ERwin

Reference Guide

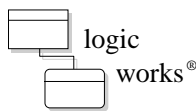*ERwin Version 3.0*

Reference Guide

Logic Works, Inc.

University Square at Princeton

111 Campus Drive

Princeton, NJ 08540

logic
works®

Contents

Erwin Reference Guide

Preface

ERwin is a data modeling tool that helps database developers in many different environments. Because of its capabilities to automate programming tasks and forward and reverse engineer databases, there are considerations that include datatypes, macros and model storage that a data modeler must address. This manual provides the valuable reference information to help get your job done.

About this Guide

This guide contains six chapters and an index:

♦ **Chapter 1** is an introduction to ODBC. It also describes how to set up an ODBC data source.

♦ **Chapter 2** describes the tables and columns in the ERwin metamodel and their valid values.

♦ **Chapter 3** explains how ERwin converts datatypes when you change target servers.

♦ **Chapter 4** is a complete reference to all ERwin macro functions.

♦ **Chapter 5** provides summary tables of the many schema generation options for all supported target servers. There is also information regarding the forward and reverse engineering of storage objects, indexes, and validation rules.

♦ **Chapter 6** is a glossary of all ERwin terms and definitions.

Related Documentation

The Logic Works ERwin documentation set includes the following manuals:

♦ *ERwin Online Help*

♦ *ERwin Reference Guide (this guide)*

♦ *ERwin Methods Guide*

♦ *ERwin Features Guide*

♦ *ERwin Tutorial Guide*

♦ *Logic Works RPTwin Users Guide*

♦ *ERwin Navigator Users Guide*

1

Defining ODBC Data Sources

Defining ODBC Data Sources

ERwin uses ODBC (Open Database Connectivity) software to access PC databases. ODBC divides data access tasks between a *database driver* and a *data source.* An ODBC database driver is a dynamic link library that communicates requests to a particular database management system, such as Access, Paradox, or dBASE. An ODBC data source is a stored specification that associates a particular driver with the database files that you want to access with that driver, such as Access .MDB files or dBASE .DBF files.

In order to connect a specific PC database to ERwin, it must be defined as an ODBC data source using the ODBC Administrator program. The ODBC Administrator comes with Microsoft Access, FoxPro, Excel, Word for Windows, and other software packages that support ODBC drivers for data access. The ODBC Administrator method is explained below. For information on using client tools like PowerBuilder to configure ODBC data sources, see the documentation that comes with your client development tools, such as PowerBuilder's ***Connecting to the Database Guide.***

Note: The ODBC Administrator is available in 16-bit and 32-bit versions. Use the appropriate version for your operating system environment.

The following example describes setting up an ODBC data source for a Microsoft Access database.

To define an ODBC data source

1. Open the Windows Control Panel in Windows 3.11, Windows NT or Windows 95.

2. Double-click on the ![ODBC icon] ODBC icon to start the ODBC Administrator program. The current list of data sources is displayed in the Data Sources dialog.
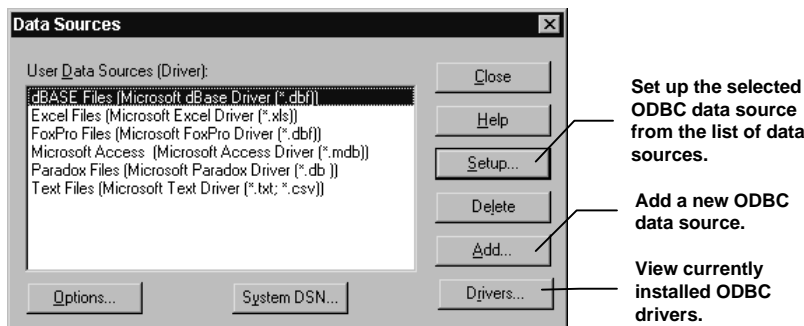
Data Sources Dialog Showing Previously Defined Data Sources

3. Click the **Add** button to open the Add Data Source dialog.
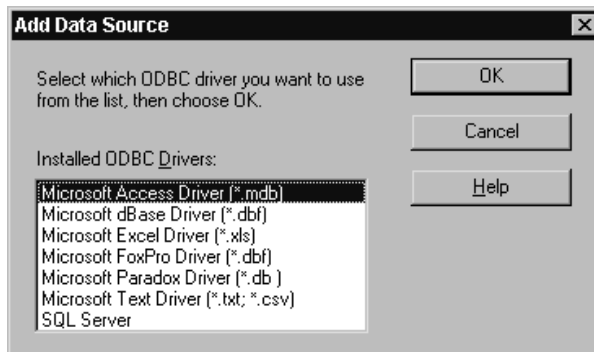
Add Data Source Dialog

4. Choose the appropriate ODBC driver for your database and click **OK**. The displayed list includes ODBC drivers previously installed for other database applications.

The ODBC driver setup dialog is displayed.

ODBC Microsoft Access 7.0 Setup

Data Source Name:	OK
Description:	Cancel
Database	Help
Database:	Advanced...
Select... Create... Repair... Compact...	
System Database	
● None	
○ Database:	
System Database...	Options>>

Complete the ODBC data source name.

Complete the ODBC data source description.

Use to select the physical database.
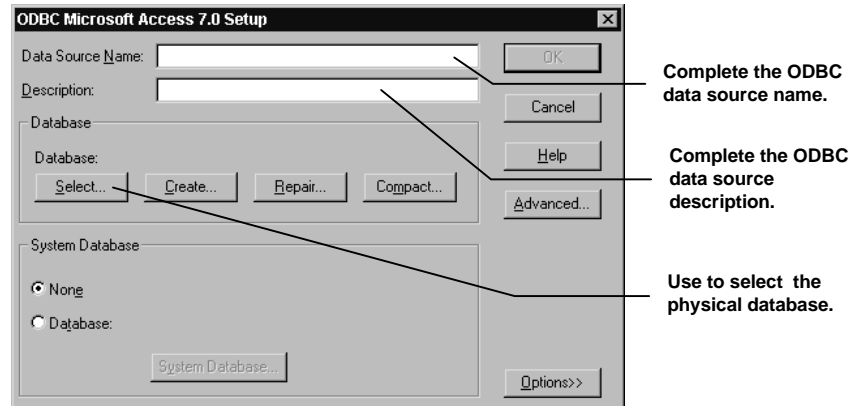
ODBC Microsoft Access 7.0 Setup Dialog

5. Complete the Data Source Name and Description text boxes by typing a data source name and description, respectively. Because each driver requires slightly different information, click the **Help** button for instructions on how to fill in the setup for a specific data source.

6. Click the **Select** button to select the physical database to which you want the data source to connect. The Select Database dialog is displayed.

Select Database

Database Name	Directories:	OK
Your Database.mdb	C:\Your Application	Cancel
Your Database.mdb	c:\ / Your Application	Help
		□ Read Only
		□ Exclusive
List Files of Type:	Drives:	
Access Databases (*.mc ▼)	c: ms-dos_6 ▼	Network...
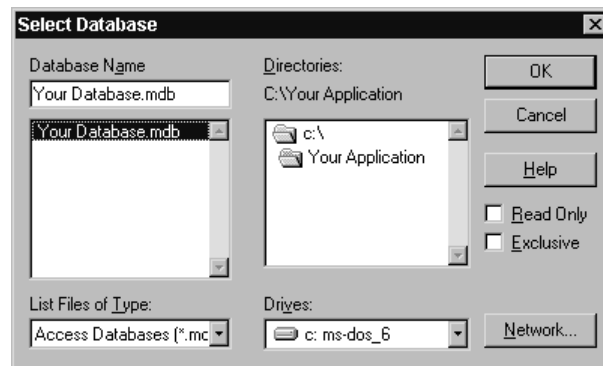
Select Database Dialog

7. Locate your physical database and click the **OK** button.

8. From the ODBC Setup dialog, verify that the selected table and path is correct.
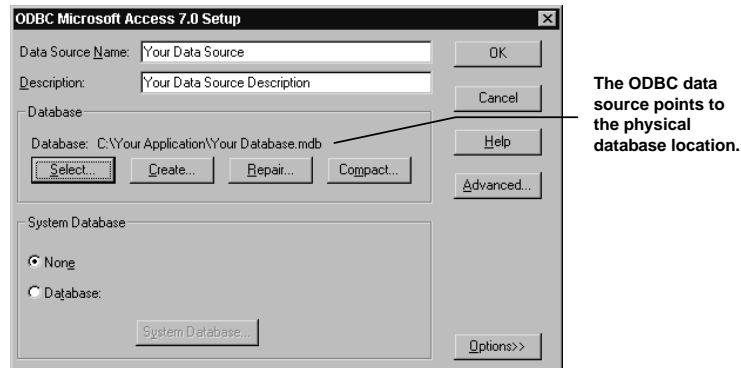
The ODBC Data Source Pointing to Your Database

9. Click the **OK** button to return to the Data Sources dialog.

10. If you want to assign an optional logon name and password, click the **Advanced** button and type the Logon Name and Password in the respective columns. You will need them later when you need to access the data source.

11. Click the **Close** button to close the Data Sources dialog. The data source is established and can be accessed by ERwin and your application.

2 Working with the ERwin Metamodel

Understanding the ERwin Metamodel

A complete ERwin data model contains all the information that is required to generate a physical database on your selected target server. Similarly, all the internal definition information that ERwin requires to draw a data model is itself contained in a special ERwin data model called the *ERwin Dictionary Metamodel.*

You can generated the ERwin metamodel to create a *data dictionary* that it stores information about the data structures used in ERwin models.

This chapter describes each entity in the ERwin metamodel in alphabetical order and lists the name, datatype description, valid values, and foreign key property of each of their attributes in a separate table. The metamodel entity information is followed by tables that list valid values for various attributes in the metamodel.

Each table includes the name of all attributes included in that entity, in the order in which they appear in the metamodel. Columns in the table include:

♦ **Attribute Name** as it appears in the logical model.

♦ **Datatype** lists the default datatype assigned to the table column. The actual datatype used is dependent on which target server you use to hold the ERwin dictionary.

♦ **Description** indicates the type of information stored by the attribute.

♦ **Valid Values** lists the valid values, the reference table that contains the valid value list, or validation expression for the attribute. Where no validation rule has been defined, the data must conform to the attribute's datatype.

♦ **FK** indicates if the attribute is a foreign key (FK) attribute. This column is blank for non-foreign key attributes.

The gray line in each table separates the key attributes (above the line) from the non-key attributes (below the line) for each entity. For entities that have only a single attribute, that attribute is a key field.

Default

This entity stores default values that can be attached to a column or a domain.

Attribute Name	Datatype	Description	Valid Values	FK
DEFAULT ID	integer	Stores the ID number of a default value that is assigned to attributes by a domain.	1 to 2^{32}	FK
DEFAULT TYPE	char(6)	Stores a string that is the type code for the DEFAULT entity. Reserved for internal use.	DEFVAL	FK
ERWIN DIAGRAM ID	integer	Stores the ID number of a diagram.	1 to 2^{32}	FK
RESERVED	integer	Reserved for internal use.	NA	FK
DEFAULT NAME	varchar(254)	Stores the name associated with a default value.	Any that meets datatype criteria.	
CLIENT DEFAULT VALUE	integer	Stores a number that specifies the client part of a default value.	1 to 2^{32}	FK
SERVER DEFAULT VALUE	integer	Stores a number that specifies the server part of a default value.	1 to 2^{32}	FK
DEFAULT FLAG	integer	Stores a numeric flag used internally to indicate the default type; for a SYBASE target, this flag specifies whether the default is generated as "sp_bind" or "DEFAULT stmt".	1 to 2^{32}	

Dictionary

This entity stores a text value that identifies the version of the metamodel that was used to generate the ERwin dictionary.

Attribute Name	Datatype	Description	Valid Values	FK
VERSION	varchar(18)	Stores a string that specifies the ERwin version to which the metamodel applies.	ERwin/ERX 1.1, ERwin/ERX 2.0, ERwin/ERX 2.5, etc.	

Domain

This entity stores information about global domains for attributes that specify properties such as validation rules, defaults, and datatypes.

Attribute Name	Datatype	Description	Valid Values	FK
DOMAIN ID	integer	Stores the unique ID number of a domain.	1 to 2^{32}	FK
DOMAIN TYPE	char(6)	Stores a string type code for a domain.	DOMAIN	FK
ERWIN DIAGRAM ID	integer	Stores the ID number of a diagram.	1 to 2^{32}	FK
RESERVED	integer	Reserved for internal use.	NA	FK
DOMAIN NAME	varchar(254)	Stores a domain's name, which is used to associate the domain with attributes.	Any that meets datatype criteria.	
PARENT DOMAIN ID	integer	Stores the ID number of the parent domain in the domain hierarchy.	1 to 2^{32}	FK
DOMAIN DEFINITION	integer	Stores the ID number of the text description of a domain.	1 to 2^{32}	FK
DOMAIN FLAGS	integer	Stores an internal numeric value that generates the domain as a user-defined datatype.	1 to 2^{32}	
DOMAIN INTERNAL	integer	Stores an internal numeric value that specifies an internal domain type.	1 to 2^{32}	

Domain PV

This entity stores the domain property values, such as physical datatype and validation rules.

Attribute Name	Datatype	Description	Valid Values	FK
DOMAIN ID	integer	Stores the unique ID number of a domain.	1 to 2^{32}	FK
DOMAIN TYPE	char(6)	Stores a string type code for a domain.	DOMAIN	FK
ERWIN DIAGRAM ID	integer	Stores the ID number of a diagram.	1 to 2^{32}	FK
RESERVED	integer	Reserved for internal use.	NA	FK
TARGET SYSTEM	char(6)	Stores a string that specifies a physical DBMS.	See Table A – Target System Valid Values later in this chapter.	FK
DOMAIN PROPERTY SEQ	integer	Stores a number that specifies the ordered position of a domain property.	1 to 2^{32}	FK
DOMAIN PROPERTY TYPE	char(6)	Stores a string that identifies a domain property's type (e.g., attribute name, attribute datatype).	See Table B – Column Domain Property Types later in this chapter.	FK
DOMAIN PROPERTY TVAL	integer	Stores the ID number of a domain property's text value; this value is null if the property is not a text property.	See Table B – Column Domain Property Types later in this chapter.	
DOMAIN PROPERTY IVAL	integer	Stores a domain property's integer value; this value is null if the domain property is not an integer.	See Table B – Column Domain Property Types later in this chapter.	FK
DOMAIN PROPERTY IDVL	integer	Stores the ID number of a domain property's object; this value is null if the domain property is not a reference to a validation rule or a default.	See Table B – Column Domain Property Types later in this chapter.	
DOMAIN PROPERTY SVAL	varchar(254)	Stores a domain-property's string value; this value is null if the domain property is not a string.	See Table B – Column Domain Property Types later in this chapter.	
DOMAIN PROPERTY INHERIT	integer	Stores a numeric flag that specifies whether a domain property is inherited from the parent domain.	0 (no), 1 (yes)	

Entity Index

This entity stores primary, alternate, and inversion entry indexes.

Attribute Name	Datatype	Description	Valid Values	FK
INDEX ID	integer	Stores the ID number of an index in an entity; an index ID can overlap across entities, but must be unique within an entity.	1 to 2^{32}	
INDEX TYPE	char(6)	Stores a string that specifies the index type (i.e., primary key, alternate key, inversion entries, or foreign key); primary and alternate key indexes are unique, inversion entries and foreign keys are not.	See Table C – Index Type Valid Values later in this chapter.	
ENTITY ID	integer	Stores the unique ID number of an entity.	1 to 2^{32}	FK
ENTITY TYPE	char(6)	Stores a string that specifies whether an entity is independent or dependent.	See Table D – Entity Type Valid Values later in this chapter.	FK
ERWIN DIAGRAM ID	integer	Stores the ID number of a diagram.	1 to 2^{32}	FK
RESERVED	integer	Reserved for internal use.	NA	FK
GENERATE	integer	Stores the state of the Generate option in the Index Editor.	1 to 2^{32}	
IDX_FLAGS	integer	Internal Integer Value	1 to 2^{32}	

Entity Physical PV

This entity stores the DBMS-specific physical parameters and values associated with the table that is used to store data for an entity.

Attribute Name	Datatype	Description	Valid Values	FK
ENTITY ID	integer	Stores the unique ID number of an entity.	1 to 2^{32}	FK
ENTITY TYPE	char(6)	Stores a string that specifies whether an entity is independent or dependent.	See Table D – Entity Type Valid Values later in this chapter.	FK
ERWIN DIAGRAM ID	integer	Stores the ID number of a diagram.	1 to 2^{32}	FK
RESERVED	integer	Reserved for internal use.	NA	FK
TARGET SYSTEM	char(6)	Stores a string that specifies a physical DBMS.	See Table A – Target System Valid Values later in this chapter.	FK
PHYS PROP TYPE	char(6)	Stores a string that specifies the physical property assigned to an entity (e.g., TABLESPACE, PCTFREE).	See Table E – Physical Property Codes later in this chapter.	FK
PHYS PROP VALUE	integer	Stores a number that specifies the value of the physical property.	1 to 2^{32}	FK

Entity Stored Procedure Template

This entity stores the information about stored procedure templates associated with a specific entity.

Attribute Name	Datatype	Description	Valid Values	FK
ENTITY ID	integer	Stores the unique ID number of an entity.	1 to 2^{32}	FK
ENTITY TYPE	char(6)	Stores a string that specifies whether an entity is independent or dependent.	See Table D – Entity Type Valid Values later in this chapter.	FK
TEMPLATE ID	integer	Stores the ID number of a template.	1 to 2^{32}	FK
TEMPLATE TYPE	char(6)	Stores a string that specifies whether the template is built-in or user-created.	See Table F – Template Types later in this chapter.	FK
ERWIN DIAGRAM ID	integer	Stores the ID number of a diagram.	1 to 2^{32}	FK
RESERVED	integer	Reserved for internal use.	NA	FK
TARGET SYSTEM	char(6)	Stores a string that specifies a physical DBMS.	See Table A – Target System Valid Values later in this chapter.	FK

Entity Synonym

This entity stores information about ERwin entity and aliases, which are user-defined table names that can be used to reference the table in relationships (Red Brick synonyms only) or stored procedures and triggers (DB2/MVS and DB2/2 only).

Attribute Name	Datatype	Description	Valid Values	FK
SYNONYM NAME	varchar(256)	Stores the string that stores the synonym name.	Any that meets datatype criteria.	
ENTITY ID	integer	Stores the unique ID number of an entity.	1 to 2^{32}	FK
ENTITY TYPE	char(6)	Stores a string that specifies whether an entity is independent or dependent.	See Table D – Entity Type Valid Values later in this chapter.	FK
ERWIN DIAGRAM ID	integer	Stores the ID number of a diagram.	1 to 2^{32}	FK
RESERVED	integer	Reserved for internal use.	NA	FK

Entity Trigger

This entity stores information about the trigger code associated with a specific entity.

Attribute Name	Datatype	Description	Valid Values	FK
ENTITY ID	integer	Stores the unique ID number of an entity.	1 to 2^{32}	FK
ENTITY TYPE	char(6)	Stores a string that specifies whether an entity is independent or dependent.	See Table D – Entity Type Valid Values later in this chapter.	FK
ERWIN DIAGRAM ID	integer	Stores the ID number of a diagram.	1 to 2^{32}	FK
RESERVED	integer	Reserved for internal use.	NA	FK
TARGET SYSTEM	char(6)	Stores a string that specifies a physical DBMS.	See Table A – Target System Valid Values later in this chapter.	FK
TRIGGER NAME	varchar(254)	Stores the name used to create and refer to a trigger in the database.	Any that meets datatype criteria.	
TRIGGER TYPE	char(6)	Stores a string that specifies whether or not the trigger is for insert, update, or delete.	See Table G – Trigger Types later in this chapter.	FK
TRIGGER DISABLED	integer	Stores a number that specifies whether the trigger is disabled.	1 to 2^{32}	
TRIGGER FIRE	char(6)	Stores a string that specifies whether a trigger is to fire before or after the transaction completes; does not apply to all databases.	See Table H – Trigger Fire Valid Values later in this chapter.	
TRIGGER SCOPE	char(6)	Stores a string that specifies whether the trigger is row or table scope; does not apply to all databases.	See Table I – Trigger Scope Valid Values later in this chapter.	
TRIGGER OLD	varchar(254)	Stores the name used to access old rows.	Any that meets datatype criteria.	
TRIGGER NEW	varchar(254)	Stores the name used to refer to new rows.	Any that meets datatype criteria.	
TRIGGER BODY	integer	Stores the ID number of a body of trigger code, which may contain ERwin macros.	1 to 2^{32}	FK

ERwin Color

This entity stores a table of colors used by the diagram.

Attribute Name	Datatype	Description	Valid Values	FK
COLOR ID	integer	Stores a number that specifies a color in an ERwin diagram.	1 to 2^{32}	FK
COLOR TYPE	char(6)	Stores the TX_COL string as a placeholder for future screen and print color implementations.	TX_COL	FK
ERWIN DIAGRAM ID	integer	Stores the ID number of a diagram.	1 to 2^{32}	FK
RESERVED	integer	Reserved for internal use.	NA	FK
COLOR VALUE	integer	Stores a number that specifies a color.	1 to 2^{32}	

ERwin Diagram

This entity stores the list of current ERwin diagrams.

Attribute Name	Datatype	Description	Valid Values	FK
ERWIN DIAGRAM ID	integer	Stores the ID number of a diagram.	1 to 2^{32}	
ERWIN DIAGRAM NAME	varchar(254)	Stores the name of a diagram.	Any that meets datatype criteria.	
ERWIN VERSION NUMBER	integer	Stores the diagram version number.	1 to 2^{32}	
MODIFICATION DATE	integer	Stores a binary number that specifies the date that the diagram was last updated.	1 to 2^{32}	
ENTITY COUNT	integer	Stores the number of entities in the diagram.	1 to 2^{32}	
CURRENT USER	varchar(254)	Stores a string that is the unique ID of the user that currently has the diagram checked out.	Any that meets datatype criteria.	
CHKIN_USER	varchar(254)	Stores the name of the user who checked in this version of the diagram.	Any that meets datatype criteria.	
CHKIN_NOTES	varchar(254)	Stores note text pertaining to this version of the diagram.	Any that meets datatype criteria.	

ERwin Diagram Option

This entity stores the state of all ERwin options and option values for a saved diagram.

Attribute Name	Datatype	Description	Valid Values	FK
OPTION NAME	char(6)	Stores a string that specifies an option (such as a display option) that can be associated with a diagram, a subject area, or a stored display.	See Table J – Diagram Options and Table K – Display Options later in this chapter.	
ERWIN DIAGRAM ID	integer	Stores the ID number of a diagram.	1 to 2^{32}	FK
SUBJECT ID	integer	Stores the ID number of a subject area.	1 to 2^{32}	FK
DISPLAY ID	integer	Stores the ID number of a stored display.	1 to 2^{32}	FK
OPTION VALUE	varchar(254)	Stores a string that specifies the value of a diagram option.	Any that meets datatype criteria.	

ERwin Diagram View Object

This entity stores the graphical coordinates, font, and color information for each diagram object.

Attribute Name	Datatype	Description	Valid Values	FK
SUBJECT ID	integer	Stores the ID number of a subject area.	1 to 2^{32}	FK
DISPLAY ID	integer	Stores the ID number of a stored display.	1 to 2^{32}	FK
OBJECT ID	integer	Stores the ID number of an object, which overlaps across types.	1 to 2^{32}	FK
OBJECT TYPE	char(6)	Stores a string that specifies an object's type (e.g., entity, text block, relationship).	Superset of all object types, including entities, attributes, relationships, etc.	FK
ERWIN DIAGRAM ID	integer	Stores the ID number of a diagram.	1 to 2^{32}	FK
RESERVED	integer	Reserved for internal use.	NA	FK
XCOORD1	integer	Stores an object's graphical X coordinate #1.	1 to 2^{32}	
YCOORD1	integer	Stores an object's graphical Y coordinate #1.	1 to 2^{32}	
XCOORD2	integer	Stores an object's graphical X coordinate #2.	1 to 2^{32}	
YCOORD2	integer	Stores an object's graphical Y coordinate #2.	1 to 2^{32}	
EDEF FONT ID	integer	Stores a number that specifies the entity-definition font if the object is an entity.	1 to 2^{32}	FK
EDEF COLOR ID	integer	Stores a number that specifies the entity-definition color if the object is an entity.	1 to 2^{32}	FK
ELINE COLOR ID	integer	Stores a number that specifies the entity-line color if the object is an entity.	1 to 2^{32}	FK
EFILL COLOR ID	integer	Stores a number that specifies the entity-fill color if the object is an entity.	1 to 2^{32}	FK
ENRN FONT ID	integer	Stores a number that specifies the entity-name font, if the object is an entity, or the relationship-name font, if it is a relationship.	1 to 2^{32}	FK
ENR COLOR ID	integer	Stores a number that specifies the entity-name color, if the object is an entity, or the relationship color, if it is a relationship.	1 to 2^{32}	FK
COLOR TYPE	char(6)	Stores the TX_COL string as a placeholder for future screen and print color implementations.	TX_COL	FK
FONT TYPE	char(6)	Stores the TX_FNT string as a placeholder for future screen and print font implementations.	TX_FNT	FK
USER TOUCHED	integer	Stores a number that specifies whether to use calculated or stored relationship lines.	1 to 2^{32}	
PATH0 X	integer	Stores the graphical X coordinate of stored relationship line segment 0.	1 to 2^{32}	
PATH0 Y	integer	Stores the graphical Y coordinate of stored relationship line segment 0.	1 to 2^{32}	
PATH1 X	integer	Stores the graphical X coordinate of stored relationship line segment 1.	1 to 2^{32}	
PATH1 Y	integer	Stores the graphical Y coordinate of stored relationship line segment 1.	1 to 2^{32}	

Summary of ERwin Diagram View Object (continued)

Attribute Name	Datatype	Description	Valid Values	FK
PATH2 X	integer	Stores the graphical X coordinate of stored relationship line segment 2.	1 to 2^{32}	
PATH2 Y	integer	Stores the graphical Y coordinate of stored relationship line segment 2.	1 to 2^{32}	
PATH3 X	integer	Stores the graphical X coordinate of stored relationship line segment 3.	1 to 2^{32}	
PATH3 Y	integer	Stores the graphical Y coordinate of stored relationship line segment 3.	1 to 2^{32}	
PATH4 X	integer	Stores the graphical X coordinate of stored relationship line segment 4.	1 to 2^{32}	
PATH4 Y	integer	Stores the graphical Y coordinate of stored relationship line segment 4.	1 to 2^{32}	
VERB1 X	integer	Stores the graphical X coordinate of parent verb phrase.	1 to 2^{32}	
VERB1 Y	integer	Stores the graphical Y coordinate of parent verb phrase.	1 to 2^{32}	
VERB2 X	integer	Stores the graphical X coordinate of child verb phrase.	1 to 2^{32}	
VERB2 Y	integer	Stores the graphical Y coordinate of child verb phrase.	1 to 2^{32}	
MID X	integer	Stores the graphical X coordinate of the relationship line midpoint.	1 to 2^{32}	
MID Y	integer	Stores the graphical Y coordinate of the relationship line midpoint.	1 to 2^{32}	

ERwin Entity

This entity stores information about ERwin entities.

Attribute Name	Datatype	Description	Valid Values	FK
ENTITY ID	integer	Stores the unique ID number of an entity.	1 to 2^{32}	FK
ENTITY TYPE	char(6)	Stores a string that specifies whether an entity is independent or dependent.	See Table D – Entity Type Valid Values later in this chapter.	FK
ERWIN DIAGRAM ID	integer	Stores the ID number of a diagram.	1 to 2^{32}	FK
RESERVED	integer	Reserved for internal use.	NA	FK
ENTITY NAME	varchar(254)	Stores the name of an entity; this text value should be unique within the model, however ERwin does allow duplicate names, depending on the Unique Name setting (in the Option menu).	Unique within model.	
ENTITY NOTE	integer	Stores the ID number of comment text that is associated with an entity.	1 to 2^{32}	FK
ENTITY DEFINITION	integer	Stores the ID number of the entity definition text in the definition-level display.	1 to 2^{32}	FK
ENTITY ICON	varchar(254)	Stores the complete pathname of the bitmap (.BMP) used to depict an entity in icon-level display. (e.g. c:\erw\cust.bmp)	Any that meets datatype criteria.	
ENT FLAGS	integer	Internal integer value	1 to 2^{32}	

ERwin Entity-Attribute Usage

This entity stores information about the usage of an attribute in an entity;. Although we do not show the subtype in this ERX storage model, all attributes are owned or foreign. Owned attribute names are unique within an entity (but need not be across a diagram).

Attribute Name	Datatype	Description	Valid Values	FK
ATTRIBUTE ID	integer	Stores the unique ID number for each attribute.	1 to 2^{32}	FK
RELATIONSHIP ID	integer	Stores the unique ID number of a relationship.	1 to 2^{32}	FK
RELATIONSHIP TYPE	char(6)	Stores a string that specifies whether a relationship is ID or non-ID	See Table L – Relationship Type Valid Values later in this chapter.	FK
ENTITY-ATTRIBUTE BASE NAME	varchar(254)	Stores the base name (other than the rolename) of an attribute.	Any that meets datatype criteria.	
ENTITY ID	integer	Stores the unique ID number of an entity.	1 to 2^{32}	FK
ENTITY TYPE	char(6)	Stores a string that specifies whether an entity is independent or dependent.	See Table D – Entity Type Valid Values later in this chapter.	FK
ERWIN DIAGRAM ID	integer	Stores the ID number of a diagram.	1 to 2^{32}	FK
RESERVED	integer	Reserved for internal use.	NA	FK
COLUMN POSITION	integer	Stores a number that specifies the ordered position of an attribute used in an entity at the logical level; column position is determined at the logical level by the position of the attribute in the attribute list of the entity.	1 to 2^{32}	
ATTRIBUTE TYPE	char(6)	Stores a string that specifies whether an attribute is a primary key or a non key.	See Table M – Attribute Type Codes later in this chapter.	FK
ENTITY-ATTRIBUTE ROLENAME	varchar(254)	Stores an attribute rolename, which is used primarily to indicate the attribute's different role as a foreign key; if it exists, the rolename (not the basename) migrates.	Any that meets datatype criteria.	
ENTITY-ATTRIBUTE PHYSICAL ORDER	integer	Stores a number that specifies the ordered position of a physical column in a table as distinct from its logical position; while primary keys must always precede non-keys at the logical level, this is not necessary at the physical level.	1 to 2^{32}	
DOMAIN ID	integer	Stores the unique ID number of a domain.	1 to 2^{32}	FK
COLOR ID	integer	Stores a number that specifies a color in an ERwin diagram.	1 to 2^{32}	FK
FONT ID	integer	Stores a number that identifies a font-specification code in the Windows LOGFONT API.	1 to 2^{32}	FK
ENTITY-ATTRIBUTE NOTE	integer	Stores the ID number of an attribute's comment text.	1 to 2^{32}	FK
ENTITY-ATTRIBUTE DEFINITION	integer	Stores the ID number of an attribute's definition text.	1 to 2^{32}	FK

ERwin Font

This entity stores the font information used by the diagram.

Attribute Name	Datatype	Description	Valid Values	FK
FONT ID	integer	Stores a number that identifies a font-specification code in the Windows LOGFONT API.	1 to 2^{32}	FK
FONT TYPE	char(6)	Stores the TX_FNT string as a placeholder for future screen and print font implementations.	TX_FNT	FK
ERWIN DIAGRAM ID	integer	Stores the ID number of a diagram.	1 to 2^{32}	FK
RESERVED	integer	Reserved for internal use.	NA	FK
FONT HEIGHT	integer	Stores a number that identifies a font-specification code in the Windows LOGFONT API.	1 to 2^{32}	
FONT WIDTH	integer	Stores a number that identifies a font-specification code in the Windows LOGFONT API.	1 to 2^{32}	
FONT ESCAPEMENT	integer	Stores a number that identifies a font-specification code in the Windows LOGFONT API.	1 to 2^{32}	
FONT ORIENTATION	integer	Stores a number that identifies a font-specification code in the Windows LOGFONT API.	1 to 2^{32}	
FONT WEIGHT	integer	Stores a number that identifies a font-specification code in the Windows LOGFONT API.	1 to 2^{32}	
FONT ITALIC	integer	Stores a Boolean value that specifies whether text is italicized.	0 (no), 1 (yes)	
FONT UNDERLINE	integer	Stores a Boolean value that specifies whether text is underlined.	0 (no), 1 (yes)	
FONT STRIKEOUT	integer	Stores a number that identifies a font-specification code in the Windows LOGFONT API.	1 to 2^{32}	
FONT CHARSET	integer	Stores a number that identifies a font-specification code in the Windows LOGFONT API.	1 to 2^{32}	
FONT OUTPRECISION	integer	Stores a number that identifies a font-specification code in the Windows LOGFONT API.	1 to 2^{32}	
FONT CLIPPRECISION	integer	Stores a number that identifies a font-specification code in the Windows LOGFONT API.	1 to 2^{32}	
FONT QUALITY	integer	Stores a number that identifies a font-specification code in the Windows LOGFONT API.	1 to 2^{32}	
FONT PITCHANDFAMILY	integer	Stores a number that identifies a font-specification code in the Windows LOGFONT API.	1 to 2^{32}	
FONT NAME	varchar(254)	Stores the name of a font; identifies a specifications in the Windows LOGFONT API.	Any that meets datatype criteria.	
OLDFONT HEIGHT	integer	Stores a number that identifies a font-specification code in the Windows LOGFONT API.	1 to 2^{32}	

ERwin Relationship

This entity stores information about the relationships between parent and child entities and the migration paths for foreign keys.

