Product Documentation

ER/Studio® Data Architect

User Guide

Version 9.7.1/XE5

© 2014 Embarcadero Technologies, Inc. Embarcadero, the Embarcadero Technologies logos,and all other Embarcadero Technologies product or service names are trademarks or registeredtrademarks of Embarcadero Technologies, Inc. All other trademarks are property of theirrespective owners.

Embarcadero Technologies, Inc. is a leading provider of award-winning tools for applicationdevelopers and database professionals so they can design systems right, build them faster andrun them better, regardless of their platform or programming language. Ninety of the Fortune100 and an active community of more than three million users worldwide rely on Embarcaderoproducts to increase productivity, reduce costs, simplify change management and complianceand accelerate innovation. The company's flagship tools include: Embarcadero® ChangeManager™, CodeGear™ RAD Studio, DBArtisan®, Delphi®, ER/Studio®, JBuilder® and RapidSQL®. Founded in 1993, Embarcadero is headquartered in San Francisco, with offices locatedaround the world. Embarcadero is online at www.embarcadero.com.

April, 2014

CONTENTS

Welcome to ER/Studio Data Architect 23
- About this Document 23
- Notice for Developer Edition Users 23
- Important Name Change Notice 24
- Product Benefits by Audience 24

Overview 25
- ER/Studio Family Products 25
- Application Design 26

Application Interface 27
Data Model Explorer 29
- Data Model 29
- Data Dictionary 30
- Data Lineage 31
- Macros 32

Data Model Window 33
Data Lineage Window 34
Pop-up Windows 35
Zoom Window 36
Overview Window 37
Menus 38
- Application Menus 38
- Accessing Shortcut Menus 38

Toolbars 39
- Moving Toolbars 39
- Undock a Toolbar 39
- Dock a Toolbar 39
- Displaying Toolbars 39

Keyboard Commands 40
- Basic Functions 40
- Function Keys 40
- Shortcuts 40
- Hot Keys 41
- Full Undo Redo 41

Status Bar 42
Configuring and Customizing 43
Customizing the Display of Diagrams and Objects 44
- Diagram tab 45
- Entity/Table tab 45
- Relationship tab 46
- View tab 46
- Schema Objects tab 47
- Transformation tab 47
- Drawing Shapes tab 47

Security Objects tab 47
Apply To tab 47
Defining Model Options for the Selected Model 48
General Options tab 48
Name Handling Options tab 49
Customizing Standard Model Features for New Models 51
Application tab 51
Logical tab 52
Physical tab 52
Name Handling tab 53
Display tab 54
Directories tab 54
Tools tab 55
Diagram tab 55
View tab 55
Schema Objects tab 56
Object Types tab 56
Object Names tab 56
Automation Options tab 56
Data Dictionary tab 56
Comments tab 57
Where Used tab 57
ERX File Import tab 57
Undo tab 57
Repository Options tab 57
Specifying the Location of Shared Data 60
Changing Cursor Popup Help Options 61
Data Modeling Fundamentals 62
Data Modeling Concepts 63
The Logical Model 63
The Physical Design 63
Data Model Design Principals 63
The Purpose of a Data Model 63
The Relational Model 64
The Entity-Relationship Model 64
The Dimensional Model 65
Developing a Data Model 66
Starting With a Basic Data Model 66
Creating a Basic Data Model 66
Reverse Engineering a Database 66
Importing a File to Create a New Database 66
Ensuring Data Integrity 66
Automating Data Model Creation and Maintenance 67
Fine Tuning the Data Model 67
Securing the Data Model 67

Preparing for Review and Implementation ... 67
Maintaining the Data Model ... 68
Using ER Studio DA ... 69
Creating and Working With Data Models ... 70
Creating a New, Blank Data Model ... 71
Reverse Engineering an Existing Database ... 72
Page 1 ... 72
Page 2 ... 74
Page 3 ... 74
Page 4 ... 74
Page 5 ... 75
Notes ... 75
Specifying Application Options for Reverse Engineering ... 76
Importing a Model ... 77
Importing a Model from External Metadata ... 78
Importing a Model from a SQL File ... 81
Importing a Model from an ERX File ... 83
Generating a Physical Data Model ... 84
Creating a New Physical Model ... 85
Using the Compare and Merge Utility ... 86
What Can the Compare/Merge Utility Do? ... 86
Comparing Models and Submodels ... 87
Page 1 ... 88
Page 2 ... 89
Page 3 ... 89
Page 4 - Select Specific Model Objects ... 90
Page 5 - Results ... 91
Forcing a Match ... 91
Create a new diagram with one entity called *Entity1* with a primary key of A. ... 91
Updating an Offline DBMS ... 95
Updating a Model or Submodel ... 95
Resolving and Reporting on Model Differences ... 96
Changing the Database Platform of a Model ... 98
Common Tasks ... 99
Using Toolbar Tools ... 100
Creating a Data Model Object ... 101
Moving Objects ... 102
Moving Objects in the Data Model Window ... 102
Moving Data Model Objects ... 102
Moving Relationship Docking Points ... 103
Moving Data Model Objects in the Data Model Explorer ... 103
Moving Objects to Different Nodes ... 103
Moving Submodels ... 103
Copying Objects ... 105
Copying an Object in the Data Model Window ... 105

