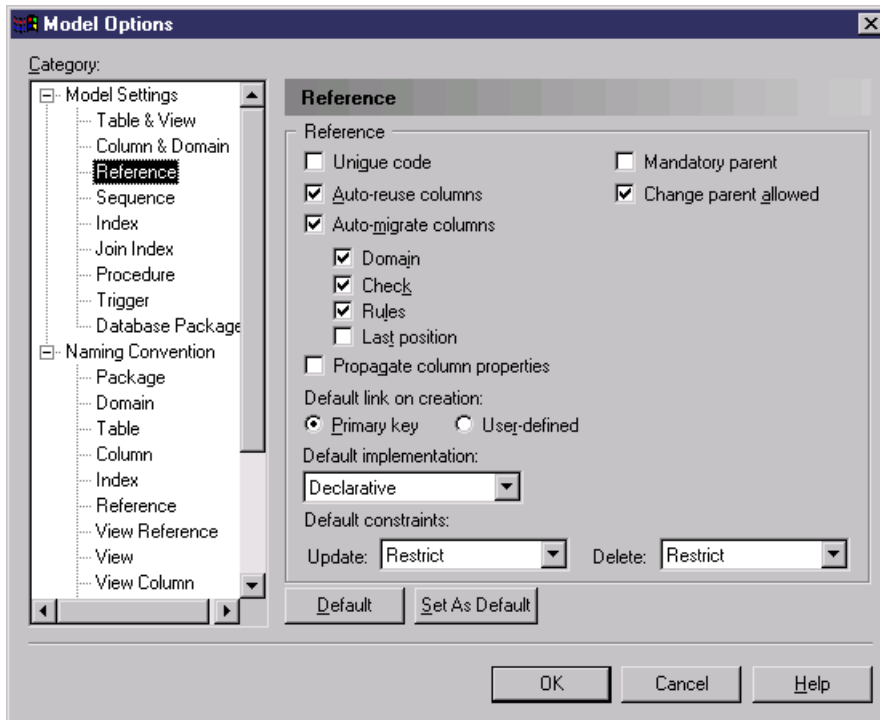
exactly the same code as before, but they will not be automatically synchronized by PowerDesigner.

Import procedures: Procedures cannot be directly imported from ERwin. You can, however transfer them by generating the triggers from ERwin, and then reverse engineering them into PowerDesigner.

Set up your object naming conventions and other model options: To control the naming conventions of model objects in PowerDesigner, select **Tools > Model Options**, and then click on the object entry in the Naming Convention category in the Category pane:



You can control other object creation defaults by clicking on the object entry in the Model Settings category:



Differences to Expect

This section lists certain differences that you may encounter when working with your imported ERwin model in PowerDesigner.

Why do I see errors in Check Model when my ERwin model was clean? PowerDesigner performs stricter checks than ERwin. For example, duplicate objects are not permitted in PowerDesigner, and the existence of orphaned items will generate a warning.

Why do some of my object symbols appear with numeric suffixes? If an object is required to appear more than once in a diagram (for, example, to improve readability), PowerDesigner will create a *graphical synonym* to represent it. Thus, if the table "Purchase" is displayed twice in a diagram, the two symbols will be labeled as "Purchase: 1" and "Purchase: 2".

Getting Started Using PowerDesigner 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 *Chapter 11, Shortcuts and Object Replications* in the *Core Features Guide*).

Packages/Subject Areas

How do I create subject areas? In PowerDesigner, you can create multiple views of your model by adding additional diagrams. You can also divide your model into smaller subdivisions using packages.

- *To add a diagram to your model:* Right-click the diagram background and select **Diagram > New Diagram > [Diagram Type]**.
- *To convert a diagram into a package:* Right-click the diagram background and select **Diagram > Convert to Package**. The Convert Diagram to Package wizard will open, permitting you to name the package and select objects to move into it. The package will appear in the Browser with its own diagram and associated objects. For more information about packages, see "Packages" in *Chapter 3, Models* of the *Core Features Guide*.

Reports

How do I create a report? PowerDesigner provides wizards to create two different types of report:

- *To create a report about a specific type of object:* Select **Report > List Report Wizard** and follow the wizard instructions.

- *To create a report about multiple object types or the whole model:* Select **Report > Report Wizard** and follow the wizard instructions.

For more information about PowerDesigner reports, see *Chapter 6, Reports* in the *Core Features Guide*

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 373.

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 345.

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 368.

Models

How do I compare or merge models? Select **Tools > Compare Models** or **Tools > Merge Model**. For more information, see *Chapter 7, Comparing and Merging Models* in the *Core Features Guide*.

PART III

DBMS Definition Reference

The chapters in this part provide information specific to the DBMSs supported by PowerDesigner.

CHAPTER 13 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: <ul style="list-style-type: none">• 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 Neoview 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 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 14 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: Support for IBM DB2 v5.x 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	<p>[v6.x and higher] Specifies a subtype for a character string column.</p> <p>Scripting name: ExtSubtypeData</p> <p>[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.</p> <p>Scripting name: ExtData</p>
Generated value	<p>[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.</p> <p>Scripting name: ExtGeneratedAs</p>
Implicitly hidden	<p>[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.</p> <p>Scripting name: ImplicitlyHidden</p>
As security label	<p>[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.</p> <p>Scripting name: SecurityLabel</p>

Domains

The following extensions are available on the DB2 tab:

Name	Description
Character Subtype	<p>[v6.x and higher] Specifies a subtype for a character string column.</p> <p>Scripting name: ExtSubtypeData</p>

References

The following extensions are available on the DB2 tab:

Name	Description
Enforced	<p>[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.</p> <p>Scripting name: Enforced</p>

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 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
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

Name	Description
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 package in the Browser, and select **New > Auxiliary Table**.

Auxiliary Table Properties

You can modify an object's properties from its property sheet. To open an auxiliary table property sheet, double-click its Browser entry in the Auxiliary Tables folder.

The following extended attributes are available on the DB2 tab:

Name	Description
Database	Specifies the database in which the LOB data will be stored. Scripting name: Database
Tablespace	Specifies the table space in which the auxiliary table is created. Scripting name: Tablespace
Table	Specifies the table that owns the LOB column. Scripting name: Table
Column	Specifies the name of the LOB column in the auxiliary table. Scripting name: Column

Name	Description
Partition	Specifies the partition of the base table for which the auxiliary table is to store the specified column. Scripting name: Partition

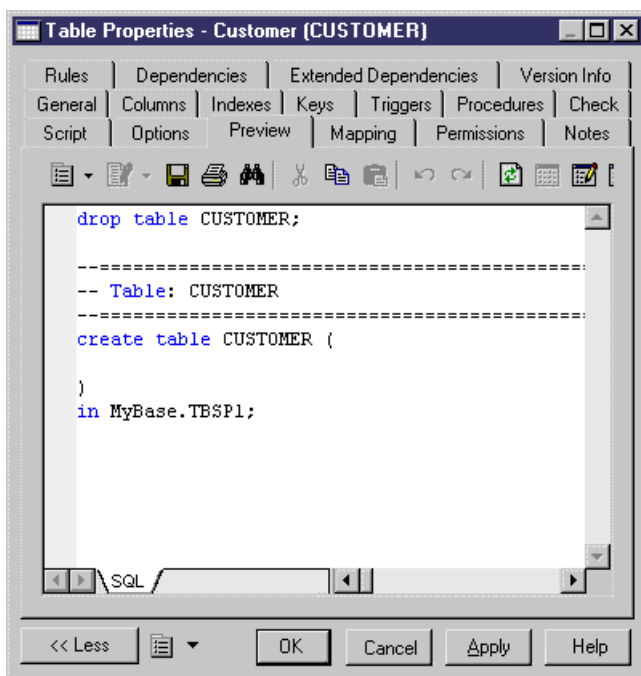
Tablespace Prefix (DB2)

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.



CHAPTER 15 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 DBMSs for IBM DB2 v5.x are deprecated.

For a list of DB2 objects and their equivalents in PowerDesigner, see *Chapter 14, IBM DB2 for z/OS (formerly OS/390)* on page 447.

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
Pctfree	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	<p>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.</p> <p>Scripting name: InlineLength</p>
Without comparison	<p>Indicates that there are no comparison functions supported for instances of the structured type.</p> <p>Scripting name: WithoutComparison</p>
Cast (ref as source) function	<p>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.</p> <p>Scripting name: RefAsSourceCastFunction</p>
Cast (source as ref) function	<p>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).</p> <p>Scripting name: SourceAsRefCastFunction</p>
With function access	<p>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.</p> <p>Scripting name: WithFunctionAccess</p>

Name	Description
Ref using	<p>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.</p> <p>Scripting name: RepType</p>
Length/ precision	<p>Specifies the precision for representation type.</p> <p>Scripting name: RepPrecision</p>

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	<p>Specifies COMPACT options for LOB data type columns.</p> <p>Scripting name: Compact</p>
Logged	<p>Specifies LOGGED options for LOB data type columns.</p> <p>Scripting name: Logged</p>

Abstract Data Type Procedures

The following extensions are available on the DB2 tab (v9.x and higher):

Name	Description
Inherit isolation level	<p>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.</p> <p>Scripting name: IsolationLevel</p>
Method is external	<p>Indicates that the CREATE METHOD statement is being used to register a method, based on code written in an external programming language.</p> <p>Scripting name: ExternalMethod</p>

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 v9.x and higher for Common Server.

A partition group is a logical layer that provides for the grouping of one or more database partitions. A partition can belong to more than one partition group. When a database is created, DB2 creates three default partition groups, which cannot be dropped.

Creating a Database Partition Group

You can create a database partition group in any of the following ways:

- Select **Model > Database Partition Groups** to access the List of Database Partition Groups, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select **New > Database Partition Group**.

Database Partition Group Properties

You can modify an object's properties from its property sheet. To open a database partition group property sheet, double-click its diagram symbol or its Browser entry in the Database Partition Groups folder.

The following extended attributes are available on the DB2 tab:

Property	Description
Database partitions	Specifies the database partitions that are in the partition group. When empty, the group includes all database partitions defined in the database at the time of its creation. Scripting name: DBPartitionNumList

Index Extensions (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
Key generation function	Specifies how the index key is generated using a user-defined table function. Multiple index entries may be generated for a single source key data value. Scripting name: KeyGenerationFunction
Parameter	Specifies parameters for the key generation function. Scripting name: KeyGenerationFunctionParameters
Search methods	Specifies the list of method details of the index search. Each detail consists of a method name, the search arguments, a range producing function, and an optional index filter function. Scripting name: SearchMethods
Source key parameters	Specifies the parameter (and its data type) that is associated with the source key column. Scripting name: SourceKeyParameters
Target key parameters	Specifies the target key parameters that are the output of the key generation function specified on the GENERATE KEY USING clause. Scripting name: TargetKeyParameters

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 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 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 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	<p>Specifies the type of component. You can choose between:</p> <ul style="list-style-type: none"> • 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 <p>Scripting name: Type</p>
Constant list	<p>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.</p> <p>Scripting name: List</p>

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 subobjects 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

CHAPTER 16 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.

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.

The following sections list the extensions provided for MS SQL Server.

Columns

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 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: ExtDefltConstName
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

Name	Description
Identity value not replicated	<p>Indicates that the IDENTITY property should not be enforced when a replication login inserts data into the table.</p> <p>Scripting name: ExtIdtNotForReplication</p>
Not null constraint name	<p>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.</p> <p>Scripting name: ExtNullConstName</p>
Row global unique identifier	<p>[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.</p> <p>Scripting name: ExtRowGuidCol</p>
Collation name	<p>[v2005 and higher] A single string that specifies the collation name for a SQL collation.</p> <p>Scripting name: ExtCollation</p>
XML schema collection	<p>[v2000 and higher] Applies only to the XML data type for associating an XML schema collection with the type.</p> <p>Scripting name: XMLSchemaCollection</p>
Content type	<p>[v2005 and higher] - CONTENT:</p> <p>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.</p> <p>- DOCUMENT:</p> <p>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.</p> <p>Scripting name: ContentType</p>

Cubes

The following extensions are available on the Microsoft tab:

Name	Description
Options	<p>[v2000] You can choose between the following:</p> <ul style="list-style-type: none"> • 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. <p>Scripting name: Options</p>
Storage mode	<p>[v2005 and higher] Specifies the storage mode for the cube.</p> <p>Scripting name: StorageMode</p>
Visible	<p>[v2005 and higher] Determines the visibility of the Cube.</p> <p>Scripting name: Visible</p>

Dimensions

The following extensions are available on the Microsoft tab:

Name	Description
Hidden	<p>[v2000] Indicates whether the dimension is hidden from clients.</p> <p>Scripting name: IsHidden</p>

Name	Description
Options	<p>[v2000] Dimension options to manage member uniqueness and specify their storage. You can choose between:</p> <ul style="list-style-type: none"> • 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. <p>Scripting name: Options</p>
Subtype	<p>[v2000] Indicates the subtype of a dimension. You can choose between:</p> <ul style="list-style-type: none"> • 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. <p>Scripting name: SubType</p>
Template	<p>[v2000] Contains a template string that is used to generate captions for system-generated data members.</p> <p>Scripting name: Template</p>

Name	Description
Time	<p>[v2000] Indicates that a dimension refers to time (year, month, week, day, and so on). You can choose between:</p> <ul style="list-style-type: none"> • TIME: Year, month, week, day, and so on. The only valid levels in a time dimension are those defined in the LevelTypes enumeration. <p>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:</p> <ul style="list-style-type: none"> • 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. <p>Scripting name: TimeDef</p>
Type	<p>[v2005 and higher] Provides information about the contents of the dimension.</p> <p>Scripting name: Type</p>
Storage mode	<p>[v2005 and higher] Determines the storage mode for the parent element.</p> <p>Scripting name: StorageMode</p>
AttributeAllMemberName	<p>[v2005 and higher] Contains the caption, in the default language, for the All member of the dimension.</p> <p>Scripting name: AttributeAllMemberName</p>
WriteEnabled	<p>[v2005 and higher] Indicates whether dimension writebacks are available (subject to security permissions).</p> <p>Scripting name: WriteEnabled</p>

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: <ul style="list-style-type: none">• 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	<p>[v2000] Options about member uniqueness, ordering and data source. You can choose between:</p> <ul style="list-style-type: none"> • 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>. <p>Scripting name: Options</p>
Root values	<p>[v2000] Determines how the root member or members of a parent-child hierarchy are identified. You can choose between:</p> <ul style="list-style-type: none"> • 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. <p>Scripting name: RootValues</p>

Name	Description
Type	<p>[v2000 and higher] Identifies the specific type of level. You can choose between:</p> <ul style="list-style-type: none"> • 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). <p>Scripting name: Type</p>
MembersWithData	<p>[v2005 and higher] Determines whether to display data members for non-leaf members in the parent attribute.</p> <p>Scripting name: MembersWithData</p>
OrderBy	<p>[v2005 and higher] Describes how to order the members contained in the attribute.</p> <p>Scripting name: OrderBy</p>
MemberNamesUnique	<p>[v2005 and higher] Determines whether member names under the parent element must be unique.</p> <p>Scripting name: MemberNamesUnique</p>
IsAggregatable	<p>[v2005 and higher] Specifies whether the values of the DimensionAttribute element can be aggregated.</p> <p>Scripting name: IsAggregatable</p>
AttributeHierarchyEnabled	<p>[v2005 and higher] Determines whether an attribute hierarchy is enabled for the attribute.</p> <p>Scripting name: AttributeHierarchyEnabled</p>
AttributeHierarchyVisible	<p>[v2005 and higher] Determines whether the attribute hierarchy is visible to client applications.</p> <p>Scripting name: AttributeHierarchyVisible</p>

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. <ul style="list-style-type: none"> • ENABLE_BROKER: Specifies that Service Broker is enabled for the specified database. • NEW_BROKER: Creates a new <code>service_broker_guid</code> value in both <code>sys.databases</code> 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 497.