Attribute Name	Datatype	Description	Valid Values	FK
RELATIONSHIP ID	integer	Stores the unique ID number of a relationship.	1 to 2^{32}	FK
RELATIONSHIP TYPE	char(6)	Stores a string that specifies whether a relationship is ID or non-ID.	See Table L – Relationship Type Valid Values later in this chapter.	FK
ERWIN DIAGRAM ID	integer	Stores the ID number of a diagram.	1 to 2^{32}	FK
RESERVED	integer	Reserved for internal use.	NA	FK
PARENT ENTITY	integer	Stores the ID number of a parent entity or subtype.	1 to 2^{32}	FK
PARENT TYPE	char(6)	Stores a string that specifies whether the parent is an independent or dependent entity or a complete or incomplete subtype.	See Table N – Entity Type Valid Values later in this chapter.	FK
CHILD ENTITY	integer	Stores the unique ID number of a child entity or subtype.	1 to 2^{32}	FK
CHILD TYPE	char(6)	Stores a string that specifies whether the child is an independent or dependent entity or a complete or incomplete subtype.	See Table N – Entity Type Valid Values later in this chapter.	FK
VERB PHRASE	varchar(254)	Stores the descriptive parent-child verb phrase labeling a relationship.	Any that meets datatype criteria.	
CARDINALITY	char(6)	Stores a string that specifies the cardinality type for a relationship.	See Table O – Relationship Cardinality Valid Values later in this chapter.	
PARENT DELETE RULE	char(6)	Stores a string that specifies the referential integrity rule that governs a child FK when a parent PK is deleted.	See Table P – Parent Delete RI Valid Values later in this chapter.	
PARENT UPDATE RULE	char(6)	Stores a string that specifies the referential integrity rule that governs a child FK when the parent PK is updated.	See Table Q – Parent Update RI Valid Values later in this chapter.	
CHILD INSERT RULE	char(6)	Stores a string that specifies the referential integrity rule that governs a parent when data is inserted into a child instance.	See Table R – Child Insert RI Valid Values later in this chapter.	
CHILD DELETE RULE	char(6)	Stores a string that specifies the referential integrity rule that governs a parent when a child instance is deleted.	See Table S – Child Delete RI Valid Values later in this chapter.	
CHILD UPDATE RULE	char(6)	Stores a string that specifies the referential integrity rule that governs a parent when the foreign key of a child instance is updated.	See Table T – Child Update RI Valid Values later in this chapter.	
PARENT INSERT RULE	char(6)	Stores a string that specifies the referential integrity rule that governs a child when data is inserted into the parent.	See Table U – Parent Insert RI Valid Values later in this chapter.	
NULL RULE	char(6)	Stores a string that specifies whether a relationship allows nulls. Only non-identifying relationships can have a null option.	See Table V – Null Options Valid Values later in this chapter.	
RELATIONSHIP DEFINITION	integer	Stores the ID number of a text description for a relationship.	1 to 2^{32}	FK
VERB PHRASE 2	varchar(256)	Stores the descriptive child-parent verb phrase labeling a relationship.	Any that meets datatype criteria.	

Summary of ERwin Relationship (continued)

Attribute Name	Datatype	Description	Valid Values	FK
SYNONYM NAME	varchar(128)	Stores a synonym name for the entity.	Any that meets datatype criteria.	FK
VIEW SEQ	integer	Integer representing relationship sequence in the FROM clause of the SELECT statement in an underlying view.	1 to 2^{32}	
IS OUTER	integer	Boolean (0 or 1) representing whether the relationship is part of an outer join in an view.	0 (no), 1 (yes)	
MANY TO MANY ASSOC	integer	ENT_ID reference into ENTITY table representing the association table for the many-to-many relationship	1 to 2^{32}	FK
REL FLAGS	integer	Internal integer value	1 to 2^{32}	

ERwin Report

This entity stores the named report formats for schema generation and other reports supported by ERwin.

Attribute Name	Datatype	Description	Valid Values	FK
ERWIN DIAGRAM ID	integer	Stores the ID number of a diagram.	1 to 2^{32}	FK
ERWIN REPORT ID	integer	Stores the ID number of a saved report format.	1 to 2^{32}	
ERWIN REPORT TYPE	char(6)	Stores a string that specifies the report type is a schema report.	RE_SCH	
ERWIN REPORT NAME	varchar(254)	Stores the name of a saved report format.	Any that meets datatype criteria.	

ERwin Report Option

This entity stores the report options used in named reports, each corresponding to a check mark in the ERwin report editor. (e.g., "Create Table" from the Schema Generation Report)

Attribute Name	Datatype	Description	Valid Values	FK
ERWIN DIAGRAM ID	integer	Stores the ID number of a diagram.	1 to 2^{32}	FK
ERWIN REPORT ID	integer	Stores the ID number of a saved report format.	1 to 2^{32}	FK
REPORT OPTION ID	integer	Stores the ID number of a single option in a saved report format (e.g., the Attribute Name option in the Attribute Report).	1 to 2^{32}	
REPORT OPTION SEQUENCE	integer	Stores a number that specifies the position of a report option in the overall report .	1 to 2^{32}	
REPORT OPTION NAME	varchar(254)	Stores the name of a report option (e.g., "Attribute Name" in the Attribute Report).	Any that meets datatype criteria.	
REPORT OPTION VALUE	integer	Stores a binary number that specifies whether an option is on, off, or disabled.	0 (no), 1(yes), null (disabled)	
REPORT OPTION TYPE	integer	Stores a number that specifies a report option type (e.g., entity option, attribute option).	1 to 2^{32}	
REPORT OPTION MULTI	integer	Stores a Boolean value that specifies whether or not an option is multi-valued (e.g., "Attribute Names" in the Entity Report).	0 (no), 1 (yes)	

ERwin Subtype Relationship

This entity stores information about the logical relationships between the child "sub-categorized" entity (subtype) and the parent "generalization" entity (supertype). Subtypes are also called "subcategories."

Attribute Name	Datatype	Description	Valid Values	FK
SUBTYPE ID	integer	Stores the ID number of a subtype generalization.	1 to 2^{32}	FK
SUBTYPE TYPE	char(6)	Stores a string that specifies whether the subtype is complete or incomplete.	See Table W – Subcategory Types later in this chapter.	FK
ERWIN DIAGRAM ID	integer	Stores the ID number of a diagram.	1 to 2^{32}	FK
RESERVED	integer	Reserved for internal use.	NA	FK
DISCRIMINATOR NAME	varchar(254)	Stores the name of a subtype relationship's discriminator.	Any that meets datatype criteria.	

ERwin Text

This entity stores text values for text entries, such as entity, attribute, and domain definitions and text blocks. Text values are related to attributes or entities that have the same TEXT ID value.

Attribute Name	Datatype	Description	Valid Values	FK
ERWIN TEXT ID	integer	Stores the ID number of descriptive or definition text.	1 to 2^{32}	
ERWIN DIAGRAM ID	integer	Stores the ID number of a diagram.	1 to 2^{32}	FK
ERWIN TEXT VALUE	long varchar	Stores descriptive or definition text from other attributes referenced by TEXT ID.	Any that meets datatype criteria.	

Global Template

This entity stores information about the trigger template attached to the diagram as global trigger code, which is expanded for the defined referential integrity slot (e.g., CHILD INSERT RESTRICT). These template attachments are specified in the Trigger Template Editor.

Attribute Name	Datatype	Description	Valid Values	FK
TEMPLATE ID	integer	Stores the ID number of a template.	1 to 2^{32}	FK
TEMPLATE TYPE	char(6)	Stores a string that specifies whether the template is built-in or user-created.	See Table F – Template Types later in this chapter.	FK
ERWIN DIAGRAM ID	integer	Stores the ID number of a diagram.	1 to 2^{32}	FK
RESERVED	integer	Reserved for internal use.	NA	FK
TARGET SYSTEM	char(6)	Stores a string that specifies a physical DBMS.	See Table A – Target System Valid Values later in this chapter.	FK
TEMPLATE PURPOSE	char(6)	Stores a string that specifies the global referential integrity or other purpose implemented by an attached template.	See Table X – Global Template Purpose Valid Values later in this chapter.	FK

Index Member

This entity stores information about the columns in an index.

Attribute Name	Datatype	Description	Valid Values	FK
INDEX ID	integer	Stores the ID number of an index in an entity; an index ID can overlap across entities, but must be unique within an entity.	1 to 2^{32}	FK
INDEX TYPE	char(6)	Stores a string that specifies the index type .	See Table C – Index Type Valid Values later in this chapter.	FK
ATTRIBUTE ID	integer	Stores the unique ID number for each attribute.	1 to 2^{32}	FK
RELATIONSHIP ID	integer	Stores the unique ID number of a relationship.	1 to 2^{32}	FK
RELATIONSHIP TYPE	char(6)	Stores a string that specifies whether a relationship is ID/non-ID, many-to-many, or subtype.	See Table L – Relationship Type Valid Values later in this chapter.	FK
ENTITY-ATTRIBUTE BASE NAME	varchar(254)	Stores the name (other than the rolename) of an attribute as it is used in an entity.	Any that meets datatype criteria.	FK
ENTITY ID	integer	Stores the unique ID number of an entity.	1 to 2^{32}	FK
ENTITY TYPE	char(6)	Stores a string that specifies whether an entity is independent or dependent.	See Table D – Entity Type Valid Values later in this chapter.	FK
ERWIN DIAGRAM ID	integer	Stores the ID number of a diagram.	1 to 2^{32}	FK
RESERVED	integer	Reserved for internal use.	NA	FK
INDEX POSITION	integer	Stores a number that specifies the position of a column in an index; this has an impact on performance and sort order.	1 to 2^{32}	

Index Member Physical PV

This entity stores information about the DBMS-specific physical properties of the columns of an index.

Attribute Name	Datatype	Description	Valid Values	FK
INDEX ID	integer	Stores the ID number of an index in an entity; an index ID can overlap across entities, but must be unique within an entity.	1 to 2^{32}	FK
INDEX TYPE	char(6)	Stores a string that specifies the index type.	See Table C – Index Type Valid Values later in this chapter.	FK
ATTRIBUTE ID	integer	Stores the unique ID number for each attribute.	1 to 2^{32}	FK
RELATIONSHIP ID	integer	Stores the unique ID number of a relationship.	1 to 2^{32}	FK
RELATIONSHIP TYPE	char(6)	Stores a string that specifies whether a relationship is ID/non-ID, many-to-many, or subtype.	See Table L – Relationship Type Valid Values later in this chapter.	FK
ENTITY-ATTRIBUTE BASE NAME	varchar(254)	Stores the name (other than the rolename) of an attribute as it is used in an entity.	Any that meets datatype criteria.	FK
ENTITY ID	integer	Stores the unique ID number of an entity.	1 to 2^{32}	FK
ENTITY TYPE	char(6)	Stores a string that specifies whether an entity is independent or dependent.	See Table D – Entity Type Valid Values later in this chapter.	FK
ERWIN DIAGRAM ID	integer	Stores the ID number of a diagram.	1 to 2^{32}	FK
RESERVED	integer	Reserved for internal use.	NA	FK
TARGET SYSTEM	char(6)	Stores a string that specifies a physical DBMS.	See Table A – Target System Valid Values later in this chapter.	FK
INDEX MEMBER PV NAME	varchar(254)	Stores the name of the index member physical property.	Any that meets datatype criteria.	
INDEX MEMBER PV VALUE	integer	Stores the numeric value of an index member physical property.	1 to 2^{32}	FK

Index Physical PV

This entity stores information about the physical properties of an index.

Attribute Name	Datatype	Description	Valid Values	FK
INDEX ID	integer	Stores the ID number of an index in an entity; an index ID can overlap across entities, but must be unique within an entity.	1 to 2^{32}	FK
INDEX TYPE	char(6)	Stores a string that specifies the index type	See Table C – Index Type Valid Values later in this chapter.	FK
ENTITY ID	integer	Stores the unique ID number of an entity.	1 to 2^{32}	FK
ENTITY TYPE	char(6)	Stores a string that specifies whether an entity is independent or dependent.	See Table D – Entity Type Valid Values later in this chapter.	FK
ERWIN DIAGRAM ID	integer	Stores the ID number of a diagram.	1 to 2^{32}	FK
RESERVED	integer	Reserved for internal use.	NA	FK
TARGET SYSTEM	char(6)	Stores a string that specifies a physical DBMS.	See Table A – Target System Valid Values later in this chapter.	FK
INDEX PV TYPE	char(6)	Stores a string that specifies a target-specific physical property identifier assigned to an index (e.g., FILLFACTOR, SEGMENT).	See Table Y – Index Physical Property Codes later in this chapter.	FK
INDEX PV PHYS OBJ	integer	Stores the ID number of a physical object.	1 to 2^{32}	FK
INDEX PV VALUE	integer	Stores the index physical property string value.	1 to 2^{32}	FK

Physical Object

This entity stores information about database physical objects, such as ORACLE TABLESPACE, ROLLBACK SEGMENT, etc.

Attribute Name	Datatype	Description	Valid Values	FK
PHYSICAL OBJECT ID	integer	Stores the ID number of a physical object.	1 to 2^{32}	FK
PHYSICAL OBJECT TYPE	char(6)	Stores a string that specifies the physical object type (e.g., ORACLE TABLESPACE, SYBASE SEGMENT).	See Table Z – Physical Object Type Codes later in this chapter.	FK
ERWIN DIAGRAM ID	integer	Stores the ID number of a diagram.	1 to 2^{32}	FK
RESERVED	integer	Reserved for internal use.	NA	FK
TARGET SYSTEM	char(6)	Stores a string that specifies a physical DBMS.	See Table A – Target System Valid Values later in this chapter.	FK
PHYSICAL OBJECT NAME	varchar(128)	Stores the name used to refer to a physical object.	Any that meets datatype criteria.	

Physical Object PV

This entity stores information about database physical object properties, such as ORACLE PCTFREE, PCTUSED, etc.

Attribute Name	Datatype	Description	Valid Values	FK
PHYSICAL OBJECT ID	integer	Stores the ID number of a physical object.	1 to 2^{32}	FK
PHYSICAL OBJECT TYPE	char(6)	Stores a string that specifies the physical object type (e.g., ORACLE TABLESPACE, SYBASE SEGMENT).	See Table FF – Physical Object Type Codes later in this chapter.	FK
ERWIN DIAGRAM ID	integer	Stores the ID number of a diagram.	1 to 2^{32}	FK
RESERVED	integer	Reserved for internal use.	NA	FK
PHYSICAL OBJECT PV TYPE	char(6)	Stores a string that specifies the physical property assigned to a physical object (e.g., PCTFREE, PCTUSED).	See Table AA – Property Codes later in this chapter.	FK
PHYSICAL OBJECT PV VALUE	integer	Stores the ID number of a physical object or its text.	1 to 2^{32}	FK
PHYSICAL OBJECT PV IVAL	integer	Stores a physical object's integer value.	1 to 2^{32}	
PHYSICAL OBJECT PV SVAL	varchar(254)	Stores a physical object's string value.	Any that meets datatype criteria.	

Rel Physical Property

This entity stores the DBMS-dependent properties of a relationship.

Attribute Name	Datatype	Description	Valid Values	FK
RELATIONSHIP ID	integer	Stores the unique ID number of a relationship.	1 to 2^{32}	FK
RELATIONSHIP TYPE	char(6)	Stores a string that specifies whether a relationship is ID/non-ID, many-to-many, or subtype.	See Table L – Relationship Type Valid Values later in this chapter.	FK
ERWIN DIAGRAM ID	integer	Stores the ID number of a diagram.	1 to 2^{32}	FK
RESERVED	integer	Reserved for internal use.	NA	FK
TARGET SYSTEM	char(6)	Stores a string that specifies a physical DBMS.	See Table A – Target System Valid Values later in this chapter.	FK
FOREIGNKEY	varchar(254)	Stores the name given to a foreign key constraint.	Any that meets datatype criteria.	

Relationship Template

This entity stores information about the ERwin RI Trigger Templates that control referential integrity.

Attribute Name	Datatype	Description	Valid Values	FK
RELATIONSHIP ID	integer	Stores the unique ID number of a relationship.	1 to 2^{32}	FK
RELATIONSHIP TYPE	char(6)	Stores a string that specifies whether a relationship is ID/non-ID, many-to-many, or subtype.	See Table L – Relationship Type Valid Values later in this chapter.	FK
TEMPLATE ID	integer	Stores the ID number of a template.	1 to 2^{32}	FK
TEMPLATE TYPE	char(6)	Stores a string that specifies whether the template is built-in or user-created.	See Table F – Template Types later in this chapter.	FK
ERWIN DIAGRAM ID	integer	Stores the ID number of a diagram.	1 to 2^{32}	FK
RESERVED	integer	Reserved for internal use.	NA	FK
TARGET SYSTEM	char(6)	Stores a string that specifies a physical DBMS.	See Table A – Target System Valid Values later in this chapter.	FK
RELATIONSHIP TEMPLATE PURPOSE	char(6)	Stores a string that specifies a referential integrity option implemented by an attached template.	See Table BB – Relationship Template Purpose Valid Values later in this chapter.	FK

Stored Display

This entity stores information for the rendering of entities and relationships in a stored display, including graphical coordinates, colors, fonts, and display options.

Attribute Name	Datatype	Description	Valid Values	FK
ERWIN DIAGRAM ID	integer	Stores the ID number of a diagram.	1 to 2^{32}	FK
SUBJECT ID	integer	Stores the ID number of a subject area.	1 to 2^{32}	FK
DISPLAY ID	integer	Stores the ID number of a stored display.	1 to 2^{32}	
RESERVED	integer	Reserved for internal use.	NA	FK
DISPLAY NAME	varchar(254)	Stores the name of the stored display that appears on the stored display tab.	Any that meets datatype criteria.	
DISPLAY DESCRIPTION	integer	Stores the ID number of the text description of a stored display.	1 to 2^{32}	FK
DISPLAY AUTHOR	varchar(254)	Stores the name of the creator of a stored display.	Any that meets datatype criteria.	
DISPLAY CREATION DATE	integer	Stores a binary number that specifies the date a stored display was created.	1 to 2^{32}	
DISPLAY MODIFICATION DATE	integer	Stores a binary number that specifies the date a stored display was last modified.	1 to 2^{32}	
ENTITY FILL COLOR ID	integer	Stores a number that specifies the default entity fill color.	1 to 2^{32}	FK
BACKGROUND COLOR ID	integer	Stores a number that specifies the default background color.	1 to 2^{32}	FK
DEFAULT RELATIONSHIP FONT ID	integer	Stores a number that specifies the default font for relationship verb-phrases or physical names.	1 to 2^{32}	FK
DEFAULT ENTITY NAME FONT ID	integer	Stores a number that specifies the default font for entity names.	1 to 2^{32}	FK
DEFAULT ENTITY DEFINITION FONT ID	integer	Stores a number that specifies the default font for entity definitions in the definition-level display.	1 to 2^{32}	FK
DEFAULT ELINE COLOR ID	integer	Stores a number that specifies the default entity-outline color.	1 to 2^{32}	FK
DEFAULT EDEF COLOR ID	integer	Stores a number that specifies the default entity-definition text color.	1 to 2^{32}	FK
DEFAULT RELATIONSHIP NAME COLOR ID	integer	Stores a number that specifies the default color for relationship verb-phrases or physical names.	1 to 2^{32}	FK
DEFAULT ENTITY NAME COLOR ID	integer	Stores a number that specifies the default entity-name color.	1 to 2^{32}	FK
COLOR TYPE	char(6)	Stores the TX_COL string as a placeholder for future screen and print color implementations.	TX_COL	FK
FONT TYPE	char(6)	Stores the TX_FNT string as a placeholder for future screen and print font implementations.	TX_FNT	FK

Subject Area

This entity stores information about subject areas, which are user-defined subsets of entities in the data model.

Attribute Name	Datatype	Description	Valid Values	FK
ERWIN DIAGRAM ID	integer	Stores the ID number of a diagram.	1 to 2^{32}	FK
SUBJECT ID	integer	Stores the ID number of a subject area.	1 to 2^{32}	
SUBJECT NAME	varchar(254)	Stores the name used to refer to the subject area.	Any that meets datatype criteria.	
SUBJECT DESCRIPTION	integer	Stores the ID number of a text description of a subject area.	1 to 2^{32}	FK
SUBJECT AUTHOR	varchar(254)	Stores the name of the creator of a subject area.	Any that meets datatype criteria.	
SUBJECT CREATION DATE	integer	Stores a binary number that specifies the date a subject area was created.	1 to 2^{32}	
SUBJECT MODIFICATION DATE	integer	Stores a binary number that specifies the date a subject area was last modified.	1 to 2^{32}	
SUBJECT FLAG	integer	Stores the flag bits for subject area options; the low order bit indicates the "Include Decomposition" option for a subject area based on a BPwin activity.	1 to 2^{32}	
SUBJECT BPWIN INDEX	integer	Stores the ID number of the BPwin activity on which a subject area is based; if not based on a BPwin activity, the value is 0.	1 to 2^{32}	

Synonym Physical PV

This entity stores information about synonym physical storage properties.

Attribute Name	Datatype	Description	Valid Values	FK
SYNONYM NAME	varchar(128)	Stores the string that specifies synonym name.	Any that meets datatype criteria.	FK
ENTITY ID	integer	Stores the unique ID number of an entity.	1 to 2^{32}	FK
ENTITY TYPE	char(6)	Stores a string that specifies whether an entity is independent or dependent.	See Table D – Entity Type Valid Values later in this chapter.	FK
ERWIN DIAGRAM ID	integer	Stores the ID number of a diagram.	1 to 2^{32}	FK
RESERVED	integer	Reserved for internal use.	NA	FK
TARGET SYSTEM	char(6)	Stores a string that specifies a physical DBMS.	See Table A – Target System Valid Values later in this chapter.	FK
SYNONYM PV TYPE	char(6)	Stores a string that specifies the physical storage type.	Any that meets datatype criteria.	
SYNONYM PV VALUE	varchar(256)	Stores a string that specifies the text value for synonym physical storage.	Any that meets datatype criteria.	

Table Constraint

This entity stores information about the validation rules attached to each entity.

Attribute Name	Datatype	Description	Valid Values	FK
ENTITY ID	integer	Stores the unique ID number of an entity.	1 to 2^{32}	FK
ENTITY TYPE	char(6)	Stores a string that specifies whether an entity is independent or dependent.	See Table D – Entity Type Valid Values later in this chapter.	FK
ERWIN DIAGRAM ID	integer	Stores the ID number of a diagram.	1 to 2^{32}	FK
RESERVED	integer	Reserved for internal use.	NA	FK
VALIDATION RULE ID	integer	Stores the ID number of a validation rule.	1 to 2^{32}	FK
VALID RULE TYPE	char(6)	Stores a string that specifies whether the validation rule is client, server, or both.	See Table CC – Validation Rule Type Codes later in this chapter.	FK
TARGET SERVER	char(6)	Stores a string that specifies a physical server DBMS.	See Table DD – Target Server Valid Values later in this chapter.	FK
TARGET CLIENT	char(6)	Stores a string that specifies a client DBMS environment.	See Table EE – Target Client Valid Values later in this chapter.	FK

Template

This entity stores information about ERwin templates, which include SQL statements and ERwin macros. Templates are used for triggers, stored procedures, and pre- and post-schema generation scripts.

Attribute Name	Datatype	Description	Valid Values	FK
TEMPLATE ID	integer	Stores the ID number of a template.	1 to 2^{32}	FK
TEMPLATE TYPE	char(6)	Stores a string that specifies whether the template is built-in or user-created.	See Table F – Template Types later in this chapter.	FK
ERWIN DIAGRAM ID	integer	Stores the ID number of a diagram.	1 to 2^{32}	FK
RESERVED	integer	Reserved for internal use.	NA	FK
TARGET SYSTEM	char(6)	Stores a string that specifies a physical DBMS.	See Table A – Target System Valid Values later in this chapter.	FK
TEMPLATE NAME	varchar(254)	Stores the name used to refer to the template.	Any that meets datatype criteria.	
TEMPLATE CODE	integer	Stores the ID number of the template code, which will be expanded on schema generation.	1 to 2^{32}	FK

Text Block

This entity stores information about text blocks in an ERwin diagram.

Attribute Name	Datatype	Description	Valid Values	FK
TEXT BLOCK ID	integer	Stores the ID number of a text block.	1 to 2^{32}	FK
TEXT BLOCK TYPE	char(6)	Stores a string that identifies an object as a text block.	TB_TB	FK
ERWIN DIAGRAM ID	integer	Stores the ID number of a diagram.	1 to 2^{32}	FK
RESERVED	integer	Reserved for internal use.	NA	FK
TEXT BLOCK TEXT	integer	Stores the ID number of the text displayed by a text block.	1 to 2^{32}	FK

Trigcols

This entity stores information about entity trigger update columns.

Attribute Name	Datatype	Description	Valid Values	FK
ENTITY ID	integer	Stores the unique ID number of an entity.	1 to 2^{32}	FK
ENTITY TYPE	char(6)	Stores a string that specifies whether an entity is independent or dependent.	See Table D – Entity Type Valid Values later in this chapter.	FK
ERWIN DIAGRAM ID	integer	Stores the ID number of a diagram.	1 to 2^{32}	FK
RESERVED	integer	Reserved for internal use.	NA	FK
TARGET SYSTEM	char(6)	Stores a string that specifies a physical DBMS.	See Table A – Target System Valid Values later in this chapter.	FK
TRIGGER NAME	varchar(254)	Stores the name used to create and refer to a trigger in the database.	Any that meets datatype criteria.	FK
UPDCOLS LOGICAL NAME	varchar(254)	Stores the logical name of update column.	Any that meets datatype criteria.	
UPDCOLS ORDER	integer	Stores a number that specifies the ordered position of an update column.	1 to 2^{32}	
UPDCOLS PHYS NAME	varchar(254)	Stores the physical name of update column.	Any that meets datatype criteria.	

Valid Values

This entity stores information about valid values, including their descriptions and associated validation rules.

Attribute Name	Datatype	Description	Valid Values	FK
VALIDATION RULE ID	integer	Stores the ID number of a validation rule.	1 to 2^{32}	FK
VALID RULE TYPE	char(6)	Stores a string that specifies whether the validation rule is client, server, or both.	See Table CC – Validation Rule Type Codes later in this chapter.	FK
ERWIN DIAGRAM ID	integer	Stores the ID number of a diagram.	1 to 2^{32}	FK
RESERVED	integer	Reserved for internal use.	NA	FK
VALUE	varchar(254)	Stores a validation rule element's value.	Any that meets datatype criteria.	
VALUE DEFINITION	integer	Stores the ID number of a text description of a validation rule element.	1 to 2^{32}	FK
DISPLAY VALUE	varchar(254)	Stores the screen display form of a validation rule element's value.	Any that meets datatype criteria.	
SEQUENCE NUMBER	integer	Stores a number that specifies the sequence order of a valid value in the list.	1 to 2^{32}	

Validation Rule

This entity stores information about validation rules, which can be constraint expressions or lists of valid values for a specific attribute. Validation rules can be attached to a domain or directly to an attribute.

Attribute Name	Datatype	Description	Valid Values	FK
VALIDATION RULE ID	integer	Stores the ID number of a validation rule.	1 to 2^{32}	FK
VALID RULE TYPE	char(6)	Stores a string that specifies whether the validation rule is client, server, or both.	See Table CC – Validation Rule Type Codes later in this chapter.	FK
ERWIN DIAGRAM ID	integer	Stores the ID number of a diagram.	1 to 2^{32}	FK
RESERVED	integer	Reserved for internal use.	NA	FK
VLDRULE_NAME	varchar(254)	Stores the name of the validation rule.	Any that meets datatype criteria.	
CLIENT RULE	integer	Stores a number that specifies the client part of a validation rule.	1 to 2^{32}	FK
SERVER RULE	integer	Stores a number that specifies the server part of a validation rule.	1 to 2^{32}	FK
CLIENT TYPE	integer	Stores a number that specifies the type of value to which a validation rule applies (e.g., string, numeric).	1 to 2^{32}	
CLIENT WARNING	varchar(254)	Stores the text of the warning message issued if the client rule is not obeyed in the client tool.	Any that meets datatype criteria.	
MINIMUM VALUE	varchar(254)	Stores a string that specifies the minimum value the column can have.	Any that meets datatype criteria.	
MAXIMUM VALUE	varchar(254)	Stores a string that specifies the maximum value the column can have.	Any that meets datatype criteria.	
ERROR MESSAGE	varchar(254)	Stores the text of the error message issued if server rule is not obeyed on the server.	Any that meets datatype criteria.	

Internal Use and Internally Derived Entities

ERwin utilizes additional entities that store internal information (internal use) or derive information that is used only when ERwin is displaying the model during a modeling session (internally derived). Internally derived entities, which are noted by brackets [that indicate an entity is not a stored entity] in the following descriptions, are used to properly connect the physical entities in the metamodel for presentation and logical modeling purposes only.

_Object_Type_

This entity stores a text code value that identifies each type of object that can be distinguished in an ERwin diagram.

Attribute Name	Datatype	Description	Valid Values	FK
OBJECT TYPE	char(6)	Stores a string that specifies an object's type (e.g., entity, text block, relationship).	Superset of all valid object types, including entities, attributes, relationships, etc.	

Target_D

This entity stores a text code value that identifies each target server ERwin currently supports.

Attribute Name	Datatype	Description	Valid Values	FK
TARGET SYSTEM	char(6)	Stores a string that specifies a physical DBMS.	See Table A – Target System Valid Values later in this chapter.	

[ERwin Attribute]

This logical entity stores information about each attribute [not a stored table].

Attribute Name	Datatype	Description	Valid Values	FK
ATTRIBUTE ID	integer	Stores the unique ID number for each attribute.	1 to 2^{32}	
ATTRIBUTE TYPE	char(6)	Stores a string that specifies whether an attribute is a primary key or a non key.	See Table M – Attribute Type Codes later in this chapter.	

[ERwin Diagram Object]

This logical entity stores information about any object identified in a diagram [not a stored table].

Attribute Name	Datatype	Description	Valid Values	FK
OBJECT ID	integer	Stores the ID number of an object, which overlaps across types.	1 to 2^{32}	
OBJECT TYPE	char(6)	Stores a string that specifies an object's type (e.g., entity, text block, relationship).	Superset of all valid object types, including entities, attributes, relationships, etc.	

[ERwin Object]

This logical entity stores the object description that identifies an ERwin object. An object can appear in the same diagram in different views and it can occur in different diagrams. [not a stored table]

Attribute Name	Datatype	Description	Valid Values	FK
OBJECT ID	integer	Stores the ID number of an object, which can overlap across object types.	1 to 2^{32}	
OBJECT DESCRIPTION	varchar(254)	Stores a diagram-independent description of an object as a string.	Any that meets datatype criteria.	

Valid Value Tables

The following tables define the valid values used in various tables in the metamodel.

Table A – Target System Valid Values

This table of valid values is used in the following metamodel tables: DOMAINPV, ENT_PROP, ENT_TRIG, ESPTMPLT, GBL_TMPL, INDXPROP, INDX_MPP, PHYS_OBJ, REL_PHYS, REL_TMPL, SYN_PROP, TEMPLATE, and TRIGCOLS.