Copying Multiple Objects ... 105
Copying an Attribute ... 106
Copying an Object from the Data Model Explorer to the Data Model Window ... 106
Copying Objects Between Models and Submodels ... 106
Resizing Objects ... 109
Manually Resize Data Model Objects ... 109
Automatically Resize Data Model Objects to Fit Text ... 109
Changing Model Alignment, Distribution, and Layout ... 110
Finding an Entity, Table or View ... 111
Locating an Object in the Data Model Window ... 112
Editing an Object ... 113
Adding and Editing Object Comments ... 114
Editing the Name of an Object ... 116
Deleting an Object ... 117
Renaming an Object ... 118
Searching for and Changing Object or Object Attribute Names ... 119
Creating and Editing Database Views ... 121
Notes ... 122
Entity/Table tab ... 123
Attribute/Column tab ... 123
Column Alias dialog ... 123
Where tab ... 124
Group By tab ... 124
Having tab ... 124
DDL tab ... 124
SQL Validation dialog ... 124
Options tab ... 124
Definition tab ... 124
Notes tab ... 124
Where Used tab ... 124
Dependencies tab ... 125
PreSQL & PostSQL tab ... 125
Permissions tab ... 125
Security Information tab ... 125
Attachment Bindings tab ... 125
Defining the Default View Settings for New Models ... 126
Customize the View Settings for a Specific Model ... 127
Creating and Editing Submodels ... 128
Edit Submodel tab ... 128
Attributes tab ... 129
Definition tab ... 129
Attachment Bindings tab ... 129
Security Information tab ... 129
Notes ... 129
Synchronizing Submodels ... 130

Example 1 130
Example 2 130
Example 3 130
Using the Submodel Synchronization Utility 132
Page 1 132
Page 2 132
Page 3 132
Page 4 132
Notes 133
Copying a Database Object to a Submodel (Apply to Submodel) 134
Establishing Database Security 135
Creating and Editing Database Users 136
General page/tab 137
System Privileges page/tab 137
Object Privileges page/tab 137
Dependencies page/tab 137
Definition page/tab 137
DDL tab 138
Attachment Bindings tab 138
Notes 138
Creating and Editing Database Roles 139
General page/tab 140
System Privileges page/tab 140
Object Privileges page/tab 140
Dependencies page/tab 140
Definition page/tab 141
DDL tab 141
Attachment Bindings tab 141
Notes 141
Associating Database Roles with New Database Users 143
Associating Database Users with New Database Roles 144
Changing Database Roles Associated with Existing Database Users 145
Changing Database Users Associated with Existing Database Roles 146
Granting and Revoking Permissions to Modify Database Objects 147
Comparing Models 148
Customizing the Data Model 149
Setting Diagram Properties 150
Attachment Bindings tab 150
Setting Data Model Properties 151
Security Information tab 151
Attachment Bindings tab 151
PreSQL & PostSQL tab 152
Name Handling tab 152
Creating Entity Relational or Dimensional Models 153
What is a Dimensional Model? 153

Dimensional Model Table Type Determination 153
Dimensional Model Table Types 154
Dimensional Table Types Indepth 156
Fact 156
Dimension 157
Snowflake 159
Bridge 160
Hierarchy Navigation 160
Undefined 160
Changing the Model Notation 161
Setting the Relationship Notation 162
IDEF1X versus IE Relationship Notation 162
IDEF1X Notation 162
IE (James Marting) Notation 163
IE (Crow's Feet) Notation 164
FItered ID (Hide Foreign Keys) Notation 165
Changing the Relationship Notation for all Diagram Models 166
Changing the Relationship Notation for a Selected Model 166
Manipulating Relationship Lines 167
Changing the Style of Relationship Lines 167
Create an N Bend 168
Straightening an N Bend 168
Change the Disposition of Relationship Lines 169
Changing the Model Background 170
Change the Background and Outline Colors of Existing Models and Views 171
Setting Default Diagram Colors and Fonts 172
Overriding Color and Font Settings for a Specific Object 173
Change Color and Font Settings for an Entity, Table, Attribute, Column or View 173
Change Entity/Table Background and Outline Colors 174
Change the Background and Outline Colors of Views in an Open Data Model ... 174
Changing the Color of Relationships 175
Change Data Model Display Preferences 176
Changing Entity/Table Display Preferences 177
Customizing the Layout 178
All tabs 178
Disconnected tab 178
Circular tab 179
Hierarchical tab 179
Orthogonal tab 180
Symmetric tab 180
Tree tab 180
Creating and Editing Shapes 181
Shape Text tab 182
Colors and Fonts tab 182
Attachment Bindings tab 182