Data Sources

The following extensions are available on the OLE DB tab:

Name	Description
Data provider	Specifies the data provider. You can choose between: <ul style="list-style-type: none"> .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
MemberNamesUnique	[v2005 and higher] Determines whether member names under the parent element must be unique. Scripting name: MemberNamesUnique
AllowDuplicateNames	[v2005 and higher] Determines whether duplicate names are allowed in a Hierarchy element. Scripting name: AllowDuplicateNames

Fact Measures

The following 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 Source-Column (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

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
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
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

Tables

The following extensions are available on the Microsoft tab:

Name	Description
File group	Specifies the name of the filegroup. Scripting name: FileGroup
Partition scheme	[v2005 and higher] Specifies the name of the partition scheme. Scripting name: PartitionScheme
Column	[v2005 and higher] Specifies the name of the partitioned column. Scripting name: PartitionSchemeColumn
Text/Image on	Specifies the name of the filegroup where text and image are stored. Scripting name: TextImageOn
Do not validate check constraints during replication	Specifies that "NOT FOR REPLICATION" keywords are used to prevent the TABLE CHECK constraint from being enforced during the distribution process used by replication. Scripting name: ExtCktNotForReplication

Users

The following extensions are available on the General tab (v2005 and higher):

Name	Description
Implicit schema	Specifies that the stored procedure <code>sp_grantdbaccess</code> 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 package in the Browser, and select **New > Partition Function**.

Partition Function Properties

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

The following extended attributes are available on the Microsoft tab:

Name	Description
Input Parameter Type	Specifies the data type of the column used for partitioning. All data types are valid, except text, ntext, image, xml, timestamp, varchar(max), nvarchar(max), varbinary(max), alias data types, or CLR user-defined data types. Scripting name: InputParameterType

Name	Description
Interval Side	<p>Specifies to which side of each boundary value interval the boundary_value [,...n] belongs. You can choose between:</p> <ul style="list-style-type: none"> • left [default] • right <p>Interval values are sorted by the Database Engine in ascending order from left to right.</p> <p>Scripting name: IntervalSide</p>
Boundary Values	<p>Specifies the boundary values for each partition of a partitioned table or index. All values must be separated by commas.</p> <p>Scripting name: BoundaryValues</p>

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 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	<p>Specifies the partition function using the scheme. Partitions created by the partition function are mapped to the filegroups specified in the partition scheme.</p> <p>Scripting name: PartitionFunction</p>

Name	Description
All Partitions	<p>Specifies that all partitions map to the filegroup specified by the File Groups property.</p> <p>Scripting name: AllPartitions</p>
File Groups	<p>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.</p> <p>Scripting name: Filegroups</p>

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 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: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • 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 package in the Browser, and select **New > Aggregate**.

Aggregate Function Properties

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

The following extended attributes are available on the Microsoft tab:

Name	Description
Owner	Specifies the name of a schema as the owner of the aggregate function. Scripting name: Owner
Assembly	Specifies the assembly to bind with the aggregate function. Scripting name: Assembly
Class name	Specifies the name of the class in the assembly that implements the aggregate function. If the class name is not specified, SQL Server assumes it is the same as the aggregate name. Scripting name: Class
Parameter name	Specifies the name of the input parameter. Scripting name: InputParameterName
Type	Specifies the type of the input parameter. All scalar data types or CLR user-defined types can be used, except text, ntext, and image. Scripting name: InputParameterType
Return type	Specifies the return type of the aggregate function. All scalar data types or CLR user-defined types can be used as return type, except text, ntext, and image. Scripting name: ReturnType

CLR User-Defined Types (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
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 package in the Browser, and select **New > Certificate**.

Certificate Properties

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

The following extended attributes are available on the Microsoft tab:

Name	Description
Authorization	Specifies the name of a user as the owner of the certificate. Scripting name: Authorization
Assembly	Specifies a signed assembly that has already been loaded into the database. Scripting name: Assembly
Assembly File	Specifies the complete path, including file name, to a DER encoded file that contains the certificate. The path name can be a local path or a UNC path to a network location. The file will be accessed in the security context of the SQL Server service account. This account must have the required file system permissions. Scripting name: AssemblyFile
Executable	If the EXECUTABLE option is used, the file is a DLL that has been signed by the certificate. Scripting name: Executable
File	Specifies the complete path, including file name, to the private key. The private key path name can be a local path or a UNC path to a network location. The file will be accessed in the security context of the SQL Server service account. This account must have the necessary file system permissions. Scripting name: PrivateKeyFile
Encryption password (private key)	Specifies the password that will be used to encrypt the private key. Scripting name: PrivateKeyEncryptionPassword
Decryption password	Specifies the password required to decrypt a private key that is retrieved from a file. Scripting name: PrivateKeyDecryptionPassword

Name	Description
Subject	Specifies the value of the subject field in the metadata of the certificate as defined in the X.509 standard. Scripting name: Subject
Encryption password	Use this option only if you want to encrypt the certificate with a password. Scripting name: EncryptionPassword
Star date	Specifies the date on which the certificate becomes valid. If not specified, StartDate will be set equal to the current date. Scripting name: StartDate
Expiry date	Specifies the date on which the certificate expires. If not specified, ExpiryDate will be set to a date one year after StartDate. Scripting name: ExpiryDate
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 package in the Browser, and select **New > Asymmetric Key**.

Asymmetric Key Properties

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

The following extended attributes are available on the Microsoft tab:

Name	Description
Authorization	Specifies the name of a user as the owner of the asymmetric key. Scripting name: Authorization
Assembly	Specifies the name of an assembly from which to load the public key. Scripting name: Assembly
Assembly file	Specifies the path of a file from which to load the key. Scripting name: AssemblyFile
Executable	If the EXECUTABLE option is used, the file attribute specifies an assembly file from which to load the public key, otherwise the file attribute specifies the path of a strong name file from which to load the key pair. Scripting name: Executable
Algorithm	Specifies the algorithm used to encrypt the key. Scripting name: Algorithm
Encryption password	Specifies the password with which to encrypt the private key. If this clause is not present, the private key will be encrypted with the database master key. Scripting name: EncryptionPassword

Symmetric Keys (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 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
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 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
Change tracking	Specifies whether or not SQL Server maintains a list of all changes to the indexed data. You can choose between: <ul style="list-style-type: none"> • manual • auto • off • off, no population Scripting name: ChangeTracking

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

Name	Description
Fill factor	Specifies a percentage that indicates how full the Database Engine should make the leaf level of each index page during index creation or rebuild. Scripting name: FillFactor
Max degree of parallelism	Overrides the max degree of parallelism configuration option for the duration of the index operation. Use MAXDOP to limit the number of processors used in a parallel plan execution. The maximum is 64 processors. Scripting name: MaxDop
Pad index	Specifies index padding. Scripting name: PadIndex
Statistics no recompute	Specifies whether distribution statistics are recomputed. Scripting name: StatisticsNoRecompute
Drop existing	Specifies that the named, preexisting clustered, nonclustered, or XML index is dropped and rebuilt. Scripting name: DropExisting
Sort in temporary database	Specifies whether to store temporary sort results in tempdb. Scripting name: SortInTempDB
Allow row locks	Specifies whether row locks are allowed. Scripting name: AllowRowLocks
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
Content type	Specifies the nature of the content to be stored in the column. You can choose between: <ul style="list-style-type: none">• 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 package in the Browser, and select **New > XML Schema Collection**.

XML Schema Collection Properties

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

The following extended attributes are available on the Microsoft tab:

Name	Description
Owner	Specifies the name of a user, role, or schema as the owner of the schema collection. Scripting name: Owner
XML model	Specifies a PowerDesigner XML model to link to the schema. Scripting name: XMLModel
Content	Specifies the content of the xml schema. By default this field contains the %xmlModelContent% template, which represents the content of the linked XML model. Scripting name: Content

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

Name	Description
Partner/ Witness	<p>Specifies the role that the database will play in the mirroring relationship. You can choose between:</p> <ul style="list-style-type: none"> • 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. <p>Scripting names: Partner, Witness</p>
Options	<p>Specifies mirroring options for the database. You can choose between:</p> <ul style="list-style-type: none"> • <None> • server • off • failover • force_service_allow_data_loss • resume • safety full • safety off • suspend • timeout <p>Scripting name: MirrorOptions</p>
Server	<p>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.</p> <p>For witness mirroring, specifies an instance of the Database Engine to act as the witness server for a database mirroring session.</p> <p>Scripting name: MirrorServer</p>
Time-out	<p>[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.</p> <p>Scripting name: TimeOut</p>

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 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: <ul style="list-style-type: none"> • started • stopped • disabled Scripting name: State
Protocol: Name	Specifies the transport protocol to be used by the endpoint. You can choose between: <ul style="list-style-type: none"> • http • tcp Scripting name: Protocol
Protocol: Argument	Allows you to enter arguments for the chosen protocol. Scripting name: ProtocolArgument

Name	Description
Language: Name	<p>Specifies the type of content to be sent. You can choose between:</p> <ul style="list-style-type: none"> • soap • tsql • service_broker • database_mirroring <p>Scripting name: Language</p>
Language: Argument	<p>Allows you to enter arguments for the chosen language.</p> <p>Scripting name: LanguageArgument</p>

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 package in the Browser, and select **New > Message Type**.

Message Type Properties

You can modify an object's properties from its property sheet. To open a message type property sheet, double-click its Browser entry.

The following extended attributes are available on the Microsoft tab:

Name	Description
Authorization	Specifies a database user or role as the owner of the message type. If the current user is dbo or sa, this may be the name of any valid user or role. Otherwise, it must be the name of the current user, a user that the current user has IMPERSONATE permission for, or a role to which the current user belongs. By default, the message type belongs to the current user. Scripting name: Owner
Validation	Specifies how the Service Broker validates the message body for messages of this type. You can choose between: <ul style="list-style-type: none"> • 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 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 501).
- Select **Model > Message Contracts** to access the List of Message Contracts, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select **New > Message Contract**.

Message Contract Properties

You can modify an object's properties from its property sheet. To open a message contract property sheet, double-click its Browser entry.

The following extended attributes are available on the Microsoft tab:

Name	Description
Sent by	Specifies which endpoint can send a message of the indicated message type. Contracts document the messages that services can use to have specific conversations. Each conversation has two endpoints: the initiator endpoint, the service that started the conversation, and the target endpoint, the service that the initiator is contacting. Scripting name: Sender
Message type	Specifies the message type of the contract. Scripting name: MessageType

Queues (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 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	<p>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.</p> <p>The default is to not retain messages in the queue in this way.</p> <p>Scripting name: Retention</p>
Activation	<p>Specifies that a stored procedure is required to activate message processing for the queue.</p> <p>Scripting name: Activation</p>
Status (activation)	<p>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.</p> <p>This is the default.</p> <p>Scripting name: ActivationStatus</p>
Procedure	<p>Specifies the name of the stored procedure to activate to process messages in this queue.</p> <p>Scripting name: ActivationProcedureName</p>
MaxQueueReaders	<p>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.</p> <p>Scripting name: ActivationMaxQueueReaders</p>
Execute as	<p>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:</p> <ul style="list-style-type: none"> • 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. <p>Scripting name: ActivationExecuteAs</p>
File group	<p>Specifies the SQL Server filegroup on which to create the queue.</p> <p>Scripting name: FileGroup</p>

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 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	<p>Specifies the scope of the event notification. You can choose between:</p> <ul style="list-style-type: none"> • 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. <p>Scripting name: AppliesOn</p>
Queue	<p>Specifies the queue to which the event notification applies. Available only if Applies on is set to "queue".</p> <p>Scripting name: Queue</p>

Name	Description
With fan in	<p>Instructs SQL Server to send only one message per event to any specified service for all event notifications that:</p> <ul style="list-style-type: none"> • 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. <p>Scripting name: WithFanIn</p>
Events	<p>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.</p> <p>Scripting name: Events</p>
Service	<p>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 <i>Services</i> on page 506.</p> <p>Scripting name: Service</p>
Instance	<p>Specifies a service broker instance against which broker_service is resolved. Use 'current database' to specify the service broker instance in the current database.</p> <p>Scripting name: Instance</p>

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 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 package in the Browser, and select **New > Route**.

Route Properties

You can modify an object's properties from its property sheet. To open a route property sheet, double-click its Browser entry.

The following extended attributes are available on the Microsoft tab:

Name	Description
Owner	Specifies the owner of the route. Scripting name: Owner
Remote service	Specifies the name of the remote service to which the route points. Scripting name: Service
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 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 as extended objects with a stereotype of <<WorkloadGroup>>.

Creating a Workload Group

You can create a workload group binding in any of the following ways:

- Select **Model > Workload groups** to access the List of Workload Groups, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select **New > Workload Group**.

Workload Group Properties

You can modify an object's properties from its property sheet. To open a workload group property sheet, double-click its Browser entry.

The following extended attributes are available on the Microsoft tab:

Name	Description
Importance	Specifies the relative importance of a request in the workload group. Scripting name: Importance

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 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 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	<p>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.</p> <p>Scripting name: SchemaOwner</p>

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 142.

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 517.

Analysis Services provide the following capabilities:

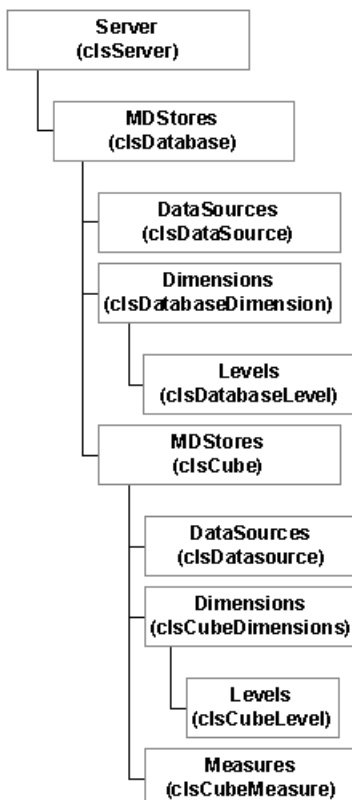
- The *Analysis server* that manages, stores multidimensional information and serves client application requests for OLAP data. The server stores cube metadata (cube definition specifications) in a repository. Completed cubes can be stored in a variety of storage

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:



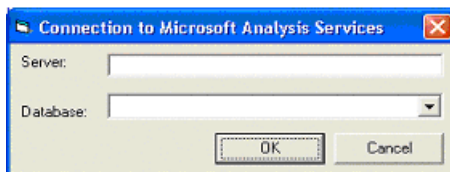
The following table lists the mappings between the objects contained within the DSO and PowerDesigner PDM metamodels:

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.)

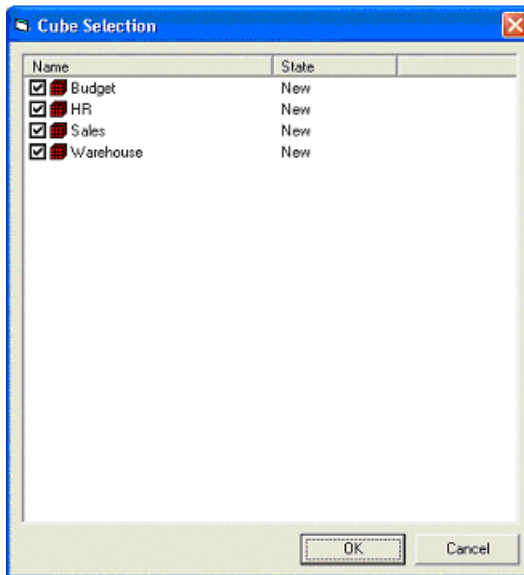
Generating Cubes

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.

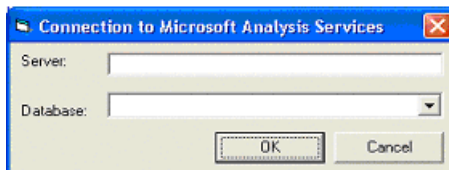
The selected cubes are generated. If a cube already exists in the database, it is dropped before being recreated. If a dimension already exists, the selected cube reuses it. To be fully generated, a cube must have a complete mapping to a table before being generated.