TARG_SYS Valid Values	TARG_SYS Description
TD_AD	Target DBMS ADABAS
TD_AS	Target DBMS AS400
TD_CL	Target DBMS CLIPPER
TD_D2	Target DBMS DB2_2
TD_D3	Target DBMS DBASE3
TD_D4	Target DBMS DBASE4
TD_DB	Target DBMS DB2
TD_FP	Target DBMS FOXPRO
TD_IN	Target DBMS INGRES
TD_IX	Target DBMS INFORMIX
TD_NW	Target DBMS NETWARE
TD_OR	Target DBMS ORACLE
TD_PG	Target DBMS PROGRESS
TD_RB	Target DBMS REDBRICK
TD_RD	Target DBMS RDB
TD_SB	Target DBMS SQLBASE
TD_SM	Target DBMS SMARTSTAR
TD_SS	Target DBMS SQLSERVER
TD_SY	Target DBMS SYBASE
TD_TE	Target DBMS TERADATA
TD_WC	Target DBMS WATCOM
TC_PB	CLIENT_TYPE_POWERB
TC_SW	CLIENT_TYPE_SQLWIN
TC_VB	CLIENT_TYPE_VB

Table B – Column Domain Property Types

This table of valid values is used in the DOMAINPV metamodel table.

DMNP_TYPE Valid Values	DMNP_TYPE Description	DMNP_TVAL Valid Values	DMNP_IVAL Valid Values	DMNP_IDVL Valid Values	DMNP_SVAL Valid Values
CD_4CL	COLDOM_TYPE_FORCOLUMN				String
CD_ACC	COLDOM_TYPE_ACCEL				String
CD_ALZ	COLDOM_TYPE_ALLOW_ZERO		1 or 0		
CD_BKC	COLDOM TYPE SQL WINDOWS swc_bkcl				String
CD_CMP	COLDOM_TYPE_COMPRESS		1 or 0		
CD_CMT	COLDOM_TYPE_COMMENT	ERWTXTID reference to ERW_TEXT table			
CD_CSE	COLDOM_TYPE_CHARCASE				String
CD_CTP	COLDOM_TYPE_CHARTYPE				String
CD_DEC	COLDOM_TYPE_DECIMALS		1 or 0		
CD_DEF	COLDOM_TYPE_DEFAULT			DFLT_ID reference to DEF_VAL table	
CD_DMC	COLDOM_TYPE_DOMAIN_COMMENT	ERWTXTID reference to ERW_TEXT table			
CD_DMN	COLDOM_TYPE_DOMAIN_NAME				String
CD_ECM	COLDOM_TYPE_EXPANDED_COMMENT	ERWTXTID reference to ERW_TEXT table			
CD_EIN	COLDOM_TYPE_EMPTY_IS_NULL		1 or 0		
CD_EXT	COLDOM_TYPE_EXTENT				String
CD_FEH	COLDOM TYPE SQL WINDOWS swc_fneh				String
CD_FMT	COLDOM_TYPE_FORMAT			POBJ_ID reference into PHYS_OBJ table	
CD_FNM	COLDOM TYPE SQL WINDOWS swc_fnnm				String
CD_FSZ	COLDOM TYPE SQL WINDOWS swc_fnsz				String
CD_HLP	COLDOM_TYPE_HELP_ID		Integer		
CD_MED	COLDOM_TYPE_MIGRATED_STYLE			POBJ_ID reference into PHYS_OBJ table	
CD_MSK	COLDOM_TYPE_SERVER_INPUT_MASK				String
CD_NAM	COLDOM_TYPE_FIELD_NAME				String
CD_NUL	COLDOM_TYPE_NULL_OPTION		Integer		
CD_ORD	COLDOM_TYPE_ORDER_NUM		Integer		

Table B – Column Domain Property Types (continued)

DMNP_TYPE Valid Values	DMNP_TYPE Description	DMNP_TVAL Valid Values	DMNP_IVAL Valid Values	DMNP_IDVL Valid Values	DMNP_SVAL Valid Values
CD_PBM	COLDOM TYPE POWERBUILDER pbc_bmap				String
CD_PCM	COLDOM TYPE POWERBUILDER pbc_cmnt	ERWTXTID reference to ERW_TEXT table			
CD_PCS	COLDOM TYPE POWERBUILDER pbc_case				String
CD_PED	COLDOM TYPE POWERBUILDER pbc_edit				String
CD_PHD	COLDOM_TYPE_HEADER				String
CD_PHO	COLDOM_TYPE_PHYSICAL_ONLY				String
CD_PHP	COLDOM TYPE POWERBUILDER pbc_hpos				String
CD_PHT	COLDOM TYPE POWERBUILDER pbc_hght				String
CD_PIN	COLDOM TYPE POWERBUILDER pbc_init				String
CD_PJT	COLDOM TYPE POWERBUILDER pbc_jtfy				String
CD_PLB	COLDOM_TYPE_LABEL				String
CD_PLP	COLDOM TYPE POWERBUILDER pbc_lpos				String
CD_PMK	COLDOM TYPE POWERBUILDER pbc_mask				String
CD_PMP	COLDOM TYPE SQL WINDOWS swc_prompt				String
CD_PTR	COLDOM TYPE POWERBUILDER pbc_ptrn				String
CD_PWD	COLDOM TYPE POWERBUILDER pbc_wdth				String
CD_REQ	COLDOM_TYPE_REQUIRED		1 or 0		
CD_RO	COLDOM_TYPE_READ_ONLY		1 or 0		
CD_TAG	COLDOM_TYPE_TAG				String
CD_TXC	COLDOM TYPE SQL WINDOWS swc_txcl				String
CD_TYP	COLDOM_TYPE_DATATYPE				String
CD_VAL	COLDOM_TYPE_VALIDATION			POBJ_ID reference into PHYS_OBJ table	
CD_VCE	COLDOM_TYPE_VIEW_COL_USER_EXPR	ERWTXTID reference to ERW_TEXT table			
CD_VEW	COLDOM_TYPE_VIEWAS				String
CD_VSB	COLDOM TYPE SQL WINDOWS swc_visb		1 or 0		

Table C – Index Type Valid Values

This table of valid values is used in the following metamodel tables: ENT_INDX, INDXPROP, INDX_MEM, and INDX_MPP.

INDEX_TYPE Valid Values	INDEX_TYPE Description
IT_AK	Index Type Alternate Key
IT_IE	Index Type Inversion Entry
IT_IF	Index Type Foreign Key
IT_PK	Index Type Primary Key

Table D – Entity Type Valid Values

This table of valid values is used in the following metamodel tables: ENT_ATT, ENT_INDX, ENT_PROP, ENT_SYN, ENT_TRIG, ESPTMPLT, ENTITY, INDXPROP, INDX_MEM, INDX_MPP, SYN_PROP, TBLCONST, and TRIGCOLS.

ENT_TYPE Valid Values	ENT_TYPE Description
ET_DE	Entity Type DEP_ENTITY
ET_DR	Entity Type DERIVED_ENTITY
ET_IE	Entity Type INDEP_ENTITY

Table E – Physical Property Codes

This table of valid values is used in the ENT_PROP metamodel table.

PROP_TYPE Valid Values	PROP_TYPE Description	PROP_VAL Valid Values
D2_DBP	DB2 DBBUFFERPOOL	ERWTXTID reference to text string in ERW_TEXT table
D2_DBS	DB2 DATABASE Physical Object Reference	POBJ_ID reference into PHYS_OBJ table
D2_DRS	DB2 ROSHARE	ERWTXTID reference to text string in ERW_TEXT table
D2_DSG	DB2 DATABASE STOGROUP	POBJ_ID reference into PHYS_OBJ table
D2_PAU	DB2 AUDIT	1 or 0
D2_PDB	DB2 IN DATABASE Physical Object Reference	POBJ_ID reference into PHYS_OBJ table
D2_PDC	DB2 DATACAPTURE	ERWTXTID reference to text string in ERW_TEXT table
D2_PEP	DB2 EDITPROC	ERWTXTID reference to text string in ERW_TEXT table
D2_POB	DB2 OBID	ERWTXTID reference to text string in ERW_TEXT table
D2_PTS	DB2 IN TABLESPACE	POBJ_ID reference into PHYS_OBJ table
D2_TSP	DB2 TABLESPACE	POBJ_ID reference into PHYS_OBJ table
EP_MXR	MAX ROWS	ERWTXTID reference to text string in ERW_TEXT table
EP_SEG	SEGMENT	POBJ_ID reference into PHYS_OBJ table
EP_SRR	STAR	1 or 0
IX_ESZ	Informix EXTENT SIZE	ERWTXTID reference to text string in ERW_TEXT table
IX_FEX	Informix FRAGMENT BY EXPRESSION	ERWTXTID reference to text string in ERW_TEXT table
IX_FRR	Informix FRAGMENT BY ROUND ROBIN IN	ERWTXTID reference to text string in ERW_TEXT table
IX_IND	Informix IN DBSPACE	POBJ_ID reference into PHYS_OBJ table
IX_INP	Informix IN PATH	ERWTXTID reference to text string in ERW_TEXT table
IX_LKM	Informix LOCK MODE	ERWTXTID reference to text string in ERW_TEXT table
IX_NSZ	Informix NEXT SIZE	ERWTXTID reference to text string in ERW_TEXT table
IX_REM	Informix REMAINDER IN	POBJ_ID reference into PHYS_OBJ table
IX_STM	Informix STORAGE OPTION	ERWTXTID reference to text string in ERW_TEXT table
PP_ACT	ACCOUNT	POBJ_ID reference into PHYS_OBJ table
PP_ASP	AS_PERMANENT	1 or 0
PP_CCH	CACHE	1 or 0
PP_CMT	Informix SYN CMT	ERWTXTID reference to text string in ERW_TEXT table
PP_CNS	CONSTRAINT	ERWTXTID reference to text string in ERW_TEXT table
PP_COM	COMMENT	ERWTXTID reference to text string in ERW_TEXT table
PP_DIN	Redbrick DATA IN	POBJ_ID reference into PHYS_OBJ table
PP_DMP	PROGDUMP	ERWTXTID reference to text string in ERW_TEXT table
PP_DS	Redbrick DOMAIN SIZE	ERWTXTID reference to text string in ERW_TEXT table
PP_DSP	Reference to a DBSPACE Physical Object	POBJ_ID reference into PHYS_OBJ table
PP_DSS	Redbrick DATA SEGS	ERWTXTID reference to text string in ERW_TEXT table

Table E – Physical Property Codes (continued)

PROP_TYPE	PROP_TYPE	PROP_VAL
Valid Values	Description	Valid Values
PP_FBK	FALLBACK	ERWTXTID reference to text string in ERW_TEXT table
PP_FNM	WATCOM FILE NAME	ERWTXTID reference to text string in ERW_TEXT table
PP_FRG	FREELIST_GROUPS	ERWTXTID reference to text string in ERW_TEXT table
PP_FRL	FREELISTS	ERWTXTID reference to text string in ERW_TEXT table
PP_GSY	Informix GEN SYN	1 or 0
PP_IGN	Access Ignore Nulls	1 or 0
PP_INI	INITIAL	ERWTXTID reference to text string in ERW_TEXT table
PP_INT	INITRANS	ERWTXTID reference to text string in ERW_TEXT table
PP_IS	Redbrick INDEX SEGS	ERWTXTID reference to text string in ERW_TEXT table
PP_IT	DB2 INDEX TYPE	ERWTXTID reference to text string in ERW_TEXT table
PP_ITB	DB2/2 INDEX TABLESPACE	POBJ_ID reference into PHYS_OBJ table
PP_JRN	JOURNAL	ERWTXTID reference to text string in ERW_TEXT table
PP_LBL	PROG TABLE LABEL	ERWTXTID reference to text string in ERW_TEXT table
PP_LKM	DB2 LOCKMAX	ERWTXTID reference to text string in ERW_TEXT table
PP_LTB	DB2/2 LONG TABLESPACE	POBJ_ID reference into PHYS_OBJ table
PP_MNE	MINEXTENTS	ERWTXTID reference to text string in ERW_TEXT table
PP_MXE	MAXEXTENTS	ERWTXTID reference to text string in ERW_TEXT table
PP_MXR	Redbrick MAXROWS	ERWTXTID reference to text string in ERW_TEXT table
PP_MXS	Redbrick MAXSEGMENTS	ERWTXTID reference to text string in ERW_TEXT table
PP_MXT	MAXTRANS	ERWTXTID reference to text string in ERW_TEXT table
PP_NXT	NEXT	ERWTXTID reference to text string in ERW_TEXT table
PP_OPT	OPTIMAL	ERWTXTID reference to text string in ERW_TEXT table
PP_OWN	OWNER	ERWTXTID reference to text string in ERW_TEXT table
PP_PCF	PCTFREE	ERWTXTID reference to text string in ERW_TEXT table
PP_PCI	PCTINCREASE	ERWTXTID reference to text string in ERW_TEXT table
PP_PCU	PCTUSED	ERWTXTID reference to text string in ERW_TEXT table
PP_PIN	Redbrick PRIMARY INDEX IN	POBJ_ID reference into PHYS_OBJ table
PP_PLL	PARALLEL	1 or 0
PP_PSS	Redbrick PK SEGS	ERWTXTID reference to text string in ERW_TEXT table
PP_RBS	Reference to a ROLLBACK_SEGMENT Physical Object	POBJ_ID reference into PHYS_OBJ table
PP_RID	Informix WITH ROWIDS	1 or 0
PP_SDB	Informix SYN DB	ERWTXTID reference to text string in ERW_TEXT table
PP_SIZ	SIZE	ERWTXTID reference to text string in ERW_TEXT table
PP_SPL	SPOOL	1 or 0
PP_SSS	Redbrick STAR SEGS	ERWTXTID reference to text string in ERW_TEXT table
PP_STO	STORAGE	ERWTXTID reference to text string in ERW_TEXT table

Table E – Physical Property Codes (continued)

PROP_TYPE Valid Values	PROP_TYPE Description	PROP_VAL Valid Values
PP_STR	Redbrick RB STORAGES	ERWTXTID reference to text string in ERW_TEXT table
PP_STY	Informix SYN TYPE	ERWTXTID reference to text string in ERW_TEXT table
PP_TDB	TERADATABASE	POBJ_ID reference into PHYS_OBJ table
PP_TMP	TEMPORARY	1 or 0
PP_TSP	Reference to a TABLESPACE Physical Object	POBJ_ID reference into PHYS_OBJ table
PP_TT	Redbrick TABLE TYPE	ERWTXTID reference to text string in ERW_TEXT table
PP_TTB	DB2/2 TABLE TABLESPACE	POBJ_ID reference into PHYS_OBJ table
PP_WDJ	WITH_DEFAULT_JOURNAL	ERWTXTID reference to text string in ERW_TEXT table
PP_WFF	Redbrick WITH FILLFACTOR	ERWTXTID reference to text string in ERW_TEXT table
PP_WJT	WITH_JOURNAL_TABLE	ERWTXTID reference to text string in ERW_TEXT table
PP_WNN	DB2 WHERE NOT NULL	1 or 0
TP_CMT	PowerBuilder pbt_cmnt	ERWTXTID reference to text string in ERW_TEXT table
TP_DFC	PowerBuilder pbd_fchr	ERWTXTID reference to text string in ERW_TEXT table
TP_DFF	PowerBuilder pbd_ffce	ERWTXTID reference to text string in ERW_TEXT table
TP_DFH	PowerBuilder pbd_fhgt	ERWTXTID reference to text string in ERW_TEXT table
TP_DFI	PowerBuilder pbd_fitl	ERWTXTID reference to text string in ERW_TEXT table
TP_DFP	PowerBuilder pbd_fptc	ERWTXTID reference to text string in ERW_TEXT table
TP_DFU	PowerBuilder pbd_funl	ERWTXTID reference to text string in ERW_TEXT table
TP_DFW	PowerBuilder pbd_fwgt	ERWTXTID reference to text string in ERW_TEXT table
TP_HFC	PowerBuilder pbh_fchr	ERWTXTID reference to text string in ERW_TEXT table
TP_HFF	PowerBuilder pbh_ffce	ERWTXTID reference to text string in ERW_TEXT table
TP_HFH	PowerBuilder pbh_fhgt	ERWTXTID reference to text string in ERW_TEXT table
TP_HFI	PowerBuilder pbh_fitl	ERWTXTID reference to text string in ERW_TEXT table
TP_HFP	PowerBuilder pbh_fptc	ERWTXTID reference to text string in ERW_TEXT table
TP_HFU	PowerBuilder pbh_funl	ERWTXTID reference to text string in ERW_TEXT table
TP_HFW	PowerBuilder pbh_fwgt	ERWTXTID reference to text string in ERW_TEXT table
TP_LFC	PowerBuilder pbl_fchr	ERWTXTID reference to text string in ERW_TEXT table
TP_LFF	PowerBuilder pbl_ffce	ERWTXTID reference to text string in ERW_TEXT table
TP_LFH	PowerBuilder pbl_fhgt	ERWTXTID reference to text string in ERW_TEXT table
TP_LFI	PowerBuilder pbl_fitl	ERWTXTID reference to text string in ERW_TEXT table
TP_LFP	PowerBuilder pbl_fptc	ERWTXTID reference to text string in ERW_TEXT table
TP_LFU	PowerBuilder pbl_funl	ERWTXTID reference to text string in ERW_TEXT table
TP_LFW	PowerBuilder pbl_fwgt	ERWTXTID reference to text string in ERW_TEXT table

Table F – Template Types

This table of valid values is used in the following metamodel tables: ESPTMPLT, GBL_TMPL, REL_TMPL, and TEMPLATE.

TMPLT_TYPE Valid Values	TMPLT_TYPE Description
SP_ENT	Stored Procedure template type Table
SP_EPO	Script template type Table Post
SP_EPR	Script template type Table Pre
SP_SCH	Stored Procedure template type schema
SP_SPO	Script template type Schema Post
SP_SPR	Script template type Schema Pre
TR_BI	Trigger template type Built-In
TR_US	Trigger template type User

Table G – Trigger Types

This table of valid values is used in the ENT_TRIG metamodel table.

TRIG_TYPE Valid Values	TRIG_TYPE Description
TR_DI	Trigger type DELETE,INSERT
TR_DIU	Trigger type DELETE,INSERT,UPDATE
TR_DU	Trigger type DELETE,UPDATE
TR_I	Trigger type INSERT
TR_D	Trigger type DELETE
TR_IU	Trigger type INSERT,UPDATE
TR_U	Trigger type UPDATE

Table H – Trigger Fire Valid Values

This table of valid values is used in the ENT_TRIG metamodel table.

TRIG_FIRE Valid Values	TRIG_FIRE Description
TR_AF	Trigger fire AFTER
TR_BF	Trigger fire BEFORE

Table I – Trigger Scope Valid Values

This table of valid values is used in the ENT_TRIG metamodel table.

TRIG_SCOPE Valid Values	TRIG_SCOPE Description
TR_RO	Trigger scope ROW
TR_TB	Trigger scope TABLE

Table J – Diagram Options

This table of valid values is used in the DIAG_OPT metamodel table.

OPT_NAME	OPT_NAME	OPT_VAL
Valid Values	**Description**	**Valid Values**
DO_ACD	Default Non-Identifying Nulls-Allowed Rel Child Delete Rule	See Relationship Child Delete Rule
DO_ACI	Default Non-Identifying Nulls-Allowed Rel Child Insert Rule	See Relationship Child Insert Rule
DO_ACU	Default Non-Identifying Nulls-Allowed Rel Child Update Rule	See Relationship Child Update Rule
DO_AI	Internal Parameter for Attribute Inheritance	Internal Value
DO_AID	Maximum Attribute Id	Integer
DO_AK	Show Key Groups	1 or 0
DO_APD	Default Non-Identifying Nulls-Allowed Rel Parent Delete Rule	See Relationship Parent Delete Rule
DO_API	Default Non-Identifying Nulls-Allowed Rel Parent Insert Rule	See Relationship Child Delete Rule
DO_APU	Default Non-Identifying Nulls-Allowed Rel Parent Update Rule	See Relationship Parent Update Rule
DO_AR	Show Attribute Role Names	1 or 0
DO_C01	Internal Color Related Value	Internal Value
DO_C02	Internal Color Related Value	Internal Value
DO_C03	Internal Color Related Value	Internal Value
DO_C04	Internal Color Related Value	Internal Value
DO_C05	Internal Color Related Value	Internal Value
DO_C06	Internal Color Related Value	Internal Value
DO_C07	Internal Color Related Value	Internal Value
DO_C08	Internal Color Related Value	Internal Value
DO_C09	Internal Color Related Value	Internal Value
DO_C10	Internal Color Related Value	Internal Value
DO_C11	Internal Color Related Value	Internal Value
DO_C12	Internal Color Related Value	Internal Value
DO_C13	Internal Color Related Value	Internal Value
DO_C14	Internal Color Related Value	Internal Value
DO_C15	Internal Color Related Value	Internal Value
DO_C16	Internal Color Related Value	Internal Value
DO_CBK	Internal Color Related Value	Internal Value
DO_CEB	Internal Color Related Value	Internal Value
DO_CEL	Internal Color Related Value	Internal Value
DO_CFK	Internal Color Related Value	Internal Value
DO_COE	Internal Color Related Value	Internal Value

Table J – Diagram Options (continued)

OPT_NAME	OPT_NAME	OPT_VAL
Valid Values	**Description**	**Valid Values**
DO_COP	Internal Color Related Value	Internal Value
DO_COR	Internal Color Related Value	Internal Value
DO_COS	Internal Color Related Value	Internal Value
DO_COT	Internal Color Related Value	Internal Value
DO_COW	Internal Color Related Value	Internal Value
DO_CSB	Internal Color Related Value	Internal Value
DO_DD	Default Column Datatype	String specifying Default Datatype
DO_DG	Internal Parameter	0
DO_DID	Maximum Domain Id	Integer
DO_DL	Diagram Display Level	Entity Level: 0 Attribute Level: 1 Definition Level: 3 Primary Key Level: 5 Physical Order Level: 6 Icon Level: 7
DO_DLM	Interbase Trigger Delimiter	String
DO_DN	Default Null-Option for Non-Key Columns	Nulls Allowed: 0
		No-Nulls: 1
DO_EE	Default Entity Editor	Internal Value
DO_EH	Entity Height	Integer value specifying entity width in characters
DO_EID	Maximum Entity Id	Integer
DO_EW	Entity Width	Integer value specifying entity width in characters
DO_FFK	Internal Font Related Value	Internal Value
DO_FK	Show Foreign Key Designator	1 or 0
DO_FOW	Internal Font Related Value	Internal Value
DO_FTE	Internal Font Related Value	Internal Value
DO_FTP	Internal Font Related Value	Internal Value
DO_FTR	Internal Font Related Value	Internal Value
DO_FTS	Internal Font Related Value	Internal Value
DO_FTT	Internal Font Related Value	Internal Value
DO_ICD	Default Identifying Rel Child Delete Rule	See Relationship Child Delete Rule
DO_ICI	Default Identifying Rel Child Insert Rule	See Relationship Child Insert Rule
DO_ICU	Default Identifying Rel Child Update Rule	See Relationship Child Update Rule
DO_IPD	Default Identifying Rel Parent Delete Rule	See Relationship Parent Delete Rule
DO_IPI	Default Identifying Rel Parent Insert Rule	See Relationship Parent Insert Rule
DO_IPU	Default Identifying Rel Parent Update Rule	See Relationship Parent Update Rule

Table J – Diagram Options (continued)

OPT_NAME	OPT_NAME	OPT_VAL
Valid Values	**Description**	**Valid Values**
DO_LG	Internal Parameter for Automatic layout	INTEGER
DO_NCD	Default Non-Identifying No Nulls Rel Child Delete Rule	See Relationship Child Delete Rule
DO_NCI	Default Non-Identifying No Nulls Rel Child Insert Rule	See Relationship Child Insert Rule
DO_NCU	Default Non-Identifying No Nulls Rel Child Update Rule	See Relationship Child Update Rule
DO_NN	Show Column Null Option	1 or 0
DO_NPD	Default Non-Identifying No Nulls Rel Parent Delete Rule	See Relationship Parent Delete Rule
DO_NPI	Default Non-Identifying No Nulls Rel Parent Insert Rule	See Relationship Child Delete Rule
DO_NPU	Default Non-Identifying No Nulls Rel Parent Update Rule	See Relationship Parent Update Rule
DO_NTT	Logical Modeling Methodology Notation	IDEF1X: 0 IE: 1
DO_PBL	Default PowerBuilder Library	File Name String
DO_PCD	Progress Create Directory	Directory Path String
DO_PDD	Progress Delete Directory	Directory Path String
DO_PF	Internal Parameter for several Display Preferences	Internal Value
DO_PF2	Internal Display Related Parameter	Internal Value
DO_PG	Show Page Grid	1 or 0
DO_PID	Maximum Physical Object Id	Integer
DO_PNT	Physical Notation Type	IDEF1X: 0 IE: 1
DO_PUD	Progress Use Dump File	File Path String
DO_PWD	Progress Write Directory	Directory Path String
DO_RC	Show Relationship Cardinality	1 or 0
DO_RE	Default Relationship Editor	Internal Value
DO_RI	Show Relationship Referential Integrity	1 or 0
DO_RID	Maximum Relationship Id	Integer
DO_RV	Show Relationship Verb Phrase	1 or 0
DO_SCD	Default Subtype Rel Child Delete Rule	See Relationship Child Delete Rule
DO_SCI	Default Subtype Rel Child Insert Rule	See Relationship Child Insert Rule
DO_SCU	Default Subtype Rel Child Update Rule	See Relationship Child Update Rule
DO_SD	Show Physical Level	1 or 0
DO_SHX	Entity Horizontal Shadow Size	Integer
DO_SHY	Entity Vertical Shadow Size	Integer
DO_SID	Maximum Stored Procedure Template Id	Integer
DO_SON	Show Entity Shadows	1 or 0
DO_SOO	Database Synchronization Owned Only Name	String Specifying Owner Name

Table J – Diagram Options (continued)

OPT_NAME	OPT_NAME	OPT_VAL
Valid Values	Description	Valid Values
DO_SPD	Default Subtype Rel Parent Delete Rule	See Relationship Parent Delete Rule
DO_SPI	Default Subtype Rel Parent Insert Rule	See Relationship Child Delete Rule
DO_SPU	Default Subtype Rel Parent Update Rule	See Relationship Parent Update Rule
DO_SR	Display Sub-Category Discriminator Name	0 or 1
DO_ST	Show Column Datatypes	1 or 0
DO_TC	Target Client	PowerBuilder: 1 SqlWindows: 2 VisualBasic: 3
DO_TD	Target DBMS	"TD_DB": DB2 "TD_NW": NETWARE "TD_SB": SQLBASE "TD_SS": SQLSERVER "TD_SY": SYBASE "TD_OR": ORACLE "TD_TE": TERADATA "TD_IN": INGRES "TD_SM": SMARTSTAR "TD_IX": INFORMIX "TD_RD": RDB "TD_AD": ADABAS "TD_D3": DBASE3 "TD_FP": FOXPRO "TD_D4": DBASE4 "TD_CL": CLIPPER "TD_AS": AS400 "TD_PG": PROGRESS "TD_WC": WATCOM "TD_RB": REDBRICK "TD_D2": DB2_2
DO_TID	Maximum Pre/Post Script Id	Integer
DO_TM	DBMS Minor Version	Integer specifying minor target DBMS version
DO_TV	Target DBMS Version	Integer Value Specifying Major Version Number
DO_UN	Unique Name Option	Allow Non-Unique: 0 Rename Non-Unique: 1 Ask on Non-Unique: 2 Dis-Allow Non-Unique: 3
DO_VEL	Maximum View Expression Length	Integer
DO_VMN	Default View Editor	Internal Value
DO_VPS	Split Relationship Verb Phrases	1 or 0
DO_VW	Internal Parameter for several Display Preferences	Internal Value
DO_WDT	Datatype Mapping Table	Internal Value
DO_WIM	Index Name Macro	String
DO_WTE	Default Table Editor	Internal Value
DO_XL	Internal Parameter for Automatic layout	INTEGER
DO_YL	Internal Parameter for Automatic layout	INTEGER

Table K – Stored Display Options

This table of valid values is used in the DIAG_OPT metamodel table.

OPT_NAME	OPT_NAME	OPT_VAL
Valid Values	**Description**	**Valid Values**
DO_DDL	Display Level	EntityLevel: 0 AttributeLevel: 1 DefinitionLevel: 3 PrimaryKeyLevel: 5 PhysicalOrderLevel: 6 IconLevel: 7
DO_DSS	Display Physical Information	1 or 0
DO_DSD	Display Column Datatypes	1 or 0
DO_DNN	Display Null Option	1 or 0
DO_DFK	Display Foreign Key Designator	1 or 0
DO_DSN	Display Discriminator Name for Sub Categories	1 or 0
DO_DZL	Display Zoom Level	Zoom factor. 0 means 100%
DO_DVP	Display Relationship Verb Phrase	1 or 0
DO_DSC	Display Relationship Cardinality	1 or 0
DO_DRI	Display Relationship Referential Integrity	1 or 0
DO_DRN	Display Attribute Role Names	1 or 0
DO_DAK	Display Key Groups	1 or 0
DO_DHR	Hide Relationships	1 or 0
DO_DDR	Show Dangling Relationships	1 or 0
DO_DSX	Entity Horizontal Shadow	Integer
DO_DSY	Entity Vertical Shadow	Integer
DO_DSO	Display Shadows	1 or 0
DO_HSP	Horizontal Scroll Position	Integer
DO_VSP	Vertical Scroll Position	Integer
DO_SVS	Display Views	1 or 0
DO_SVR	Display View Relationships	1 or 0
DO_SVD	Display View Column Data Types	1 or 0
DO_SVN	Display View Column Null Options	1 or 0
DO_SVM	Display View Column Aliases	1 or 0
DO_SUN	Display Ungenerated Columns	1 or 0
DO_SMA	Display Migrated Attributes	1 or 0
DO_PDL	Physical Display Level	EntityLevel: 0 AttributeLevel: 1 DefinitionLevel: 3 PrimaryKeyLevel: 5 PhysicalOrderLevel: 6 IconLevel: 7

Table K – Stored Display Options (continued)

OPT_NAME	OPT_NAME	OPT_VAL
Valid Values	**Description**	**Valid Values**
DO_SPR	Display Physical Relationship Names	1 or 0
DO_SPG	Display Relationship Referential Integrity in Physical Mode	1 or 0
DO_SPC	Display Cardinality in Physical Mode	1 or 0
DO_SPA	Display Key Groups in Physical Mode	1 or 0
DO_LDT	Display Logical Datatypes	1 or 0
DO_LFK	Display Logical Foreign Key Designators	1 or 0
DO_P00	Internal Printer Information Parameter	Internal Value
DO_P01	Internal Printer Information Parameter	Internal Value
DO_P02	Internal Printer Information Parameter	Internal Value
DO_P03	Internal Printer Information Parameter	Internal Value
DO_P04	Internal Printer Information Parameter	Internal Value
DO_P05	Internal Printer Information Parameter	Internal Value
DO_P06	Internal Printer Information Parameter	Internal Value
DO_P07	Internal Printer Information Parameter	Internal Value
DO_P08	Internal Printer Information Parameter	Internal Value
DO_P09	Internal Printer Information Parameter	Internal Value
DO_P10	Internal Printer Information Parameter	Internal Value
DO_P11	Internal Printer Information Parameter	Internal Value
DO_P12	Internal Printer Information Parameter	Internal Value
DO_P13	Internal Printer Information Parameter	Internal Value
DO_P14	Internal Printer Information Parameter	Internal Value
DO_P15	Internal Printer Information Parameter	Internal Value
DO_ARL	Automatically layout relationships	1 or 0

Table L – Relationship Type Valid Values

This table of valid values is used in the following metamodel tables: ENT_ATT, INDX_MEM, INDX_MPP, REL_PHYS, REL_TMPL, and RELATION.

REL_TYPE Valid Values	REL_TYPE Description
RT_DR	Relationship Type DERIVED_RELATION
RT_ID	Relationship Type ID_RELATION
RT_MM	Relationship Type MANYMANY
RT_NI	Relationship Type NONID_RELATION
RT_SC	Relationship Type SUBCAT_RELATION

Table M – Attribute Type Codes

This table of valid values is used in the ENT_ATT metamodel table.

ATT_TYPE Valid Values	ATT_TYPE Description
AT_DR	Attribute Type DERIVED_COLUMN
AT_NK	Attribute Type NONKEY
AT_PK	Attribute Type KEY

Table N – Entity Type Valid Values

This table of valid values is used in the RELATION metamodel table.