Locking Data Model Objects to a Drawing Shape 183
Adding and Editing Lines 184
Properties tab 184
Attachment Bindings tab 184
Notes 184
Adding and Editing Title Blocks 186
Adding Text Blocks 187
Changing Data Model Layout 188
Redraw a Data Model Using Auto Layout 188
Notes 188
Enforcing Data Integrity Using Check Constraints 190
Checking Out a Concurrent License for Offline Usage 191
Developing the Logical Model 192
Logical Design Concepts 193
Logical Design Documentation 194
Object Name Rules 195
Normalization 196
Creating and Editing Entities and Tables 198
Creating and Editing Entities 199
All tabs 200
Attributes/Columns tab 200
Keys tab 200
Relationships tab 200
Definition tab 200
Note tab 201
Where Used tab 201
User-Defined Mappings tab 201
Constraints tab 201
Dependencies tab 201
Permissions tab 201
Naming Standards tab 202
Data Lineage tab 202
Security Information tab 202
Attachment Bindings tab 202
Notes 203
Creating and Editing Tables 204
Columns tab 205
Dimensional tab 205
DDL tab 205
Indexes tab 205
Foreign Keys tab 205
Definition tab 205
Note tab 206
Where Used tab 206
User-Defined Mappings tab 206

Storage tab ... 206
Dimensions tab ... 206
Properties tab ... 206
Partition Columns tab ... 207
Distribute Columns tab ... 207
Distribution tab ... 207
Organization tab ... 207
Partitions tab ... 207
Overflow tab ... 207
Constraints tab ... 207
Dependencies tab ... 208
Capacity Panning tab ... 208
Permissions tab ... 208
PreSQL & Post SQL tab ... 208
Naming Standards tab ... 208
Compare Options tab ... 208
Data Lineage tab ... 208
Security Information tab ... 209
Attachment Bindings tab ... 209
Notes ... 209
Creating and Editing Attributes and Columns ... 211
Creating and Editing Attributes ... 212
Main page ... 212
Datatype tab ... 213
Default tab ... 214
Rule/Constraint tab ... 214
Definition tab ... 214
Notes tab ... 214
Where Used tab ... 214
User-Defined Mappings tab ... 215
Reference Values tab ... 215
Naming Standards tab ... 215
Compare Options tab ... 215
Data Lineage tab ... 215
Security Information tab ... 215
Attachment Bindings tab ... 215
Data Movement Rules tab ... 216
Notes ... 216
Creating and Editing Columns ... 218
Main page ... 219
Datatype tab ... 220
Default tab ... 223
Definition tab ... 224
Notes tab ... 224
Where Used tab ... 224

User-Defined Mappings tab 224
Reference Values tab 224
Naming Standards tab 224
Compare Options tab 225
LOB Storage tab 225
Data Lineage tab 225
Security Information tab 225
Attachment Bindings tab 225
Data Movement Rules tab 225
Notes 225
Creating and Editing Keys 227
Create a Primary Key 227
Create an Alternate Key or Inversion Entry 228
Create a Foreign Key/Relationship 228
Specify FK Attribute Name and Datatype 229
Associate Similar Objects Using User-Defined Mappings 230
Create User-Defined Mappings 231
Definition tab 232
Attachment Bindings tab 232
Notes 232
Working with Universal Mappings 234
Creating Universal Mappings through the Compare/Merge Utility 235
Creating a Universal Mapping using the Universal Mapping Editor 238
Viewing Universal Mappings 240
Viewing and Editing Denormalization Mappings 241
Working with Relationships 244
Creating and Editing Relationships 245
All tabs 246
Properties tab 248
Phrases tab 250
Name tab 250
Trigger tab 250
Definition tab 250
Compare Options tab 251
Note tab 251
Rolename tab 251
Attachment Bindings tab 251
Notes 251
Creating and Editing Recursive Relationships 253
Creating View Relationships 255
Creating Subtype Cluster Relationships 256
Subtype tab 258
Attachment Bindings tab 258
Notes 258
Resolving Name Duplication in Parent and Child Entities (Unification) 260