Reverse Engineering Cubes

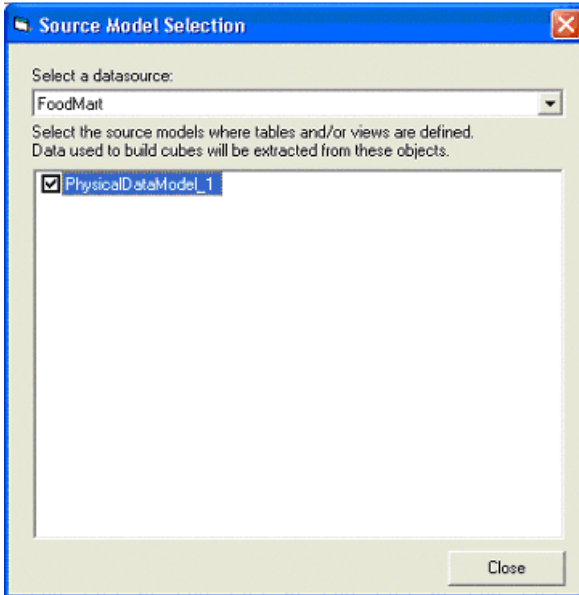
The Microsoft Analysis Services add-in lets you reverse engineer cubes.

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.

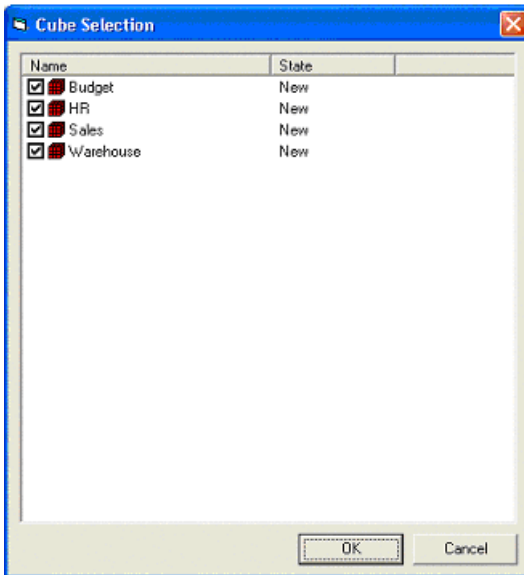
1. Select **Tools > Microsoft Analysis Services > Reverse Engineer Cubes** to open the connection dialog box.



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)

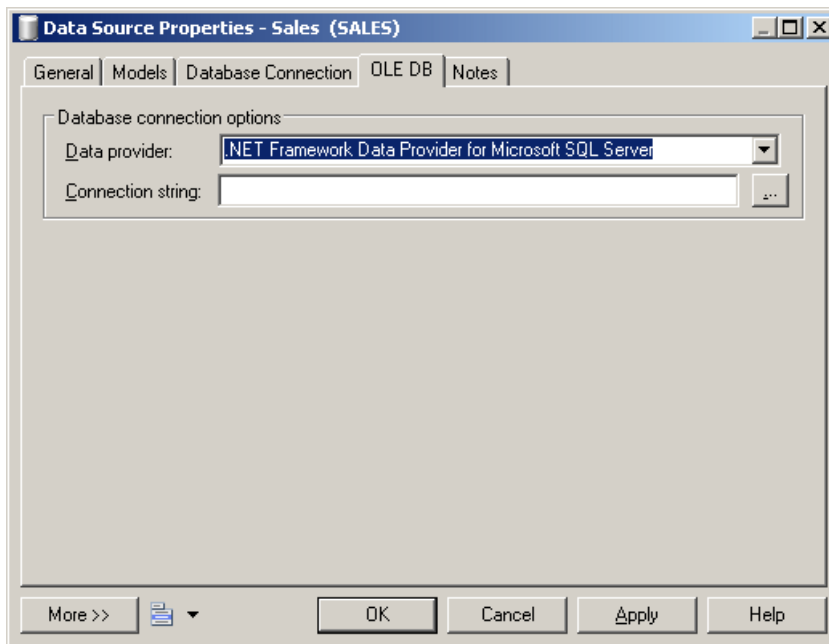
PowerDesigner allows you to retrieve multiple dimension objects in a PDM in order to build cubes, and to create a new multiple-dimension diagram. From this diagram, you can generate cubes to a Microsoft SQL Server 2005 Analysis Server (SSAS). To enable analysis services, select **Tools > General Options**, 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.

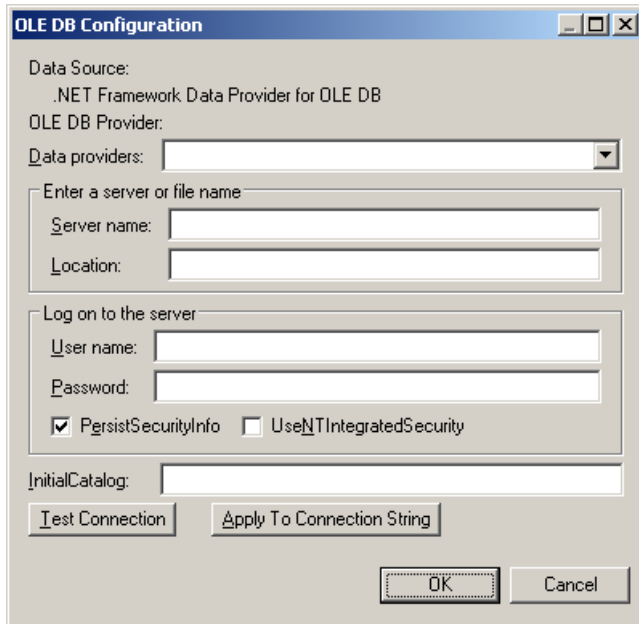
Specifying a Data Source for Cubes

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.



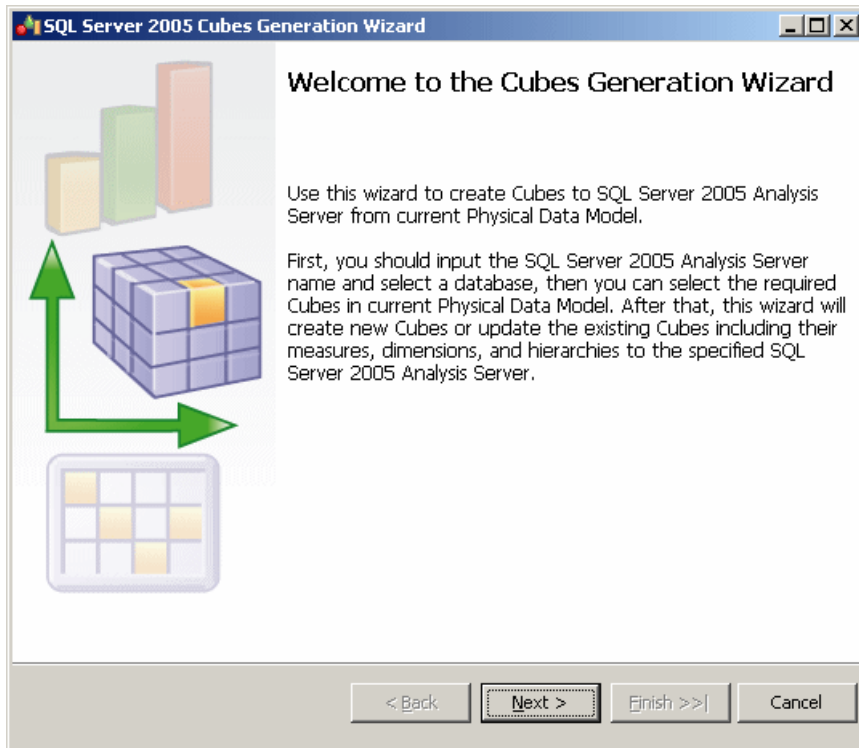
4. Complete the parameters appropriately, click Apply to Connection String, and then Test Connection. Then click Ok to return to the data source property sheet.
5. Click OK to return to your model.

When you have created the appropriate data sources, you can proceed with generating your cubes.

Generating Cubes for Microsoft SQL Server 2005

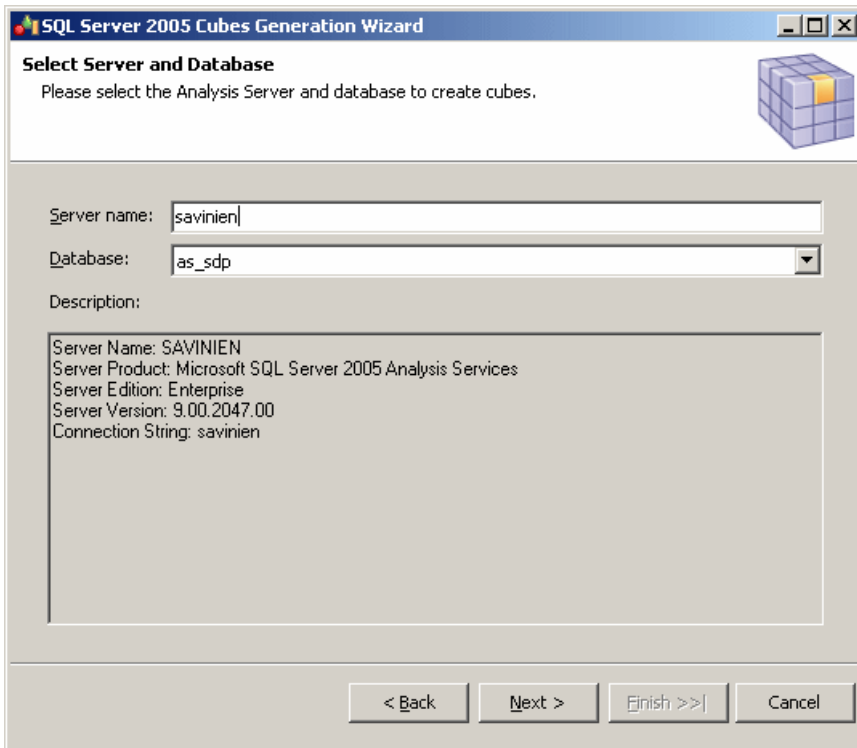
The Microsoft SQL Server 2005 Analysis Services add-in enables the generation of cubes.

1. Select **Tools > Microsoft SQL Server 2005 Analysis Services > Generate Cubes** to open the wizard.



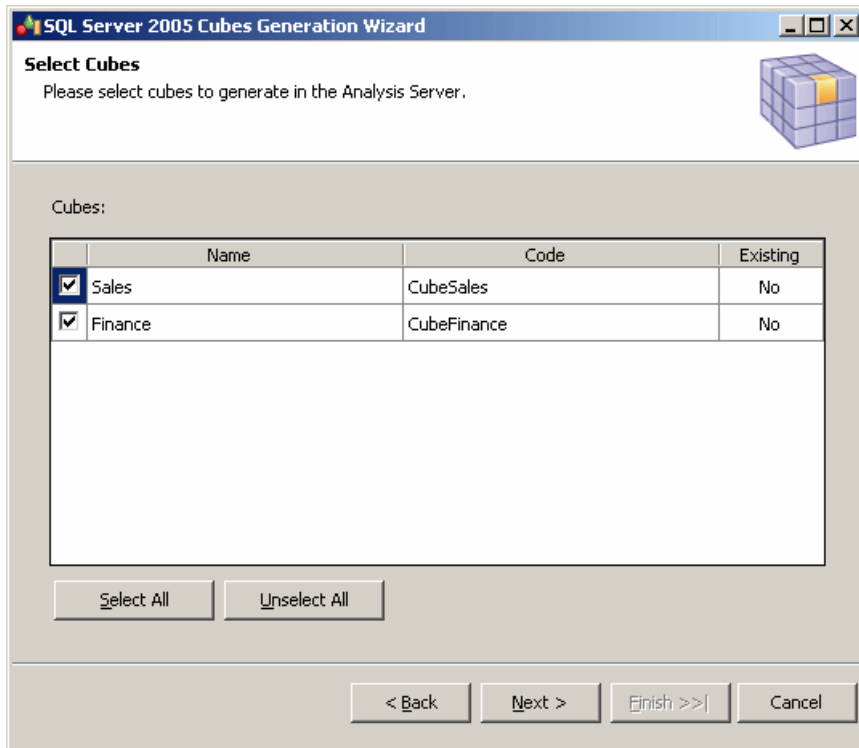
Click Next to continue.

2. Enter a server name, and select the database you want to generate to:



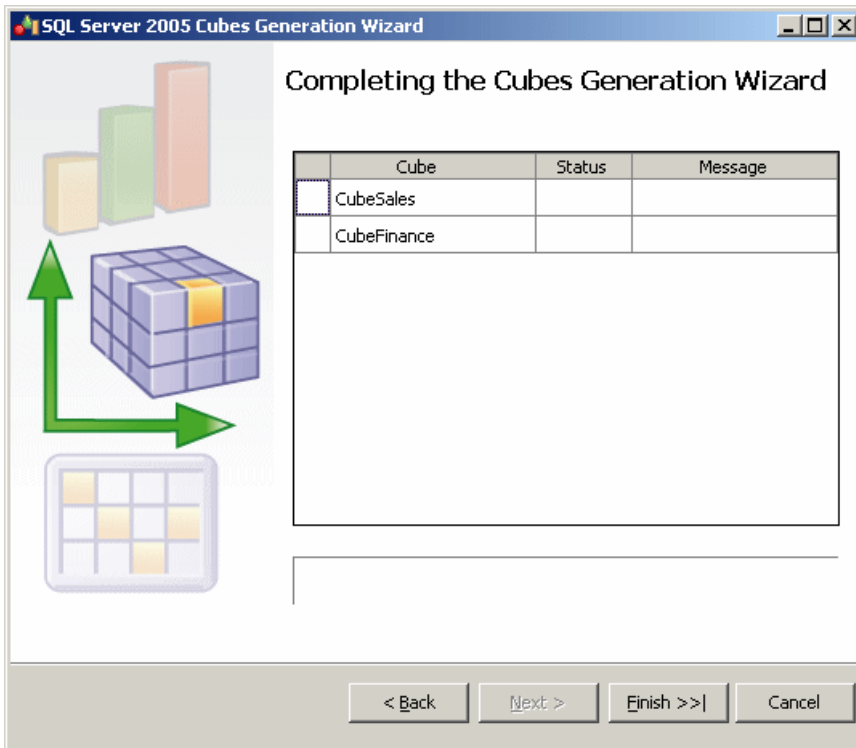
Click Next to continue.

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.

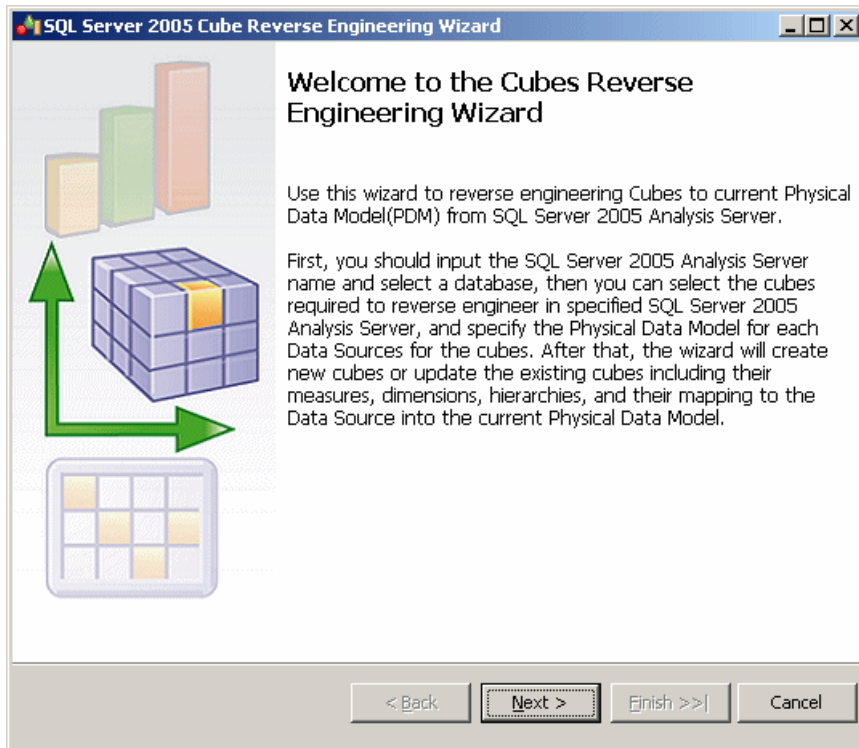
If a cube already exists in the database, it is dropped and recreated. If a related dimension already exists, it is reused. To fully generate a cube, your model must include a complete mapping to a table.