PENT_TYPE CENT_TYPE Valid Values	PENT_TYPE CENT_TYPE Description
ET_DE	Entity Type DEP_ENTITY
ET_DR	Entity Type DERIVED_ENTITY
ET_IE	Entity Type INDEP_ENTITY

Table O – Relationship Cardinality

This table of valid values is used in the RELATION metamodel table.

CARDINAL Valid Values	CARDINAL Description
CT_PR	Relationship Cardinality One or More
CT_ZM	Relationship Cardinality Zero, One, Or More
CT_ZR	Relationship Cardinality Zero or One
Character representation of a number	Relationship Cardinality Zero or One to Exactly n; (e.g. n = "2")

Table P – Parent Delete RI Valid Values

This table of valid values is used in the RELATION metamodel table.

PAR_DEL Valid Values	PAR_DEL Description
RI_NO	Referential Integrity None
DR_CA	Referential Integrity On Parent Delete Cascade
DR_RE	Referential Integrity On Parent Delete Resctrict
DR_SD	Referential Integrity On Parent Delete Set Default
DR_SN	Referential Integrity On Parent Delete Set Null

Table Q – Parent Update RI Valid Values

This table of valid values is used in the RELATION metamodel table.

PAR_UPD Valid Values	PAR_UPD Description
RI_NO	Referential Integrity None
UR_CA	Referential Integrity On Parent Update Cascade
UR_RE	Referential Integrity On Parent Update Resctrict
UR_SD	Referential Integrity On Parent Update Set Default
UR_SN	Referential Integrity On Parent Update Set Null

Table R – Child Insert RI Valid Values

This table of valid values is used in the RELATION metamodel table.

CHILD_INS	CHILD_INS
Valid Values	Description
RI_NO	Referential Integrity None
IR_CA	Referential Integrity On Child Insert Cascade
IR_RE	Referential Integrity On Child Insert Resctrict
IR_SD	Referential Integrity On Child Insert Set Default
IR_SN	Referential Integrity On Child Insert Set Null

Table S – Child Delete RI Valid Values

This table of valid values is used in the RELATION metamodel table.

CHILD_DEL	CHILD_DEL
Valid Values	Description
RI_NO	Referential Integrity None
DR_CCA	Referential Integrity On Child Delete Cascade
DR_CRE	Referential Integrity On Child Delete Resctrict
DR_CSD	Referential Integrity On Child Delete Set Default
DR_CSN	Referential Integrity On Child Delete Set Null

Table T – Child Update RI Valid Values

This table of valid values is used in the RELATION metamodel table.

CHILD_UPD	CHILD_UPD
Valid Values	Description
RI_NO	Referential Integrity None
UR_CCA	Referential Integrity On Child Update Cascade
UR_CRE	Referential Integrity On Child Update Resctrict
UR_CSD	Referential Integrity On Child Update Set Default
UR_CSN	Referential Integrity On Child Update Set Null

Table U – Parent Insert RI Valid Values

This table of valid values is used in the RELATION metamodel table.

PAR_INS Valid Values	PAR_INS Description
RI_NO	Referential Integrity None
IR_PCA	Referential Integrity On Parent Insert Cascade
IR_PRE	Referential Integrity On Parent Insert Resctrict
IR_PSD	Referential Integrity On Parent Insert Set Default
IR_PSN	Referential Integrity On Parent Insert Set Null

Table V – Null Option Valid Values

This table of valid values is used in the RELATION metamodel table.

NULL_RULE Valid Values	NULL_RULE Description
NR_NA	Nulls Allowed
NR_NN	No Nulls

Table W – Subcategory Types

This table of valid values is used in the SUBTYPE metamodel table.

SUBT_TYPE Valid Values	SUBT_TYPE Description
ST_CS	Subcategory Type COMPLETE
ST_IS	Subcategory Type INCOMPLETE

Table X – Global Template Purpose

This table of valid values is used in the GBL_TMPL metamodel table.

TMPLT_PURP Valid Values	TMPLT_PURP Description
DR_CA	Referential Integrity On Parent Delete Cascade
DR_RE	Referential Integrity On Parent Delete Resctrict
DR_SD	Referential Integrity On Parent Delete Set Default
DR_SN	Referential Integrity On Parent Delete Set Null
DR_CCA	Referential Integrity On Child Delete Cascade
DR_CRE	Referential Integrity On Child Delete Resctrict
DR_CSD	Referential Integrity On Child Delete Set Default
DR_CSN	Referential Integrity On Child Delete Set Null
IR_PCA	Referential Integrity On Parent Insert Cascade
IR_PRE	Referential Integrity On Parent Insert Resctrict
IR_PSD	Referential Integrity On Parent Insert Set Default
IR_PSN	Referential Integrity On Parent Insert Set Null
IR_CA	Referential Integrity On Child Insert Cascade
IR_RE	Referential Integrity On Child Insert Resctrict
IR_SD	Referential Integrity On Child Insert Set Default
IR_SN	Referential Integrity On Child Insert Set Null
UR_CCA	Referential Integrity On Child Update Cascade
UR_CRE	Referential Integrity On Child Update Resctrict
UR_CSD	Referential Integrity On Child Update Set Default
UR_CSN	Referential Integrity On Child Update Set Null
UR_CA	Referential Integrity On Parent Update Cascade
UR_RE	Referential Integrity On Parent Update Resctrict
UR_SD	Referential Integrity On Parent Update Set Default
UR_SN	Referential Integrity On Parent Update Set Null
TR_CB	Trigger template for Custom Body
TR_CF	Trigger template for Custom Footer
TR_CH	Trigger template for Custom Header
TR_DF	Trigger template for Delete Footer
TR_DH	Trigger template for Delete Header
TR_IF	Trigger template for Insert Footer
TR_IH	Trigger template for Insert Header
TR_UF	Trigger template for Update Footer
TR_UH	Trigger template for Update Header

Table Y – Index Physical Property Codes

This table of valid values is used in the INDXPROP metamodel table.

INX_PV_TP	INX_PV_TP	IDX_PV_POBJ	IDX_PV_VAL
Valid Values	Description	Valid Values	Valid Values
D2_DBP	DB2 DBBUFFERPOOL		ERWTXTID reference to text string in ERW_TEXT table
D2_DBS	DB2 DATABASE Physical Object Reference	POBJ_ID reference into PHYS_OBJ table	
D2_DRS	DB2 ROSHARE		ERWTXTID reference to text string in ERW_TEXT table
D2_DSG	DB2 DATABASE STOGROUP	POBJ_ID reference into PHYS_OBJ table	
D2_ICL	DB2 INDEX CLUSTER		1 or 0
D2_IDF	DB2 INDEX DEFER		1 or 0
D2_ISP	DB2 INDEX SUBPAGES		ERWTXTID reference to text string in ERW_TEXT table
D2_PAU	DB2 AUDIT		1 or 0
D2_PDB	DB2 IN DATABASE Physical Object Reference	POBJ_ID reference into PHYS_OBJ table	
D2_PDC	DB2 DATACAPTURE		ERWTXTID reference to text string in ERW_TEXT table
D2_PEP	DB2 EDITPROC		ERWTXTID reference to text string in ERW_TEXT table
D2_POB	DB2 OBID		ERWTXTID reference to text string in ERW_TEXT table
D2_PTS	DB2 IN TABLESPACE	POBJ_ID reference into PHYS_OBJ table	
D2_TSP	DB2 TABLESPACE	POBJ_ID reference into PHYS_OBJ table	
EP_ALR	ALLOW_DUP_ROW		ERWTXTID reference to text string in ERW_TEXT table
EP_DPR	DUP_ROW		ERWTXTID reference to text string in ERW_TEXT table
EP_FIL	FILLFACTOR		ERWTXTID reference to text string in ERW_TEXT table
EP_IGD	IGNORE_DUP_KEY		ERWTXTID reference to text string in ERW_TEXT table
EP_IGR	IGNORE_DUP_ROW		ERWTXTID reference to text string in ERW_TEXT table
EP_MXR	MAX ROWS		ERWTXTID reference to text string in ERW_TEXT table
EP_SEG	SEGMENT	POBJ_ID reference into PHYS_OBJ table	
EP_SRR	STAR		1 or 0
EP_SRT	SORTED_DATA		ERWTXTID reference to text string in ERW_TEXT table
IP_ABB	ABBREVIATED		1 or 0
IP_ACT	ACTIVE		1 or 0
IP_CLU	CLUSTER		1 or 0
IP_CMT	INDEX COMMENT		ERWTXTID reference to text string in ERW_TEXT table
IP_CPT	COMPACT		1 or 0
IP_DEF	INDEX DEFINITION		ERWTXTID reference to text string in ERW_TEXT table
IP_EXP	KEY EXPRESSION		ERWTXTID reference to text string in ERW_TEXT table
IP_FIL	FILE NAME		ERWTXTID reference to text string in ERW_TEXT table

Table Y – Index Physical Property Codes (continued)

INX_PV_TP Valid Values	INX_PV_TP Description	IDX_PV_POBJ Valid Values	IDX_PV_VAL Valid Values
IP_FOR	FOR CONDITION		ERWTXTID reference to text string in ERW_TEXT table
IP_LNM	INDEX LOGICAL NAME		ERWTXTID reference to text string in ERW_TEXT table
IP_MDX	MDX		1 or 0
IP_NAM	NAME		ERWTXTID reference to text string in ERW_TEXT table
IP_NOT	INDEX NOTE		ERWTXTID reference to text string in ERW_TEXT table
IP_SRT	SORT ORDER		ERWTXTID reference to text string in ERW_TEXT table
IP_SZR	SIZE ROWS		ERWTXTID reference to text string in ERW_TEXT table
IP_UNQ	UNIQUE		1 or 0
IP_WNN	WHERE NOT NULL		1 or 0
IP_WRD	WORD INDEX		1 or 0
IX_ESZ	Informix EXTENT SIZE		ERWTXTID reference to text string in ERW_TEXT table
IX_ESZ	Informix EXTENT SIZE		ERWTXTID reference to text string in ERW_TEXT table
IX_FEX	Informix FRAGMENT BY EXPRESSION		ERWTXTID reference to text string in ERW_TEXT table
IX_FEX	Informix FRAGMENT BY EXPRESSION		ERWTXTID reference to text string in ERW_TEXT table
IX_FRR	Informix FRAGMENT BY ROUND ROBIN IN		ERWTXTID reference to text string in ERW_TEXT table
IX_FRR	Informix FRAGMENT BY ROUND ROBIN IN		ERWTXTID reference to text string in ERW_TEXT table
IX_IND	Informix IN DBSPACE	POBJ_ID reference into PHYS_OBJ table	
IX_IND	Informix IN DBSPACE	POBJ_ID reference into PHYS_OBJ table	
IX_INP	Informix IN PATH		ERWTXTID reference to text string in ERW_TEXT table
IX_INP	Informix IN PATH		ERWTXTID reference to text string in ERW_TEXT table
IX_LKM	Informix LOCK MODE		ERWTXTID reference to text string in ERW_TEXT table
IX_LKM	Informix LOCK MODE		ERWTXTID reference to text string in ERW_TEXT table
IX_NSZ	Informix NEXT SIZE		ERWTXTID reference to text string in ERW_TEXT table
IX_NSZ	Informix NEXT SIZE		ERWTXTID reference to text string in ERW_TEXT table
IX_REM	Informix REMAINDER IN	POBJ_ID reference into PHYS_OBJ table	
IX_REM	Informix REMAINDER IN	POBJ_ID reference into PHYS_OBJ table	
IX_STM	Informix STORAGE OPTION		ERWTXTID reference to text string in ERW_TEXT table
IX_STM	Informix STORAGE OPTION		ERWTXTID reference to text string in ERW_TEXT table
PP_ACT	ACCOUNT	POBJ_ID reference into PHYS_OBJ table	
PP_ASP	AS_PERMANENT		1 or 0

Table Y – Index Physical Property Codes (continued)

INX_PV_TP Valid Values	INX_PV_TP Description	IDX_PV_POBJ Valid Values	IDX_PV_VAL Valid Values
PP_CCH	CACHE		1 or 0
PP_CLS	CLUSTERED		1 or 0
PP_CMT	Informix SYN CMT		ERWTXTID reference to text string in ERW_TEXT table
PP_CNS	CONSTRAINT		ERWTXTID reference to text string in ERW_TEXT table
PP_COM	COMMENT		ERWTXTID reference to text string in ERW_TEXT table
PP_DIN	Redbrick DATA IN	POBJ_ID reference into PHYS_OBJ table	
PP_DMP	PROGDUMP		ERWTXTID reference to text string in ERW_TEXT table
PP_DS	Redbrick DOMAIN SIZE		ERWTXTID reference to text string in ERW_TEXT table
PP_DSP	Reference to a DBSPACE Physical Object	POBJ_ID reference into PHYS_OBJ table	
PP_DSS	Redbrick DATA SEGS		ERWTXTID reference to text string in ERW_TEXT table
PP_FBK	FALLBACK		ERWTXTID reference to text string in ERW_TEXT table
PP_FNM	WATCOM FILE NAME		ERWTXTID reference to text string in ERW_TEXT table
PP_FRG	FREELIST_GROUPS		ERWTXTID reference to text string in ERW_TEXT table
PP_FRL	FREELISTS		ERWTXTID reference to text string in ERW_TEXT table
PP_GSY	Informix GEN SYN		1 or 0
PP_HSH	HASH_IS		ERWTXTID reference to text string in ERW_TEXT table
PP_HSN	HASNKEYS		ERWTXTID reference to text string in ERW_TEXT table
PP_IDX	INDEX		ERWTXTID reference to text string in ERW_TEXT table
PP_IGN	Access Ignore Nulls		1 or 0
PP_INI	INITIAL		ERWTXTID reference to text string in ERW_TEXT table
PP_INT	INITRANS		ERWTXTID reference to text string in ERW_TEXT table
PP_IS	Redbrick INDEX SEGS		ERWTXTID reference to text string in ERW_TEXT table
PP_IT	DB2 INDEX TYPE		ERWTXTID reference to text string in ERW_TEXT table
PP_ITB	DB2/2 INDEX TABLESPACE	POBJ_ID reference into PHYS_OBJ table	
PP_JRN	JOURNAL		ERWTXTID reference to text string in ERW_TEXT table
PP_LBL	PROG TABLE LABEL		ERWTXTID reference to text string in ERW_TEXT table
PP_LKM	DB2 LOCKMAX		ERWTXTID reference to text string in ERW_TEXT table
PP_LTB	DB2/2 LONG TABLESPACE	POBJ_ID reference into PHYS_OBJ table	
PP_MNE	MINEXTENTS		ERWTXTID reference to text string in ERW_TEXT table
PP_MXE	MAXEXTENTS		ERWTXTID reference to text string in ERW_TEXT table
PP_MXR	Redbrick MAXROWS		ERWTXTID reference to text string in ERW_TEXT table
PP_MXS	Redbrick MAXSEGMENTS		ERWTXTID reference to text string in ERW_TEXT table
PP_MXT	MAXTRANS		ERWTXTID reference to text string in ERW_TEXT table
PP_NOS	NOSORT		1 or 0

Table Y – Index Physical Property Codes (continued)

INX_PV_TP Valid Values	INX_PV_TP Description	IDX_PV_POBJ Valid Values	IDX_PV_VAL Valid Values
PP_NXT	NEXT		ERWTXTID reference to text string in ERW_TEXT table
PP_OPT	OPTIMAL		ERWTXTID reference to text string in ERW_TEXT table
PP_OWN	OWNER		ERWTXTID reference to text string in ERW_TEXT table
PP_PCF	PCTFREE		ERWTXTID reference to text string in ERW_TEXT table
PP_PCI	PCTINCREASE		ERWTXTID reference to text string in ERW_TEXT table
PP_PCU	PCTUSED		ERWTXTID reference to text string in ERW_TEXT table
PP_PIN	Redbrick PRIMARY INDEX IN	POBJ_ID reference into PHYS_OBJ table	
PP_PLL	PARALLEL		1 or 0
PP_PSS	Redbrick PK SEGS		ERWTXTID reference to text string in ERW_TEXT table
PP_RBS	Reference to a ROLLBACK_SEGMENT Physical Object	POBJ_ID reference into PHYS_OBJ table	
PP_RID	Informix WITH ROWIDS		1 or 0
PP_SDB	Informix SYN DB		ERWTXTID reference to text string in ERW_TEXT table
PP_SIZ	SIZE		ERWTXTID reference to text string in ERW_TEXT table
PP_SPL	SPOOL		1 or 0
PP_SSS	Redbrick STAR SEGS		ERWTXTID reference to text string in ERW_TEXT table
PP_STO	STORAGE		ERWTXTID reference to text string in ERW_TEXT table
PP_STR	Redbrick RB STORAGES		ERWTXTID reference to text string in ERW_TEXT table
PP_STY	Informix SYN TYPE		ERWTXTID reference to text string in ERW_TEXT table
PP_TDB	TERADATABASE	POBJ_ID reference into PHYS_OBJ table	
PP_TMP	TEMPORARY		1 or 0
PP_TSP	Reference to a TABLESPACE Physical Object	POBJ_ID reference into PHYS_OBJ table	
PP_TT	Redbrick TABLE TYPE		ERWTXTID reference to text string in ERW_TEXT table
PP_TTB	DB2/2 TABLE TABLESPACE	POBJ_ID reference into PHYS_OBJ table	
PP_WDJ	WITH_DEFAULT_JOURNAL		ERWTXTID reference to text string in ERW_TEXT table
PP_WFF	Redbrick WITH FILLFACTOR		ERWTXTID reference to text string in ERW_TEXT table
PP_WJT	WITH_JOURNAL_TABLE		ERWTXTID reference to text string in ERW_TEXT table
PP_WNN	DB2 WHERE NOT NULL		1 or 0
TP_CMT	PowerBuilder pbt_cmnt		ERWTXTID reference to text string in ERW_TEXT table
TP_DFC	PowerBuilder pbd_fchr		ERWTXTID reference to text string in ERW_TEXT table
TP_DFF	PowerBuilder pbd_ffce		ERWTXTID reference to text string in ERW_TEXT table
TP_DFH	PowerBuilder pbd_fhgt		ERWTXTID reference to text string in ERW_TEXT table

Table Y – Index Physical Property Codes (continued)

INX_PV_TP Valid Values	INX_PV_TP Description	IDX_PV_POBJ Valid Values	IDX_PV_VAL Valid Values
TP_DFI	PowerBuilder pbd_fitl		ERWTXTID reference to text string in ERW_TEXT table
TP_DFP	PowerBuilder pbd_fptc		ERWTXTID reference to text string in ERW_TEXT table
TP_DFU	PowerBuilder pbd_funl		ERWTXTID reference to text string in ERW_TEXT table
TP_DFW	PowerBuilder pbd_fwgt		ERWTXTID reference to text string in ERW_TEXT table
TP_HFC	PowerBuilder pbh_fchr		ERWTXTID reference to text string in ERW_TEXT table
TP_HFF	PowerBuilder pbh_ffce		ERWTXTID reference to text string in ERW_TEXT table
TP_HFH	PowerBuilder pbh_fhgt		ERWTXTID reference to text string in ERW_TEXT table
TP_HFI	PowerBuilder pbh_fitl		ERWTXTID reference to text string in ERW_TEXT table
TP_HFP	PowerBuilder pbh_fptc		ERWTXTID reference to text string in ERW_TEXT table
TP_HFU	PowerBuilder pbh_funl		ERWTXTID reference to text string in ERW_TEXT table
TP_HFW	PowerBuilder pbh_fwgt		ERWTXTID reference to text string in ERW_TEXT table
TP_LFC	PowerBuilder pbl_fchr		ERWTXTID reference to text string in ERW_TEXT table
TP_LFF	PowerBuilder pbl_ffce		ERWTXTID reference to text string in ERW_TEXT table
TP_LFH	PowerBuilder pbl_fhgt		ERWTXTID reference to text string in ERW_TEXT table
TP_LFI	PowerBuilder pbl_fitl		ERWTXTID reference to text string in ERW_TEXT table
TP_LFP	PowerBuilder pbl_fptc		ERWTXTID reference to text string in ERW_TEXT table
TP_LFU	PowerBuilder pbl_funl		ERWTXTID reference to text string in ERW_TEXT table
TP_LFW	PowerBuilder pbl_fwgt		ERWTXTID reference to text string in ERW_TEXT table

Table Z – Physical Object Type Codes

This table of valid values is used in the PHYS_OBJ metamodel table.

POBJ_TYPE Valid Values	POBJ_TYPE Description
D2_DBS	DB2 DATABASE
D2_STG	DB2 STOGROUP
EP_SEG	SEGMENT
PP_DBS	DATABASE
PP_RBS	ROLLBACK_SEGMENT
PP_TDB	TERADATA DATABASE
PP_TSP	TABLESPACE

Table AA – Property Codes

This table of valid values is used in the POBJPV metamodel table.

POBJPVTYP Valid Values	PVOBJTYP Description	POBJPVVAL Valid Values	POBJPVIVAL	POBJPVSVAL
D2_DBP	DB2 DBBUFFERPOOL			Character String
D2_DBS	DB2 DATABASE Physical Object Reference			POBJ_ID reference into PHYS_OBJ table
D2_DRS	DB2 ROSHARE			Character String
D2_DSG	DB2 DATABASE STOGROUP	POBJ_ID reference into PHYS_OBJ table		
D2_PAU	DB2 AUDIT		1 or 0	
D2_PDB	DB2 IN DATABASE Physical Object Reference	POBJ_ID reference into PHYS_OBJ table		
D2_PDC	DB2 DATACAPTURE			Character String
D2_PEP	DB2 EDITPROC			Character String
D2_POB	DB2 OBID			Character String
D2_PTS	DB2 IN TABLESPACE	POBJ_ID reference into PHYS_OBJ table		
D2_SPW	DB2 STOGROUP PASSWORD			Character String
D2_STG	DB2 STOGROUP			Character String
D2_SVC	DB2 STOGROUP VCAT			Character String
D2_SVL	DB2 STOGROUP VOLUMES			Character String
D2_TCL	DB2 TABLESPACE CLOSE		1 or 0	
D2_TCP	DB2 TABLESPACE COMPRESS		1 or 0	
D2_TDB	DB2 TABLESPACE DATABASE Reference	POBJ_ID reference into PHYS_OBJ table		
D2_TLS	DB2 TABLESPACE LOCKSIZE			Character String
D2_TNP	DB2 TABLESPACE NUMPARTS			Character String
D2_TPA	DB2 TABLESPACE PART			Character String
D2_TPW	DB2 TABLESPACE PASSWORD			Character String
D2_TSP	DB2 TABLESPACE	POBJ_ID reference into PHYS_OBJ table		
D2_TSZ	DB2 TABLESPACE SEGSIZE			Character String
D2_TUS	DB2 TABLESPACE USING			Character String
EP_MXR	MAX ROWS			Character String
EP_SEG	SEGMENT	POBJ_ID reference into PHYS_OBJ table		
EP_SRR	STAR		1 or 0	
IX_ESZ	Informix EXTENT SIZE			Character String
IX_FEX	Informix FRAGMENT BY EXPRESSION			Character String

Table AA – Property Codes (continued)

POBJPVTYP	PVOBJTYP	POBJPVVAL	POBJPVIVAL	POBJPVSVAL
Valid Values	**Description**	**Valid Values**		
IX_FRR	Informix FRAGMENT BY ROUND ROBIN IN			Character String
IX_IND	Informix IN DBSPACE	POBJ_ID reference into PHYS_OBJ table		
IX_INP	Informix IN PATH			Character String
IX_LKM	Informix LOCK MODE			Character String
IX_NSZ	Informix NEXT SIZE			Character String
IX_REM	Informix REMAINDER IN	POBJ_ID reference into PHYS_OBJ table		
IX_STM	Informix STORAGE OPTION			Character String
PP_ACT	ACCOUNT	POBJ_ID reference into PHYS_OBJ table		
PP_ARC	ARCHIVELOG		1 or 0	
PP_ASP	AS_PERMANENT		1 or 0	
PP_CCH	CACHE		1 or 0	
PP_CHR	CHARSET			Character String
PP_CMT	Informix SYN CMT			Character String
PP_CNS	CONSTRAINT			Character String
PP_COM	COMMENT			Character String
PP_CTR	CONTROLFILE_REUSE		1 or 0	
PP_DAT	DATAFILE			Character String
PP_DBL	SYNDBLINK			Character String
PP_DIN	Redbrick DATA IN	POBJ_ID reference into PHYS_OBJ table		
PP_DMP	PROGDUMP			Character String
PP_DS	Redbrick DOMAIN SIZE			Character String
PP_DSP	Reference to a DBSPACE Physical Object	POBJ_ID reference into PHYS_OBJ table		
PP_DSS	Redbrick DATA SEGS			Character String
PP_EXC	EXCLUSIVE		1 or 0	
PP_FBK	FALLBACK			Character String
PP_FNM	WATCOM FILE NAME			Character String
PP_FRG	FREELIST_GROUPS			Character String
PP_FRL	FREELISTS			Character String
PP_GSY	Informix GEN SYN		1 or 0	
PP_IGN	Access Ignore Nulls		1 or 0	
PP_INI	INITIAL			Character String
PP_INT	INITRANS			Character String
PP_IS	Redbrick INDEX SEGS			Character String
PP_IT	DB2 INDEX TYPE			Character String
PP_ITB	DB2/2 INDEX TABLESPACE	POBJ_ID reference into PHYS_OBJ table		
PP_JRN	JOURNAL			Character String

Table AA – Property Codes (continued)

POBJPVTYP	PVOBJTYP	POBJPVVAL	POBJPVIVAL	POBJPVSVAL
Valid Values	**Description**	**Valid Values**		
PP_LBL	PROG TABLE LABEL			Character String
PP_LGF	LOGFILE			Character String
PP_LKM	DB2 LOCKMAX			Character String
PP_LTB	DB2/2 LONG TABLESPACE	POBJ_ID reference into PHYS_OBJ table		
PP_MNE	MINEXTENTS			Character String
PP_MXD	MAXDATAFILES			Character String
PP_MXE	MAXEXTENTS			Character String
PP_MXF	MAXLOGFILES			Character String
PP_MXH	MAXLOGHIST			Character String
PP_MXI	MAXINSTANCES			Character String
PP_MXM	MAXLOGMBRS			Character String
PP_MXR	Redbrick MAXROWS			Character String
PP_MXS	Redbrick MAXSEGMENTS			Character String
PP_MXT	MAXTRANS			Character String
PP_NXT	NEXT			Character String
PP_OFL	OFFLINE		1 or 0	
PP_OPT	OPTIMAL			Character String
PP_OWN	OWNER			Character String
PP_PCF	PCTFREE			Character String
PP_PCI	PCTINCREASE			Character String
PP_PCU	PCTUSED			Character String
PP_PIN	Redbrick PRIMARY INDEX IN	POBJ_ID reference into PHYS_OBJ table		
PP_PLL	PARALLEL		1 or 0	
PP_PSS	Redbrick PK SEGS			Character String
PP_PUB	PUBLIC		1 or 0	
PP_RBS	Reference to a ROLLBACK_SEGMENT Physical Object	POBJ_ID reference into PHYS_OBJ table		
PP_RID	Informix WITH ROWIDS		1 or 0	
PP_SDB	Informix SYN DB			Character String
PP_SIZ	SIZE			Character String
PP_SPB	SYNPUBLIC		1 or 0	
PP_SPL	SPOOL		1 or 0	
PP_SSS	Redbrick STAR SEGS			Character String
PP_STO	STORAGE			Character String
PP_STR	Redbrick RB STORAGES			Character String
PP_STY	Informix SYN TYPE			Character String
PP_TDB	TERADATABASE	POBJ_ID reference into PHYS_OBJ table		

Table AA – Property Codes (continued)

POBJPVTYP Valid Values	PVOBJTYP Description	POBJPVVAL Valid Values	POBJPVIVAL	POBJPVSVAL
PP_TMP	TEMPORARY		1 or 0	
PP_TSP	Reference to a TABLESPACE Physical Object	POBJ_ID reference into PHYS_OBJ table		
PP_TT	Redbrick TABLE TYPE			Character String
PP_TTB	DB2/2 TABLE TABLESPACE	POBJ_ID reference into PHYS_OBJ table		
PP_WDJ	WITH_DEFAULT_JOURNAL			Character String
PP_WFF	Redbrick WITH FILLFACTOR			Character String
PP_WJT	WITH_JOURNAL_TABLE			Character String
PP_WNN	DB2 WHERE NOT NULL		1 or 0	
TP_CMT	PowerBuilder pbt_cmnt			Character String
TP_DFC	PowerBuilder pbd_fchr			Character String
TP_DFF	PowerBuilder pbd_ffce			Character String
TP_DFH	PowerBuilder pbd_fhgt			Character String
TP_DFI	PowerBuilder pbd_fitl			Character String
TP_DFP	PowerBuilder pbd_fptc			Character String
TP_DFU	PowerBuilder pbd_funl			Character String
TP_DFW	PowerBuilder pbd_fwgt			Character String
TP_HFC	PowerBuilder pbh_fchr			Character String
TP_HFF	PowerBuilder pbh_ffce			Character String
TP_HFH	PowerBuilder pbh_fhgt			Character String
TP_HFI	PowerBuilder pbh_fitl			Character String
TP_HFP	PowerBuilder pbh_fptc			Character String
TP_HFU	PowerBuilder pbh_funl			Character String
TP_HFW	PowerBuilder pbh_fwgt			Character String
TP_LFC	PowerBuilder pbl_fchr			Character String
TP_LFF	PowerBuilder pbl_ffce			Character String
TP_LFH	PowerBuilder pbl_fhgt			Character String
TP_LFI	PowerBuilder pbl_fitl			Character String
TP_LFP	PowerBuilder pbl_fptc			Character String
TP_LFU	PowerBuilder pbl_funl			Character String
TP_LFW	PowerBuilder pbl_fwgt			Character String

Table BB – Relationship Template Purpose Valid Values

This table of valid values is used in the REL_TMPL metamodel table.

RTMPLPURP	RTMPLPURP
Valid Values	**Description**
DR_CA	Referential Integrity On Parent Delete Cascade
DR_RE	Referential Integrity On Parent Delete Resctrict
DR_SD	Referential Integrity On Parent Delete Set Default
DR_SN	Referential Integrity On Parent Delete Set Null
DR_CCA	Referential Integrity On Child Delete Cascade
DR_CRE	Referential Integrity On Child Delete Resctrict
DR_CSD	Referential Integrity On Child Delete Set Default
DR_CSN	Referential Integrity On Child Delete Set Null
IR_PCA	Referential Integrity On Parent Insert Cascade
IR_PRE	Referential Integrity On Parent Insert Resctrict
IR_PSD	Referential Integrity On Parent Insert Set Default
IR_PSN	Referential Integrity On Parent Insert Set Null
IR_CA	Referential Integrity On Child Insert Cascade
IR_RE	Referential Integrity On Child Insert Resctrict
IR_SD	Referential Integrity On Child Insert Set Default
IR_SN	Referential Integrity On Child Insert Set Null
UR_CCA	Referential Integrity On Child Update Cascade
UR_CRE	Referential Integrity On Child Update Resctrict
UR_CSD	Referential Integrity On Child Update Set Default
UR_CSN	Referential Integrity On Child Update Set Null
UR_CA	Referential Integrity On Parent Update Cascade
UR_RE	Referential Integrity On Parent Update Resctrict
UR_SD	Referential Integrity On Parent Update Set Default
UR_SN	Referential Integrity On Parent Update Set Null

Table CC – Validation Rule Type Codes

This table of valid values is used in the VLDRULE metamodel table.

VLDRULE_TP	VLDRULE_TP
Valid Values	**Description**
VR_CLI	Validation Rule type Client
VR_SVR	Validation Rule type Server

Table DD – Target Server Valid Values

This table of valid values is used in the TBLCONST metamodel table.

TARG_SVR Valid Values	TARG_SVR Description
TD_AD	Target DBMS ADABAS
TD_AS	Target DBMS AS400
TD_CL	Target DBMS CLIPPER
TD_D2	Target DBMS DB2_2
TD_D3	Target DBMS DBASE3
TD_D4	Target DBMS DBASE4
TD_DB	Target DBMS DB2
TD_FP	Target DBMS FOXPRO
TD_IN	Target DBMS INGRES
TD_IX	Target DBMS INFORMIX
TD_NW	Target DBMS NETWARE
TD_OR	Target DBMS ORACLE
TD_PG	Target DBMS PROGRESS
TD_RB	Target DBMS REDBRICK
TD_RD	Target DBMS RDB
TD_SB	Target DBMS SQLBASE
TD_SM	Target DBMS SMARTSTAR
TD_SS	Target DBMS SQLSERVER
TD_SY	Target DBMS SYBASE
TD_TE	Target DBMS TERADATA
TD_WC	Target DBMS WATCOM

Table EE – Target Client Valid Values

This table of valid values is used in the TBLCONST metamodel table.