Attribute Name Editing Options ... 260
Examples of Duplicate Name Resolutions ... 261
Replace Native Attribute with Propagating Attribute ... 261
Rolename Propagating Attribute ... 261
Change Native Attribute Name to Allow Propagating Attribute Name ... 262
Unify Native Attribute with Propagating Attribute ... 262
Validating the Model ... 263
Validation Options tab ... 263
Object Selection tab ... 263
Notes ... 264
Customizing Datatype Mappings ... 265
Developing the Physical Model ... 267
Transforming Model Objects from Logical to Physical ... 269
Physical Storage Considerations ... 271
Defining Table Storage ... 272
GreenPlum Table Storage Options ... 272
HiRDB Table Storage Options ... 273
HiRDB LOB Column Storage ... 273
IBM DB2 Common Server and IBM DB2 for LUW Table Storage Options ... 273
IBM DB2 for OS/390 Table Storage Options ... 274
Informix Table Storage Options ... 275
Interbase Table Storage Options ... 275
Microsoft SQL Server Table Storage Options ... 275
Teradata Table Storage Options ... 276
Oracle 7 Table Storage Options ... 277
Oracle 8.x, 9i, 10g, and 11g Table Storage Options ... 278
Oracle LOB Segment Storage Options ... 280
PostgreSQL Table Storage Options ... 282
Sybase Table Storage Options ... 282
Define Table Overflow Options ... 283
Distributing Columns for GreenPlum ... 286
Distributing Columns for IBM DB2 for LUW 9.x (and later versions) ... 288
Distributing Columns for Netezza ... 289
Organizing Columns for Netezza ... 291
Partitioning a Table ... 292
Table Partition Options for Sybase ... 292
Table Partition Options for Oracle 8.x, 9i, 10g, 11g and 12c ... 292
Table Partition Editor for Oracle 8.x, 9i, 10g, 11g and 12c ... 294
Table Partition Options for IBM DB2 for OS/390 8.x, 9.x, and 10.x ... 295
Table Partition Editor for IBM DB2 for OS/390 8.x, 9.x, and 10.x ... 296
Table Partition Options for IBM DB2 for LUW 9.x and 10.x ... 296
Table Partition Editor for IBM DB2 for LUW 9.x (and later versions) ... 296
Table Partition Columns for IBM DB2 for LUW 5.x, 6.x, 7.x, 8.x ... 298
Defining Index Storage ... 299
GreenPlum Index Storage Options ... 299

Hitachi HiRDB Index Storage Options ... 299
IBM DB2 for OS/390 Index Storage Options ... 300
Informix Index Storage Options ... 300
Interbase Index Storage Options ... 300
Microsoft SQL Server Index Storage Options ... 300
Oracle 7.x Index Storage Options ... 301
Postgre SQL Index Storage Options ... 302
Sybase SQL Anywhere Index Storage Options ... 302
Sybase System 10, ASE 10, 11 Index Storage Options ... 302
Partitioning a Table Index ... 303
IBM DB2 for OS/390 Index Partitions Options ... 303
Oracle 8.x, 9i, 10g, 11g and 12c Index Partitions Options ... 304
Index Partition Editor for Oracle 8.x, 9i, 10g, 11g and 12c Options ... 305
Customizing Table DDL ... 307
Creating and Editing Indexes ... 308
Create an Index ... 309
Edit an Index ... 309
Index Editor Options ... 309
Columns tab ... 311
Properties tab ... 311
Options tab ... 313
Hitachi HiRDB Index Options ... 314
IBM DB2 for AS/400, Common Server, and UDB Versions 5 through 9 Index Options ... 314
IBM DB2 for OS/390 Index Options ... 314
Informix Index Options ... 317
Interbase Index Options ... 317
Microsoft Access and SQL Server Index Options ... 317
MySQL Index Options ... 318
Teradata Index Options ... 318
Oracle 7 Index Options ... 319
Postgre SQL Index Options ... 319
Sybase Index Options ... 319
Storage tab ... 321
Partitions tab ... 321
Compare Options tab ... 321
Attachment Bindings tab ... 321
Creating and Editing Database Dependent Objects ... 322
Creating and Editing Aliases ... 323
Name page/tab ... 323
Reference page/tab ... 323
Definition page/tab ... 324
DDL page/tab ... 324
Attachment Bindings tab ... 324
Creating and Editing Auxiliary Tables ... 325