Reverse Engineering Microsoft SQL Server 2005 Cubes

The Microsoft SQL Server 2005 Analysis Services add-in enables the reverse engineering of cubes.

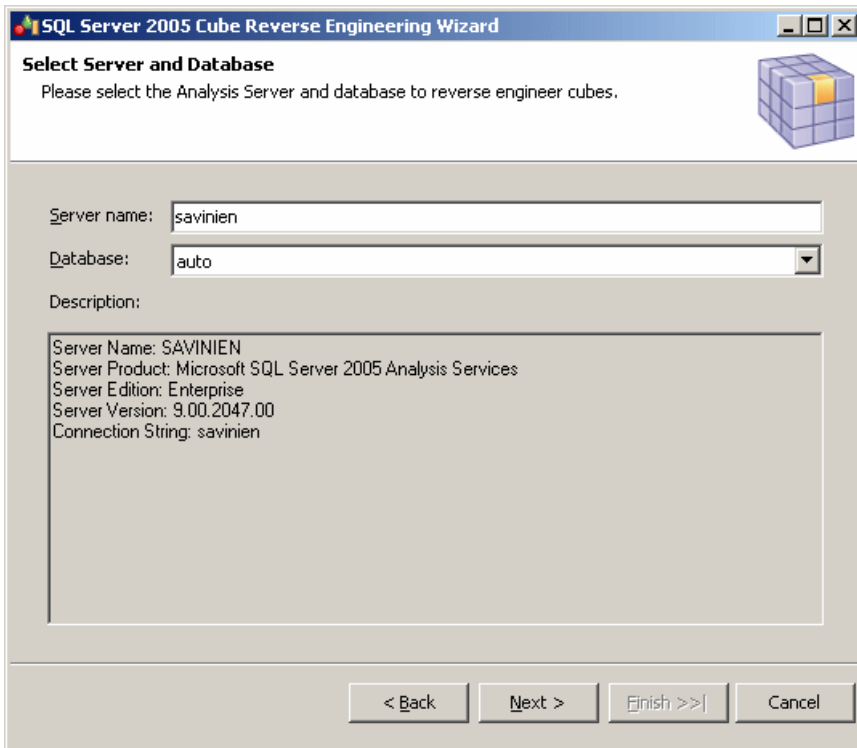
Before reverse-engineering cubes, you should create one or more PDMs to model the tables which provide its data. As part of the reverse-engineering process, PowerDesigner will create links from the reversed cubes to these tables.

1. Select **Tools > Microsoft SQL Server 2005 Analysis Services > Reverse Engineer Cubes** to open the wizard.



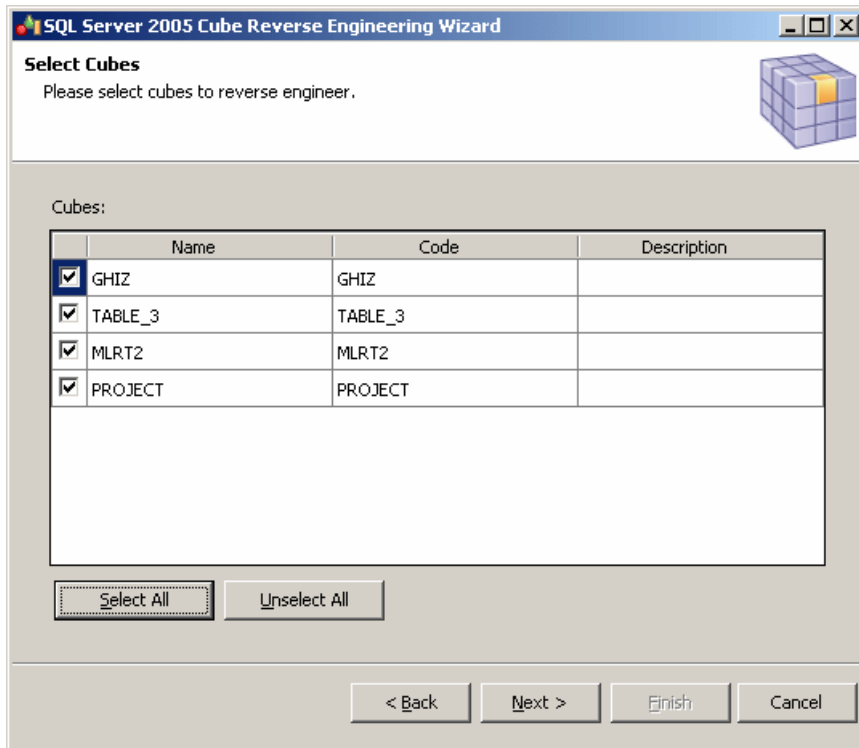
Click Next to continue.

2. Enter a server name, and select the database you want to reverse from:

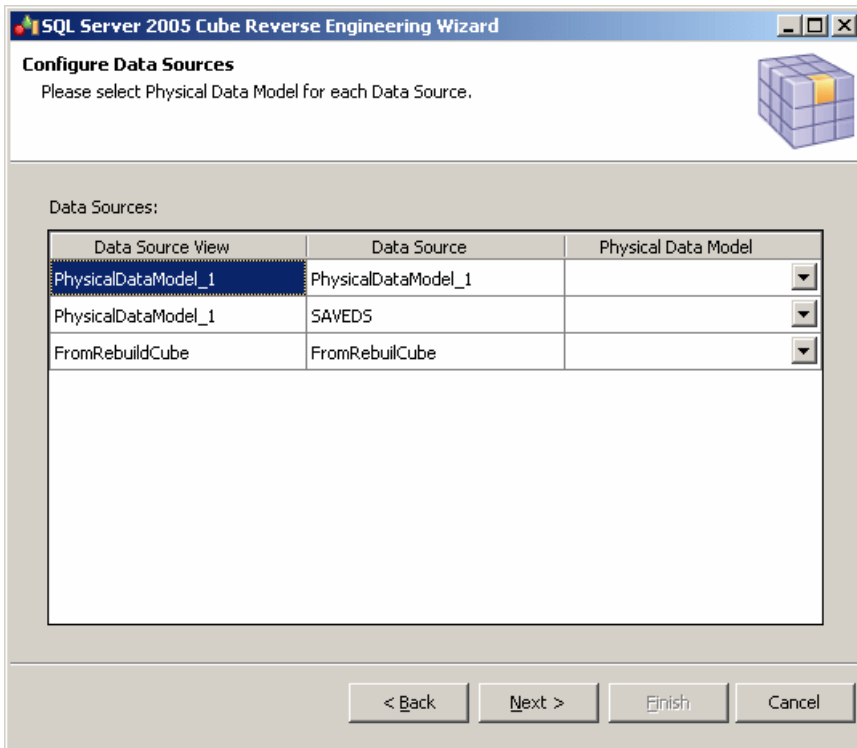


Click Next to continue.

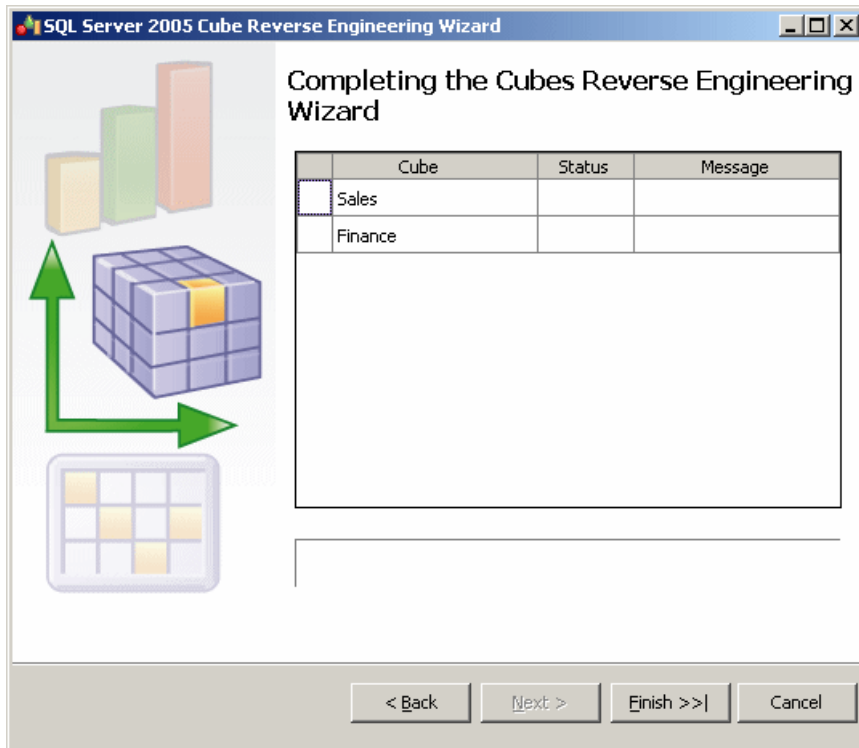
3. The Select Cubes page lists the available cubes. Select the cubes you want to reverse engineer and then click Next to continue:



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 17 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 and v8i (8.1.5) are deprecated.

When working with Oracle triggers, you can use the TRGBODY and TRGDESC variables. For information about working with variables, see *PDM Variables* on page 404.

The following table lists Oracle dimension objects and their equivalents in PowerDesigner:

Oracle object	PowerDesigner object
Dimension	Dimension
Hierarchy	Dimension hierarchy
Level	Dimension attribute used in a hierarchy
Attribute	Dimension attribute used as detail attribute

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.

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
Deferred option of check constraint	Defines the deferred option of a column constraint check. It is used in the definition or create and add items statements. Scripting name: ExtColumnDeferOption
Name of not null constraint	[v8i and higher] Defines the name of the not null constraint for a column. Scripting name: ExtNotNullConstraintName
Deferred option of not null constraint	[v8i and higher] Defines the deferred option of a column not null constraint. It is used in "create" and "add" statement items definition. An empty value means "Not deferrable". Scripting name: ExtNotNullDeferOption
Encrypted	[v10gR2 and higher] Specifies if column is encrypted. Scripting name: Encrypted
Algorithm	[v10gR2 and higher] Specifies the algorithm used for encryption. Scripting name: Algorithm
With salt	[v10gR2 and higher] Specifies if encryption adds salt to encoded data. Scripting name: EncryptionWithSalt
Identified by Password	[v10gR2 and higher] Identifies by password. Scripting name: IdentifiedByPassword

Database Packages

The following 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

References

The following extensions are available on the Oracle tab:

Name	Description
Deferred option of foreign key constraint	Defines the deferred option of a reference. It is used in the definition of create and add items statements. Scripting name: ExtReferenceDeferOption
Exceptions into	Specifies a table into which Oracle places the ROWIDs of all rows violating the constraint. Scripting name: ExceptionsInto
Rely	[v8i and higher] Specifies whether an enabled constraint is to be enforced. Specify RELY to enable an existing constraint without enforcement. Specify NORELY to enable and enforce an existing constraint. Scripting name: Rely
Disable	Disables the integrity constraint. Scripting name: Disable
Validate	Checks that all old data also obeys the constraint. Scripting name: Validate

Tables

The following 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
Object properties	Specifies that the properties of object tables are essentially the same as those of relational tables. However, instead of specifying columns, you specify attributes of the object. Scripting name: XmlTypeObjProperty

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: <ul style="list-style-type: none"> 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 table-space	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: <ul style="list-style-type: none"> 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 9, Generating and Reverse-Engineering Databases* on page 345). 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 package in the Browser, and select **New > Database Package**.

Database Package Properties

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

The following properties are available on the General tab:

Property	Description
Name	The name of the item which should be clear and meaningful, and should convey the item's purpose to non-technical users.
Code	The technical name of the item used for generating code or scripts, which may be abbreviated, and should not generally include spaces.
Comment	Additional information about the database package.
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined.
Owner	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 <i>Database Package Templates</i> on page 542). 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.

The following tabs are also available:

- Procedures – Lists the procedures associated with the database package (see *Database Package Procedures* on page 539).
- Variables - Lists the variables associated with the database package (see *Database Package Variables* on page 539).
- Cursors - Lists the cursors associated with the database package (see *Database Package Cursors* on page 540).
- Exceptions – Lists the exceptions associated with the database package (see *Database Package Exceptions* on page 541).
- Types - Lists the types associated with the database package (see *Database Package Types* on page 541).
- 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 255).

In addition to the standard Name, Code, Comment, and Stereotype, a database package procedure has the following properties:

Property	Description
DB Package	Name of the database package to which the procedure belongs.
Type	Allows you to choose between procedure and function.
Return data type	Allows you to define the return data type of a function.
Pragma	Allows you to type a compiler directive, that is, a string for specifying compilation parameters for the procedure.
Public	Allows you to declare the procedure in the package specification and to permit use from outside the database package. A private procedure (check box deselected) is only defined in the package body.

The following tabs are also available:

- **Parameters** – Lists the input and output parameters required by the procedure (see *Database Package Parameters* on page 542).
- **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.

In addition to the standard Name, Code, and Comment, a database package variable has the following properties:

Property	Description
DB Package	Name of the database package to which the variable belongs.
Data Type	Data type of the variable. You can use the Question Mark button to display the list of Standard Data Types.
Mandatory	If selected, indicates that the not null clause is set on the variable, thus making it mandatory.
Length	Allows you to define the variable length.
Precision	Number of places after the decimal point, for data values that can take a decimal point.
Default value	Default value of the variable.
Constant	Indicates that the variable is a constant. A constant has a value assigned. For example: Credit_Limit constant REAL := 500 000;
Public	Allows you to declare the variable in the package specification and to permit use from outside the database package. A private variable (check box deselected) is only defined in the package body.

Database Package Cursors

A cursor is a multi-row query, which lets you name a work area and access its stored information. You create database package cursors on the **Cursors** tab of a database package using the **Add a Row** tool.

In addition to the standard Name, Code, and Comment, a database package cursor has the following properties:

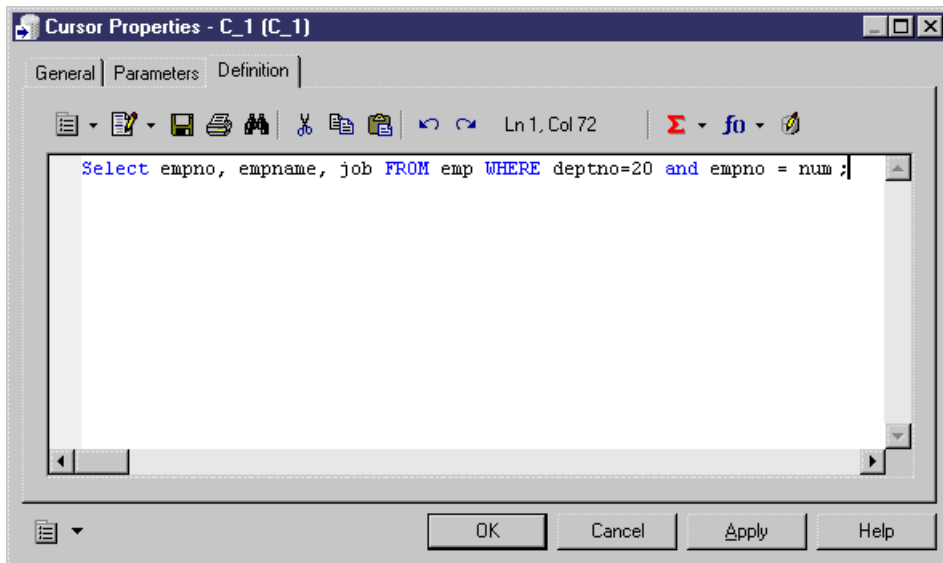
Property	Description
DB Package	Name of the database package to which the cursor belongs.
Return Data Type	Allows you to define the data type of a cursor result value.
Public	Allows you to declare the cursor in the package specification and to permit use from outside the database package. A private cursor (check box deselected) is only defined in the package body.