TARG_CLI Valid Values	TARG_CLI Description
TC_PB	CLIENT_TYPE_POWERB
TC_SW	CLIENT_TYPE_SQLWIN
TC_VB	CLIENT_TYPE_VB

Table FF – Physical Object Type Codes

This table of valid values is used in the POBJPV metamodel table.

POBJPVVTYPE Valid Values	POBJPVVTYPE Description
D2_DBS	DB2 DATABASE
D2_STG	DB2 STOGROUP
EP_SEG	SEGMENT
PP_DBS	DATABASE
PP_RBS	ROLLBACK_SEGMENT
PP_TDB	TERADATA DATABASE
PP_TSP	TABLESPACE

3

Using Datatype Mapping

ERwin Datatype Mapping Rules

The tables in this chapter show how ERwin converts datatypes from one DBMS to another. The columns list the database systems ERwin supports; SQL and desktop systems are grouped separately and arranged alphabetically within each group.

The rows in the DBMS Datatype Mapping tables list the supported datatypes for each DBMS. The numbers beside the entries for a logical datatype ("Var Char #1," "Var Char #2," etc.) indicate minor variants among the systems. Where there are multiple like-named datatypes used in a single DBMS entry, an asterisk (*) indicates the default type ERwin uses when it converts instances of that datatype.

To determine how ERwin converts a datatype from a source DBMS to a target DBMS, locate the asterisk-marked instance of that datatype for your source DBMS, and then trace that row across to your target DBMS. For example, the OpenIngres **c** datatype has an asterisk in the row "Char #1". The datatype for Progress in "Char #1" is CHAR*()*. ERwin converts the **c** datatype to CHAR*()* if you switch your server from OpenIngres to Progress. In the reverse scenario however, ERwin converts Progress CHAR*()* (the default) to OpenIngres char*()*.

Logical Datatype	...	OpenIngres	...	Progress
...		...		...
Char #1		c*		CHAR*()*
Char #2		char*()*		CHARACTER*()**
Char #3		char*()**		CHAR*()**
...		...		...

Converting One Datatype to Another Using the Datatype Mapping Tables.

Datatype Mapping Parameters

In the following DBMS Datatype Mapping tables, parentheses following the datatype name denote optional or required parameters for size, precision (,), or scale. If the parentheses are italicized (for example, char*()*), they are required for that datatype; if the parentheses are not italicized (for example, char()), they are optional. A comma included in the parentheses denotes that precision is allowed or required for that datatype.

When converting datatypes in a model, ERwin uses several rules to determine if the resulting datatype includes parentheses. The rules are listed below; each has a corresponding example in the following table.

♦ If the source datatype has parameters and the target datatype has:

- Required or optional parameters, the resulting datatype includes the source datatype's parameters (see Example 1 in the table below).

- No parameters, the resulting datatype does not include parameters (see Example 2 in the table below).

♦ If the source datatype does not have parameters and the target datatype has:

- Required parameters, the resulting datatype includes parameters with the default value of 100 (see Example 3 in the table below).

- Optional or no parameters, the resulting datatype does not include parameters (see Example 4 in the table below).

Example	Source Datatype	Target Datatype	Resulting Datatype
1	char(18)	char( ) or char*( )*	char(18)
2	char(18)	char	char
3	char	char*( )*	char(100)
4	char	char( ) or char	char

Summary of SQL DBMS Datatype Mapping

Logical Datatype	AS/400	DB2/MVS	DB2/2 2.0	INFORMIX	INGRES
Long Integer #1	INTEGER*	INTEGER*	INTEGER*	integer*	integer*
Long Integer #2	INTEGER	INTEGER	INTEGER	int*	integer
Long Integer #3	INTEGER	INTEGER	SMALLINT	int	integer4*
Boolean	SMALLINT	SMALLINT	SMALLINT	byte	integer1
Bit	SMALLINT	SMALLINT	SMALLINT	byte	integer1
Byte	SMALLINT	SMALLINT	SMALLINT*	byte*	integer1*
Short Integer #1	SMALLINT*	SMALLINT*	INTEGER	smallint*	smallint*
Short Integer #2	SMALLINT	SMALLINT	SMALLINT	smallint	integer2*
Quad Word (Rdb)	FLOAT	FLOAT	INTEGER	numeric(,)	float8
Big Integer (Rdb)	FLOAT	FLOAT	INTEGER	numeric(,)	float8
Decimal	DECIMAL(,)*	DECIMAL(,)*	DECIMAL(,) *	decimal(,)*	float
Number #1	DECIMAL(,)	DECIMAL(,)	DECIMAL(,)	dec(,)*	float
Number #2	DECIMAL(,)	DECIMAL(,)	NUMERIC(,) *	numeric(,)*	float
Money	DECIMAL(,)	DECIMAL(,)	NUMERIC(,)	money(,)*	money*
Short Money	DECIMAL(,)	DECIMAL(,)	DECIMAL (,)	money(,)	money
Small Float (Ingres)	REAL	REAL	DOUBLE	smallfloat*	float4
Real	REAL*	REAL*	DOUBLE	real*	float4*
Float #1	FLOAT*	FLOAT*	DECIMAL(,)	float*	float*
Float #2	FLOAT	FLOAT	DECIMAL(,)	float	float
Long Float	DOUBLE PRECISION	FLOAT	DOUBLE*	double precision*	float8*
Interval	FLOAT	FLOAT	DOUBLE	interval <lgst> to <smlst>*	float8
Id	INTEGER	INTEGER	INTEGER	serial*	integer
Char #1	CHAR()	CHAR()	CHAR(,) *	char()	c*
Char #2	CHARACTER()*	CHARACTER()*	CHAR(,)	character()*	char( )
Char #3	CHAR()*	CHAR()*	CHAR(,)	char()*	char( ) *
Var Char #1	VARCHAR( ) *	VARCHAR()*	VARCHAR(,)*	varchar( ) *	varchar( ) *
Var Char #2	VARCHAR()	VARCHAR()	VARCHAR(,)	varchar( )	varchar( )
Text #1	VARCHAR()	LONG VARCHAR	VARCHAR(,)	text	varchar( )
Text #2	VARCHAR()	LONG VARCHAR*	LONG VARCHAR*	text	varchar( )
Text #3	VARCHAR()	LONG VARCHAR	LONG VARCHAR	text	varchar( )
Text #4	VARCHAR()	LONG VARCHAR	LONG VARCHAR	text*	text( ) *
Text #5	VARCHAR()	LONG VARCHAR	LONG VARCHAR	text in table*	text( )
Date	DATE*	DATE*	DATE*	date*	date*
Time	TIME*	TIME*	TIME	datetime year to second	date
Timestamp	TIMESTAMP*	TIMESTAMP*	TIMESTAMP	datetime year to second	date
Date/Time	TIMESTAMP	TIMESTAMP	TIMESTAMP	datetime year to second	date
Date/Time #2	TIMESTAMP	TIMESTAMP	TIMESTAMP	datetime*	date
Small Date Time	TIMESTAMP	DATE	DATE	datetime year to second	date
Date ANSI (Rdb)	DATE	DATE	DATE	date	date
Date VMS (Rdb)	DATE	DATE	DATE	datetime year to second	date
Image (SYBASE)	VARGRAPHIC()	LONG VARGRAPHIC	VARGRAPHIC( )	byte in table	text( )
Binary (SQL Svr)	GRAPHIC()*	GRAPHIC()*	GRAPHIC( ) *	byte in table	text( )
Long Vargraphic (DB2)	VARGRAPHIC( ) *	LONG VARGRAPHIC*	LONG VARGRAPHIC*	byte in table	text( )
Raw (ORACLE)	VARGRAPHIC()	VARGRAPHIC()*	VARGRAPHIC( ) *	byte in table*	text( )
Long Raw (ORACLE)	VARGRAPHIC()	LONG VARGRAPHIC	LONG VARGRAPHIC	byte in table	text( )
MLSLabel (ORACLE)	VARCHAR()	VARCHAR()	BLOB( ) *	varchar( )	varchar( )
Raw MLSLabel (ORACLE)	GRAPHIC()	GRAPHIC()	CLOB( ) *	byte in table	varchar( )
OLE Object (Access)	VARGRAPHIC()	LONG VARGRAPHIC	DBCLOB( ) *	byte in table	text( )
National Char #1	CHAR()	CHAR()	CHAR( )	nchar()*	char( )
National Char #2	CHAR()	CHAR()	CHAR( )	nchar()	char( )
National Char #3	CHAR()	CHAR()	CHAR( )	nchar()	char( )
National Char Varying #1	VARCHAR( )	VARCHAR()	VARCHAR( )	nvarchar()*	varchar( )
National Char Varying #2	VARCHAR( )	VARCHAR()	VARCHAR( )	nvarchar()	varchar( )
National Char Varying #3	VARCHAR( )	VARCHAR()	VARCHAR( )	nvarchar()	varchar( )

Summary of SQL DBMS Datatype Mapping (continued)

Logical Datatype	InterBase	Netware SQL	ORACLE	OpenIngres	Progress
Long Integer #1	INTEGER*	INT(2)	INTEGER*	integer*	INTEGER*
Long Integer #2	INTEGER	INT(2)*	INTEGER	integer	INTEGER
Long Integer #3	SMALLINT	INT(2)	INTEGER	integer	INTEGER
Boolean	SMALLINT	LOGICAL(1)*	SMALLINT	integer1*	LOGICAL*
Bit	SMALLINT	BIT(1)*	SMALLINT	byte*	LOGICAL
Byte	SMALLINT	INT(2)	SMALLINT	integer4*	SMALLINT
Short Integer #1	INTEGER	INT(2)	SMALLINT*	smallint*	SMALLINT*
Short Integer #2	SMALLINT	INT(2)	SMALLINT	Integer2*	SMALLINT
Quad Word (Rdb)	FLOAT	INT(2)	FLOAT	float8	FLOAT
Big Integer (Rdb)	FLOAT	INT(2)	FLOAT	float8	FLOAT
Decimal	DECIMAL(,)*	DECIMAL(6,0)*	DECIMAL(,)	decimal(,)*	DECIMAL(,)*
Number #1	NUMERIC(,)	DECIMAL(6,0)	NUMBER(,)	float	NUMERIC(,)
Number #2	NUMERIC(,)	NUMERIC(6,0)*	NUMBER(,)	float	NUMERIC(,)*
Money	NUMERIC(,)	MONEY(6,2)*	NUMBER(,)	money*	NUMERIC(,)
Short Money	NUMERIC(,)	MONEY(6,2)	NUMBER(,)	money	NUMERIC(,)
Small Float (Ingres)	FLOAT	FLOAT(4)	FLOAT	float4	REAL
Real	FLOAT	FLOAT(4)*	FLOAT*	float4*	REAL*
Float #1	FLOAT	FLOAT(4)	FLOAT	float*	FLOAT*
Float #2	FLOAT	BFLOAT(4)*	FLOAT	float	FLOAT
Long Float	DOUBLE PRECISION*	FLOAT(4)	FLOAT	float8*	NUMERIC(,)
Interval	NUMERIC(,)	FLOAT(4)	NUMBER(,)	float8	NUMERIC(,)
Id	INTEGER	AUTOINC(4)*	ROWID*	integer	INTEGER
Char #1	CHAR()	CHARACTER(1)	CHAR()	c*	CHAR()
Char #2	CHARACTER()*	CHARACTER(1)*	CHARACTER()*	char()	CHARACTER()*
Char #3	CHAR()*	CHARACTER(1)	CHAR()*	char()*	CHAR()*
Var Char #1	VARCHAR()*	ZSTRING(1)*	VARCHAR2()*	varchar()*	CHARACTER()
Var Char #2	VARCHAR()	ZSTRING(1)	VARCHAR()*	varchar()	CHARACTER()
Text #1	VARCHAR()	LSTRING(1)*	LONG VARCHAR	varchar()	CHARACTER()
Text #2	VARCHAR()	LVAR(2)*	LONG VARCHAR*	long varchar*	CHARACTER()
Text #3	VARCHAR()	LVAR(2)	LONG*	varchar()	CHARACTER()
Text #4	VARCHAR()	NOTE(1)*	LONG VARCHAR	text()*	CHARACTER()
Text #5	VARCHAR()	NOTE(1)	LONG VARCHAR	text()	CHARACTER()
Date	DATE*	DATE(4)*	DATE*	date*	DATE*
Time	DATE	TIME(4)*	DATE	date	DATE
Timestamp	DATE	TIME(4)	DATE	date	DATE
Date/Time	DATE	DATE(4)	DATE	date	DATE
Date/Time #2	DATE	DATE(4)	DATE	date	DATE
Small Date Time	DATE	DATE(4)	DATE	date	DATE
Date ANSI (Rdb)	DATE	DATE(4)	DATE	date	DATE
Date VMS (Rdb)	DATE	TIME(4)	DATE	date	DATE
Image (SYBASE)	BLOB*	LVAR(2)	LONG RAW	text()	CHARACTER()
Binary (SQL Svr)	BLOB	LVAR(2)	RAW()	byte varying*	CHARACTER()
Long Vargraphic (DB2)	BLOB	LVAR(2)	LONG RAW	text()	CHARACTER()
Raw (ORACLE)	BLOB	LVAR(2)	RAW()*	long byte	CHARACTER()
Long Raw (ORACLE)	BLOB	LVAR(2)	LONG RAW*	text()	CHARACTER()
MLSLabel (ORACLE)	BLOB	ZSTRING(1)	MLSLABEL*	varchar()	CHARACTER()
Raw MLSLabel (ORACLE)	BLOB	LVAR(2)	RAW MLSLABEL*	varchar()	CHARACTER()
OLE Object (Access)	BLOB	LVAR(2)	LONG RAW	text()	CHARACTER()
National Char #1	NCHAR()*	CHARACTER(1)	CHAR()	char()	CHAR()
National Char #2	NCHAR()	CHARACTER(1)	CHAR()	char()	CHAR()
National Char #3	NCHAR()	CHARACTER(1)	CHAR()	char()	CHAR()
National Char Varying #1	NVARCHAR()*	ZSTRING(1)	VARCHAR()	varchar()	CHARACTER()
National Char Varying #2	NVARCHAR()	ZSTRING(1)	VARCHAR()	varchar()	CHARACTER()
National Char Varying #3	NVARCHAR()	ZSTRING(1)	VARCHAR()	varchar()	CHARACTER()

Summary of SQL DBMS Datatype Mapping (continued)

Logical Datatype	Red Brick	Rdb	SQL Anywhere	SQLServer 4.2	SQL Server 6.0
Long Integer #1	INTEGER*	INTEGER*	INTEGER*	int	int
Long Integer #2	INTEGER*	INTEGER	INT*	int*	int*
Long Integer #3	INTEGER*	INTEGER	BINARY(1)	int	int
Boolean	INTEGER*	TINYINT	BINARY(1)*	bit	bit
Bit	TINYINT*	TINYINT	SMALLINT	bit*	bit*
Byte	SMALLINT*	TINYINT*	SMALLINT*	tinyint*	tinyint*
Short Integer #1	INTEGER	SMALLINT*	INTEGER	smallint*	smallint*
Short Integer #2	INTEGER	SMALLINT	SMALLINT	smallint	smallint
Quad Word (Rdb)	NUMERIC(,) *	QUADWORD*	FLOAT	float	float
Big Integer (Rdb)	NUMERIC(,)	BIGINT*	FLOAT	float	float
Decimal	DECIMAL(,) *	DECIMAL(,)*	DECIMAL(,) *	float	decimal(,)*
Number #1	DECIMAL(,)	DECIMAL(,)	DECIMAL(,)	float	decimal(,)
Number #2	NUMERIC(,)	NUMERIC(,)*	NUMERIC(,) *	float	numeric(,)*
Money	NUMERIC(,)	DECIMAL(,)	DECIMAL(,)	money*	money*
Short Money	NUMERIC(,)	DECIMAL(,)	DECIMAL(,)	smallmoney*	smallmoney*
Small Float (Ingres)	NUMERIC(,)	REAL	REAL	real	real
Real	REAL*	REAL*	REAL*	real*	real*
Float #1	NUMERIC(,)	FLOAT*	FLOAT*	float*	float*
Float #2	NUMERIC(,)	FLOAT	FLOAT	float	float
Long Float	DOUBLE PRECISION*	DOUBLE PRECISION*	DOUBLE*	float	float
Interval	FLOAT*	INTERVAL()<lgst> to <smlst>*	DOUBLE	float	float
Id	INTEGER	INTEGER	INTEGER	int	int
Char #1	CHAR()	CHAR()	CHAR()	char( )	char()
Char #2	CHAR()	CHAR()	CHAR()	char( )	char()
Char #3	CHAR()	CHAR()*	CHAR() *	char( ) *	char()*
Var Char #1	CHAR()	VARCHAR() *	VARCHAR() *	varchar( ) *	varchar()*
Var Char #2	CHAR()	VARCHAR()	VARCHAR()	varchar( )	varchar()
Text #1	CHAR()	LONG VARCHAR	LONG VARCHAR	text	text
Text #2	CHAR()	LONG VARCHAR*	LONG VARCHAR*	text	text
Text #3	CHAR()	LONG VARCHAR	LONG VARCHAR	text	text
Text #4	CHAR()	LONG VARCHAR	LONG VARCHAR	text*	text*
Text #5	CHAR()	LONG VARCHAR	LONG VARCHAR	text	text
Date	DATE*	DATE*	DATE*	datetime	datetime
Time	TIME() *	TIME()*	TIME*	datetime	datetime
Timestamp	TIMESTAMP() *	TIMESTAMP()*	TIMESTAMP	timestamp*	timestamp*
Date/Time	TIMESTAMP()	TIMESTAMP	TIMESTAMP*	datetime*	datetime*
Date/Time #2	TIMESTAMP()	TIMESTAMP	TIMESTAMP	datetime	datetime
Small Date Time	DATE()	DATE	DATE	smalldatetime*	smalldatetime*
Date ANSI (Rdb)	DATE	DATE ANSI*	DATE	datetime	datetime
Date VMS (Rdb)	DATE	DATE VMS*	DATE	datetime	datetime
Image (SYBASE)	CHAR()	LIST OF BYTE VARYING	LONG BINARY	image*	image*
Binary (SQL Svr)	CHAR()	LIST OF BYTE VARYING	BINARY()	binary( ) *	binary()*
Long Vargraphic (DB2)	CHAR()	LIST OF BYTE VARYING	LONG BINARY	image	image
Raw (ORACLE)	CHAR()	LIST OF BYTE VARYING*	BINARY()	varbinary( ) *	varbinary()*
Long Raw (ORACLE)	CHAR()	LIST OF BYTE VARYING	LONG BINARY	image	image
MLSLabel (ORACLE)	CHAR()	VARCHAR()	VARCHAR()	varchar( )	varchar()
Raw MLSLabel (ORACLE)	CHAR()	LIST OF BYTE VARYING	BINARY()	binary( )	binary()
OLE Object (Access)	CHAR()	LIST OF BYTE VARYING	LONG BINARY	image	image
National Char #1	CHAR()	NCHAR()*	CHAR()	char( )	char()
National Char #2	CHAR()	NATIONAL CHAR()*	CHAR()	char( )	char()
National Char #3	CHAR()	NATIONAL CHARACTER()*	CHAR()	char( )	char()
National Char Varying #1	CHAR()	NCHAR VARYING()*	VARCHAR()	varchar( )	varchar()
National Char Varying #2	CHAR()	NATIONAL CHAR VARYING()*	VARCHAR()	varchar( )	varchar()
National Char Varying #3	CHAR()	NATIONAL CHARACTER VARYING()*	VARCHAR()	varchar( )	varchar()

Summary of SQL DBMS Datatype Mapping (continued)

Logical Datatype	SQLBase	SYBASE System 10 and 11	Teradata	Watcom
Long Integer #1	INTEGER*	int	INTEGER*	INTEGER*
Long Integer #2	INTEGER	int*	INTEGER	INT*
Long Integer #3	INTEGER	int	INTEGER	INTEGER
Boolean	SMALLINT	bit	BYTEINT	BINARY(1)
Bit	SMALLINT	bit*	BYTEINT	BINARY(1)*
Byte	SMALLINT	tinyint*	BYTEINT*	SMALLINT
Short Integer #1	SMALLINT*	smallint*	SMALLINT*	SMALLINT*
Short Integer #2	SMALLINT	smallint	SMALLINT	SMALLINT
Quad Word (Rdb)	NUMBER()	float	FLOAT	FLOAT
Big Integer (Rdb)	NUMBER()*	float	FLOAT	FLOAT
Decimal	DECIMAL(,)*	decimal(,)*	DECIMAL(,)*	DECIMAL(,) *
Number #1	DECIMAL(,)	decimal(,)	DECIMAL(,)	DECIMAL(,)
Number #2	DECIMAL(,)	numeric(,)*	DECIMAL(,)	NUMERIC(,) *
Money	DECIMAL(,)	money*	DECIMAL(,)	DECIMAL(,)
Short Money	DECIMAL(,)	smallmoney*	DECIMAL(,)	DECIMAL(,)
Small Float (Ingres)	FLOAT	real	FLOAT	REAL
Real	FLOAT*	real*	FLOAT	REAL*
Float #1	FLOAT	float*	FLOAT*	FLOAT*
Float #2	FLOAT	float	FLOAT	FLOAT
Long Float	DOUBLE PRECISION*	float	FLOAT	DOUBLE*
Interval	DOUBLE PRECISION	float	FLOAT	DOUBLE
Id	INTEGER	int	INTEGER	INTEGER
Char #1	CHAR()	char( )	CHAR()	CHAR( )
Char #2	CHAR()	char( )	CHAR()*	CHAR( )
Char #3	CHAR( ) *	char( ) *	CHAR()	CHAR( ) *
Var Char #1	VARCHAR( ) *	varchar( ) *	VARCHAR()*	VARCHAR( ) *
Var Char #2	VARCHAR()	varchar( )	VARCHAR()	VARCHAR( )
Text #1	LONG VARCHAR	text	LONG VARCHAR*	LONG VARCHAR
Text #2	LONG VARCHAR*	text	LONG VARCHAR	LONG VARCHAR*
Text #3	LONG VARCHAR	text	LONG VARCHAR	LONG VARCHAR
Text #4	LONG VARCHAR	text*	LONG VARCHAR	LONG VARCHAR
Text #5	LONG VARCHAR	text	LONG VARCHAR	LONG VARCHAR
Date	DATE*	datetime	DATE*	DATE*
Time	TIME*	datetime	DATE	TIME*
Timestamp	TIMESTAMP*	timestamp*	DATE	TIMESTAMP
Date/Time	DATETIME*	datetime*	DATE	TIMESTAMP*
Date/Time #2	DATETIME	datetime	DATE	TIMESTAMP
Small Date Time	DATETIME	smalldatetime*	DATE	DATE
Date ANSI (Rdb)	DATE	datetime	DATE	DATE
Date VMS (Rdb)	DATETIME	datetime	DATE	DATE
Image (SYBASE)	LONG VARCHAR	image*	VARBYTE()	LONG BINARY
Binary (SQL Svr)	VARCHAR()	binary( ) *	BYTE()*	BINARY( )
Long Vargraphic (DB2)	LONG VARCHAR	image	VARBYTE()	LONG BINARY
Raw (ORACLE)	VARCHAR()	varbinary( ) *	VARBYTE()*	BINARY( ) *
Long Raw (ORACLE)	LONG VARCHAR	image	VARBYTE()	LONG BINARY*
MLSLabel (ORACLE)	VARCHAR()	varchar( )	VARCHAR()	VARCHAR( )
Raw MLSLabel (ORACLE)	VARCHAR()	binary( )	VARBYTE()	BINARY( )
OLE Object (Access)	LONG VARCHAR	image	VARBYTE()	LONG BINARY
National Char #1	CHAR( )	char( )	CHAR()	CHAR( )
National Char #2	CHAR( )	char( )	CHAR()	CHAR( )
National Char #3	CHAR( )	char( )	CHAR()	CHAR( )
National Char Varying #1	VARCHAR()	varchar( )	VARCHAR()	VARCHAR( )
National Char Varying #2	VARCHAR()	varchar( )	VARCHAR()	VARCHAR( )
National Char Varying #3	VARCHAR()	varchar( )	VARCHAR()	VARCHAR( )

Summary of Desktop DBMS Datatype Mapping

Logical Datatype	Access 1.1, 2.0, and 7.0	Clipper/ dBASE3	FoxPro/ dBASE4	Paradox 4.x	Paradox 5.0
Long Integer #1	Long Integer*	Numeric(,)	Numeric(,)	Number	Long Integer*
Long Integer #2	Long Integer	Numeric(,)	Numeric(,)	Number	Long Integer
Long Integer #3	Long Integer	Numeric(,)	Numeric(,)	Number	Long Integer
Boolean	Yes/No	Logical*	Logical*	Binary	Logical*
Bit	Yes/No*	Logical	Logical	Binary*	Logical
Byte	Byte*	Numeric(,)	Numeric(,)	Short Number	Bytes( )*
Short Integer #1	Integer*	Numeric(,)	Numeric(,)	Short Number*	Short*
Short Integer #2	Integer	Numeric(,)	Numeric(,)	Short Number	Short
Quad Word (Rdb)	Double	Numeric(,)	Numeric(,)	Number	Number
Big Integer (Rdb)	Double	Numeric(,)	Numeric(,)	Number	Long Integer
Decimal	Double	Numeric(,)	Numeric(,)	Number	Number
Number #1	Double	Numeric(,)	Numeric(,)	Number	BCD*
Number #2	Double	Numeric(,)*	Numeric(,)*	Number*	Number*
Money	Currency*	Numeric(,)	Numeric(,)	Currency*	Money*
Short Money	Currency	Numeric(,)	Numeric(,)	Currency	Money
Small Float (Ingres)	Single*	Numeric(,)	Float(,)	Number	Number
Real	Single	Numeric(,)	Float(,)	Number	Number
Float #1	Double*	Numeric(,)	Float(,)*	Number	Number
Float #2	Double	Numeric(,)	Float(,)	Number	Number
Long Float	Double	Numeric(,)	Numeric(,)	Number	Number
Interval	Double	Numeric(,)	Numeric(,)	Number	Number
Id	Counter*	Numeric(,)	Numeric(,)	Number	Autoincrement*
Char #1	Text( )	Character( )	Character( )	Alphanumeric( )	Alpha( )
Char #2	Text( )	Character( )*	Character( )*	Alphanumeric( )	Alpha( )
Char #3	Text( )*	Character( )	Character( )	Alphanumeric( )*	Alpha( )*
Var Char #1	Text( )	Character( )	Character( )	Memo()	Memo( )
Var Char #2	Text( )	Character( )	Character( )	Memo()	Memo( )
Text #1	Memo	Character( )	Character( )	Memo()	Memo( )
Text #2	Memo	Memo	Memo	Memo()	Memo( )
Text #3	Memo	Memo	Memo	Memo()	Memo( )
Text #4	Memo*	Memo*	Memo*	Memo()*	Memo( )*
Text #5	Memo	Memo	Memo	Formatted Memo*	Formatted Memo*
Date	Date/Time	Date*	Date*	Date*	Date*
Time	Date/Time	Date	Date	Date	Time*
Timestamp	Date/Time	Date	Date	Date	Timestamp*
Date/Time	Date/Time*	Date	Date	Date	Timestamp
Date/Time #2	Date/Time	Date	Date	Date	Timestamp
Small Date Time	Date/Time	Date	Date	Date	Timestamp
Date ANSI (Rdb)	Date/Time	Date	Date	Date	Date
Date VMS (Rdb)	Date/Time	Date	Date	Date	Date
Image (SYBASE)	OLE Object	Memo	Memo	Graphic	Binary( )
Binary (SQL Svr)	OLE Object	Memo	Memo	Graphic*	Binary( )*
Long Vargraphic (DB2)	OLE Object	Memo	Memo	Graphic	Graphic()*
Raw (ORACLE)	OLE Object	Memo	Memo	Binary	Binary( )
Long Raw (ORACLE)	OLE Object	Memo	Memo	Graphic	Binary( )
MLSLabel (ORACLE)	Memo	Memo	Memo	Memo()	Memo( )
Raw MLSLabel (ORACLE)	OLE Object	Memo	Memo	Short	Binary( )
OLE Object (Access)	OLE Object*	Memo	Memo	OLE*	OLE*
National Char #1	Text( )	Character( )	Character( )	Alphanumeric( )	Alpha( )
National Char #2	Text( )	Character( )	Character( )	Alphanumeric( )	Alpha( )
National Char #3	Text( )	Character( )	Character( )	Alphanumeric( )	Alpha( )
National Char Varying #1	Text( )	Character( )	Character( )	Memo()	Memo( )
National Char Varying #2	Text( )	Character( )	Character( )	Memo()	Memo( )
National Char Varying #3	Text( )	Character( )	Character( )	Memo()	Memo( )
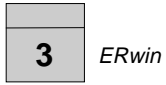
3 | *ERwin*

4 ERwin Macros

Target Server Support for Macros

You can use ERwin macros in the templates you create for triggers, stored procedures, and pre- and post- schema generation scripts when these features are supported by the target DBMS.

All macros, except those specifically noted, can be used on all of the target servers supported by ERwin. See your server documentation for detailed information on its support for RI triggers, stored procedures, and/or scripts.

Using the Macro Reference

The macro reference information is organized alphabetically by macro name. The information for each macro includes a description, macro syntax, the *scope* of the macro (i.e., the statements or code segments in which it can be used), the return value; and, typically, an example of the macro template code and corresponding expanded code. A description of each element follows:

♦   **Macro Syntax**. Provides the syntax for each ERwin macro, including keywords, user-specified variables, and required punctuation.

♦ **Scope**. For each macro, this element lists:

- The "%ForEach" commands in which a given macro can be inserted.

- Whether the macro can be used as the predicate in a conditional "%If" statement.

- The template types in which a given macro can be inserted. Entries include: *RI or Rel Override, Trigger Override*, and *Global*. The specific templates of each type are listed in the following table.

Scope	Templates	
RI or Rel Override	PARENT INSERT RESTRICT	CHILD INSERT RESTRICT
	PARENT INSERT CASCADE	CHILD INSERT CASCADE
	PARENT INSERT SET NULL	CHILD INSERT SET NULL
	PARENT INSERT SET DEFAULT	CHILD INSERT SET DEFAULT
	PARENT UPDATE RESTRICT	CHILD UPDATE RESTRICT
	PARENT UPDATE CASCADE	CHILD UPDATE CASCADE
	PARENT UPDATE SET NULL	CHILD UPDATE SET NULL
	PARENT UPDATE SET DEFAULT	CHILD UPDATE SET DEFAULT
	PARENT DELETE RESTRICT	CHILD DELETE RESTRICT
	PARENT DELETE CASCADE	CHILD DELETE CASCADE
	PARENT DELETE SET NULL	CHILD DELETE SET NULL
	PARENT DELETE SET DEFAULT	CHILD DELETE SET DEFAULT
	Stored procedures attached to tables (macros used within %ForEachFKAtt); pre- and post-scripts (macros used within %ForEachEntity).	
Trigger Override	TRIGGER UPDATE HEADER	TRIGGER INSERT FOOTER
	TRIGGER DELETE HEADER	CUSTOM TRIGGER HEADER
	TRIGGER INSERT HEADER	CUSTOM TRIGGER FOOTER
	TRIGGER UPDATE FOOTER	CUSTOM TRIGGER DEFAULT BODY
	TRIGGER DELETE FOOTER	
	Stored procedures attached to tables (macros used within %ForEachAtt); pre- and post-scripts (macros used within %ForEachEntity).	
Global	All triggers, stored procedures, and pre- and post-scripts.	

♦ **Return Value**. Explains what SQL code the macro returns when it is generated.

♦ **Example**. Includes sample template syntax and the corresponding expanded code. All expanded code examples are based on a subset of the tables in the ERwin MOVIES model (SQL Server target) shown in the ERwin Template Toolbox. An example is not included if the macro returns TRUE or FALSE.