Add an Auxiliary Table 325
Name page/tab 325
Base Table page/tab 326
Storage page/tab 326
Index Options page/tab 326
Definition page/tab 328
DDL page/tab 328
Attachment Bindings tab 328
Creating and Editing Buffer Pools 329
Create a Buffer Pool 329
Name page/tab 329
Nodes page/tab 330
Definition page/tab 330
DDL page/tab 330
Attachment Bindings tab 330
Creating and Editing DataBases 331
Add a Database 331
Name & Type page/tab 331
Option page/tab 332
Definition page/tab 332
DDL page/tab 332
Attachment Bindings tab 332
Creating and Editing Generators 334
To Create a Generator 334
Name page/tab 334
Definition page/tab 334
DDL page/tab 334
PreSQL & PostSQL tab 334
Attachment Bindings tab 335
Creating and Editing SQL Procedures 336
Create a Scripted Procedure 338
SQL page/tab 338
Description page/tab 338
Permissions page/tab 338
Attachment Bindings tab 338
Apply a Scripted Procedure to a Submodel or Model 340
Creating and Editing PreSQL and PostSQL for the CREATE TABLE Statement 341
Creating and Editing Procedures 342
Create Reusable Procedure 344
Create a Scripted User-Defined Procedure 345
Create a Templated User-Defined Procedure 346
Bind a Reusable Procedure to a Table 347
Create an SQL Procedure 348
Description tab 348
Attachment Bindings tab 348

Permissions tab ... 348
Notes ... 348
Export an SQL Procedure ... 350
Import an SQL Procedure ... 351
Creating and Editing Functions ... 352
Create a Stored SQL Function ... 352
Description tab ... 352
Permissions tab ... 352
Attachment Bindings tab ... 353
Notes ... 353
Creating and Editing Packages ... 354
Create a Package ... 354
Header tab ... 354
Body tab ... 354
Definition tab ... 355
Attachment Bindings tab ... 355
Permissions tab ... 355
Creating and Editing Triggers ... 356
Notes ... 357
Trigger Example ... 357
Notes ... 359
Create a Referential Integrity Trigger ... 359
Create a Scripted User-Defined Trigger ... 360
Notes ... 361
SQL tab ... 361
Description tab ... 361
Attachment Bindings tab ... 361
Create a Templated User-Defined Trigger ... 362
Notes ... 362
Description page/tab ... 362
Attachment Bindings tab ... 362
Create a Reusable User-Defined Trigger ... 362
Creating and Editing Materialized Query Tables ... 364
Add a Materialized Query Table ... 364
Name page/tab ... 364
Location page/tab ... 365
SQL page/tab ... 365
Definition page/tab ... 365
DDL page/tab ... 365
Attachment Bindings tab ... 365
Notes ... 365
Creating and Editing Materialized Views ... 366
Name page/tab ... 366
Refresh page/tab ... 367
Query page/tab ... 367

Transaction page/tab ... 368
Extent page/tab ... 368
Options page/tab ... 369
Definition page/tab ... 369
DDL page/tab ... 369
PreSQL & PostSQL tab ... 369
Permissions tab ... 369
Attachment Bindings tab ... 370
Requirements for the Fast Refresh Method ... 370
Creating and Editing Node Groups ... 373
Create a Node Group ... 373
Name page/tab ... 373
Definition page/tab ... 373
DDL page/tab ... 374
Attachment Bindings tab ... 374
Creating and Editing Object Types ... 375
Create an Object Type ... 375
Header and Body tab ... 376
Description tab ... 376
Attachment Bindings tab ... 376
Creating and Editing Rollback Segments ... 377
Create or Edit a Rollback Segment ... 377
Name tab ... 377
Storage tab ... 377
Definition tab ... 378
DDL tab ... 378
Attachment Bindings tab ... 378
Creating and Editing Sequences ... 379
Create or edit a Sequence ... 379
Name page/tab ... 379
Numbers Page/Tab ... 379
Options page/tab ... 380
Definition page/tab ... 380
DDL page/tab ... 380
PreSQL & PostSQL page/tab ... 380
Attachment Bindings page/ tab ... 380
Permissions page/tab ... 380
Creating and Editing StoGroups ... 382
Create a StoGroup ... 382
Options page/tab ... 382
Definition page/tab ... 383
DDL page/tab ... 383
Attachment Bindings tab ... 383
Creating and Editing Synonyms ... 384
Create a Synonym ... 384