The following tabs are also available:

- Parameters – Lists the input and output parameters required by the cursor (see *Database Package Parameters* on page 542).
- 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.

In addition to the standard Name, Code, and Comment, a database package exception has the following properties:

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.

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.

In addition to the standard Name, Code, and Comment, a database package type has the following properties:

Property	Description
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.

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.

In addition to the standard Name, Code, and Comment, a database package parameter has the following properties:

Property	Description
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.

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	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	Provides a description of the template.
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 *Chapter 4, Customizing Generation with GTL in Customizing and Extending PowerDesigner*.

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 package in the Browser, and select **New > Cluster**

Cluster Properties

You can modify an object's properties from its property sheet. To open a cluster property sheet, double-click its Browser in the Clusters folder.

The following extended attributes are available on the General tab:

Name	Description
Owner	Specifies the owner of the cluster

In addition, the following tabs are available:

- Columns – lists the columns associated with the cluster
- Indexes – lists the indexes defined for the cluster

Database Links (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 package in the Browser, and select **New > Database link**.

Database Link Properties

You can modify an object's properties from its property sheet. To open a database link property sheet, double-click its Browser in the Database links folder.

The following extended attributes are available on the Oracle tab:

Name	Description
Shared	<p>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.</p> <p>Scripting names: Shared, AuthenticatedBy, AuthenticationPassword</p>
Connect to	<p>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.</p> <p>Scripting names: Username, Password</p>
Service name	<p>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.</p> <p>Scripting name: ServiceName</p>
Physical data model	<p>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.</p> <p>Scripting name: LinkModel</p>

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	<p>Specifies the method for partitioning the table. You can choose between:</p> <ul style="list-style-type: none"> • 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. <p>When you select a type, additional options are displayed, to allow you to specify the appropriate parameters.</p>

CHAPTER 18 Sybase ASE

To create a PDM with support for features specific to the Sybase ASE 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 Sybase AS Enterprise v11.0 and v11.5-11.9 are 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	<p>Indicates how records are distributed on table partitions. You must choose between:</p> <ul style="list-style-type: none">• 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). <p>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.</p> <p>Scripting name: Partition</p>
Columns	<p>[range and hash] Specifies an ordered list of columns used to determine into which partition a row belongs.</p> <p>Scripting name: PartitionByRangeColumnListColumn, PartitionByHashColumnListColumn</p>

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 create or select a key (see <i>Encryption Keys</i> on page 552). 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

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: <ul style="list-style-type: none"> • [for standard databases] <code>inmemory</code>, <code>temporary</code>, or <code>inmemory temporary</code> • [for cluster databases] <code>temporary</code>, <code>global temporary</code>, or <code>system temporary</code> 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 586.

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 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
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

Name	Description
Password phrase	<p>[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.</p> <p>Scripting name: PasswordPhrase</p>
Default encryption key	<p>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.</p> <p>Scripting name: Default</p>

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 19 Sybase IQ

To create a PDM with support for features specific to the Sybase AS 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 DBMSs for Sybase IQ v12.0 and v12.4.3 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 database file (dbspace) in which to create the table. Scripting name: DBSpace
Global temporary table	[v12.4.3 and higher] Specifies that the table is a global temporary table. Scripting name: ExtGlobalTemporaryTable
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
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
Number of distinct value (Iq unique)	Defines the cardinality of the column (to optimize the indexes internally). Scripting name: ExtIqUnicity

Indexes

The following extensions are available on the Sybase tab (v15.0 and higher):

Name	Description
Tablespace	[Non-text indexes] Specifies the index dbspace. 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
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 <i>Text Configurations</i> on page 565) 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, References, and Table and Column Partitions

The following extensions are available on the General tab (v15.0 and higher):

Name	Description
Values	[table partitions] Denotes the infinite upper bound and can only be specified for the last partition. Scripting name: Values

Name	Description
Partition	[column partitions] Specifies the partition on which column rows are stored. Scripting name: PortNumber
DBSpace	Specifies the DBSpace where the object is stored. 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
Connection string	Specifies the connection string used to connect to the external database in the format: <ul style="list-style-type: none"> • 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	<p>[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.</p> <p>Scripting name: TempFunction</p>
Return data type	<p>Specifies the procedure return data type.</p> <p>Scripting name: ReturnDttp</p>
Routine characteristics	<p>[standard functions] Transact-SQL-like error handling and deterministic options.</p> <p>Scripting name: RoutineCharacteristics</p>
Sql security	<p>[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.</p> <p>Scripting name: SqlSecurity</p>
URL	<p>[web functions] Specifies the URL of the web service.</p> <p>Scripting name: URL</p>
Type	<p>[web functions] Specifies the format used when making the web service request.</p> <p>Scripting name: URLType</p>
Header	<p>[HTTP web functions] When creating HTTP web service client functions, use this clause to add or modify HTTP request header entries.</p> <p>Scripting name: Header</p>
Soap header	<p>[SOAP web functions] When declaring a SOAP web service as a function, use this clause to specify one or more SOAP request header entries.</p> <p>Scripting name: SoapHeader</p>
Certificate	<p>[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.</p> <p>Scripting name: Certificate</p>
Client port	<p>[HTTP web functions] Identifies the port number on which the HTTP client procedure communicates using TCP/IP.</p> <p>Scripting name: ClientPort</p>

Name	Description
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 <i>Login Policies</i> on page 564). 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

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 289.

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 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

Name	Description
Schedule definition	<p>Enter the schedule of event trigger times here. Click the New button to launch a dedicated editor window.</p> <p>Scripting name: SchedulesText</p>
Event is triggered	<p>Specifies that the server carries out a set of actions when a predefined type of system event occurs.</p> <p>This option is the default and, if selected, disables the "Event is scheduled" option.</p> <p>Scripting name: TypedEvent</p>
Event type	<p>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.</p> <p>Scripting name: EventType</p>
Trigger condition	<p>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:</p> <p>WHERE event_condition('LogDiskSpacePercentFree') < 20</p> <p>The argument to the event_condition function must be valid for the event type.</p> <p>You can use multiple AND conditions to make up the WHERE clause, but you cannot use OR conditions or other conditions.</p> <p>Scripting name: TriggerCondition</p>
Handler	<p>Each event has one handler.</p> <p>The actions of an event handler are committed if no error is detected during execution, and rolled back if errors are detected.</p> <p>Scripting name: Handler</p>
Enable	<p>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.</p> <p>Scripting name: Enable</p>
At (databases)	<p>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.</p> <p>Scripting name: Database</p>

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 models dbspaces as tablespaces (see *Tablespaces and Storages (PDM)* on page 298 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 299).
- 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
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

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 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

Name	Description
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 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:

Name	Description
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

Name	Description
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 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
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
Password expires	Specifies that the user's password will expire in the next login. Scripting name: PasswordExpiryOnNextLogin
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 life time	Specifies the maximum number of days before a password must be changed. Scripting name: PasswordLifeTime

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 566).

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 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 123), and selecting the type TEXT. For information about the properties specific to text indexes, see *Chapter 19, Sybase IQ* on page 555.

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

It is best to consider all criteria in combination, rather than individually. To decide on indexes, look closely at the data in each column.

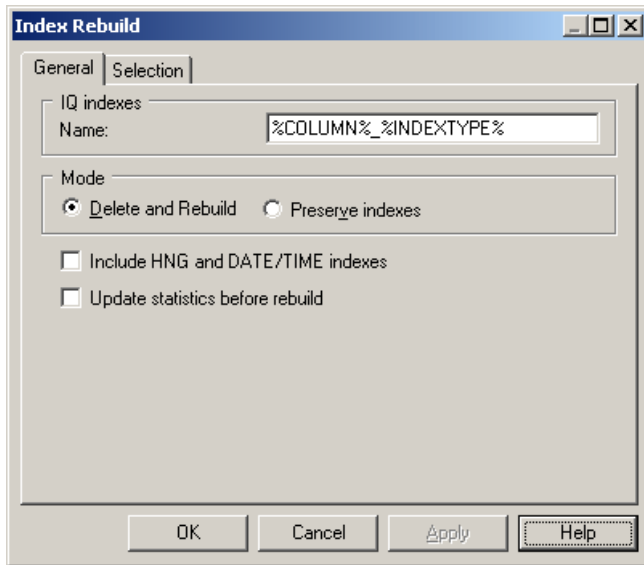
Try to anticipate the number of unique and total values, the query results users will want from it, and whether the data will be used in ad hoc joins or join indexes.

For more information about choosing index types, see chapter Adaptive Server IQ Indexes in the *Adaptive Server IQ Administration and Performance Guide*.

Rebuilding IQ Indexes

As you develop a PDM or modify an existing one, you may change data types, alter the percentage of distinct values or change the number of values in tables. You must then rebuild the IQ indexes to reflect these changes.

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.

Index Types

When you rebuild indexes, PowerDesigner determines the index type based on information contained from the table statistics. It uses the number field, which indicates the estimated number of records per table, and the percentage of distinct values to compute the number of unique values. If the user has not specified a number of rows for the table, PD assumes that the table will include at least 1 row of data.

Usually the rebuild process creates a FASTPROJECTION index for all columns. Otherwise, the following rules are applied:

Criteria	Index type
If no statistics are provided and the column has an undefined data type	No index is created
Low number of unique values in a column Column used in join predicate	LOWFAST
High number of unique values in a column No COUNT DISTINCT, SELECT DISTINCT, or GROUP BY queries required	HIGHNONGROUP
Column used in join predicate High number of unique values in a column (more than 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

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 571).

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:

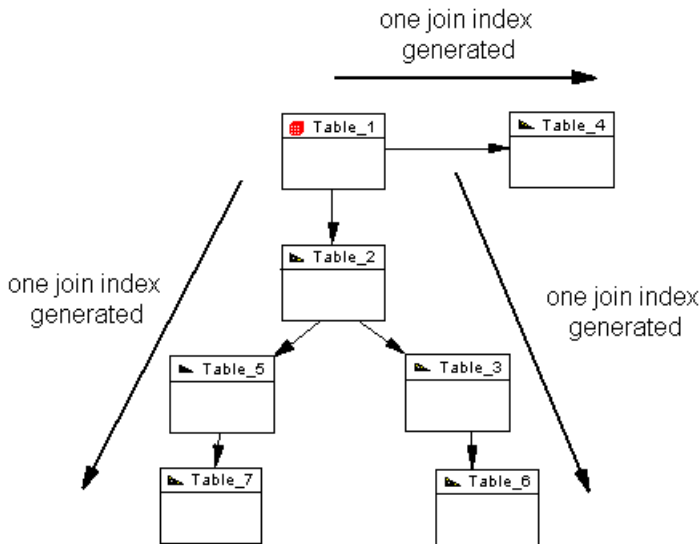
Property	Description
Name	The name of the item which should be clear and meaningful, and should convey the item's purpose to non-technical users.
Code	The technical name of the item used for generating code or scripts, which may be abbreviated, and should not generally include spaces.
Stereotype	Sub-classification used to extend the semantics of an object without changing its structure; it can be predefined or user-defined.
Comment	Descriptive label for the join index.
Base table	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.

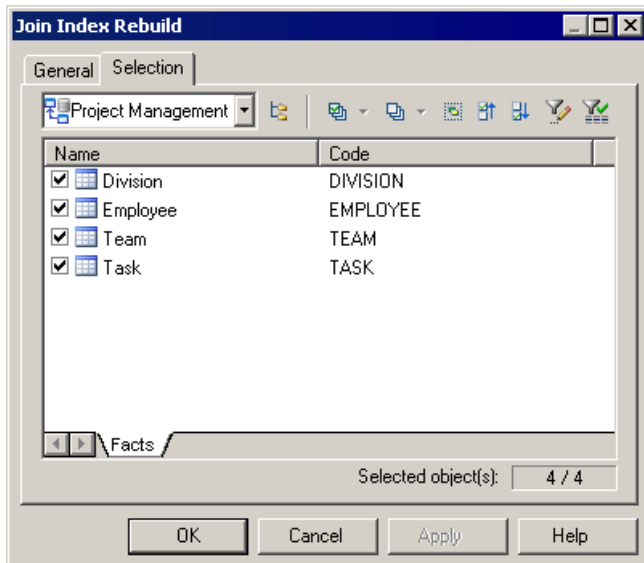
Automatically Creating Join Indexes Through Rebuilding

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.

Generating IQ Data Movement Scripts

PowerDesigner provides the capability to generate data movement scripts to populate your AS IQ data warehouse from your other databases.

The script can be used to:

- Generate a flat file for loading to the AS IQ data warehouse
- Create Insert Location statements for use with a proxy data base (for ASE and ASA only)

To create a data movement script, you must:

- Attach the Data Movement IQ extension file to your AS IQ model.
To enable these extensions in your model, select **Model > Extensions**, click the **Import** tool, select the Data Movement IQ (on the **General Purpose** tab), and click **OK** to attach it.
- Specify your data movement options
- [optional] Create a data source linked to a model of the database from which you want to draw the data to be moved
- [optional] Specify mappings between the tables in your data source and your AS IQ database
- Generate the data movement script

Model Properties Data Movement Tab

The model properties sheet **Data Movement** tab contains properties to control the files used during data movement.

Property	Description
Field delimiter	Specifies the delimiter to be used between fields in the dump file.
Row delimiter	Specifies the delimiter to be used between rows in the dump file.
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.

Creating a Data Source to Populate Your IQ Data Warehouse

You must create a data source to populate your IQ Data Warehouse.

1. Create a PDM to model your source database, and ensure that it is open in your workspace.
2. In your AS IQ PDM, right-click the model name in the Browser and select **New > Data Source**.
3. Enter a name for the source and then click the Models tab.
4. Click the Add Models tool, and select your source model.
5. Click the Database Connection tab, and complete the fields to enable a connection to your source database.
6. Complete the fields on the Data Movement tab and click OK.

Data Source Properties Data Movement Tab

The data source properties sheet **Data Movement** tab contains properties to enable access to 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.
Local user name	Specifies the database user name.

Specifying Data Movement Options

You specify data movement options on the model property sheet.

1. Right-click the model item in the Browser and select Properties from the contextual menu.
2. Click the Data Movement tab and enter the appropriate values for the model as a whole.
3. [optional] To override these global data movement options for a specific table, open its property sheet and enter table-specific values on the Data Movement tab. This tab also allows you to specify a table-specific dump file for importing into the table

Table Properties Data Movement Tab

If the **Data Movement Method** generation option is set to `Insert Location`, a **Data Movement** tab is available on each table properties sheet.

Property	Description
Dump file name	Specifies the name of the 'dump' file (external flat file) that contains the data to be imported.
Field Delimiter	Specifies the delimiter to be used between fields in the dump file.
Row Delimiter	Specifies the delimiter to be used between rows in the dump file.
Maximum Image or Text Size	Specifies the maximum length of an image (or text) record, to which it will be truncated if necessary.

Specifying Mappings Between the Tables in Your Data Source and Your AS IQ Database

You specify mappings using the Mapping Editor.