Note: *The sample model above shows the physical names for both tables and columns, indicated by the underscore character in the object names (e.g., MOVIE_COPY, movie_copy_number). Different ERwin macros are available to return either the ERwin logical name (which can contain hyphens or spaces) or database physical names (in which hyphens and spaces are replaced by underscores) depending on your requirements.*

Reading ERwin Macro Syntax

The ERwin macro syntax commands are composed of keywords, punctuation, and variables.

♦ *Keywords* always start with a percent sign (%) and must be entered exactly as shown in the table (e.g. %AttFieldName).

♦ Braces {} and parentheses () in the syntax must be included; commas are required if you include more than one variable in the command.

♦ *Variables* are shown in angle brackets (e.g., <macro code1>). You replace variables with your own specific text, number, or macro code when you type the macro command into a trigger, stored procedure, or script.

Macro Variables

The following list defines common variables used in the macro syntax:

♦ <default value>, <value>, and <initial value> represent a user-specified text or numeric value, which appears exactly as written when the macro code is expanded.

♦ < action> represents a valid SQL action (i.e., INSERT, UPDATE, or DELETE).

♦ <macro code>, <macro code1>...<macro coden> represents any macro code sequence that is valid in the current scope. When multiple macro code sequences can be included in the same command, they are numbered from 1 to n.

♦ <predicate> represents any valid Boolean expression that evaluates to TRUE or FALSE, including those supported directly by an ERwin macro (e.g., %AttIsFK) and those created using supported Boolean operators (e.g., %!= and other comparison operators, %And, %Or, and %Not).

♦ <prefix>, <old prefix>, <new prefix> represents a text or numeric value (e.g., @ins) or expanded macro (e.g., %Parent) that is added to the beginning of a table or column name when the macro code is expanded. If both old and new prefix elements are specified, the list created by the macro has two sections: The first section declares <old prefix> <element>, the second section declares <new prefix> <element>.

♦ <separator> represents one or more characters used to delimit a list created by the macro. Separators are set off by quotation marks (e.g., "OR ").

♦ <table name>, <default name>, <domain name>, <validation> are replaced by the name of an ERwin table, default, domain, or validation rule or the corresponding macro (e.g., %TableName, %Child, %DefaultName, %DomainName, %ValidationName).

♦ <variable> represents the name of a variable used within the trigger, stored procedure, or script.

%!=

A Boolean predicate that returns "true" if the expansion of <macro code1> is not equal to the expansion of <macro code2>.

```
%!=(<macro code1>, <macro code2>)
```

Scope

Global, %If

Return Value

TRUE/FALSE

Example

Template Code	Expanded Code
`%ForEachAtt() {%If(%!=(%AttFK)` `{%AttName}}`	`general-condition`

%%

Inserts a single percent (%) symbol in expanded trigger code.

```
%%
```

Scope

Global

Return Value

```
%
```

Example

Template Code	Expanded Code
`/* %%JoinFKPK`	`/*%JoinFKPK`

%*

Multiplies the expansions of <macro code1> and <macro code2>.

```
%*(<macro code1>,<macro code2>)
```

Scope

Global

Return Value

Numeric

Example

Template Code	Expanded Code
```string1="aaaaa" string2="bbbbbbbbb" %*(%Len(%:string1), %Len(%:string2))```	45

%+

Adds the expansions of <macro code1> and <macro code2>.

```
%+(<macro code1>,<macro code2>)
```

Scope

Global

Return Value

Numeric

Example

Template Code	Expanded Code
```string1="aaaaa" string2="bbbbbbbbb" %*(%Len(%:string1), %Len(%:string2))```	14

%-

Subtracts the expansions of <macro code2> from <macro code1>.

```
%-(<macro code1>,<macro code2>)
```

Scope

Global

Return Value

Numeric

Example

Template Code	Expanded Code

`string1="aaaaa"`
`string2="bbbbbbbbb"`
`%*(%Len(%:string1),`
`%Len(%:string2))`
	`-4`

%/

Divides the expansions of <macro code1> by <macro code2>.

```
%/(<macro code1>,<macro code2>)
```

Scope

Global

Return Value

Numeric

Example

Template Code	Expanded Code

`string1="aaaaa"`
`string2="bbbbbbbbb"`
`%*(%Len(%:string1),`
`%Len(%:string2))`
	`.5556`

%:

Returns the value of <variable>.

```
%:<variable>
```

Scope

Global

Return Value

Value of the variable <variable>.

Example

Template Code	Expanded Code
`%:@ins_customer_id`	`1213`

%<

A Boolean predicate that returns "true" if the expansion of <macro code1> is less than the expansion of <macro code2>.

```
%<(<macro code1>, <macro code2>)
```

Scope

Global, %If

Return Value

TRUE/FALSE

%<=

A Boolean predicate that returns "true" if the expansion of <macro code1> is less than or equal to the expansion of <macro code2>.

```
%<=(<macro code1>, <macro code2>)
```

Scope

Global, %If

Return Value

TRUE/FALSE

%=

Assigns the expansion of <macro code> to a declared <variable> (See %Decl). The result of the statement is variable assignment only, the result does not appear in the expanded code.

```
%=(<variable>,<macro code>)
```

Scope

Global

Return Value

None

Example

Template Code	Expanded Code
`%=(@var1,%Tablename)`	`Internally, @var1 is set to the value of %Tablename`

%==

A Boolean predicate that returns "true" if the expansion of <macro code1> is equal to the expansion of <macro code2>.

```
%==(<macro code1>, <macro code2>)
```

Scope

Global, %If

Return Value

TRUE/FALSE

%>

A Boolean predicate that returns "true" if the expansion of <macro code1> is greater than the expansion of <macro code2>

```
%>(<macro code1>, <macro code2>)
```

Scope

Global, %If

Return Value

TRUE/FALSE

%>=

A Boolean predicate that returns "true" if the expansion of <macro code1> is greater than or equal to the expansion of <macro code2>

```
%>=(<macro code1>, <macro code2>)
```

Scope

Global, %If

Return Value

TRUE/FALSE

%Action

Returns the name of the action that caused the trigger to fire. Depending on the context of the command, %Action will return DELETE, INSERT, or UPDATE. For example, if the trigger was fired on a delete action, %Action returns DELETE.

```
%Action
```

Scope

Trigger Override

Return Value

DELETE, INSERT, or UPDATE

Example

Template Code	Expanded Code
%Action	DELETE

%Actions

Returns a list of actions that were implemented either before or after the trigger fired. If the trigger is a multiple action trigger, the macro reports all actions.

```
%Actions(<separator>)
```

Scope

Trigger Override

Return Value

DELETE, INSERT, and/or UPDATE

Example

Template Code	Expanded Code
`%Actions(",")`	`DELETE,DELETE`

%And

Lets you combine Boolean expressions to create more complex comparisons. %And performs a "logical and" on Boolean predicates defined by <macro code1> and <macro code2>. For example, the code %If {%And(%AttIsFK, %AttIsRolenamed)…} lets you execute additional steps if the given attribute is both a foreign key and has a rolename. Both elements must evaluate to "TRUE" for the %And condition to be satisfied.

```
%And(<macro code1>, <macro code2>)
```

Scope

Global, %If

Return Value

TRUE/FALSE

%AttDatatype

Returns the datatype of the current attribute. *Note*: Attributes can be assigned a user-defined datatype using ERwin domains. If an attribute has a user-defined datatype, the name of the datatype is provided by this macro.

```
%AttDatatype
```

Scope

%ForEachAtt, %ForEachFKAtt

Return Value

Attribute datatype

Example

Template Code	Expanded Code

`%ForEachAtt() {%AttName ->`
`%AttDatatype}`
	`master-number -> int`
	`movie-copy-number -> int`
	`general-condition -> varchar(10)`

%AttDef

Returns the attribute definition

```
%AttDef
```

Scope

%ForEachAtt, %ForEachFKAtt

Return Value

Attribute definition.

Example

Template Code	Expanded Code

`/*If the current att. is`
`master_number, then */`
 	
`%AttDef`
	`The unique id of a movie.`

%AttDefault

Returns the name of the default attached to the attribute.

```
%AttDefault
```

Scope

%ForEachAtt, %ForEachFKAtt

Return Value

Name of the default attached to the attribute.

Example

Template Code	Expanded Code
%AttDefault	SPACES

%AttDomain

Returns the name of the domain attached to the attribute.

```
%AttDomain
```

Scope

%ForEachAtt, %ForEachFKAtt

Return Value

Name of the domain.

Example

Template Code	Expanded Code
/*If the current att. is cust_phone, then */	PHONE NUMBER
%AttDomain	

%AttFieldName

Returns the column name associated with the current attribute.

```
%AttFieldName
```

Scope

%ForEachAtt, %ForEachFKAtt

Return Value

Column name.

Example

Template Code	Expanded Code
```/*If the current att. is renting-customer, then */``` ```%AttFieldName```	```renting_customer```

%AttFieldWidth

Returns the length of the attribute datatype

```
%AttFieldWidth
```

Scope

%ForEachAtt, %ForEachFKAtt

Return Value

Numeric

Example

Template Code	Expanded Code
```/* For a datatype of varchar(50)*/``` ```%AttFieldWidth```	```50```

%AttId

Returns the attribute ID.

```
%AttId
```

Scope

%ForEachAtt, %ForEachFKAtt

Return Value

Integer

%AttIsFK

A Boolean predicate that lets you determine whether or not the current attribute is a foreign key member.

```
%AttIsFK
```

Scope

%ForEachAtt, %ForEachFKAtt, %If

Return Value

TRUE/FALSE

Example

Template Code	Expanded Code

`%ForEachAtt() {`
`    %If{`
`      %AttIsFK}`
`    %AttName}`
	`master-number movie-copy-number`

%AttIsRolenamed

A Boolean predicate that lets you determine whether or not the current attribute is rolenamed. See "%AttIsFK" for expansion information.

```
%AttIsRolenamed
```

Scope

%ForEachAtt, %ForEachFKAtt, %If

Return Value

TRUE/FALSE

%AttIsPK

A Boolean predicate that lets you determine whether or not the current attribute is a primary key member. See "%AttIsFK" for expansion information.

```
%AttIsPK
```

Scope

%ForEachAtt, %ForEachFKAtt, %If

Return Value

TRUE/FALSE

%AttName

Returns the logical name of the current attribute.

```
%AttName
```

Scope

%ForEachAtt, %ForEachFKAtt

Return Value

Attribute name.

Example

Template Code	Expanded Code
/*If the current att. is master-number, then */	master-number
%AttName	

%AttNullOption

Returns the null option of the current attribute.

```
%AttNullOption
```

Scope

%ForEachAtt, %ForEachFKAtt

Return Value

NULL/NOT NULL

%AttPhysDatatype

Returns the physical datatype of the current attribute, even for user-defined datatypes.

```
%AttPhysDatatype
```

Scope

%ForEachAtt, %ForEachFKAtt

Return Value

Datatype

Example

Template Code	Expanded Code
`/*If the current att. is master-number, then */` `%AttPhysDatatype`	`int`

%Atts

Lists all the attributes of the trigger entity, performing the specified function on each member.

```
%Atts(<separator>,<action>, <prefix>)
```

Scope

Trigger Override

Return Value

List of actions and attributes.

Example

Template Code	Expanded Code
`%Atts(",` `",update,%Parent)`	`update(MOVIE_COPY.master_number),` `update(MOVIE_COPY.movie_copy_number)`

%AttValidation

Returns the name of the validation attached to a given attribute.

```
%AttValidation
```

Scope

%ForEachAtt, %ForEachFKAtt

Return Value

Validation Name

Example

Template Code	Expanded Code
`/*If the current att. is due_date, then */` `%AttValidation`	DUE_DATE_RULE

%Cardinality

Returns relationship cardinality:

♦ *<nothing>* = 1:0,1, or more

♦ P = 1:1 or more

♦ Z = 1:0 or 1

♦ *n* = 1:n, where *n* is an integer.

```
%Cardinality
```

Scope

RI or Rel Override

Return Value

<nothing>, P, Z, *<any integer>*

%Child

Returns the physical table name of the child entity in a relationship.

```
%Child
```

Scope

RI or Rel Override

Return Value

Table name.

Example

Template Code	Expanded Code
/*If the current relationship is <is in stock as>, then */ %Child	MOVIE_RENTAL_RECOR

%ChildAtts

Lists all the attributes of the child entity in a relationship, and performs the specified action on each.

```
%ChildAtts(<separator>,<action>, <prefix>)
```

Scope

RI or Rel Override

Return Value

Attribute names and actions.

Example

Template Code	Expanded Code
/*If the current relationship is <is in stock as>, then */ %ChildAtts(" or ",update)	update(movie_copy_number) or update(master_number) or ...

%ChildFK

Lists the foreign keys of the child entity in a relationship and performs the specified action on each.

```
%ChildFK(<separator>,<action> <prefix>)
```

Scope

RI or Rel Override

Return Value

Attribute names and actions.

Example

Template Code	Expanded Code

`/*If the current relationship is <is in stock as>, then */`
 	
`%ChildFK(" or ",update)`
	`update(master_number) or update(movie_copy_number)`

%ChildFKDecl

Lists the foreign keys of the child entity in the relationship followed by their datatypes.

```
%ChildFKDecl(<old prefix>, <new prefix>,<separator>>,<attribute/type separator>)
```

Scope

RI or Rel Override

Return Value

FK attribute names and datatypes

Example

Template Code	Expanded Code

`/*If the current relationship is <is in stock as>, then */`
 	
`%ChildFKDecl(,@ins_,", ")`
	`@ins_master_number int, @ins_movie_copy_number int`

%ChildNK

Lists non-key elements of the child entity and performs the specified action on each.

```
%ChildNK(<separator>,<action>, <prefix>)
```

Scope

RI or Rel Override

Return Value

Non-key attribute names and action

Example

Template Code	Expanded Code
`/*If the current relationship is <is in stock as>, then */` `%ChildNK(" or ",update,@)`	`update(@general_condition)`

%ChildNKDecl

Lists the non-key elements of the child entity in the relationship followed by their datatypes.

```
%ChildNKDecl(<old prefix>, <new
prefix>,<separator>>,<attribute/type separator>)
```

Scope

RI or Rel Override

Return Value

Non-key attribute names and datatypes

Example

Template Code	Expanded Code
`/*If the current relationship is <is in stock as>, then */` `%ChildNKDecl(,@ins_,", ")`	`@ins_general_condition varchar(10)`

%ChildParamDecl

Lists all the attributes of the child entity in the relationship followed by their datatypes. This is useful when declaring parameters in stored procedures.

```
%ChildParamDecl(<old prefix>, <new prefix>,
<separator>>,<attribute/type separator>)
```

Scope

RI or Rel Override

Return Value

Attribute names and datatypes

Example

Template Code	Expanded Code

`/*If the current relationship is <is in stock as>, then */`
 	
`%ChildParamDecl(,@ins_,", ")`
	`@ins_master_number int,`
	`@ins_movie_copy_number int,`
	`@ins_general_condition`
	`varchar(10),`

%ChildPK

Lists primary key elements of the child entity, and performs the specified action on each.

```
%ChildPK(<separator>,<action>, <prefix>)
```

Scope

RI or Rel Override

Return Value

Non-key attribute names and actions

Example

Template Code	Expanded Code

`/*If the current relationship is <is in stock as>, then */`
 	
`%ChildPK(" or ",update,@)`
	`update(@movie_copy_number) or`
	`update(@master_number)`

%ChildPKDecl

Lists the primary key elements of the child entity in the relationship followed by their datatypes.

```
%ChildPKDecl(<old prefix>, <new
prefix>,<separator>>,<attribute/type separator>)
```

Scope

RI or Rel Override

Return Value

Attribute names and datatypes

Example

Template Code	Expanded Code
/*If the current relationship is <is in stock as>, then */ %ChildPKDecl(,@ins_,", ")	@ins_master_number int, @ins_movie_copy_number int,

%Concat

Concatenates <value1> and <value2> and returns the result.

```
%Concat(<value1>,<value2>)
```

Scope

Global

Return Value

String

%CurrentDatabase

Returns the name of the database in use when the script was generated, as entered in the <DB> Connection dialog.

```
%CurrentDatabase
```

Scope

Global

Return Value

String

Example

Template Code	Expanded Code
%CurrentDatabase	testdb

%CurrentFile

Returns the name of the ERwin file from which the script was generated.

```
%CurrentFile
```

Scope

Global

Return Value

String

Example

Template Code	Expanded Code
%CurrentFile	C:\MYFILES\MOVIES.ER1

%CurrentServer

Returns the name of the target server in use when the script was generated, as entered in the <DB> Connection dialog.

```
%CurrentServer
```

Scope

Global

Return Value

String

Example

Template Code	Expanded Code
%CurrentServer	SQL6

%CurrentUser

Returns the name of the user who generated the script, as entered in the <DB> Connection dialog.

```
%CurrentUser
```

Scope

Global

Return Value

String

Example

Template Code	Expanded Code
%CurrentUser	ssmith

%CustomTriggerDefaultBody

Inserts the code from the trigger body template into the trigger in expanded form.

```
%CustomTriggerDefaultBody
```

Scope

Trigger Override

Return Value

All code contained in the CustomTriggerDefaultBody trigger template.

%CustomTriggerDefaultFooter

Inserts the code from the trigger footer template into the trigger in expanded form.

```
%CustomTriggerDefaultFooter
```

Scope

Trigger Override

Return Value

All code contained in the CustomTriggerDefaultFooter trigger template.

%CustomTriggerDefaultHeader

Inserts the code from the trigger header template into the trigger in expanded form.

```
%CustomTriggerDefaultHeader
```

Scope

Trigger Override

Return Value

All code contained in the CustomTriggerDefaultHeader trigger template.

%DatatypeName

Returns the name of the datatype.

```
%DatatypeName()
```

Scope

Global

Return Value

Datatype

Example

Template Code	Expanded Code
`%ForEachAtt(%Tablename,", "){` ` %DatatypeName(%AttPhysDatatype)}`	varchar

%DatatypeScale

For decimal datatypes, %DatatypeScale returns the number of decimal places.

```
%DatatypeScale()
```

Scope

Global

Return Value

Integer

Example

Template Code	Expanded Code
`/* For example, in decimal(10,2)*/` `%DatatypeScale()`	2

%DatatypeWidth

Returns the field width of the datatype.

```
%DatatypeWidth()
```

Scope

Global

Return Value

Integer

Example

Template Code	Expanded Code
`/* For example, in decimal(10,2)*/` `%DatatypeWidth()`	10

%Datetime

Returns the current date and time.

`%Datetime`

Scope

Global

Return Value

String

Example

Template Code	Expanded Code
`%Datetime`	`Wed Feb 17 12:56:22 1998`

%DBMS

Returns the database name.

`%DBMS`

Scope

Global

Return Value

String

Example

Template Code	Expanded Code
`%DBMS`	`ORACLE`

%DBMSDelim

Returns the statement delimiter of the DBMS.

```
%DBMSDelim
```

Scope

Global

Return Value

, or ; or other delimiter

%Decl

Declares a <variable>and, optionally, initializes it with <initial value>. For example, if you include the %Decl(var1,0) statement, on expansion, var1 is declared and initialized with the value 0. No expansion code is generated.

```
%Decl(<variable>,<initial value>)
```

Scope

Global

Return Value

None

Example

Template Code	Expanded Code
`%Decl(var1,0)`	`Internally, var1 is set to the value of 0.`

%DefaultName

Returns the default's name.

```
%DefaultName
```

Scope

%ForEachDefault

Return Value

Name of the default.

Example

Template Code	Expanded Code
`%ForEachDefault(", "){` `%DefaultName}`	`SPACES, PERSON-NAME, ADDRESS`

%DefaultValue

Returns the default's value.

```
%DefaultValue(<default name>) or %DefaultValue
```

Scope

Global with an argument (<default name>) or %ForEachDefault without an argument

Return Value

Value of the default.

Example

Template Code	Expanded Code
`%ForEachDefault(", "){` `%DefaultValue}`	`NEW RELEASE, COMEDY, DRAMA,` `CHILDREN, HORROR`

%DomainDatatype

Returns the physical datatype of the domain.

```
%DomainDatatype(<domain name>)or %DomainDatatype
```

Scope

Global with an argument (<domain name>) or %ForEachDomain

Return Value

Datatype

Example

Template Code	Expanded Code
`%ForEachDomain(", "){` `%DomainDatatype}`	`varchar(10), int, char`

%DomainDef

Returns the name of the default attached to the domain.

```
%DomainDef(<domain name>) or %DomainDef
```

Scope

Global with an argument (<domain name>) or %ForEachDomain

Return Value

Domain's default value.

Example

Template Code	Expanded Code
`%ForEachDomain(", "){` `%DomainDef}`	`NEW RELEASE`

%DomainName

Returns the name of the domain.

```
%DomainName
```

Scope

%ForEachDomain

Return Value

Domain name.

Example

Template Code	Expanded Code
`%ForEachDomain{` `%DomainName}`	PERSON NAME ADDRESS

%DomainNullOption

Returns the null option of the domain.

```
%DomainNullOption(<domain name>) or %DomainNullOption
```

Scope

Global with an argument (<domain name>) or %ForEachDomain

Return Value

NULL, NOT NULL, IDENTITY, WITH NULL, or NOT NULL WITH
DEFAULT

%DomainValidation

Returns the name of a validation attached to the domain.

```
%DomainValidation(<domain name>) or %DomainValidation
```

Scope

Global with an argument (<domain name>) or %ForEachDomain

Return Value

Domain name.

Example

Template Code	Expanded Code
`%ForEachDomain{` `%DomainValidation}`	IS VALID CUSTOMER, DUE DATE

%EntityId

Returns the ID of the entity or table.

```
%EntityId(<entity or tablename>)
```

Scope

RI or Rel OverrideTrigger Override

Return Value

Integer

Example

Template Code	Expanded Code
`%EntityId(%Parent)`	1234

%EntityName

Returns the name of the entity or table.

```
%EntityName(<entity or tablename>)
```

Scope

RI or Rel OverrideTrigger Override

Return Value

String

Example

Template Code	Expanded Code
%EntityName(%Parent)	MOVIE-COPY

%File

Expands the macro code within the braces and writes the result to the given file. This macro provides a workaround to the Windows 3.x 64K string size limit.

```
%File(<filename>) {<macro code>}
```

Scope

Global

Return Value

String

Example

Template Code	Expanded Code
%File(C:\erw\myfile.tst) {%ForEachAtt(Parent, ", ") {%AttFieldName} ...}	<creates a file and includes the specified information>

%Fire

Specifies when the trigger is fired. Available on INFORMIX, ORACLE7, and Rdb only.

```
%Fire
```

Scope

Trigger Override

Return Value

BEFORE or AFTER

Example

Template Code	Expanded Code
`%Fire`	`BEFORE`

%ForEachAtt

Expands macro code for each of the attributes in the given table.

```
%ForEachAtt(<table name>,<separator>) { <macro code> }
```

Scope

Trigger Override

Return Value

Result of <macro code> for each attribute in a given table.

Example

Template Code	Expanded Code

`%ForEachAtt(%Parent, ", ")`
`{%AttFieldName}`
	`master_number,`
	`movie_copy_number,`
	`general_condition`

%ForEachChildRel

Expands the <relationship code> for every relationship in which the trigger entity is a child.

```
%ForEachChildRel(<separator>) {<relationship code>}
```

Scope

Trigger Override

Return Value

Result of <macro code> for each relationship.

Example

Template Code	Expanded Code

```%ForEachChildRel("go") {```
 ``` insert %Parent values```
 ```(%ParentAtts(", ",,@ins_) }```
	```insert MOVIE_COPY```
	 ```values(@ins_master_number,```
	 ```@ins_movie_copy_number,```
	 ```@ins_general_condition)```
	 ```go```

%ForEachDefault

Expands the macro code for each default.

```
%ForEachDefault(<separator>) {<macro code>}
```

Scope

Global

Return Value

Result of <macro code> for each default.

Example

Template Code	Expanded Code

```%ForEachDefault(```
 ```){%DefaultName }```
	```FILL_WITH_SPACES```
	 ```SET_TO_NULL```
	 ```SET_TO_ZERO```

%ForEachDomain

Expands the macro code for all domains that are implemented as user-defined datatypes.

```
%ForEachDomain(<separator>) {<macro code>}
```

Scope

Global

Return Value

Result of <macro code>

Example

Template Code	Expanded Code
`%ForEachDomain(","){%DomainName }`	ADDRESS, PERSON-NAME

%ForEachEntity

Expands the macro code for all entities in the current subject area.

```
%ForEachEntity(<separator>) {<macro code>}
```

Scope

Trigger Override

Return Value

Result of <macro code> for each entity.

Example

Template Code	Expanded Code
`%ForEachEntity() {%TableName}`	MOVIE_COPY MOVIE_RENTAL_RECOR

%ForEachFKAtt

Expands macro code for each of the foreign key attributes migrated through the current relationship.

```
%ForEachFKAtt(<separator>) {<macro code>}
```

Scope

RI or Rel Override

Return Value

Result of <macro code> for each FK attribute.

Example

Template Code	Expanded Code
`%ForEachFKAtt(", ")` `{%AttFieldName}`	`master_number, movie_copy_number`

%ForEachIndex

Expands the macro code for all indexes in the current subject area. <table> defaults to the table in the current scope; can be used to name another table (%Parent, MOVIE_COPY).<type>filters type of index (AK, IE, IF, PK, AK1, IE2, etc.). Defaults to all.

```
%ForEachIndex([<table>],[<type>],[<name>],[<separator>])
{<macro code>}
```

Scope

Trigger Override

Return Value

Result of <macro code> for each index.

Example

Template Code	Expanded Code
`%ForEachIndex {  %IndexName` `%IndexType}`	`XPKMOVIE_COPY PK`

%ForEachIndexMem

Expands the macro code for all members of the index in the current subject area. <sequence> can be used to enter the assigned number of a specific index member. Defaults to all members.

```
%ForEachIndexMem([<sequence>],[<separator>]) {<macro code>}
```

Scope

Trigger Override

Return Value

Result of <macro code> for each index member.

Example

Template Code	Expanded Code

`%ForEachIndex {`
`%ForEachIndexMem() {`
`%AttDatatype }}`
	`int`
	`int`
	`varchar(10)`

%ForEachKey

Expands the macro code for all alternate keys and inversion entries in the current subject area.

```
%ForEachKey([<table>],[<type>],[<separator>]) {<macro code>}
```

Scope

Trigger Override

Return Value

Result of <macro code> for each key.

Example

Template Code	Expanded Code
`%ForEachKey(%Parent,,,` `) {%KeyName}`	`AK1,` `IE1`

%ForEachKeyMem

Expands the macro code for all members of the key.

```
%ForEachKeyMem([<sequence>],[<separator>]) {<macro code>}
```

Scope

Trigger Override

Return Value

Result of <macro code> for each key member.

Example

Template Code	Expanded Code
%ForEachKey {%ForEachKeyMem() {%AttName}}	renting-customer

%ForEachParentRel

Expands the <relationship code> for every relationship in which the trigger entity is a parent.

```
%ForEachParentRel(<separator>) { <relationship code> }
```

Scope

Trigger Override

Return Value

Result of <macro code> for each relationship.

Example

Template Code	Expanded Code
%ForEachParentRel() {insert %Parent values(%ParentAtts(", ",@ins_) }	insert MOVIE_COPY values(@ins_master_number, @ins_movie_copy_number, @ins_general_condition)

%ForEachValidValue

Expands the macro code for all valid values within the validation rule.

```
%ForEachValidValue(<separator>) {<macro code>}
```

Scope

Trigger Override, %ForEachValidation

Return Value

Result of <macro code> for each valid value.

Example

Template Code	Expanded Code

```%ForEachValidValue(","){```
``` %ValidValue =>%ValidValueDef }```
	```N =>New (recent release),```
	```S =>Standard release,```
	```O =>Old movie,```
	```C =>Classic```

%ForEachValidation

Expands the macro code for all validation rules.

```
%ForEachValidation(<separator>) {<macro code>}
```

Scope

Trigger Override

Return Value

Result of <macro code> for each validation rule.

Example

Template Code	Expanded Code
```%ForEachValidation(){``` ```%ValidationName }```	```Due Date Validation```

%If %Else

Conditionally expands either the %If<macro code1> if the predicate evaluates to "true" or the %Else <macro code2> if the predicate evaluates to "false". The %Else clause is optional.

```
%If (<predicate>) {<macro code1>} %Else {<macro code2>}
```

Scope

Global

Return Value

Result of <macro code1> each time <predicate> evaluates to TRUE.
Result of <macro code2> each time <predicate> evaluates to FALSE.

Example

Template Code	Expanded Code
`%ForEachAtt() {   %If(%AttIsPK) {` `%AttName -> PK}}`	`master-number -> PK` `movie-copy-number -> PK`

%Include

Allows you to include trigger macro code across multiple files or trigger templates.

```
%Include("<path name>")
```

```
%Include(<trigger template name>)
```

Scope

Global

Return Value

String

Example

Template Code	Expanded Code
`%Include("<c:\er\trg.tdl")`	`<All code included in the trg.tdl` `file>`

%IndexName

Returns the index name.

`%IndexName`

Scope

Trigger Override, %ForEachIndex, %ForEachIndexMem

Return Value

String

Example

Template Code	Expanded Code
`%IndexName`	`XPK_MOVIE_COPY`

%IndexType

Returns the index type.

`%IndexType`

Scope

Trigger Override, %ForEachIndex, %ForEachIndexMem

Return Value

FK, PK, AK, IE1, etc.

Example

Template Code	Expanded Code
`%IndexType`	`PK`

%JoinFKPK

Lets you join the foreign key of a child entity to the primary key of the parent entity in a relationship.

```
%JoinFKPK([<child table>, <parent table>,<comparison
op>,<separator>)
```

Scope

RI or Rel Override, WHERE clause search condition

Return Value

Code supporting comparison of FK and PK values in a relationship.

Example

Template Code	Expanded Code
`%JoinFKPK(%Child, deleted," <> "," or")`	`MOVIE_RENTAL_RECOR.master_number <> deleted.master_number or MOVIE_RENTAL_RECOR.movie_copy_number <> deleted.movie_copy_number`

%JoinPKPK

Lets you join the primary key of two correlations, or a base table and a correlation.

```
%JoinPKPK(<table>, <correlation>, <comparison op>,<separator>)
```

Scope

RI or Rel OverrideTrigger Override, WHERE clause search condition

Return Value

Code supporting comparison of PK values of two tables.

Example

Template Code	Expanded Code
`%JoinPKPK(%Child,ins," <> "," or")`	`MOVIE_RENTAL_RECOR.renting_custome r <> ins.renting_customer or MOVIE_RENTAL_RECOR.master_number <> ins.master_number or …`

%KeyName

Returns the key name.

```
%KeyName
```

Scope

Trigger Override, %ForEachKey, %ForEachKeyMem

Return Value

String

Example

Template Code	Expanded Code
`%KeyName`	AK1

%Len

Returns the string length of <macro code>.

```
%Len(<macro code>)
```

Scope

Global

Return Value

Integer

Example

Template Code	Expanded Code
`string1='aaaaa'` `%Len(%:string1)`	5

%Lower

Converts the expansion of <macro code> to lowercase.

```
%Lower(<macro code>)
```

Scope

Global

Return Value

Lowercase string

Example

Template Code	Expanded Code
`%Lower(%Parent)`	`movie`

%Max

Compares <value1> and <value2> and returns the maximum.

```
%Max(<value1>,<value2>)
```

Scope

Global

Return Value

Larger value in comparison

Example

Template Code	Expanded Code

`var1=8`
`var2=5`
`%Max(var1,var2)`
	`8`

%Min

Compares <value1> and <value2> and returns the minimum.

```
%Min(<value1>,<value2>)
```

Scope

Global

Return Value

Smaller value in comparison

Example

Code	Expansion
var1=8 var2=5 %Min(var1, var2)	5

%NK

Lists all the non-key attributes of the trigger entity, and performs the specified action on each.

```
%NK(<separator>,<action>, <prefix>)
```

Scope

Trigger Override

Return Value

Attribute and action

Example

Template Code	Expanded Code
`%NK(" or ",update,%Parent)`	`update(MOVIE_COPY.general_condition)` `or...`

%NKDecl

Lists the non-key attributes of the trigger entity followed by their datatypes.

```
%NKDecl(<old prefix>,<new prefix>,<separator>,<attribute/type
separator>)
```

Scope

Trigger Override

Return Value

Attribute names and datatypes

Example

Template Code	Expanded Code
`%NKDecl(,@ins_,", ")`	`@ins_general_condition varchar(10)`

%Not

Lets you combine Boolean expressions to create more complex comparisons. %Not performs a "logical not" on the Boolean predicate defined by <macro code>.

```
%Not(<macro code>)
```

Scope

Global, %If

Return Value

Result of macro code.