Name page/tab 384
Reference page/tab 385
Definition page/tab 385
DDL page/tab 385
Attachment Bindings tab 385
Creating and Editing Tablespaces 387
Create a TableSpace 387
Name & Type page/tab 387
Name page/tab 388
Datafile page/tab 388
Storage page/tab 389
Options page/tab 389
Options 1 page/tab 389
Options 2 page/tab 390
Definition page/tab 390
DDL page/tab 390
Attachment Bindings page/ tab 390
Creating and Editing Database Schemas 392
Create a Database Schema 392
Object Privilege page/tab 392
Definition page/tab 392
DDL page/tab 392
Attachment Bindings tab 393
Upgrading Diagrams with Denormalization Mappings 394
Optimizing Query Performance on the Physical Model (Denormalizing the Physical Model) 397
Removing and Consolidating Child Columns 398
About Roll-Up Denormalization 398
Create a New Roll-up Mapping 399
Removing Unnecessary Parent Tables 400
About Roll Down Denormalization 400
Create a New Roll-Down Mapping 400
Partitioning a Table and Duplicating Columns 401
About Horizontal Split Denormalization 401
Create a New Horizontal Split Mapping 402
Replacing a Table with New Tables That Share the Primary Key 402
About Vertical Split Denormalization 402
Create a Vertical Split Mapping 403
Eliminating Joins by Copying Columns 403
About Column Map Denormalization 403
Create a New Column Map Mapping 403
Removing Unnecessary Tables 404
About Table Merge Denormalization 404
Create a New Table Merge Mapping 404
Synchronizing Physical and Logical Models 405

Planning for and Predicting Database Growth 406
Planning for Table and Index Storage 406
Predicting Table Growth 406
Predicting Database Growth 407
Reporting Database Growth Analysis 408
Connecting to Database Sources and Targets 409
Configuring ODBC Data Source and Target Connections 410
Using Native or Direct Database Connections 411
Troubleshooting Database Connections 413
IBM DB2 Connectivity Checklist 413
Informix Connectivity Checklist 414
Interbase Connectivity Checklist 414
Microsoft Access Connectivity Checklist 414
Microsoft SQL Server Connectivity Checklist 414
Oracle Connectivity Checklist 415
Sybase Connectivity Checklist 415
Working with the Data Dictionary 417
Importing a Data Dictionary 419
Copying and Pasting Data Dictionary Objects 420
Attaching External Documents to the Data Model 421
Create and Edit Attachment Types 421
Create and Edit Attachments 422
Bind an Attachment to an Object 422
Bind an Attachment to a Specific Object 423
Override an Attachment Value 423
Enforcing Security Using Data Security Types and Properties 424
Generate Default Security Information 424
Create and Edit a Data Security Type 425
Create and Edit a Data Security Property 425
Enhancing Data Integrity Using Defaults 426
Create and Edit Defaults 426
Promoting Data Integrity Through Rules 427
Example 427
Create and Edit Data Dictionary Rules 428
Defining Valid Attribute Data Using Reference Values 430
Reference Value tab 430
Definition tab 431
Attachment Bindings tab 431
Binding Information tab 431
Enforcing Naming Standards Using Naming Standards Templates 432
Creating a Naming Standards Template 432
Name tab 433
Logical and Physical tabs 433
Mapping tab 434
Order tab 434

General tab ... 435
Attachment Bindings tab ... 435
Binding Information tab ... 435
Track Versions of Naming Standards Templates ... 435
Ensuring Consistent Domain Definitions Using User Datatypes ... 436
Create and Edit User Datatypes ... 436
Reusing Attribute Definitions Using Domains ... 438
Create a Domain Folder ... 438
Create and Edit a Domain or Subdomain ... 439
Datatype tab ... 439
Default tab ... 440
Rule/Constraint tab ... 440
Reference Values tab ... 440
Naming Standards tab ... 440
Definition tab ... 440
Note tab ... 440
Override Controls tab ... 441
Attachment Bindings tab ... 441
Binding Information tab ... 442
Reusing Procedural Logic ... 443
Create and Edit Reusable Triggers ... 443
Create and Edit Reusable Procedures ... 444
Create and Edit Libraries ... 445
Library Example ... 445
Call to Library Example ... 446
Determining Which Objects are Bound by Data Dictionary Objects ... 447
Changing Data Dictionary Object Bindings and Values ... 448
Working with the Enterprise Data Dictionary ... 450
Documenting Data Extraction, Transformation, and Load ... 451
Introduction to Data Lineage ... 452
Data Lineage Workflow ... 454
Defining Source Systems ... 455
Creating a New Source ... 455
Importing External Source into Other Sources ... 456
Relating Source and Target Models ... 458
Relating Source and Target Tables and Columns ... 459
Assigning Model Data Movement Properties ... 460
Creating and Editing Data Lineage Data Flows ... 461
Create a Data Lineage Data Flow ... 461
All tabs ... 463
Columns tab ... 463
Definition tab ... 463
Data Movement Rules tab ... 463
Attachments tab ... 463
Documenting Column/Attribute ETL ... 465