1. Select **Tools > Mapping Editor** to open the Mapping Editor.
2. Create the necessary mappings and then click OK. For detailed information about using the Mapping Editor, see *Chapter 12, Object Mappings* in the *Core Features Guide*.

Generating the Data Movement Script

You can generate the data movement script from the Tools menu.

1. Select **Tools > Extended Generation** to open the Generation window.
2. Specify a directory in which to generate your data movement files.
3. [optional] Click the Selection tab and specify for which Tables and/or Data Sources you want to generate a data movement script.
4. Click the Options tab and specify your data movement script generation options. You can set the following options:
 - Use Mappings – specifies whether any previously-created mappings should be used for the data movement
 - Data Movement Method – specifies which kind of script to generate. You can choose between:
 - Insert Location – [IQ or ASE only] PowerDesigner will create a loadscript for connecting the source database to the IQ server. Note that if the data source is not an IQ or ASE database, then no loadscript will be generated.
 - External File – PowerDesigner will create a dump file from the source database together with a loadscript to upload it to the IQ server.
5. [optional] Click the Generated Files tab to review the names and locations of the files to be generated.
6. Click OK to begin the generation of the data movement script.

CHAPTER 20 Sybase SQL Anywhere

To create a PDM with support for features specific to the 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 DBMSs for Sybase Adaptive Server Anywhere v7 and v8 are 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
Dbospace (table-space)	Specifies the dbospace in which the table is to be created Scripting name: DbospaceIn

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
Configuration	[Text indexes v12 and higher] Specifies the text configuration (see <i>Text Configurations</i> on page 565) 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
Login policy	Specifies the login policy to assign to the user (see <i>Login Policies</i> on page 564). 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 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	<p>Specifies the type of mirror server to create. You can choose between:</p> <ul style="list-style-type: none"> • 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. <p>Scripting name: Type</p>
Using auto parent	<p>[copy only] Specifies that the primary server will assign a parent for this server.</p> <p>Scripting name: UsingAutoParent</p>
Parent	<p>[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.</p> <p>Scripting name: ParentServer</p>
Alternate parent	<p>[copy only] Specifies an alternate parent for the copy node.</p> <p>Scripting name: AlternateParentServer</p>
Primary	<p>[copy only] Specifies that the parent server is the primary server.</p> <p>Scripting name: PrimaryParentServer</p>
Connection string	<p>Specifies the connection string to be used to connect to the server.</p> <p>Scripting name: ConnectionString</p>

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.

Spatial Reference Systems (SQL Anywhere)

Sybase SQL Anywhere (v12 and higher) supports 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 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

Name	Description
Organization	Specifies the organization that created the spatial reference system that the new spatial reference system is based on. Scripting name: Organization
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
<i>Axis/Bounded/Unbounded</i>	Specifies whether the axis is bounded or unbounded and, if it is bounded, the minimum and maximum values. Scripting names: <i>BoundedCoordinateAxis</i> , <i>MinCoordinateAxis</i> , <i>MaxCoordinateAxis</i>
Ellipsoid axis length	[round earth systems] Specifies the values to use for representing the Earth as an ellipsoid. Scripting names: <i>SemiMajorAxisLength</i> , <i>SemiMinorAxisLength</i> , <i>InverseFlattening</i>
Grid Size	[planar systems] Specifies the size of the grid to use when performing calculations. Scripting name: <i>GridSize</i>
Tolerance	[planar systems] Specifies the precision to use when comparing points. Scripting name: <i>Tolerance</i>
Linear/Angular unit of measure	Specify the linear and angular units of measure for the spatial reference system. Scripting name: <i>LinearUnitOfMeasure</i> , <i>AngularUnitOfMeasure</i>

Spatial Units of Measure (SQL Anywhere)

Sybase SQL Anywhere (v12 and higher) supports 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 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

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 560, *Login Policies (IQ/SQL Anywhere)* on page 564, and *Full Text Searches (IQ/SQL Anywhere)* on page 565.

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 it 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. To enable the proxy table extensions in your model, select **Model > Extensions**, click the **Import** tool, select the Proxy Tables file (on the **General Purpose** tab), and click **OK** to attach it.

After designing the proxy tables, you can use the *build data source* feature that will create a data source for each target model of the current model. Target models are models containing the target tables of the replica or external shortcuts, they are also called *remote servers*.

Once the data sources are properly defined, you can use the *extended generation* feature to generate the proxy table and remote server creation scripts.

The ProxyTables extension file contains generation templates, extended attributes, custom checks and custom methods to support the definition of external proxy tables. Double-click the Proxy Tables extension file (in the Extensions folder) in the Browser to open it in the Resource Editor for review. The following extensions must be defined in the Profile category to fully support proxy tables:

- BasePackage:
 - Generation template - for generating proxy tables.

- DataSource:
 - Connection information custom check - verifies that the connection information is sufficient to connect to the database. You must specify the data source name, user login and password in the Database Connection tab of the data source property sheet.
 - GenerateAsProxyServer extended attribute - when set to true, defines the data source model as the proxy remote server.
 - [various templates] - used for proxy table generation.
- Model:
 - Proxy Servers and Tables generated files - to generate proxy server and table script files.
 - Menu – provides a contextual menu for building data sources and commands in the Tools menu for rebuilding data sources and generating proxy tables.
 - BuildProxyTableDataSourcesand GenerateProxyTables methods - used in the menu.
 - [various templates] - required for proxy server and proxy table script generation.
- Shortcut:
 - Data source existence custom check - verifies that data sources are defined for the shortcuts.
- Table:
 - Proxy table is child of reference custom check - verifies that the model replica is not the child of another table via a reference link.
 - [various templates] - required for proxy table, remote server and access definition creation syntax.

Creating a Proxy Table

You use external shortcuts and/or replica to design proxy tables in your model.

An external shortcut is a non-modifiable reference to an object in another model. For more information on shortcuts, see "Shortcuts" in the Shortcuts and Object Replications chapter of the *Core Features Guide*.

A replica is an exact copy of an object that can be updated when the original object is modified. For more information on replications, see "Replications" in the Shortcuts and Object Replications chapter of the *Core Features Guide*.

One interesting aspect of using replica, is that you can modify the replica code in order to make it different from the target table. A custom check verifies that replica are not used as child tables of a reference.

1. Select a table in a target model and drag it to the model where you want to create proxy tables using the appropriate key combination to create either an external shortcut or a replica.
2. Repeat this operation for each proxy table.

Defining the Remote Server of a Proxy Table

The remote server is the model containing the target tables of the external shortcut or replica. The remote server is defined using a data source in the proxy tables model; this data source provides access to the remote data on the server.

Note: the same data source can contain information for several models that share the same remote servers.

When you attach the ProxyTables extension file to the model containing the proxy tables, a new command is added to the contextual menu for the physical data model item. The target models must be open in the workspace in order for the command to build data sources for them.

1. Create a new data source and set the `GenerateAsProxyServer` extended attribute to True.
2. Add the target models in the Models tab of the data source property sheet or right-click the model that contains replica and/or shortcuts in the Browser and select the Build Proxy Tables Data Sources command. A data source is automatically created for each target model.
3. Double-click a data source in the Browser to display its property sheet.
4. Click the Database Connection tab, and define the data source name, login and password.
5. Click OK.
6. Repeat steps 2 to 5 for each data source.

Generating the Remote Server and Proxy Tables Creation Scripts

You can generate the remote server and proxy tables creation scripts in order to run them in the database. You must launch the generation from the model containing proxy tables.

The ProxyTables extension file contains the creation script syntax for ASA or ASE.

1. Select **Tools > Proxy Tables > GenerateProxy Tables** to open the Generation dialog box, and click the Options tab.
2. Set a value for the UserReplica and UserShortcut options to allow you to generate the proxy tables corresponding to replica and/or external shortcuts.
3. Set the Generate proxy servers option to one of the following values:
 - True – to generate proxy servers. You can deselect any proxy servers you do not want to generate.
 - False – to not generate proxy servers
4. Click OK to begin generation.

The generated script is displayed in the Result dialog box.

5. [optional] Double-click the generated SQL file or click the Edit button to open the script in a text editor.

6. Run the script on your database in order to create the proxy tables.

CHAPTER 21 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.

The following sections list the extensions provided for Teradata.

Abstract Data Types

The following extensions are available on the Teradata tab if the type is distinct (V2R6 and higher):

Name	Description
Predefined data type	Indicates that character column comparison uses character case (upper and lower) to raise differences. Scripting name: PredefinedDataType

Abstract Data Type Procedures

The following 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
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

Name	Description
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
Case specific	Specifies that character column comparison is case-sensitive. Scripting name: CaseSpecific
Character set	Specifies the character set to be used. Scripting name: CharacterSet

Name	Description
System generated only	<p>Specifies that identity column values are always system-generated. You cannot insert values into, nor update, an identity column defined as GENERATED ALWAYS.</p> <p>If not selected, identity column values are system-generated unless the user does not enter a non-null value.</p> <p>Scripting name: ExtGenAlways</p>
Compressed values	<p>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.</p> <p>Attribute must be enclosed in parenthesis when it is composed of multiple values.</p> <p>Scripting name: Compress</p>

Databases

The following extensions are available on the Teradata tab:

Name	Description
Account	<p>Specifies the account ID identifiers.</p> <p>Scripting name: Account</p>
After journal	<p>Specifies the type of image to be maintained by default for data tables created in the new database.</p> <p>Scripting name: AfterJournal</p>
Default journal table	<p>Specifies the default table that is to receive the journal images of data tables created in the new database.</p> <p>Scripting name: DefaultJournalTable</p>
Fallback	<p>Specifies whether to create and store a duplicate copy of each table created in the new database.</p> <p>Scripting name: Fallback</p>
Owning database	<p>Specifies the name of the immediate owning user or database. The default is the user name associated with the current session.</p> <p>Scripting name: FromDatabaseName</p>

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
Permanent space	Specifies the number of bytes to be reserved for permanent storage of the new user database. The space is taken from unallocated space in the database of the immediate owner. Scripting name: PermanentSpace
Spool space	Specifies the number of bytes (n) to be allocated for spool files. The default is the largest value that is not greater than the owner spool space, and that is a multiple of the number of AMPs on the system. Scripting name: SpoolSpace
Temporary space	Specifies how much space (in bytes) is to be allocated for creating temporary tables by this user. Note that temporary space is reserved prior to spool space for any user defined with this characteristic. Scripting name: TemporarySpace

Indexes

The following extensions are available on the Teradata tab:

Name	Description
Primary Index	Specifies that the index is the primary index. Scripting name: PrimaryIndex
Partition by	[primary key] Lets you select the used function to evaluate partition condition. <ul style="list-style-type: none"> • 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

Name	Description
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
On commit action	Specifies the action to take with the contents of a global temporary table when a transaction ends: <ul style="list-style-type: none"> • DELETE ROWS - clears the temporary table of all rows. • PRESERVE ROWS - retains the rows in the table after the transaction is committed. Scripting name: CommitRowAction

Name	Description
Type	<p>Specifies whether the table to be created is a global temporary table or a volatile table:</p> <ul style="list-style-type: none"> • 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. <p>Scripting name: GlobalTemporary</p>
Duplicate row control	<p>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 MULTISSET. Some client utilities have restrictions with respect to MULTISSET tables.</p> <p>Scripting name: SetOrMultiset</p>

Users

The following extensions are available on the Teradata tab :

Name	Description
Owner	<p>Specifies the database (or user) that owns the current user.</p> <p>Scripting name: DBOwner</p>
Permanent	<p>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.</p> <p>Scripting name: PermanentSpace</p>
Spool	<p>Specifies the number of bytes (n) to be allocated for spool files. The default is the largest value that is not greater than the owner spool space, and that is a multiple of the number of AMPs on the system.</p> <p>Scripting name: SpoolSpace</p>
Temporary	<p>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.</p> <p>Scripting name: TemporarySpace</p>

Name	Description
Account	Specifies the account ID identifiers. Scripting name: Account
Fallback	Specifies whether to create and store a duplicate copy of each table created in the new database. Scripting name: Fallback
Journal	Specifies the number of before change images to be maintained by default for each data table created in the new database. Scripting name: Journal
After journal	Specifies the type of image to be maintained by default for data tables created in the new database. Scripting name: AfterJournal
Default table	Specifies the default table that is to receive the journal images of data tables created in the new database. Scripting name: DefaultJournalTable
Database	Specifies the default database name. Scripting name: DefaultDatabase
Role	Specifies the default role for the user. Scripting name: DefaultRole
Character set	Specifies the default character data type. Scripting name: DefaultCharacterSet
Collation	Specifies the default collation for this user. Scripting name: Collation
Time zone	Specifies the default time zone displacement for the user. Scripting name: TimeZone
Date format	Specifies the default format for importing and exporting DATE values for the user. Scripting name: DateForm
Profile name	Specifies a profile to the user. Scripting name: Profile

Name	Description
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

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 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
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 136.

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 encountered 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 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 changes 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 package in the Browser, and select **New > Join Index**.

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 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

Name	Description
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
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 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 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 subobjects 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

Name	Description
Replication group	Specifies the name of the replication group to which the rule set is assigned. Scripting name: ReplicationGroup

CHAPTER 22 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.

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
External password	Allows a user's password to be authenticated externally to Ingres. The password is passed to an external authentication server for authentication. Scripting name: ExternalPassword
Limiting security label	Allows a security administrator to restrict the highest security label with which users can connect to Ingres when enforcing mandatory access control (MAC). Scripting name: LimitingSecurityLabel
Profile	Allows a profile to be specified for a particular user. If the profile clause is omitted, the default is noprofile. Scripting name: Profile

Interbase

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: Support for Microsoft Access 95 & 97 is deprecated.

The following sections list the extensions provided for MS Access.

Columns

The following extended attributes are available on the Access 2000 tab:

Name	Description
Allow Zero Length	Specifies whether a zero-length string ("") is a valid entry in a table column. Applies only to Text, Memo, and Hyperlink table fields. Scripting name: ExtAllowZeroLength

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.

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
Character set	Set of symbols and encodings. Scripting name: CharSet
Collation	Set of rules for comparing characters in a character set. Scripting name: Collate
National	A way to indicate that a CHAR column should use UTF8 character set. Scripting name: National
Unsigned	Indicates negative values are not allowed for the column. Scripting name: Unsigned

Name	Description
Retrieve with leading zeros	<p>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.</p> <p>If you specify ZEROFILL for a numeric column, MySQL automatically adds the UNSIGNED attribute to the column.</p> <p>Scripting name: ZeroFill</p>

Indexes

The following extended attributes are available on the MySQL tab:

Name	Description
Full text index	<p>Indicates that the index is a full text index.</p> <p>Scripting name: FullText</p>

Keys

The following extended attributes are available on the MySQL tab:

Name	Description
Unique key	<p>When set to True, indicates that the key is unique. False implies that the key allows duplicate values.</p> <p>Scripting name: ExtUnique</p>

Models

The following extended attributes are available on the MySQL tab:

Name	Description
Database type	<p>Indicates the type of the database, as specified in the extended attribute type DatabaseType.</p> <p>Scripting name: DatabaseType</p>

References

The following extended attributes are available on the MySQL tab:

Name	Description
Reference match type	<p>Indicates the reference match type, as specified in the extended attribute type ReferenceMatchType.</p> <p>Scripting name: ReferenceMatch</p>

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.

Tables

The following extensions are available on the General tab.