Example

Template Code	Expanded Code

`%ForEachAtt() {`
`%If(%Not(%==(%AttDatatype, int)))`
`{%AttName}}`
	`general-condition`

%NotnullFK

Compares the foreign key of a child entity <child table>in a relationship to <null expression>. This macro expands if and only if the relationship is non-identifying , nulls allowed.

```
%NotnullFK(<child table>, <not null
expression>,<prefix>,<separator>)
```

Scope

RI or Rel Override, WHERE clause search condition

Return Value

Null expression for each FK allowing null values.

Example

Template Code	Expanded Code
`where %NotnullFK(,"is null",," or ")`	`where master_number is null or` `movie_copy_number is null`

%Or

Lets you combine Boolean expressions to create more complex comparisons. %Or performs a "logical or" on Boolean predicates defined by <macro code1> and <macro code2>. For example, %If {%Or(%AttIsFK,%AttIsRolenamed)…} lets you execute additional steps if the given attribute is either a foreign key or is rolenamed. The macro is executed if any element evaluates to "TRUE".

```
%Or(<macro code1>,<macro code2>)
```

Scope

Global, %If

Return Value

TRUE/FALSE

Example

Template Code	Expanded Code

`%If (%Or(%AttIsFK,`
`%AttIsRolenamed){`
`%AttName}`
	`master_copy_number`

%ParamDecl

Lists all attributes of the trigger entity followed by their datatypes.

```
%ParamDecl(<old prefix>,<new
prefix>,<separator>,<attribute/type separator>)
```

Scope

Trigger Override

Return Value

Attributes and datatypes

Example

Template Code	Expanded Code

`%ParamDecl(,`
`@ins_,", ")`
	`@ins_master_number int,`
	`@ins_movie_copy_number int,`
	`@ins_general_condition varchar(10)`

%ParamPass

Assigns values to procedure parameters specified by the <old prefix> and/or the <new prefix>, for all attributes of the trigger entity. Available on Ingres only.

```
%ParamPass(<old prefix>,<new prefix>,<param/value
separator>,<param separator>)
```

Scope

Trigger Override

Return Value

Parameter list for Ingres

Example

Template Code	Expanded Code
`%ParamPass(,new," = ")`	`master_number = new.master_number`

%Parent

Returns the physical table name of the parent entity in a relationship.

```
%Parent
```

Scope

RI or Rel Override

Return Value

String

Example

Template Code	Expanded Code
%Parent	MOVIE_COPY

%ParentAtt

Expands any attribute macro (e.g., %AttFieldName, %AttDatatype) on the parent primary key attribute from which the current attribute is migrated.

```
%ParentAtt(<attribute macro>)
```

Scope

%ForEachAtt, %ForEachFKAtt

Return Value

Result of <attribute macro>

Example

Template Code	Expanded Code
%ForEachAtt() {%ParentAtt (%AttFieldName)}	master_number

%ParentAtts

Lists all the attributes of the parent entity in a relationship, and performs the specified action on each.

```
%ParentAtts(<separator>, <action>,<prefix>)
```

Scope

RI or Rel Override

Return Value

Attributes and actions

Example

Template Code	Expanded Code
`%ParentAtts(" or ",update,%Parent)`	`update(MOVIE.master_number) or update(MOVIE.customer_number)…`

%ParentNK

Lists the non-key attributes of the parent entity in a relationship, and performs the specified action on each.

```
%ParentNK(<separator>, <action>,<prefix>)
```

Scope

RI or Rel Override

Return Value

Non-key attributes and actions

Example

Template Code	Expanded Code
`%ParentNK(" or ",update,%Parent)`	`update(MOVIE.movie_name) or update(MOVIE.movie_rating)or update(MOVIE.movie_year)`

%ParentNKDecl

Lists the non-key attributes of the parent entity in the relationship followed by their datatypes.

```
%ParentNKDecl(<old prefix>, <new
prefix>,<separator>,<attribute/type separator>)
```

Scope

RI or Rel Override

Return Value

FK attribute names and datatypes

Example

Template Code	Expanded Code

`/*If the current relationship is <is in stock as>, then */`
`%ParentNKDecl(,@ins_,", ")`
	`@ins_movie_name varchar(50),`
	`@ins_movie_rating varchar(5),`
	`@ins_movie_rental_rate money`

%ParentParamDecl

Lists the attributes of the parent entity in the relationship followed by their datatypes.

```
%ParentParamDecl(<old prefix>, <new prefix>,
<separator>,<attribute/type separator>)
```

Scope

RI or Rel Override

Return Value

Attribute names and datatypes

Example

Template Code	Expanded Code

`/*If the current relationship is <is in stock as>, then */`
`%ParentParamDecl(, @ins_,", ")`
	`@ins_movie_name varchar(50),`
	`@ins_movie_rating varchar(5),`
	`@ins_movie_rental_rate money`

%ParentPK

Lists the primary key elements of the parent entity in a relationship, and performs the specified action on each.

```
%ParentPK(<separator>,<action>)
```

Scope

RI or Rel Override

Return Value

Primary key attributes and actions

Example

Template Code	Expanded Code
`%ParentPK(,update,%Parent)`	`update(MOVIE_COPY.master_number)` `update(MOVIE_COPY.movie_copy_number)`

%ParentPKDecl

Lists the primary key elements of the parent entity in the relationship followed by their datatypes.

```
%ParentPKDecl(<old prefix>, <new
prefix>,<separator>,<attribute/type separator>)
```

Scope

RI or Rel Override

Return Value

Primary key attribute names and datatypes

Example

Template Code	Expanded Code

`/*If the current relationship is`
`<is in stock as>, then */`
 	
`%ParentPKDecl(,@ins_)`
	`@ins_master_number varchar(10)`
	`@ins_movie_copy_number varchar(10)`
	`@ins_customer_number varchar(10)`

%PhysRelName

Returns the physical relationship name.

```
%PhysRelName
```

Scope

RI or Rel Override

Return Value

String

Example

Code	Expansion
%PhysRelName	is_rented_under

%PK

Lists the primary key elements of the trigger entity, and performs the specified action on each.

```
%PK(<separator>,<action>,<prefix>)
```

Scope

Trigger Override

Return Value

Primary key attributes and actions

Example

Template Code	Expanded Code
%PK(,update,%Parent)	update(MOVIE_COPY.master_number) update(MOVIE_COPY.movie_copy_number)

%PKDecl

Lists the primary key elements of the trigger entity followed by their datatypes.

```
%PKDecl(<old prefix>,<new prefix>,<separator>>,<attribute/type
separator>))
```

Scope

Trigger Override

Return Value

Primary key attribute names and datatypes

Example

Template Code	Expanded Code
`%PKDecl(,@ins_)`	`@ins_master_number varchar(10)` `@ins_movie_copy_number varchar(10)`

%RefClause

Generates the references clause from the OLD and NEW information added in the Entity Trigger Editor. Available on INFORMIX, Ingres, ORACLE7, and Rdb only.

```
%RefClause
```

Scope

Trigger Override

Return Value

Reference clause

Example

Template Code	Expanded Code
`%RefClause`	`REFERENCES NEW as inserted OLD as deleted`

%RelId

Returns the relationship ID.

```
%RelId
```

Scope

RI or Rel Override

Return Value

Integer

%RelIsNonull

Examines the relationship null expression and returns "true" if it is nulls are not allowed and "false" if nulls are allowed.

```
%RelIsNonull
```

Scope

RI or Rel Override, %If

Return Value

TRUE/FALSE

%RelRI

Returns the given referential integrity option for the specified <action> (insert, delete, update) and <RI type> (parent or child).

```
%RelRI(<action>,<RI type>)
```

Scope

RI or Rel Override

Return Value

CASCADE, RESTRICT, SET NULL, SET DEFAULT, or NONE

Example

Template Code	Expanded Code
%RelRI(Update, Child)	CASCADE

%RelTemplate

Expands the template code attached to the current relationship. If no code is attached, the corresponding diagram-wide referential integrity template is expanded.

```
%RelTemplate
```

Scope

RI or Rel Override

Return Value

Expands the code fragment for the relationship corresponding to trigger type.

%RelType

Returns the relationship type. Relationship type valid values and their definitions include: RT_ID (identifying), RT_NI (non-identifying), RT_SC (subtype), or RT_MM (many-to-many).

```
%RelType
```

Scope

RI or Rel Override

Return Value

RT_ID, RT_NI, RT_SC, or RT_MM

Example

Template Code	Expanded Code
%RelType	RT_ID

%Scope

Returns the specification of how the trigger is to be executed. Available on ORACLE7 only.

```
%Scope
```

Scope

Trigger Override

Return Value

FOR EACH ROW or FOR EACH TABLE

Example

Template Code	Expanded Code
%Scope	FOR EACH ROW

%SetFK

Lists the foreign key elements of a child entity in a relationship and sets each member to the specified value.

```
%SetFK(<child table>,<value>)
```

Scope

RI or Rel Override

Return Value

Foreign key attribute name and value

Example

Template Code	Expanded Code

`%SetFK(%Child,NULL)`	`MOVIE_RENTAL_RECOR.master_number = NULL`
	`MOVIE_RENTAL_RECOR.movie_copy_number = NULL`
	`MOVIE_RENTAL_RECOR.customer_number = NULL`
	`MOVIE_RENTAL_RECOR.rental_record_id = NULL`

%SetPK

Lists the primary key elements of the specified table and sets each member to the specified value.

```
%SetPK(<table>,<value>)
```

Scope

RI or Rel OverrideTrigger Override

Return Value

Primary key attribute name and value

Example

Template Code	Expanded Code
`%SetPK(%Parent,0)`	`MOVIE_COPY.master_number = 0,` `MOVIE_COPY.movie_copy_number = 0`

%Substitute

Substitutes string <pattern> in string <value> with string <substitute>.

```
%Substitute(<value>,<pattern>,<substitute>)
```

Scope

Global

Return Value

String

Example

Template Code	Expanded Code
/*If the name of the child table is movie_copy_number*/	ins_movie_copy_number
%Substitute(%Child, movie,ins_movie)	

%Substr

Generates a substring of the expansion of the given <macro code>. <length> is not required.

```
%Substr(<macro code>,<initial pos>,<length>)
```

Scope

Global

Return Value

String

Example

Template Code	Expanded Code
%Substr(macro,1,3)	mac

%Switch

Evaluates the argument against the specified choices. ERwin expands the macro code corresponding to the matching choice. If no match is found, ERwin expands the default macro code.

```
%Switch(<argument>) {%Choose(<choice 1>) {<macro code 1>}
    %Choose(<choice 2>) {<macro code 2>}   <etc...> %Default
{<default macro code>}}
```

Scope

Global

Return Value

Macro code value

Example

Template Code	Expanded Code

`/* If %TableName is MOVIE_COPY */`
 	
`%Switch(%TableName){`
`%Choose(MOVIE_COPY){`
`   %FK}`
`%Choose(MOVIE_RENTAL_RECOR){`
`   %PK}`
`%Default{%PK}}`
	`master_number`

%TableName

Returns the physical table name of the trigger entity.

```
%TableName
```

Scope

Trigger Override

Return Value

Name of table

Example

Template Code	Expanded Code
`%Tablename`	`MOVIE_COPY`

%TemplateName

Returns the name of the trigger, stored procedure, or script template.

```
%TemplateName
```

Scope

Trigger Override

Return Value

Template name

Example

Template Code	Expanded Code
%TemplateName	CHILD_DELETE_CASCADE

%TriggerName

Returns the physical trigger name.

```
%TriggerName
```

Scope

Trigger Override

Return Value

Trigger name

Example

Template Code	Expanded Code
%TriggerName	MOVIE_COPY_CHILD_DELETE_CASCADE

%TriggerRelRI

Boolean predicate that is "true" if the given trigger and relationship are of the given <action> (Update/Delete/Insert), <RI type> (Child/Parent), and <integrity > (Cascade/Restrict/Set Null/Set Default).

```
%TriggerRelRI(<action>, <RI type>, <integrity>)
```

Scope

RI or Rel Override, %If

Return Value

TRUE/FALSE

Example

Template Code	Expanded Code

`%ForEachChildRel() {`
`%If(%TriggerRelRI`
`(Update, Child, Cascade))`
 	
`{%VerbPhrase Child Update`
`Cascade}}`
	`is rented under Child Update`
	`Cascade`

%UpdateChildFK

Lists the foreign key elements of the child entity in a relationship and performs the update action on each member. Available on ORACLE7, SQL Server, and SYBASE only.

```
%UpdateChildFK()
```

Scope

RI or Rel Override

Return Value

Foreign key value and action

Example

Template Code	Expanded Code

`%UpdateChildFK()`	`update(master_number) or`
	`update(movie_copy_number)`
	`...`

%UpdateParentPK

Lists the primary key elements of the parent entity in a relationship and performs the update action on each member. Available on ORACLE7, SQL Server, and SYBASE only.

```
%UpdateParentPK()
```

Scope

RI or Rel Override

Return Value

Primary key value and action

Example

Template Code	Expanded Code
`%UpdateParentPK()`	`update(master_number)`

%UpdatePK

Lists the primary key elements of the trigger entity and performs the update action on each member. Available on ORACLE7, SQL Server, and SYBASE only.

```
%UpdatePK()
```

Scope

Trigger Override

Return Value

Primary key value and action

Example

Template Code	Expanded Code
`%UpdatePK()`	`update(master_number)` `update(movie_copy_number)`

%Upper

Converts the expansion of <macro code> to uppercase.

```
%Upper(<macro code>)
```

Scope

Global

Return Value

String

Example

Template Code	Expanded Code
%Upper(%EntityName)	MOVIE COPY

%ValidationHasValidValues

A Boolean predicate that lets you determine whether or not the current validation has valid values.

```
%ValidationHasValidValues(<validation name>)
```

Scope

Global with an argument (<validation name>) or %ForEachValidation, %If

Return Value

TRUE/FALSE

%ValidationName

Returns the name of the validation.

```
%ValidationName
```

Scope

%ForEachValidation

Return Value

Validation Name

Example

Template Code	Expanded Code
%ValidationName	Due Date Validation

%ValidationRule

Returns the server-side validation rule.

```
%ValidationRule(<validation name>)or %ValidationRule
```

Scope

Global with an argument (<validation name>) or %ForEachValidation

Return Value

Validation rule string

Example

Template Code	Expanded Code
%ValidationRule (Valid CustNumber)	@col BETWEEN 1000 AND 9999

%ValidValue

Returns the data value of a valid value.

```
%ValidValue
```

Scope

%ForEachValidValue

Return Value

String

Example

Template Code	Expanded Code
`%ValidValue`	`CHILDREN`

%ValidValueDef

Returns the definition of a valid value.

```
%ValidValueDef
```

Scope

%ForEachValidValue

Return Value

String

Example

Template Code	Expanded Code

`/* Where valid values are: COMEDY, HORROR, DRAMA, NEW RELEASE and CHILDREN, one definition could read: */`
 	
`%ValidValueDef`
	`A movie that is of interest to children.`

%VerbPhrase

Returns the verb phrase of a relationship.

`%VerbPhrase`

Scope

RI or Rel Override

Return Value

Relationship verb phrase

Example

Template Code	Expanded Code
`%VerbPhrase`	`is rented under`

5

Schema Generation Options

Summary of Schema Generation Options

The options available in the Schema Generation Report Editor vary by target server. Supported options for both SQL and desktop servers are displayed in the appropriate group box. The group boxes include: Referential Integrity, Trigger Option, Statement Format, Table Option, Index Option, Column Option, View Option, Schema Option, and Other Options.

Referential Integrity Options

Referential Integrity (RI) options let you control how related records are treated when the value in a key field is modified or deleted. Select one or more of the following options:

Primary Key	To enforce the unique identification of each row in a table.
Foreign Key	To enforce the specified referential integrity rule when the value in a foreign key field is updated.
On Delete	To enforce the selected referential integrity option if the value is deleted in either a primary or foreign key field.
On Update	To enforce the selected referential integrity option if the value is updated in either a primary or foreign key field.
Unique (AK)	To enforce the referential integrity rule requiring that alternate keys values must be unique.
sp_primary key	To include the system procedure that creates the primary key in each table.
sp_foreign key	To include the system procedure that creates foreign keys.

Trigger Options

Trigger options let you override ERwin*'s* default RI templates to enforce referential integrity. Select one or more of the following options:

ERwin Generated	Includes ERwin-generated RI triggers in the schema for all RI options.
RI Type Override	Includes either an RI Type Override trigger (if available) or ERwin-generated trigger in the schema for all RI options.
Relationship Override	Includes either a Relationship Override trigger (if available) or ERwin-generated trigger in the schema for all RI options.
User Defined	Includes Entity Overrride triggers in the schema. If no other options are selected, Entity Override triggers include the default ERwin-generated trigger code.
RI Type Override	Includes RI Type Override trigger code (if available) in Entity Override triggers in the schema.
Relationship Override	Includes Relationship Override trigger code (if available) in Entity Override triggers in the schema.

? Search for **Trigger, overriding built-in** in ERwin Online Help for more information.

Statement Format Options

Statement Format options let you choose if the schema includes a CREATE or ALTER statement for each primary or foreign key. If the table that requires a new key that already exists in the database, use the ALTER option. Select one of the following options for both the primary key and the foreign key:

CREATE/PK	To include a PRIMARY KEY clause in a CREATE TABLE statement.
ALTER/PK (default)	To include the PRIMARY KEY clause in an ALTER TABLE statement.
CREATE/FK	To include a FOREIGN KEY clause in a CREATE TABLE statement.
ALTER/FK (default)	To include the FOREIGN KEY clause in an ALTER TABLE statement.

Note: *In the Red Brick schema generation options, the CREATE/FK and ALTER/FX controls are found in the FK Format group box.*

Table Options

Table options let you specify what data definition statements to use in generating the schema. Select one or more of the following options:

CREATE TABLE	To execute SQL CREATE TABLE statements when generating the schema.
Entity Integ	To include SQL statements that create constraint rules for each entity.
DROP TABLE	To execute SQL DROP TABLE statements before executing CREATE TABLE statements when the schema is generated.
Integrity/Check	To include statements for table-level check constraint in the schema that ERwin generates. For OpenIngres, you can choose the way check constraints are generated in the schema (i.e., as CREATE INTEGRITY ON statements or as CHECK statements).
Physical Storage	To include physical storage objects and parameters in the schema.
Table CHECK	To include SQL statements that create constraint rules for each table.
Table Pre-Script	To execute pre-schema generation scripts immediately before the schema is generated.
Table Post-Script	To execute post-schema generation scripts immediately after the schema is generated.
VALIDPROC/Check	To include statements for table-level check constraint in the schema that ERwin generates. For DB2/MVS version 4.0, you can choose the way table-level check constraints are generated in the schema (i.e., as VALIDPROC statements or as CHECK statements).
Validation	To include any validation rules associated with tables in the schema statement. For Progress, this option is only available when you choose to generate the schema using the Progress 4GL option.
Create Alias	To include any alias table names that you defined in ERwin in the schema that ERwin generates.
Drop Alias	To include statements that drop previously defined alias table names in the schema that ERwin generates.
Create Macro	To include any macros that you defined in ERwin in the schema that ERwin generates. (Teradata only).
Drop Macro	To include statements that drop previously defined macros in the schema that ERwin generates. (Teradata only).
CREATE SYNONYM	To include synonym table names that you defined in ERwin in the schema that ERwin generates.
DROP SYNONYM	To include statements that drop previously defined synonym table names in the schema that ERwin generates.
Create Procedure	To include any stored procedures that you defined in ERwin in the schema that ERwin generates.
Drop Procedure	To include statements that drop previously defined stored procedures in the schema that ERwin generates.

Index Options

Index options let you control how indexes are created and stored and which key attributes are indexed. Select one or more of the following options:

Primary Key (PK)	To create an index on the primary key in each entity.
Alternate Key (AK)	To create an index on the alternate keys in each entity.
Foreign Key (FK)	To create an in index on the foreign keys in each entity.
Inversion Entry (IE)	To create an index on the inversion keys in each entity.
CLUSTERED or CLUSTERED HASHED	To create a clustered or clustered hashed index in the schema.
Physical Storage	To include index physical storage information in the schema.
PRIMARY	To include a UNIQUE PRIMARY INDEX clause in the schema.
Write .PRG File	To create an ERwin ASCII file that must be run on the target server to create the index.
.ndx Index (or any desktop Xbase single index file format)	To include single index files in the generated schema (Clipper and dBase).
.ntx Index Files	To include multiple index files in the generated schema (Clipper).
.idx Index Files (or any desktop Xbase single index file format)	To include single index files in the generated schema (FoxPro).
.cdx Index Files (or any desktop Xbase multiple index structure)	To include multiple index files in the generated schema (FoxPro).
.mdx Index (or any desktop Xbase multiple index structure)	To include multiple index files in the generated schema. (dBase)

View Options

View Options let you control the existence of view tables. Select one or more of the following options.

CREATE VIEW	To execute SQL CREATE VIEW statements when generating the schema.
DROP VIEW	To execute SQL DROP VIEW statements before executing CREATE VIEW statements when the schema is generated.
Pre-Script	To execute pre-schema generation scripts immediately before the schema is generated.
Post-Script	To execute post-schema generation scripts immediately after the schema is generated.

Column Options

Column options add constraint clauses to SQL CREATE TABLE statements. Select one or more of the following options:

Attribute Integ	To include SQL statements that create constraint rules for each column.
Column CHECK	To include SQL statements that create constraint rules for each column.
Default or DEFAULT Value	To include the default value for the column in the schema statement.
Integrity/Check	To include statements for column-level check constraint in the schema that ERwin generates. For OpenIngres, you can choose the way check constraints are generated in the schema (i.e., as CREATE INTEGRITY ON statements or as CHECK statements).
Physical Order	To preserve the physical order of the columns as ERwin generates the new schema.
sp_bindrule	To include a statement that binds the constraint rule to the column.
sp_bindefault	To include a statement that binds the default constraint rule to the column.
User Datatype	To include the User Datatype for the column in the schema statement (SQL Anywhere only).
Validation	To include the validation rule for the column in the schema statement.
FIELDPROC/ Check	To include statements for column-level check constraints in the schema that ERwin generates. For DB2/MVS version 4.0, you can choose the way column-level check constraints are generated in the schema (i.e., as FIELDPROC statements or as CHECK statements).
Initial Value	To include a statement assigning column initial values.
Column Label	To include a statement assigning column labels.
Label	To include a statement assigning column labels.
Column Heading	To include a statement assigning column headings.
Check Constr	To include SQL statements that create constraint rules for each column.
Use Domain	To include the Domain for the column in the schema statement (InterBase and Rdb only).
Use Distinct Type	To include the user Distinct datatype for the column in the schema statement (DB2/2 only).
BETWEEN	To include the validation rule for the column in the schema statement (Teradata only).
TITLE	To include a statement assigning column titles.

Schema Options

These options let you control schema-level features of the generated schema. Select one or more of the following options:

<PHYSICAL OBJECT>	To include any physical object definitions in the schema that ERwin generates.
Create Procedure	To include any schema stored procedures in the schema that ERwin generates.
Drop Procedure	To include DROP PROCEDURE statements in the schema that ERwin generates.
Drop Macro	To include DROP MACRO statements in the schema that ERwin generates (Teradata only).
Pre-Script	To execute pre-schema generation scripts immediately before the schema is generated.
Post-Script	To execute post-script schema generation scripts immediately after the schema is generated.
DISTINCT DATATYPE	To include CREATE DISTINCT statements for each DB2/2 user datatype in the schema that ERwin generates. You can define DB2/2 user datatypes in the ERwin Domain Editor.
CREATE DOMAIN	To include CREATE DOMAIN statements for each InterBase or Rdb user datatype in the schema that ERwin generates. You can define InterBase and Rdb user datatypes in the ERwin Domain Editor.
Create DATATYPE	To include CREATE DATATYPE statements for each SQL Anywhere user datatype in the schema that ERwin generates. You can define SQL Anywhere user datatypes in the ERwin Domain Editor.
TABLESPACE	To include CREATE TABLESPACE statements in the schema that ERwin generates.
ROLLBACK SEG	To include CREATE ROLLBACK SEGMENT statements in the schema that ERwin generates.
DATABASE	To include CREATE DATABASE statements in the schema that ERwin generates.
SEGMENT	To include CREATE SEGMENT statements in the schema that ERwin generates.
CREATE DBSPACE	To include CREATE DBSPACE statements in the schema that ERwin generates.
sp_addtype	To include sp_addtype statements in the schema that ERwin generates.
CREATE RULE	To include validation rule definitions in the schema that ERwin generates.
CREATE DEFAULT	To include default value definitions in the schema that ERwin generates.

Other Options

Other options may be available to support special features offered by the selected target server. Select one or more of the following options:

Comments	To include ERwin table and column comments in the schema that ERwin generates.
Constraint Name	To include the names of constraints in the schema that ERwin generates.
Quote Names	To insert quotation marks around table and column names in the schema that ERwin generates.
Owner	To include the individual table owner in the CREATE TABLE statement in the schema that ERwin generates.
No Delim	(Rdb only)
No CR	(Rdb only)
Use Labels for Logical Names	To include labels for tables based on entity names in the schema that ERwin generates (AS/400 only).

Note: *When ERwin generates a schema on the server, any changes made in ERwin to table properties, such as changing the table's name, columns, or relationships, are not updated on the server unless you DROP the modified table and then CREATE it again. Select the* **DROP TABLE** *and* **CREATE TABLE** *option boxes on the Schema Generation Report to replace the old table with the new table.*

The following summary chart shows the schema generation options available for each target server.

Summary of SQL Schema Generation Options

Target Server	Referential Integrity	Trigger Overrides	Statement Format	Table Option
AS/400	PK, FK ON DELETE ON UPDATE Unique (AK)	Ref Integrity Relationship Entity	CREATE/ ALTER PK, FK	CREATE/DROP TABLE Pre-Script/Post-Script
DB2/MVS	PK, FK On Delete	N/A	CREATE/ ALTER PK, FK	CREATE/DROP TABLE Pre-Script/Post-Script Physical Storage VALIDPROC/Check CREATE/DROP SYNONYM Create/Drop Procedure
DB2/2	PK, FK On DELETE On UPDATE	Ref Integrity Relationship Override Entity Override	CREATE/ ALTER PK, FK	CREATE/DROP TABLE Pre-Script/Post-Script Table CHECK Physical Storage CREATE/DROP ALIAS
Informix	PK, FK ON DELETE	Ref Integrity Relationship Entity	CREATE/ ALTER PK, FK	CREATE/DROP TABLE Pre-Script/Post-Script Table CHECK Create/Drop Procedure Physical Storage CREATE/DROP SYNONYM
Ingres	N/A	Ref Integrity Relationship Entity	N/A	CREATE/DROP TABLE Pre-Script/Post-Script Integrity Create/Drop Procedure
InterBase	PK, FK	Ref Integrity Relationship Entity	CREATE/ ALTER PK, FK	CREATE/DROP TABLE Pre-Script/Post-Script Table CHECK Create/Drop Procedure
ORACLE	PK, FK On Delete Unique (AK)	Ref Integrity Relationship Entity	CREATE/ ALTER PK, FK	CREATE/DROP TABLE Table CHECK Pre-Script/Post-Script Create/Drop Procedure Physical Storage CREATE/DROP SYNONYM
OpenIngres	PK, FK, AK	Ref Integrity Relationship Entity	CREATE/ ALTER PK, FK	CREATE/DROP TABLE Pre-Script/Post-Script Integrity/Check Create/Drop Procedure

Summary of SQL Schema Generation Options (continued)

Target Server	Index Options	Column Options	Schema Options	Other Options
AS/400	PK, AK, FK, IE	Physical Order DEFAULT Value Label Column Heading	Pre-Script/Post-Script	Comments Quote Names Constraint Name Use labels for logical Names
DB2/MVS	PK, AK, FK, IE Physical Storage	FIELDPROC/ Check Physical Order	STOGROUP DATABASE TABLESPACE Pre-Script/Post-Script	Constraint Name Comments Quote Names Owner
DB2/2	PK, AK, FK, IE	Column CHECK DEFAULT Value Physical Order Use Distinct Type	Pre-Script/Post-Script DISTINCT TYPE	Constraint Name Comments Quote Names
Informix	PK, AK, FK, IE CLUSTER Physical Storage	Column CHECK DEFAULT Value Physical Order	Pre-Script/Post-Script Create Procedure Drop Procedure	Constraint Name Owner
Ingres	PK, AK, FK, IE	Integrity Physical Order DEFAULT Value	Pre-Script/Post-Script Create Procedure Drop Procedure	Comments Quote Names Owner
InterBase	PK, AK, FK, IE	Column CHECK DEFAULT Value Physical Order Use DOMAIN	Pre-Script/Post-Script Create Procedure Drop Procedure CREATE DOMAIN	Constraint Name Owner
ORACLE	PK, AK, FK, IE Physical Storage	CHECK Constr DEFAULT Value Physical Order	TABLESPACE ROLLBACK SEG DATABASE Pre-Script/Post-Script Create Procedure Drop Procedure	Comments Constraints Quote Names Owner
OpenIngres	PK, AK, FK, IE	Integrity/Check DEFAULT Value Physical Order	Pre-Script/Post-Script Create Procedure Drop Procedure	Constraint Name Comments Quote Names Owner

Summary of SQL Schema Generation Options (continued)

Target Server	Referential Integrity	Trigger Overrides	Statement Format	Table Option
Progress [ODBC SQL option]	Unique Constraint	N/A	N/A	CREATE/DROP Pre-Script/Post-Script
Progress [4GL option]	N/A	Ref Integrity Relationship Entity	N/A	CREATE/DROP TABLE Table Validation Pre-Script/Post-Script Create Procedure Physical Storage
Rdb	PK, FK	Ref Integrity Relationship Entity	CREATE/ ALTER PK, FK	CREATE/DROP TABLE Pre-Script/Post-Script Table CHECK
Red Brick	PK, FK ON DELETE Create / Alter FK format	N/A	N/A	CREATE/DROP TABLE Pre-Script/Post-Script Physical Storage CREATE/DROP SYNONYM
SQL Anywhere & WATCOM	PK, FK On DELETE On UPDATE UNIQUE (AK)	Ref Integrity Relationship Entity	CREATE/ ALTER PK, FK	CREATE/DROP TABLE Pre-Script/Post-Script Table CHECK Create/Drop Procedure Physical Storage
SQLBase	PK, FK ON DELETE	Ref Integrity Relationship Entity	CREATE/ ALTER PK, FK	CREATE/DROP TABLE Pre-Script/Post-Script Create/Drop Procedure CREATE/DROP SYNONYM
SQL Server	PK, FK sp_primarykey sp_foreignkey Unique (AK)	Ref Integrity Relationship Entity	CREATE/ ALTER PK, FK	CREATE/DROP TABLE Physical Storage Table CHECK Pre-Script/Post-Script Create/Drop Procedure
SYBASE	PK, FK sp_primary key sp_foreign key Unique (AK)	Ref Integrity Relationship Entity	CREATE/ ALTER PK, FK	CREATE/DROP TABLE Physical Storage Table CHECK Pre-Script/Post-Script Create/Drop Procedure
Teradata	N/A	N/A	N/A	CREATE/DROP TABLE Physical Storage Pre-Script/Post-Script Create/Drop MACRO

Summary of SQL Schema Generation Options (continued)

Target Server	Index Options	Column Options	Schema Options	Other Options
Progress [ODBC SQL option]	PK, AK, FK, IE	Physical Order	Pre-Script/Post-Script	Quote Names
Progress [4GL option]	PK, AK, FK, IE	Column Label Label Validation Initial Value Physical Order	Pre-Script/Post-Script Create Procedure	Comments Quote Names
Rdb	PK, AK, FK, IE	Column CHECK DEFAULT Value Physical Order Use DOMAIN	Pre-Script Post-Script CREATE DOMAIN	Constraint Name Comments Quote Names Owner No Delim No CR
Red Brick	PK, AK, FK, IE Physical Storage	Physical Order DEFAULT Value	CREATE SEGMENT Pre-Script Post-Script	Comments Constraint Name
SQL Anywhere & WATCOM	PK, AK, FK, IE Physical Storage	Column CHECK DEFAULT Value Physical Order Use User DATATYPE	Pre-Script/Post-Script Create Procedure Drop Procedure CREATE DATATYPE CREATE DBSPACE	Constraint Name Comments Quote Names Owner
SQLBase	PK, AK, FK, IE CLUSTERED (HASHED)	Physical Order	Pre-Script/Post-Script Create Procedure Drop Procedure	Comments Quote Names Owner
SQL Server	PK, AK, FK, IE Physical Storage CLUSTERED	User Datatype Physical Order Validation Default	sp_addtype CREATE RULE CREATE DEFAULT Pre-Script/Post-Script Create Procedure Drop Procedure	Constraint Name Quote Names Owner
SYBASE	PK, AK, FK, IE Physical Storage CLUSTERED	User Datatype Physical Order Validation Default	sp_addtype CREATE RULE CREATE DEFAULT Pre-Script/Post-Script Create Procedure Drop Procedure	Constraint Names Quote Names Owner
Teradata	PK, AK, FK, IE PRIMARY	BETWEEN DEFAULT TITLE Physical Order	DATABASE Pre-Script/Post-Script Create/Drop MACRO	Comments

Summary of Desktop Schema Generation Options

Desktop Database Type	Referential Integrity Options	Schema/ Index Options	Index Options / Index On	Include Index Files	Column Options	Table/Other Options
Access [Access Basic]	Create Relation Delete Relation	*Access BASIC* Pre-Script Post-Script	PK AK FK IE	NA	Column Validation Column Default Physical Order Caption	Create Table Delete Table Table Validation Table Prescript Table Postscript Comments
Access [ODBC SQL]	PK FK AK *Stmnt. Format:* CREATE/ ALTER PK, FK	*ODBC SQL* Pre-Script Post-Script	PK AK FK IE	N/A	Physical Order	Create Table DROP TABLE Table Prescript Table Postscript Quote Names
Clipper	NA	.PRG File	PK AK FK IE	.ndx .ntx	Physical Order	CREATE/DROP TABLE
dBASE III	NA	.PRG File	PK AK FK IE	.ndx	Physical Order	CREATE/DROP TABLE
dBASE IV	NA	ODBC .PRG File	PK AK FK IE	.ndx .mdx	Physical Order	CREATE/DROP TABLE
FoxPro	Referential Integrity	.PRG File	PK AK FK IE	.idx .cdx	Physical Order	CREATE Table DROP Table Table Prescript Table Postscript Physical Order
Paradox	NA	Pre-Script Post-Script	PK AK FK IE	NA	Physical Order	CREATE/DROP TABLE Table Prescript Table Postscript Quote Names Constraint Names

Forward and Reverse Engineering Physical Storage Objects

If you generate a physical schema on a database server, you can include the physical storage objects you've defined in ERwin as part of the schema. ERwin automatically translates the physical object definitions into the appropriate SQL CREATE statements and inserts the specified parameter information in the correct syntax.