Documenting Table/Entity Data ETL 466
Saving and Using Quick Launch Settings 467
Generating RTF and HTML Model Reports 469
Generate a Report 471
Exporting the Data Model 474
Exporting Metadata for a Particular Model 475
Exporting Metadata for all Models and Submodels 476
Exporting a Data Model to XML 477
Generating a Document Type Definition (DTD) 479
Generating a Schema Definition (XSD) 480
Generating a Customized XML Schema 481
Generating a Script File or Database 498
    Page 1 498
    Page 2 499
    Tables tab 499
    General Options tab 500
Creating SQL Procedures to Run Before and After Generating a Physical Model View 501
Creating SQL Procedures to Run Before and After Generating a Table 502
Creating SQL to Run Before and After Generating a Physical Database 503
Exporting an Image of the Data Model 504
Printing the Data Model 505
Working with ERStudio Repository 506
Specifying Repository and ER/Studio TS Options 508
Setting Repository and ER/Studio TS Options 509
Repository Properties 512
Logging In and Out of the Repository and ER/Studio TS 513
Synchronizing Comments 514
Multi-threading in the Repository 515
Establishing Security for the Repository 522
Creating and Managing Groups 525
Creating and Managing Roles 529
    Notes 529
Creating and Managing Users 535
Working with Objects in the Repository 538
Repository Status Icons 540
Adding a Diagram or Data Dictionary to the Repository 541
Relating models to ER/Studio TS Data Sources 542
Securing Repository Objects 543
Changing the Diagram File Name 544
Retrieving a Repository Diagram, Submodel or Named Release 545
Creating Special Search Strings (Using the Query Parser) 547
Getting the Latest Version of a Diagram 548
Saving Changed Items to the Repository (Checking In) 549
Checking Out Repository Items 554
Discarding Changes Made to a Checked Out Diagram 556

Checking Out Objects When Not Logged In 557
Deleting Diagrams from the Repository 558
Forcing a Checkin 559
Determining Who has an Object Checked Out 560
Viewing the Checkin History of a Repository Object 561
Working with Named Releases of Repository Diagrams 562
Branching and Merging Diagrams 565
Rolling Back a Diagram to a Previous Version 566
Working with Data Dictionaries in the Repository 567
Determining Where an Enterprise Data Dictionary is Used 568
Creating an Enterprise Data Dictionary 569
Associating a Data Dictionary with a Diagram 570
Unbinding an Enterprise Data Dictionary from a Diagram 571
Checking in the Data Dictionary 572
Checking out the Data Dictionary 574
Undoing a Data Dictionary Check Out 575
Redoing a Data Dictionary Redo Check Out 576
Associating the Enterprise Data Dictionary with Repository Diagrams 577
Retrieving an Enterprise Data Dictionary 578
Working with Repository Projects 579
    Create, Delete, Rename, and Define Repository Projects 579
Cancelling a Repository Operation 581
Automating ERStudio DA 582
Why Use Automation Objects? 584
    Objects and Object Models 584
    Object-Oriented Programming 584
Automation Objects Programmer's Guide 586
    Note on Properties Get/Let 586
    Note on Use of "Set" 586
    Note on Dim [what] As [ever] 586
    Related Topics 587
Object Model Hierarchy Diagram 588
Instantiating the Parent Object in an External Application 589
Instantiating the Parent Object in the Macro Editor 591
Accessing Collections of Objects 592
    Iterating Through Collections 592
Assigning Object Appearance and Behavior Using Properties 594
Performing Complex Actions Using Methods 595
Sample Macros Installed with ER Studio Data Architect 596
    Meta Data Management Macros 596
    Model Layout Macros 597
    Modeling Productivity Macros 597
    Physical Modeling Macros 600
    IBM DB2 600
    MS SQL Server 600

Oracle 600
Sybase 601
Running ER Studio Data Architect Macros from Windows 602
Access ER Studio Data Architect from Command Line 602
Access ER Studio Data Architect from Windows 602
Creating Event Handlers 604
Handling Update Events 606
Adding Macros to GUI Modeling Menus 608
Using the SAX Basic Macro Editor 610
Tutorials 613
Introduction 614
Installing Embarcadero ER Studio Data Architect 617
Tutorials: Getting Started 620
Interface Features 620
Tutorials: Logical and Physical Modeling 623
Tutorials: Using Data Dictionary Domains to Populate New Entity 624
Tutorials: Establishing Relationships Between Entities 627
Tutorials: Creating and Working with Submodels 628
Tutorials: Generating Physical Models from a Logical Model 631
Tutorials: Denormalizing the Physical Model 633
Tutorials: Finding out How an Entity Maps to the Physical Model 637
Tutorials: Documenting an Existing Database 639
PRE-REQUISITE: 639
Conclusion 645
Tutorials: Documenting Data Lineage 646
Tutorials: Creating a Data Flow 647
Tutorials: Creating a Data Movement Rule 649
Create a data movement rule 649
Tutorials: Defining External Source and Target Systems 651
Import external source or target data 651
Tutorials: Creating a Data Lineage and Transformation Visualization 653
Tutorials: Diagram Navigation and Aesthetics 656
Navigating the Diagram 656
Diagram Aesthetics 658
Setting the Logical Model Display 659
Setting the Physical Model Display 660
Conclusion 661

Welcome to ER/Studio Data Architect

ER Studio Data Architect is a visual modeling application used for platform-independent logical data architecture analysis and design, in addition to platform-specific physical database design and construction. The powerful, multi-level design environment addresses the everyday needs of database administrators, developers, and data architects who build and maintain large, complex database applications and strive to consolidate, report, and re-use metadata across the enterprise.