Name	Description
Distribution	Specifies the type of row distribution. You can choose between None, hash, and random. Scripting name: Distribution
Column(s)	[hash distribution] Specifies the hash distribution columns. Scripting name: DistributeOnExplicitColumnList

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
Array delimiter	[Base Type] Specifies the delimiter character for the array. Scripting name: ExtTypeDelimiter
Array Element type	[Base Type] Specifies the type of the array elements. Scripting name: ExtTypeElement

Name	Description
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
Length	[Base Type] Specifies the internal length of the new type. Scripting name: ExtTypeLength
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
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
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
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

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	<p>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.)</p> <p>Scripting name: ProcLanguage</p>

References

The following extensions are available on the PostgreSQL tab (v8 and higher):

Name	Description
Deferrable	<p>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.</p> <p>Only foreign key constraints currently accept this clause. All other constraint types are not deferrable.</p> <p>Scripting name: Deferrable</p>
Foreign key constraint	<p>If a constraint is deferrable, this clause specifies the default time to check the constraint.</p> <p>False means the constraint is INITIALLY IMMEDIATE, it is checked after each statement. This is the default.</p> <p>True means the constraint is INITIALLY DEFERRED, it is checked only at the end of the transaction.</p> <p>Scripting name: ForeignKeyConstraintDeferred</p>

Tables

The following extensions are available on the PostgreSQL tab (v8 and higher):

Name	Description
Temporary state	<p>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.</p> <p>Scripting name: Temporary</p>

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
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

Name	Description
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

- abstract data type 84, 160
 - abstract object 309
 - categories 160
 - check model 309
 - create 161
 - instantiable object 309
 - link to Java class 164
 - properties 161
- abstract data type procedure
 - check model 310
- Access 97 358, 383
- access.mdb 358
- aggregate function (CLR) 485
- AKCOLN 417
- alias 142
- ALLCOL 418
- alter (script) 368
- alternate key 79, 118
 - constraint 122
 - designate 121
 - number 122
 - variable 122
- Analysis services (MS SQL Server) 512
- apm 10
- archive 10
- ASK column default 103
- assertion
 - template 14
- association 67, 202, 218
 - cardinality 69
 - change to entity 71
 - check model 311, 312
 - create 68, 219
 - dependent 70
 - entity attribute 71
 - properties 69, 219
 - reflexive 70
- association attribute 72
- association link 67
 - cardinality 69
 - properties 69
- asymmetric key 490
- attribute 52, 202, 213

- association 72
 - constraint 105, 107, 108
 - create 52, 214
 - delete 54
 - identifier 54
 - link to identifier 60
 - migrate 60
 - properties 53, 214
 - reuse 54
- auto-increment column in Sybase ASA 580
- auto-migrate 169
 - check parameter 169
 - column 169
 - default join 173
 - domain 169
 - validation rule 169
- auto-migrate columns (model option) 18
- auto-reuse columns 169
- auto-reuse columns (model option) 18
- auxiliary table
 - IBM DB2 450

B

- Barker notation 21
- bitmap join index 535, 536
 - check model 330
 - options in Oracle 536
 - properties 536
 - rebuild 535
- BLOB 366
- build
 - index 128
- business rule 193
 - apply 195
 - create 194
- business rule (PDM)
 - properties 194

C

- candidate key 118
- cardinality 58
 - association 69

- define for an association link 69
 - reference 175
 - relationship 65
- case sensitivity (model option) 16
- CDM
 - association 67
 - association link 67
 - attribute 52
 - check model 309
 - create 6
 - data dictionary 31
 - data item 45
 - description 3
 - domain 148
 - entity 47
 - Entity Relationship Diagram 3
 - ERD 3
 - generate 388
 - identifier 54
 - import 429
 - inheritance 72
 - model options 12
 - notation 21
 - open PAM 11
 - option 63
 - relationship 56
 - role 3
- certificate 488
- change parent allowed (model option) 18
- check model 309
 - abstract data type 309
 - abstract data type procedure 310
 - association (CDM) 311
 - association (PDM) 312
 - bitmap join index 330
 - column 313
 - cube 315
 - data format 318
 - data item 318
 - data source 320
 - database 316
 - database package 316
 - database package subobject 317
 - default 320
 - dimension 321
 - dimension attribute 327
 - dimension hierarchy 327
 - domain 322
 - entity 325
 - entity attribute 323
 - entity identifier 324
 - fact 326
 - fact measure 327
 - group 343
 - horizontal partitioning 328
 - index 328
 - inheritance 329
 - join index 330
 - key 331
 - lifecycle 332
 - package 333
 - phase 332
 - procedure 335
 - reference 336
 - relationship 337
 - role 343
 - sequence 338
 - synonym 338
 - table 339
 - table collapsing 328
 - tablespace 342
 - trigger 342
 - user 343
 - vertical partitioning 328
 - view 339, 344
 - view index 328
 - view reference 336
 - Web operation 344
 - Web service 344
- check on commit (model option) 18
- check parameter
 - auto-migrate 169
 - column 117
 - constraint 86, 117
 - data type 115
 - table 86
 - validation rule 193
- circular extended dependency 267
- CLIENTEXPRESSION 425
- CLR
 - aggregate function 485
 - assembly 484
 - function 487
 - integration 484
 - procedure 487
 - trigger 487
 - user-defined type 486
- cluster

- key 119
 - Oracle 545
 - referential integrity 175
 - clustered index 127
 - CODASYL 16
 - code
 - preview 397
 - collapse tables 91
 - column 79, 103
 - alternate key 121
 - assign défaut 147
 - auto-migrate 169
 - check model 313
 - computed 103, 113
 - constraint 105, 107, 108, 117
 - create 103
 - data type 115
 - domain 116, 154
 - duplicate 116
 - foreign key 103, 122
 - link to key 173
 - list 117
 - migrate 173
 - permission 138
 - primary key 103, 120
 - properties 103, 182
 - reuse 169
 - statistics 384
 - test data profile 113
 - variable 404, 419
 - view 182
 - column denormalization 93
 - delete replica 93
 - duplicate columns 93
 - replication 93
 - revert 93
 - select column 93
 - wizard 93
 - column mandatory by default (model option) 17
 - computed column 103, 113
 - conceptual 16
 - conceptual diagram
 - association 67
 - association link 67
 - attribute 52
 - create 31
 - data item 45
 - entity 47
 - identifier 54
 - inheritance 72
 - relationship 56
 - connect database 345, 372
 - connection profile 345
 - constraint 105, 107, 108
 - alternate key 122
 - column 117
 - create 196
 - data format 107
 - delete 245
 - generate 196
 - insert 245
 - name 86, 122
 - reverse engineer 196
 - table 86
 - update 245
 - constraint name
 - default 122
 - contract 501
 - cube 202, 203
 - check model 315
 - create 203
 - generate cube data 210
 - generate in MS SQL Server 514
 - generate in MS SQL Server 2005 518
 - properties 204
 - rebuild 205
 - reverse engineer in MS SQL Server 515
 - reverse engineer in MS SQL Server 2005 522
 - cube data generation 210
 - currency 299
 - customize
 - function 248
 - procedure 248
 - script 358
- ## D
- data access 87
 - data dictionary 31
 - data fields 210
 - data format 107
 - check model 318
 - data item 45
 - association 72
 - check model 318
 - copy 48
 - create 45
 - delete 54

- error 47
- option 48
- properties 45
- reuse 47, 54
- unique code 47
- data lifecycle 289
- data movement script (Sybase IQ) 572
- data profiling 105, 107, 108
- data source
 - check model 320
 - disconnect 345
- data type
 - average length 366
 - BLOB 366
 - column 115
 - domain 149
 - length 151
 - precision 151
 - undefined 149
- database
 - add user to group 140
 - check model 316
 - connect 345, 372
 - create 9, 346, 358
 - define 9
 - denormalization 87
 - display 345, 372
 - estimate size 366
 - generate 346, 354, 358, 368
 - generate PDM 381
 - generate privileges 135
 - group 131
 - information 345
 - modify 346, 368
 - MS Access 97 358, 383
 - optimize reverse engineering 384
 - owner 132
 - parameter 370
 - permission 136
 - prefix tablespace in DB2 for z/OS 451
 - prefix tablespace in DB2 OS/390 304
 - privilege 133
 - properties 9
 - quick launch 350
 - reverse engineer 373
 - role 131
 - script 368
 - settings sets 350
 - size 366
 - statistics 105, 384
 - user 131
 - variable 404
- database link
 - Oracle 545
- database mirroring 497, 581
- database package 537
 - check model 316
 - cursors 540
 - exceptions 541
 - procedure 539
 - types 541
 - variable 539
- database package cursor 540
- database package exception 541
- database package procedure dependencies 255
- database package subobject
 - check model 317
- database package template 542
- database package type 541
- database partition group 459
- DATE 568
- DB2
 - column default 103
- DB2 for Common Server 453
- DB2 for z/OS 447
- DBCCreateAfter 187, 267
- DBMS
 - physical option 302
- DBMS trigger 234
 - properties 224
- DBMS triggers
 - creating 235
- dbspace 562
- default 146
 - assign 147
 - automatic rebuild 148
 - check model 320
 - column 103
 - constraint name 122
 - create 146
 - delete and rebuild 148
 - physical option 302
 - physical options objects 302
 - properties 147
 - rebuild 148
 - reuse 148
- default constraints delete (model option) 18
- default constraints update (model option) 18

- default data type (model option) 17
 - default implementation (model option) 18
 - default link on creation (model option) 18
 - DEFINE 418
 - DEFINEIF 419
 - denormalization 87
 - column denormalization 93
 - horizontal partitioning 87
 - table collapsing 91
 - vertical partitioning 89
 - dependency
 - procedure 250
 - trigger 250
 - dependent association 70
 - detail attribute 214
 - dimension 202, 210, 212
 - check model 321
 - create 212
 - detail attribute 214
 - properties 212
 - dimension attribute
 - check model 327
 - dimension hierarchy
 - check model 327
 - disconnect data source 345
 - discriminant column 87
 - display preference
 - entity 49
 - reference 178
 - table 101
 - display preferences 21
 - domain
 - CDM 148
 - check model 322
 - constraint 105, 107, 108
 - create 149
 - data type 149
 - LDM 148
 - modify 153
 - PDM 148
 - properties 149
 - domain (PDM)
 - assign default 147
 - attach column 116
 - auto-migrate 169
 - column 154
 - data type 151
 - enforce 154
 - length 151
 - precision 151
 - variable 404
 - domain mandatory by default (model option) 17
 - DSO metamodel 512
 - DTTM 568
 - duplicate columns 93
- ## E
- E/R + Merise 12
 - enable links to requirements (model option) 16
 - enabledbprefix 451
 - encoding for reverse engineering 380
 - encryption 488
 - encryption key
 - Sybase ASE 552
 - end point 499
 - enforce non-divergence (model option) 17
 - entity 47
 - add to inheritance 76
 - associative 64
 - attribute 54
 - change from relationship 64
 - check model 325
 - constraint 107, 108
 - copy 48
 - create 47
 - create from a relationship 64
 - create from association 71
 - delete 54
 - display preferences 49
 - entity attribute 48
 - generate table 390
 - identifier 48, 54
 - inheritance 72, 390
 - properties 48
 - entity attribute
 - add to identifier 55
 - association 71
 - check model 323
 - copy 48
 - identifier 48, 55
 - relationship 64
 - entity identifier
 - check model 324
 - entity/relationship 12
 - ERD
 - CDM 3
 - Entity Relationship Diagram 3
 - ERROR 419

- error message 419
 - data item 47
 - relationship 63
 - unique code 47, 63
 - user-defined 269, 270
 - error table 600
 - ERwin
 - import 429
 - migrate CDM/PDM pair 430
 - migrate multiple files 432
 - migrating from 430, 432
 - user-defined properties 429
 - estimate
 - database size 366
 - event
 - EventDelimiter 234
 - multiple 234
 - event monitor 463
 - event monitor group 463
 - event notification 505
 - EventDelimiter 234
 - events 560, 586
 - extended attribute
 - IBM DB2 for Common Server 453
 - PowerBuilder 99
 - extended dependencies
 - circular 187, 267
 - stored procedures 267
 - view 187
 - extended dependency 28
 - extension 27
 - extension file 27
 - extensions
 - HP Neoview 441
 - IBM DB2 for z/OS 447
 - Informix SQL 607
 - Ingres 608
 - Interbase 609
 - MS Access 609
 - MS SQL Server 467
 - MySQL 610
 - Netezza 612
 - NonStop SQL 612
 - Oracle 529
 - PostgreSQL 613
 - PowerBuilder 99
 - Red Brick Warehouse 617
 - Sybase ASA 577
 - Sybase ASE 549
 - Sybase IQ 555
 - Sybase SQL Anywhere 577
 - Teradata 591
 - external table 87
- ## F
- fact 202, 215
 - check model 326
 - create 215
 - properties 216
 - fact measure
 - check model 327
 - FASTPROJECTION 568
 - FKCOLN 419
 - FOREACH_CHILD 420
 - FOREACH_COLUMN 421
 - FOREACH_PARENT 422
 - foreign identifier
 - change attribute 61
 - foreign key 79, 118
 - auto-migrate 169
 - change column 173
 - column 103
 - designate 122
 - generate from identifier 388
 - index 127
 - referential integrity 175
 - variable 423
 - format
 - variable 416
 - full text indexes 586
 - full text search 565
 - full-text catalog 492
 - full-text index 493
 - full-text search 492
 - function
 - create 248
 - custom 248
 - define 248
 - permission 136
- ## G
- generate
 - CDM 393
 - constraint 196
 - cube data 210

- cubes in MS SQL Server 514
 - data source 99
 - database 346, 354, 358, 368
 - error message 269, 270
 - extraction script 207
 - for MS Access 97 358
 - foreign key 388
 - from a CDM 388
 - from a PDM 391
 - inheritance 390
 - join index 571
 - PDM from database 381
 - PowerBuilder extended attribute 99
 - primary key 388
 - privileges 135
 - procedure 266
 - proxy table script 588
 - public names 99
 - quick launch 350
 - reference 177
 - script 368
 - select object 356
 - settings sets 350
 - stored procedures order 267
 - table 390
 - test data 361
 - trigger 266
 - XML model via XML Builder Wizard 393
 - generate automatic archive 10
 - generation
 - preserve horizontal partitioning 95
 - glop set 602
 - group 131
 - add user 140
 - assign role 141
 - check model 343
 - create 131
 - privilege 133
 - properties 131
- ## H
- hash index 601
 - hierarchy 202, 217
 - create 217
 - properties 218
 - HIGHGROUP 568
 - HIGHNONGROUP 568
 - horizontal partitioning
 - check model 328
 - create 87
 - MS SQL Server 2005 482
 - object 94
 - preserve during generation 95
 - properties 94
 - remove 98
 - HP Neoview
 - extensions 441
 - materialized view groups 444
- ## I
- IBM DB2
 - auxiliary table 450
 - database partition group 459
 - event monitor 463
 - event monitor group 463
 - index extension 460
 - security label 461
 - security label component 462
 - security policy 460
 - trusted context 449
 - web services 275
 - IBM DB2 for Common Server
 - extended attributes 453
 - IBM DB2 for z/OS
 - extensions 447
 - IDEFIX 12, 16
 - identifier 54
 - attribute 54
 - copy 48
 - create 55
 - delete 54
 - entity attribute 48, 55
 - generate key 388
 - link to attribute 60
 - properties 55
 - identity
 - column 103
 - column in Sybase ASE 580
 - Sybase 103
 - ignore identifying owner 20
 - import
 - CDM 429
 - deprecated PDM logical model 43
 - ERwin 429
 - interconnected PDM logical models 44
 - model 429