When you generate the physical schema, ERwin creates the specified parent storage objects first, and then it creates child storage objects and physical tables in the specified storage location. The Schema Options group on the Schema Generation Report Editor contains the options for forward engineering physical storage objects.

In addition, when you reverse engineer a database, ERwin can import the names and definitions of physical storage objects defined on the target server in the same way it imports physical tables, indexes, and other physical schema information.

After you import physical storage objects into ERwin, you can view or modify the object definitions and table associations in the Physical Object Editor and the Table Property Editor in the same way you work with physical storage objects originally created in ERwin.

The following table shows the physical storage objects that ERwin can forward and reverse engineer.

Summary of Physical Storage Generation Options

Server	Forward Engineering	Reverse Engineering
DB2/MVS	Stogroup, Database, Tablespace	Stogroup, Database, Tablespace
DB2/2	Tablespace	Tablespace
INFORMIX	—	blobspace, dbspace
ORACLE	Database, Tablespace, Rollback Segment	Database, Tablespace, Rollback Segment
Red Brick	Segment	Segment
SQL Server	—	Segment
SQL Anywhere and WATCOM	DBSPACE	DBSPACE
SYBASE	—	Segment
Teradata	Database	Database

Forward and Reverse Engineering Indexes

If you generate a physical schema, you can include any indexes you've defined in ERwin as part of the schema. ERwin automatically translates the index definitions into schema statements and inserts the specified parameter information in the correct syntax.

The Index Option group box in the Schema Generation Report Editor lets you select which indexes ERwin includes when it generates a schema. The Schema Generation Report Editor includes all index options supported by the target server.

ERwin generates schema statements for primary key, alternate key, foreign key, inversion entry key, and clustered indexes.

ERwin also generates schema statements for physical storage objects if the option is supported by the server and selected in the Schema Generation Report Editor.

When you reverse engineer a database, ERwin can import the name, definition and parameters of each index defined on the server. When you import the index information from a server, ERwin maintains the storage location information for each index, so later, you can recreate the database using the same storage assignments. You do *not* have to manually reassign the storage location for each index.

After you import indexes into ERwin, you can view or modify the index properties, definitions and table associations in the Index Editor. For DB2/MVS, INFORMIX, ORACLE, SQL Server, and SYBASE, you can assign an index to a physical storage object in the <DB> Index Editor. For DB2/MVS, INFORMIX, and ORACLE, you can also modify the storage parameters in the <DB> Index Editor.

? Search for **Properties, Specifying for an index** in ERwin Online help for more information.

See the documentation for your selected target server for more information about its index features.

Note: *If the target server is DB2/MVS, INFORMIX, ORACLE, or SYBASE and any Physical Storage option is selected, the schema includes index physical storage parameters in the schema statement.*

Forward and Reverse Engineering Validation Rules

If you generate a physical schema, you can include any table-level or column-level validation rules you've defined in ERwin as part of the schema. ERwin automatically translates the validation rule definitions into schema statements and inserts the specified parameter information in the correct syntax.

To include column-level validation rules, click the "Validation" check box (or the equivalent option for your server) in the Column Option group box in the Schema Generation Report editor. Similarly, to include table-level validation rules, click the **Table CHECK** box (or the equivalent option for your server) in the Table Option group box in the Schema Generation Report editor.

The following table lists the options on the <DB> Schema Generation Report dialog for each target server that let you include table and column-level validation rules in the schema.

Summary of Validation Rule Generation Options

SQL Target Server	Column-Level Validation	Table-Level Validation
AS/400	N/A	N/A
DB2/MVS 2.0 and 3.0	FIELDPROC	VALIDPROC
DB2/MVS 4.0	FIELDPROC/Check	VALIDPROC/Check
DB2/2	Column CHECK	Table CHECK
INFORMIX	Column CHECK	Table CHECK
INGRES	Integrity	Integrity
InterBase	Column CHECK	Table CHECK
OpenIngres	Integrity/Check	Integrity/Check
ORACLE	CHECK Constr	Table CHECK
PROGRESS [4GL]	Validation	Table Validation
PROGRESS [ODBC SQL]	N/A	N/A
Rdb	Column CHECK	Table CHECK
Red Brick	N/A	N/A
SQL Anywhere/WATCOM	Column CHECK	Table CHECK
SQL Server 4.2	sp_bindrule	N/A
SQL Server 6.0	Validation	Table CHECK
SQLBase	N/A	N/A
SYBASE 4.2	sp_bindrule	N/A
SYBASE 10 and 11	Validation	Table CHECK
Teradata	BETWEEN	N/A

Summary of Validation Rule Generation Options *(continued)*

Desktop Target Server	Column-Level Validation	Table-Level Validation
Access [Access Basic]	Column Validation	Table Validation
Access [ODBC SQL]	N/A	N/A
Clipper	N/A	N/A
dBASE III/IV	N/A	N/A
FoxPro	N/A	N/A
Paradox	N/A	N/A

Note: *ERwin cannot generate a validation rule statement in the schema if the target database does not support the required validation syntax (indicated by N/A in the above table). Consult your DBMS documentation for additional information on support and syntax for data validation rules.*

When reverse engineering from a schema file, script, or system catalog, ERwin automatically imports validation rules and attaches them to the appropriate table or column in the resulting data model. The naming convention that ERwin uses to name rules as they are imported is: VALID_RULE*n*, where *n* is a sequential number starting at zero (i.e., the first validation rule that ERwin encounters when reverse engineering is named VALID_RULE0, the next rule VALID_RULE1, etc., until the entire schema has been processed).

ERwin Features Supported Via Reverse Engineering

ERwin can reverse engineer any supported DDL script or database into an ERwin diagram. The following table illustrates the ERwin features supported with respect to each database type.

Summary of ERwin Features Supported via Reverse Engineering

ERwin Feature	AS/400	DB2/MVS	DB2/2	Informix	Ingres	InterBase
Entities	Y	Y	Y	Y	Y	Y
Attributes	Y	Y	Y	Y	Y	Y
Primary Keys	Y	Y	Y	Y	N	Y
Foreign Keys	Y	Y	Y	Y	N	Y
Alternate Keys (Unique Index)	Y	Y	Y	Y	Y	Y
Inversion Entries (Non-Unique Index)	Y	Y	Y	Y	Y	Y
Rolenames (Derived data)	N	Y	Y	Y	N	Y
Datatypes	Y	Y	Y	Y	Y	Y
User-Defined Datatypes	N	N	Y	N	N	Y called Domains
Indexes	Y	Y	Y	Y	Y	Y
Identifying Relationships (Derived data)	Y	Y	Y	Y	N	Y
Non-Identifying Relationships (Mandatory) (Derived data)	Y	Y	Y	Y	N	Y
Non-Identifying Relationships (Optional) (Derived data)	Y	Y	Y	Y	N	Y
Verb Phrases (Relationship Physical Name)	Y	Y	Y	Y	N	Y
NULL Options	Y	Y	Y	Y	Y	Y
Delete Rule (Referential Constraint)	Y	Y	Y	Y	N	Y
Subtype Relationship	N	N	N	N	N	N
Physical Objects	N	Y	Y	Y	N	N
Synonyms/Alias	N/A	Y	Y	Y	N/A	N/A
Triggers	N	N	Y	Y	Y	N
Stored Procedures	N	N	N	Y	Y	N
Pre & Post Script	N	N	N	N	N	N

Summary of ERwin Features Supported via Reverse Engineering (continued)

ERwin Feature	NetWare SQL *	ORACLE	OpenIngres	Progress	Rdb	Red Brick
Entities	Y	Y	Y	Y	Y	Y
Attributes	Y	Y	Y	Y	Y	Y
Primary Keys	N	Y	Y	Y	Y	Y
Foreign Keys	N	Y	Y	Y	Y	Y
Alternate Keys (Unique Index)	Y	Y	Y	Y	Y	Y
Inversion Entries (Non-Unique Index)	Y	Y	Y	Y	Y	Y
Rolenames (Derived data)	N	Y	Y	N	Y	Y
Datatypes	Y	Y	Y	Y	Y	Y
User-Defined Datatypes	N	N	N	N	Y called Domains	N
Indexes	Y	Y	Y	Y	Y	Y
Identifying Relationships (Derived data)	N	Y	Y	Y	Y	Y
Non-Identifying Relationships (Mandatory) (Derived data)	N	Y	Y	Y	Y	Y
Non-Identifying Relationships (Optional) (Derived data)	N	Y	Y	Y	Y	Y
Verb Phrases (Relationship Physical Name)	N	Y	Y	N	Y	Y
NULL Options	N/A	Y	Y	Y	Y	Y
Delete Rule (Referential Constraint)	N	Y	Y	Y	Y	Y
Subtype Relationship	N	N	N	N	N	N
Physical Objects	N	Y	N	N	N	N
Synonym/Alias	N/A	Y	N/A	N/A	N/A	Y
Triggers	N	Y	Y	Y	Y	N
Stored Procedures	N	Y	Y	Y	Y	N
Pre & Post Script	N	N	N	N	N	N

> **Note**: *Information on Netware SQL provided for reference only. Netware SQL is no longer supported in ERwin.*

Summary of ERwin Features Supported via Reverse Engineering (continued)

ERwin Feature	SQL Anywhere	SQL Base	SQL Server	Sybase	WATCOM	Teradata
Entities	Y	Y	Y	Y	Y	Y
Attributes	Y	Y	Y	Y	Y	Y
Primary Keys	Y	Y	Y	Y	Y	Y
Foreign Keys	Y	Y	Y	Y	Y	Y
Alternate Keys (Unique Index)	Y	Y	Y	Y	Y	Y
Inversion Entries (Non-Unique Index)	Y	Y	Y	Y	Y	Y
Rolenames (Derived data)	Y	Y	Y	Y	Y	N
Datatypes	Y	Y	Y	Y	Y	Y
User-Defined Datatypes	Y	N	Y	Y	N	N
Indexes	Y	Y	Y	Y	Y	Y
Identifying Relationships (Derived data)	Y	Y	Y	Y	Y	Y
Non-Identifying Relationships (Mandatory) (Derived data)	Y	Y	Y	Y	Y	Y
Non-Identifying Relationships (Optional) (Derived data)	Y	Y	Y	Y	Y	Y
Verb Phrases (Relationship Physical Name)	Y	Y	N	N	Y	N
NULL Options	Y	Y	Y	Y	Y	Y
Delete Rule (Referential Constraint)	Y	Y	N	N	Y	N
Subtype Relationship	N	N	N	N	N	N
Physical Objects	N	N	Y	Y	Y	Y
Synonym/Alias	N/A	Y	N/A	N/A	N/A	N/A
Triggers	N	N	Y	Y	Y	N
Stored Procedures	N	Y	Y	Y	Y	Y called MACROs
Pre & Post Script	N	N	N	N	N	N

The features supported on your target server may vary depending on which version you have installed. For example:

♦   Triggers and Stored Procedures are available on ORACLE version 7 (and later).

♦   Triggers and Stored Procedures are available on WATCOM version 4 (and later).

♦   Stored Procedures are available on SQLBase version 6.0 (and later).

6 ERwin Terms and Definitions

alternate key

1) An attribute or attributes that uniquely identify an instance of an entity.

2) If more than one attribute or group of attributes satisfies rule 1, the alternate keys are those attributes or groups of attributes not selected as the primary key.

ERwin will generate a unique index for each alternate key.

application database

A database that stores business information.

architect

A ModelMart security profile that is typically assigned to a more experienced modeler who is responsible for managing one or more ModelMart libraries, including the diagrams and shared objects in these libraries.

archive

A record of the changes made to a diagram since the diagram was last saved to the ModelMart. If the Auto Archive option for a ModelMart library is enabled, each time a user saves a diagram to that library, ERwin automatically generates an archive. You can use an archive to rollback all or part of the parent diagram in the ModelMart to the state represented by the archive. ERwin automatically assigns a default name to a diagram archive according to the following convention: **<diagram name>:<user name> on <date>;<archive number>.**

attribute

An attribute represents a type of characteristic or property associated with a set of real or abstract things (people, places, events, etc.). The logical equivalent to a column.

attribute group

A set of attributes that are used to index the table. An attribute group can be an a primary key, an alternate key (AK) or an inversion entry (IE).

basename

The original name of a rolenamed foreign key.

binary relationship

A relationship in which exactly one instance of the parent is related to zero, one, or more instances of a child. In IDEF1X, identifying, non-identifying, and subtype relationships are all binary relationships.

business rule

Logic applied to the use of data within the business, such as "Quantity_Ordered must be less than Quantity_On_Hand." Business rules are implemented using defaults, validation rules, referential integrity, relationship cardinality, triggers, and stored procedures.

cardinality

The ratio of parent instances to child instances. In IDEF1X, the cardinality of binary relationships is 1:n, whereby n may be one of the following:

♦ zero, one, or more - signified by a blank space

♦ one or more - signified by the letter P

♦ zero or one - signified by the letter Z

♦ exactly n - where n is some number

cascade delete

A referential integrity option. Using the cascade delete option, when you delete a parent instance, all dependent child instances are also deleted (e.g., deleting Order #123 causes the software to also delete all Line Items for Order #123.)

change control

The process of reviewing your changes and/or comparing them with changes saved by other users to the ModelMart master model, resolving conflicts between your changes and the ModelMart master model, and selectively merging your changes back to ModelMart.

column display property

Column definition information that is used to format column data or the column header text when it is displayed in a client application (e.g., display formats, number of decimal places displayed, and column header text).

column storage property

Column definition information that is used to validate information stored on the server (e.g., valid values, range).

column property

A specification of data attributes for the client or the server. Server-side column properties include datatype, null string, validation rules, and default/initial values. You can also define client-side column properties, including edit style, display format, client-side validation rule, and client-side default/initial value.

complete subtype cluster

If the subtype cluster includes all of the possible subtypes (every instance of the generic parent is associated with one subtype), then the subtype cluster is complete. For example, every employee is either male or female, and therefore the subtype cluster of male-employee and female-employee is a complete subtype cluster.

conflict resolution

The process of using the ModelMart Change Control Manager - Conflict Resolution dialog, to compare conflicting changes between your copy of a ModelMart diagram and the master model of the same diagram. To resolve a conflict, you can choose to cancel or save your changes to the ModelMart master model.

database

A database is a reserved amount of space on one or more storage devices that is used to store data and the definitions of database objects such as tables and indexes.

datatype

A predefined set of characteristics for a column that specifies field length, acceptable characters, and optional and required parameters. For example, char(18) specifies that the column can store up to 18 alpha and numeric characters.

DataWindow

A PowerBuilder DataWindow is a control that lets you display and manipulate data from a data source.

DataWindow Wizard

A set of dialogs that let you use information from an ERwin diagram as a data source for a PowerBuilder DataWindow.

default/initial value

A column property that sets a default or initial entry in a column if a user does not enter a value; a column can be assigned different default values for the server and client.

dependent entity

An entity whose instances cannot be uniquely identified without determining its relationship to another entity or entities.

discriminator

An attribute whose values are used to determine the subtypes of the instance. For example, the value of the attribute "gender" in an instance of "employee" determines to which particular subtype (male-employee or female-employee) that instance belongs.

display format

A column property that lets data be displayed in a different format from how it is stored; for example, a data or currency value.

domain

A group of predefined column property characteristics that can be saved and attached to columns to speed model development.

edit style

A column property that determines how data is displayed in an application form; for example, as an edit field, a drop-down box, a check box, or a set of option buttons.

entity

An entity represents a set of real or abstract things (people, places, events, etc.) which have common attributes or characteristics. Entities may be either independent or dependent. The logical equivalent to a table.

ERwin dictionary

The database that is generated from the ERwin metamodel, it stores information about the data structures used in models rather than the business information stored in other databases.

ERwin Custom Control

A Visual Basic custom control that you can draw on a form to access the ERwin Form Wizard dialog.

ERwin Form Wizard

A special dialog that lets you use information from an ERwin diagram as a data source for a Visual Basic form.

ERX format

ERwin's native text file format, lets you save the information stored in a graphical data model as a text description.

extended attributes

Also called "extended column definition information," represents information you define to control the display and validation of data in a column.

foreign key

An attribute that has migrated through a relationship from a parent entity to a child entity.

foreign key migration

When the key from a parent entity automatically appears in the key of the child entity with the designation of (FK) for foreign key.

forward engineering

The process of generating the physical database schema from the logical data model.

guest

The ModelMart security profile that is assigned by default to new users. The Guest profile has no associated permissions.

IDEF1X

ICAM Definition Method 1 Extended. A methodology for graphically depicting entities, attributes, and entity relationships.

identifying relationship

A relationship whereby an instance of the child entity is identified through its association with a parent entity. The primary key attributes of the parent entity become primary key attributes of the child.

Information Engineering

(IE). A methodology for graphically depicting entities, attributes, and entity relationships. IE uses a different notation than IDEF1X for depicting relationships and cardinality.

incomplete subtype cluster

If the subtype cluster does not include all of the possible subtypes (every instance of the generic parent is not associated with one subtype), then the subtype cluster is incomplete. For example, if some employees are commissioned, a subtype cluster of salaried-employee and part-time employee would be incomplete.

independent entity

An entity whose instances can be uniquely identified without determining its relationship to another entity.

inheritance

When ERwin migrates a primary key attribute, by default, the foreign key created in the child entity inherits the name but not the definition of the primary key attribute. If you also want the primary key attribute's definition to migrate to the child entity, you must change ERwin*'s* default setting.

input mask

A string of symbols that controls how you enter data in a field; for example, the mask (999) 999-9999 requires digits to be entered for a telephone number and automatically inserts the format characters and space.

instance

An occurrence of data within a table. Also called a record or a tuple.

inversion entry

An attribute or set of attributes that do not uniquely identify an instance of an entity, but are often used to access instances of entities. ERwin will generate a non-unique indexes for each inversion entry.

key

Any field or set of fields which uniquely identify an instance in a table.

library

An environment in the ModelMart that includes one or more related ModelMart diagrams and the set of library-level objects (e.g., domains, validation rules, stored procedures, etc.) shared by those diagrams.

library-level object

An object definition stored in a ModelMart library that is available to all ModelMart diagrams stored in the library. Library-level objects include: display formats, domains, edit styles, pre- and post-script templates, rollback segments, stored procedures, tablespaces, trigger templates, and validation rules.

lock mode

The current lock state of a ModelMart diagram. The possible modes are: Locked, Unlocked, or Read-Only, which correspond to the locking option chosen when the diagram is opened. A user can change the lock mode of a ModelMart diagram while working on it, unless it was opened in read-only mode or another user has a locked copy of the same diagram.

locking option

An option available when you open a ModelMart diagram. The available options are: Locked, Unlocked, or Read-Only. The Locked option gives a user exclusive rights to the master model. The Unlocked option lets users save changes to the master model in the ModelMart on a first-come-first-served basis. The Read-Only options lets you view the master model in the ModelMart, but you cannot save changes to the ModelMart.

logical level

Thinking and modeling things directly from the real world.

many-to-many

A relationship between two entities where instances in one entity have zero, one, or more related instances in the other entity, and vice versa (e.g., a many-to-many relationship exists between students and classes. Each student can attend many classes; each class can include many students).

master model

The master copy of a ModelMart diagram, which is stored in the ModelMart.

metamodel

The ERwin model that defines the data structures needed to store all the definition, location, font, color, and other required information about your diagram.

model management system

A system for storing, retrieving, versioning, and maintaining models used to support software development projects. Logic Works ModelMart is designed to be a common model management system for all Logic Works products. By letting you store ERwin, BPwin, and OOwin objects in our ModelMart, Logic Works provides an integrated environment for entity-relationship, business-process, and object-oriented model development.

modeler

A ModelMart default security profile that is typically assigned to ModelMart users that need to create, update, and delete ERwin objects in a ModelMart diagram, but are restricted from updating library-level objects.

ModelMart

A special database located on a DBMS server that stores ModelMart diagrams in SQL tables. The ModelMart also contains the stored procedures and triggers ModelMart uses to manipulate ModelMart diagrams, and merge changes when diagrams are replicated and saved.

ModelMart Administrator

By default, the person who creates the ModelMart is automatically assigned an Administrator security profile, which gives him/her unlimited access to all ModelMart objects and lets him/her create additional users and assign their security privileges.

ModelMart Control Tables

Two special SQL tables that ModelMart uses in conjunction with several stored procedures to track ModelMart software license usage information.

ModelMart Diagram

An ERwin diagram saved in the ModelMart that can be edited on a workstation.

ModelMart License

A license that specifies the maximum number of users that can log on to the ModelMart. When it's necessary, you can upgrade your ModelMart License to add more users.

ModelMart Model Merge

The combining of two independent ERwin diagrams into a single ModelMart diagram with one set of unique objects. ERwin eliminates redundancy between the two models by consolidating duplicate objects.

ModelMart Workgroup

The members of a modeling team who use ERwin to create and edit a shared entity-relationship model and who store a master version of that model in the ModelMart. The role of each workgroup member is defined by his/her assigned security profile (e.g., Administrator, Architect, Modeler), which may limit or grant permission to save changes back to ModelMart.

MPD ModelPro format

A type of file that stores data model information in text format.

n-ary relationship

A relationship that occurs when two or more tables are parents to a single child table. (When a single parent-child relationship exists, the relationship is called binary.)

non-key attribute

Any attribute that is not part of the entity's primary key. Non-key attributes may be part of an inversion entry and / or alternate key, and may also be foreign keys.

nonidentifying relationship

A relationship whereby an instance of the child entity is not identified through its association with a parent entity. The primary key attributes of the parent entity become non-key attributes of the child.

nonspecific relationship

Both parent-child connection and subtype relationships are considered to be specific relationships because they define precisely how instances of one entity relate to instances of another. However, in the initial development of a model, it is often helpful to identify "non-specific relationships" between two entities. A nonspecific relationship, also referred to as a "many-to-many relationship," is an association between two entities in which each instance of the first entity is associated with zero, one, or many instances of the second entity and each instance of the second entity is associated with zero, one, or many instances of the first entity.

normalization

The process by which data in a relational construct is organized in order to minimize redundancy and non-relational constructs.

null

Having no value, an "empty" or non value.

one-to-many

a relationship where each parent instance has zero, one, or more child instances.

operation

An action you can perform (i.e., create, update, or delete) on an ERwin object as specified in a security profile.

owned attribute

An attribute that is not a foreign key. An owned attribute represents the primary reference to an attribute within a single model.

parent diagram

In ModelMart, a parent diagram is the only read/write version in a set of related diagrams. In addition to the parent diagram, the set may include one or more archive versions and/or snapshots, which are read-only.

permission

An authorization granted in a security profile that lets a user perform an operation (i.e., create, update, delete) on a specific type of object (e.g., entity).

permission object class

A container object that groups ERwin objects together so that you can assign permissions to the group in a security profile. ERwin uses five permission objects classes: ModelMart, Library, Diagram, Subject Area, and Entity. When you assign a permission to an object class (e.g., Entity), you automatically assign the permission to each type of ERwin object in the object class.

physical level

Database- and DBMS-specific model information; e.g., tables, columns, datatypes, etc.

physical storage object

A named partition or area on a storage device used to store object definitions and/or data.

pre/post script

SQL scripts that you want ERwin to execute immediately before or after the rest of the schema is generated.

primary key

1) An attribute or attributes that uniquely identify an instance of an entity.

2) If more than one attribute or group of attributes satisfies rule 1, the primary key is chosen from this list of candidates based on its perceived value to the business as an identifier. Ideally, primary keys should not change over time, and should be as small as possible.

ERwin will generate a unique index for each primary key.

recursive relationship

A relationship that exists when instances in a table have a relationship with other instances in that same table (e.g., in the EMPLOYEE table, an employee can manage other employees).

referential integrity

(RI). The assertion that the foreign key values in an instance of a child entity have corresponding values in a parent entity.

referential integrity trigger

A trigger that is used to maintain integrity between two related tables.

refresh

The ability to update an open ModelMart diagram with changes saved to the master model in the ModelMart during a modeling session. While you are working on a ModelMart diagram, ERwin lets you import changes made by others to keep your copy of the model current, and lets you review and compare your changes with information already saved to the master model in the ModelMart.

relationship

Represent connections, links or associations between entities.

report browser

A versatile reporting module for browsing and generating reports on ERwin diagrams and ModelMart information.

restrict

A referential integrity option that prohibits the deletion of a parent instance if one or more dependent child instances exist.

results set

The results returned by an ERwin or ModelMart report.

reverse engineering

The process of generating a logical model from a physical database.

revert

The process of resetting the master model in the ModelMart to the state captured in an archive version of a ModelMart diagram. By overwriting the ModelMart master model in the ModelMart with a specific archive version, you can rollback (undo) changes made to the master model.

rolename

A new name for a foreign key. A rolename is used to indicate that the foreign key performs a specific function (or role) in the entity.

schema

The structure of a database. Usually refers to the DDL (data definition language) script file. DDL consists of CREATE TABLE, CREATE INDEX, and other statements.

security profile

A set of permissions that you can assign to one or more ModelMart users. You can assign different security profiles to a user for different permission objects (i.e., ModelMart, Library, Diagram, Subject Area) in the ModelMart.

session

A period of time in which a user has one or more ModelMart diagrams open or locked.

set null

A referential integrity option that automatically sets the foreign key of a child instance to null when the parent instance is deleted.

SML format

A special type of file designed to store information about an Entity-Relationship model in text format.

snapshot

A special version of a ModelMart diagram that records the status of the diagram when you opened it. ERwin uses the snapshot as a baseline to track your changes and generate a transaction list that shows the changes made to a model when you attempt to save your copy of the ModelMart diagram back to the master model in the ModelMart. You can also create a snapshot when you save an ER1 version of a ModelMart diagram to work on off-line. Later, when you want to merge your changes into the ModelMart diagram, ModelMart uses the snapshot to compare your changes, and the changes made by other users, to the master model in the ModelMart. ERwin automatically assigns a default name to a snapshot according to the following convention: **[nn: <user name> on <date>]**.

source diagram

The diagram that contains the objects that will be merged into a target diagram when you merge two diagrams. When the merge is complete, the source diagram is unaltered.

specific relationship

A specific relationship is an association between entities in which each instance of the parent entity is associated with zero, one, or many instances of the child entity.

stored display

An alternative presentation of a Subject Area or model that highlights a particular aspect of the total data structure. A Stored Display includes all the objects in the parent Subject Area or model, but the objects may positioned differently and the diagram can be set to a different display level.

stored procedure

A block of SQL Code, similar to a trigger, that is stored on the server for quick execution. It resembles a trigger except it is invoked by another program that passes the procedure name to the server.

subject area

A named version of a data model that may include all the entities, relationships, subtypes, and text blocks, or any subset of the objects in the complete data model.

subtype entity

In the real world, we often encounter entities which are specific types of other entities. For example, a salaried employee is a specific type of employee. Subtype entities are useful for storing information that only applies to a specific subtype. They are also useful for expressing relationships that are only valid for that specific subtype, such as the fact that a salaried employee will qualify for a certain benefit, while a part-time-employee will not.

In IDEF1X, subtypes within a subtype cluster are mutually exclusive.

subtype relationship

A subtype relationship (also known as a categorization relationship) is a relationship between a subtype entity and its generic parent. A subtype relationship always relates one instance of a generic parent with zero or one instance of the subtype.

synchronizing

The process of comparing an ERwin model with an existing database, and updating the model, the database definition, or both to ensure that they are identical.

target diagram

The diagram in which the source diagram objects are merged when you merge two diagrams. When the merge is complete, the target diagram contains all the unique objects from both the source and the target diagrams.

target server

DBMS in which the physical schema is created.

template

Stores diagram settings and ERwin objects that are automatically applied to each new diagram based on the template. Templates provide the ideal way of maintaining a consistent look and feel across models.

trigger

A named set of precompiled SQL statements stored on the server that is automatically executed when a specified event occurs.

unification

The merging of two or more foreign key attributes into a single foreign key attribute based on the assertion that the values of the original foreign key attributes must be identical.

valid value

One item in a fixed list of all the acceptable values that can be stored in a column; the list can be assigned to a validation rule for control of data entry; a class style can be applied to present the valid values as a combo box or as a set of option buttons.

validation rule

A column property that restricts what values can be stored in a column. A column can be assigned different validation rules for the server and client.

version comparison

The process of comparing differences between an earlier version (archive version or snapshot) and a parent diagram. Optionally, you can revert back to an earlier version by saving a version under the same name as the parent diagram.

viewer

A ModelMart default security profile that is typically assigned to users who should be able to view, but not alter ModelMart diagrams.

View

A SQL query that is permanently stored in the database under an assigned name. The result of the view query is a virtual table. A view is similar to a table, however, unlike a table a view does is not a permanently stored set of data values. Instead, the rows and columns of data visible through the view are the results of the database query that defines the view.

View Column Expression

A user-defined or arbitrary expression that is stored as a view column.

View Relationship

A relationship between a source table and a view that indicates that one or more of the view columns are references to columns in the source table. A view relationship is sometimes referred to as a derivation.

Version

A special read-only version of a ModelMart diagram that reflects the state of the parent diagram at some point in its history. You can use a version to rollback all or part of the parent diagram in the ModelMart to the state captured in the version. Workgroup members can create versions and annotate the purpose or content of each version. ERwin automatically assigns a default name to a version according to the following convention: **Vn: <diagram name>**, where V indicates a version, and n is a unique number.

Index

Notes

Notes

Documentation Comments Form   **ERwin Version 3.0 Reference Guide**

Logic Works is interested in your feedback on this documentation. You can use this form if you have compliments or questions, or would like to report problems in the documentation.

Please fax or mail completed forms to:

Documentation Manager
Logic Works Inc
University Square at Princeton
111 Campus Drive
Princeton, NJ 08540

Fax: (609) 514-0184

Comments:

Please enter your comments in the space provided below:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Please include your name, address, and telephone number in the space below:

_____

_____

_____

Would you like a Logic Works representative to contact you?   Yes ☐   No ☐