- web service as service provider 274
- INCOLN 423
- index 79, 123
 - alternate key 127
 - check model 328
 - clear 128
 - clustered 127
 - create 123
 - estimate database size 366
 - foreign key 127
 - generate 128
 - primary key 127
 - properties 127
 - query table 130
 - rebuild 128
 - type 568
 - unique 127
 - variable 404
- index extension 460
- Informix SQL
 - extensions 607
- Ingres
 - extensions 608
- inheritance 72
 - add child entity 76
 - check model 329
 - create 73
 - generate 74
 - generation mode 74
 - mutually exclusive 77
 - properties 74
- insert
 - trigger template 238
- Interbase
 - extensions 609
 - sequences 156
- IQ index rebuild 567
- J**
- Java class
 - link 164
 - reverse engineer 167
- join
 - change child attribute 61
 - change child column 173
 - create 60, 173
 - default 60, 173
 - migrate column 173
 - primary key 173
 - reuse column 173, 191
 - table collapsing 91
 - view reference 191
- JOIN 423
- join index 600
 - check model 330
 - generate 571
 - Oracle 569
 - Sybase IQ 569
 - variable 404
- K**
- key
 - add column 122
 - alternate 118
 - check model 331
 - create 119
 - foreign 118
 - link column 173
 - primary 118
 - properties 119
 - variable 404
- L**
- LDM
 - attribute 52
 - check model 309
 - create 6
 - domain 148
 - entity 47
 - identifier 54
 - inheritance 72
 - logical diagram 41
 - logical model 3
 - migration settings 15
 - model options 12
 - notation 21
 - relationship 56
- lifecycle 289
 - check model 332
 - create 290
 - data source wizard 296
 - external database 295, 296
 - generating data movement scripts 291
 - mapping editor 296

- phase 297
 - properties 293
- link
 - Java class 160
 - reverse Java class 160
- list
 - column 117
- logical diagram
 - attribute 52
 - create 43
 - entity 47
 - identifier 54
 - inheritance 72
 - objects 42
 - relationship 56
- logical model 3
- login policies 586
- login policy 564
- loop macro 404
- LOWFAST 568
- M**
- macro
 - CLIENTEXPRESSION 425
 - SERVEREXPRESSION 425
 - SQLXML 426
 - trigger 417
 - trigger template 417
- mandatory parent (model option) 18
- mapping
 - relational to relational mapping 207
- materialized query table 186
- materialized view 186
- materialized view groups 444
- materialized view log
 - Oracle 547
- measure 202, 216
 - create 216
 - properties 217
- member 210
- Merise 12
 - association 67
- message contract 502
- message type 500
- metadata variables 404
- migrate
 - ERwin model into CDM/PDM pair 430
 - migrate from ERwin 430, 432
- migration settings
 - LDM 15

- mirror servers 581
- model
 - copy DBMS 6
 - create 6
 - ERwin 429
 - import 429
 - open 11
 - option 63
 - preview code 397
 - properties 8
 - script 359
 - share DBMS 6
- model option 16
- modeling environment
 - customize 11
- MS Access
 - MS Access extensions 609
- MS SQL Server
 - Analysis Services 512
 - asymmetric key 490
 - certificate 488
 - CLR aggregate function 485
 - CLR assembly 484
 - CLR function 487
 - CLR integration 484
 - CLR procedure 487
 - CLR trigger 487
 - CLR user-defined type 486
 - contract 501
 - database mirroring 497
 - DSO metamodel 512
 - encryption 488
 - end point 499
 - event notification 505
 - extensions 467
 - full-text catalog 492
 - full-text index 493
 - full-text search 492
 - generate cubes 514, 518
 - message contract 502
 - message type 500
 - partition function 482
 - partition scheme 483
 - queue 503
 - remote service binding 508
 - Resource Governor 509
 - resource pool 510
 - reverse engineer cubes 515, 522
 - route 507

- schema 511
 - service 506
 - service broker 500
 - symmetric key 491
 - synonym 512
 - workload group 509
 - XML data type 495
 - XML index 494
 - XML schema collection 496
- MS SQL Server 2005
 - horizontal partitioning 482
- multidimensional diagram
 - association 218
 - attribute 213
 - basics 201
 - create 203
 - cube 203
 - dimension 212
 - fact 215
 - hierarchy 217
 - measure 216
 - retrieve objects 204
- multiple triggers 233
- multiplex server 563
- MySQL
 - extensions 610
- N**
- Netezza
 - extensions 612
- NMFCOL 424
- NonStop SQL
 - extensions 612
- normalization 87
- notation 12
 - Barker 21
- number
 - alternate key 122
- O**
- object
 - attach to user 132
 - owner 132
 - PDM 79
- object abstract data type 162
 - inherit procedure 162
- ODBC 381
- OOM
 - link 164
 - link Java class 160
- operation procedure 256
- option
 - data item 48
- Oracle
 - bitmap join index 535, 536
 - bitmap join index properties 536
 - cluster 545
 - database link 545
 - database package 537
 - database package template 542
 - extensions 529
 - join index 569
 - materialized view log 547
 - rebuild bitmap join index 535
 - rebuild table database package 543
 - sequences 156
 - transparent data encryption (TDE) 544
- order trigger 233
- owner 132, 136
- P**
- package
 - check model 333
- PAM opened into CDM 11
- parameter
 - modify database 370
- partition 300
 - add 94
 - corresponding table 94
 - create 300
 - delete 94
 - horizontal 87
 - manage 94
 - properties 301
 - vertical 89
- partition function 482
- partition scheme 483
- PBCatCol 99
- PBCatTbl 99
- PDM 4
 - archive 10
 - changing 26
 - check model 309
 - create 6
 - domain 148

- edit definition file 25
 - generate from 391
 - generate from database 381
 - object 79
 - save as 10
 - table 81
- PDM model options
 - notation 16
- permission
 - column 138
 - database objects 136
 - object owner 136
- phase 297
 - check model 332
 - create 297
 - properties 297
- physical diagram
 - abstract data type 160
 - column 103
 - create 81
 - default 146
 - define 79
 - index 123
 - reference 168
 - synonym 142
 - table 81
 - view 179
 - view reference 190
 - web parameter 280
- physical environment 289
- physical option 301
 - default 302
 - define 304
 - storage 304
 - tablespace 304
- PKCOLN 424
- PostgreSQL
 - extensions 613
- PowerBuilder
 - extended attribute 99
 - extensions 99
 - generate extended attributes 99
 - PBCatCol 99
 - PBCatTbl 99
 - reverse extended attributes 100
- precision 151
- preview
 - SQL 397
- preview code 397
- primary identifier
 - join 60
- primary index
 - Teradata 599
- primary key 79, 118
 - column 103, 120
 - generate from identifier 388
 - index 127
 - join 173
 - rebuild 120, 378
 - referential integrity 175
 - variable 423
- privilege
 - generate 135
 - grant 133
 - revoke 133
- procedure
 - attach to table 256
 - check model 335
 - custom 248
 - define 248
 - dependency 250
 - generate 266
 - inherit in abstract data type 162
 - OOM operation 256
 - permission 136
 - properties 250
 - SQL tools 401
 - stored 248
 - trigger template 233
 - variable 404
- procedure (database package) 539
- procedure dependencies 255
- procedure template 258, 260
 - create 259
 - modify 261
 - properties 261
- propagate column properties (model option) 18
- properties
 - horizontal partitioning 94
 - vertical partitioning 94
- proxy table
 - create 587
 - define remote server 588
 - generate script 588
 - Sybase ASA 586
 - Sybase ASE 552, 586

Q

- query
 - execute 372
 - performance 87
- query table
 - index 130
- queue 503
- quick launch 350

R

- rebuild
 - bitmap join index 535
 - cube 205
 - database package procedure dependencies 255
 - database package template 543
 - defaults 148
 - index 128
 - IQ index 567
 - primary key 378
 - primary keys 120
 - procedure dependencies 255
 - reference 177, 378
 - trigger dependencies 255
- rebuild automatically triggers 20
- rebuild triggers 226
 - create trigger automatically 222
 - template item 240
 - trigger template 240
- Red Brick Warehouse
 - extensions 617
- reference 79, 168
 - add to join index 572
 - auto-migration 169
 - cardinality 175
 - check model 336
 - create 168
 - delete 177
 - display preferences 178
 - generate 177
 - properties 172
 - rebuild 177, 378
 - reuse column 169
 - variable 404
 - view 183
- referential integrity 175, 222
- reflexive association 70
- relational 16

- relational to multidimensional
 - mapping 210
- relationship 56
 - associative entity 64
 - cardinality 58, 65
 - change to entity 64
 - check model 337
 - create 56
 - create an associative entity 64
 - define 64
 - entity attribute 64
 - example 65
 - option 63
 - properties 56
 - reflexive 63
 - unique code 63
- remote server 588
- remote service binding 508
- replication (PDM) 93
- replication group 603
- replication rule 603
- replication rule set 603
- Resource Governor 509
- resource pool 510
- result column
 - data type 280
 - definition 280
 - is element 280
- retrieve
 - multidimensional objects 204
 - WarehouseArchitect object 204
- reverse engineer
 - Access 97 383
 - administrator permissions 373
 - constraint 196
 - cubes in MS SQL Server 515
 - from a data source 375
 - from script files 373
 - from scripts 373
 - generate PDM from database 381
 - Java class 167
 - link Java class 160
 - optimization 384
 - options 378
 - PowerBuilder extended attributes 100
 - public names 100
 - script files order 373
 - shortcuts 382
 - variable 404

- reverse engineering
 - file encoding 380
 - statistics 384
- role 131
 - assign role 141
 - assign to a group 141
 - assign to a user 141
 - check model 343
 - create 131
 - privilege 133
 - properties 131
- route 507
- S**
- schema 131, 511
- script
 - alter 368
 - begin 359, 360
 - create database 358
 - create table 358
 - customize 358
 - database create 359
 - end 359, 360
 - extraction 207
 - generate 368
 - model 359
 - table 360
 - tablespace 360
- security label 461
- security label component 462
- security policy 460
- sequence 156
 - check model 338
 - create 156
 - properties 158
 - variable 404
- SERVEREXPRESSION 425
- service 506
- service broker 500
- service provider in web service import 274
- settings sets 350
- shortcuts
 - reverse engineering in PDM 382
- size
 - database 366
- snapshot 186
- SOAP 274
- spatial data 583
- spatial reference systems 583
- spatial units of measure 585
- SQL
 - editor 399
 - preview 397
- SQL Editor
 - tools 401
- SQL query 372
- SQL/XML wizard 262
- SQLBase column default 103
- SQLXML macro 426
- statistics 105
- storage 298
 - check model 342
 - create 299
 - define 304
 - not used 342
 - properties 299
 - select 304
- stored procedure
 - create 248
 - extended dependencies 267
 - generation order 267
- summary table 186
- Sybase AS IQ
 - add reference to join index 572
 - data movement script 572
 - generate join index 571
 - index type 568
 - rebuild IQ indexes 567
- Sybase ASA
 - auto-increment columns 580
 - define remote server of proxy 588
 - extensions 577
 - generate script for proxy tables 588
 - proxy tables 586
 - web services 274
- Sybase ASE
 - define remote server of proxy 588
 - encryption key 552
 - extensions 549
 - generate script for proxy tables 588
 - identity columns 580
 - proxy table 552
 - proxy tables 586
 - web services 274
- Sybase IQ
 - dbspace 562
 - events 560
 - extensions 555

- full text search 565
 - information lifecycle management 560
 - join index 569
 - login policy 564
 - multiplex server 563
 - text configuration 565
 - text index 566
 - web services 274
 - Sybase SQL Anywhere
 - events 560, 586
 - extensions 577
 - full text indexes 586
 - full text search 565
 - login policies 586
 - login policy 564
 - mirror servers 581
 - spatial data 583
 - spatial reference systems 583
 - spatial units of measure 585
 - text configuration 565
 - text index 566
 - symmetric key 491
 - synonym 142, 512
 - check model 338
 - create 143
 - create view 145
 - properties 145
 - System Administrator 131
 - system privilege 133
 - System Security Officer 131
- T**
- table 79, 81
 - alternate key 121
 - attach procedure 256
 - based on abstract data type 84
 - bitmap join index 536
 - check model 339
 - collapsing 91
 - constraint 86, 107, 108
 - create 82, 358
 - create trigger 221
 - default owner 20
 - display preferences 101
 - estimate database size 366
 - external 87
 - foreign key 122, 169
 - lifecycle 82
 - message 269
 - modify trigger 228
 - multidimensional type 204
 - permission 136
 - preview code 397
 - primary key 120
 - properties 82
 - script 360
 - statistics 384
 - variable 122, 404
 - view 183
 - XML type 85
 - table collapsing
 - check model 328
 - object 91
 - references 91
 - remove 98
 - wizard 91
 - table generation 390
 - tablespace 298
 - check model 342
 - create 299
 - database prefix in DB2 for z/OS 451
 - database prefix in DB2 OS/390 304
 - define 304
 - enabledbprefix 451
 - properties 299
 - script 360
 - select 304
 - TDE 544
 - template item 221, 241
 - add to trigger template 240
 - create from existing template item 242
 - create new 242
 - declare in trigger 231
 - define 241
 - identify 245
 - insert in trigger 229
 - insert in trigger template 229
 - modify 247
 - properties 247
 - rebuild triggers 240
 - teradata
 - error table 600
 - Teradata
 - extensions 591
 - glop set 602
 - hash index 601
 - join index 600

- primary index 599
- primary key 599
- replication group 603
- replication rule 603
- replication rule set 603
- transform group 598
- test data
 - generate 361
 - number of rows 361
 - triggers 361
- test data profile 109
 - assigning to column 113
 - properties 110
- text configuration 565
- text index 566
- TIME 568
- transform group
 - Teradata 598
- transformation
 - column denormalization 93
 - horizontal partitioning 87
 - table collapsing 91
 - vertical partitioning 89
- transparent data encryption (TDE) 544
- trigger 221
 - check model 342
 - create automatically 222
 - create from template 222
 - create manually 221
 - declare template item 231
 - dependency 250
 - edit 228
 - function insert 228
 - generate 266
 - insert template item 229
 - macro 417
 - macro insert 228
 - modify 228
 - modify from table 228
 - multiple 233
 - multiple events 234
 - name convention 232
 - operator insert 228
 - order 233
 - properties 224
 - referential integrity 222
 - same type 233
 - SQL tools 401
 - test data 361

- user-defined 221
 - variable 404
 - variable insert 228
- trigger dependencies 255
- trigger template 221, 235
 - add template item 240
 - create 236
 - declare template item 231
 - delete 238
 - insert 238
 - insert template item 229
 - macro 417
 - modify 240
 - name convention 232
 - procedure 233
 - properties 240
 - rebuild triggers 240
 - type 238
 - update 238
- triggers
 - rebuild triggers 226
- trusted context
 - IBM DB2 449

U

- undefined data type 115, 149
- unique
 - index 127
- unique code (model option) 18
- user 131
 - add to group 140
 - assign role 141
 - attach object 132
 - check model 343
 - create 131
 - permission 136
 - privilege 133
 - properties 131
- user-defined
 - generate 270

V

- validation rule
 - auto-migrate 169
- variable 404
 - alternate key 122

- column 419
- foreign key 423
- format 416
- primary key 423
- table 122
- variable (database package) 539
- vertical partitioning 89
 - check model 328
 - create 89
 - properties 94
 - remove 98
- view 79, 179
 - check model 339, 344
 - columns 182
 - create 179
 - create from synonym 145
 - default owner 20
 - define query 183
 - extended dependencies 187
 - generation order 187
 - materialized query 186
 - modify column properties 182
 - multidimensional type 204
 - permission 136
 - properties 180
 - reference 183
 - select table 179
 - select view 179
 - table 183
 - variable 404
 - XML type 85
- view index
 - check model 328
- view reference 190
 - check model 336
 - create 190
 - join 191
 - properties 190

W

WarehouseArchitect retrieve object 204

- web operation 277
 - create 277
 - properties 277
- Web operation 79
 - check model 344
- web parameter 280
 - create 280
 - properties 280
- web service
 - create 275
 - DADX extension 282
 - DADX files 275
 - general definition 273
 - generate web services for IBM DB2 282
 - import as service provider 274
 - prefix name 274
 - properties 275
 - reverse engineer web services 285
 - types 274
- Web service 79, 274
 - check model 344
 - generate 282
- with default (column properties) 103
- workload group 509
- WSDL 274

X

- xem 27
- XML
 - data type 495
 - index 494
 - table 85
 - view 85
 - XML schema collection 496
- XML Builder Wizard 393
- XML index 494
- XML schema collection 496