ER Studio Data Architect's progressive interface and simplicity has been designed to effectively address the ease-of-use issues which have plagued data modeling and CASE tools for the past decade and more. The application equips you to create, understand, and manage the life-cycle of mission-critical database designs and business metadata within the enterprise.

ER Studio Data Architect is rich and customizable, offering:

Strong logical design capabilities

- The ability to spawn many physical designs from a corporate logical design.
- Bi-directional model comparison and information synchronization.
- Visual-Basic for Applications API for product customization.
- Powerful DDL reverse engineering and generation.
- Metadata import and export capabilities.
- Data lineage documentation.
- Sophisticated XML, HTML, and RTF-based documentation and reporting facilities.

About this Document

The ER/Studio Data Architect User Guide is the primary reference for the product. It provides an overview of the product and gives detailed instructions on common and less-common tasks. In HTML Help format it serves as the online help for ER/Studio DA. It is also distributed in PDF format for easy downloading and printing.

For licensing and installation information, see the ER/Studio Data Architect Quick Start Guide.

For more information and the latest documentation versions, see the Embarcadero website at www.embarcadero.com.

Notice for Developer Edition Users

The Developer Edition of ER Studio Data Architect provides the user with all the standard functionality of ER/Studio except for the following limitations:

- No access to the ER/Studio Repository and ER/Studio Team Server.
- Limited import and export functionality; the Import and Export Metadata Model Bridges are not supported.
- Limited database support. For a list of supported database platforms, see Supported Platforms.

Important Name Change Notice

The following describes recent product names changes in the ER/Studio family

Old Name	New Name	Short Form of New Name
ER/Studio	ER/Studio Data Architect	ER/Studio DA
ER/Studio Repository	ER/Studio Repository	Repository
Embarcadero Connect	ER/Studio Team Server	ER/Studio TS
EA/Studio	ER/Studio Business Architect	ER/Studio BA

Product Benefits by Audience

- *Data Modelers and Data Architects*: ER Studio Data Architect is critical for organizations concerned with eliminating data redundancy, creating an enterprise view of data assets, and assisting development with making informed decisions about how best to reuse elements pre-defined by the enterprise. Its powerful logical (non-database or technology specific) analysis and design environment helps to normalize and create an enterprise view of the objects concerning the data managed by an organization. More importantly, it can communicate this quickly through powerful reporting and metadata exchange mechanisms throughout the enterprise.
- *Database Administrators and Database Developers*: Managing databases can be incredibly difficult without a blueprint or roadmap to understand important object dependencies. ER Studio Data Architect's round-trip engineering capabilities including database reverse-engineering provide database administrators (DBAs) or developers with important *physical* data models in seconds. These models can be used as powerful and efficient change management platforms, allowing users to update a model with the required changes which need to be implemented at the database and automatically generate DBMS-specific, syntactically correct, alteration or database DDL.
- *Business and IT Managers*: ER Studio Data Architect's robust reporting facilities allow critical information about designs to the enterprise to be delivered in seconds. This heavily leveraged and beneficial capability of ER/Studio DA allows users to provide, in literally seconds, clear, easily navigable and safe-to-distribute documentation about a database or enterprise data model to those who need to review it.

See Also

- What's New
- Upgrading ER Studio Data Architect
- Supported Platforms
- Release Notes

Overview

ER Studio Data Architect is an industry-leading data modeling tool for designing and understanding databases, helping companies discover, document, and re-use data assets. With round-trip database support, you can reverse-engineer, analyze, and optimize existing databases. Productivity gains and organizational standards enforcement can be achieved with ER Studio Data Architect's strong collaboration capabilities.

ER Studio Data Architect offers:

- Model-driven design environment
- Complete database life cycle support
- Enterprise model management
- Data lineage documentation
- Enterprise communication capabilities
- Data warehouse and integration support
- Quality database designs

This overview provides a basic introduction to ER Studio Data Architect, the products in the ER Studio Data Architect family, and Embarcadero applications that can further enable data architects.

ER/Studio Family Products

There are several products in the immediate ER/Studio family:

- ER Studio Data Architect: Provides a complete environment for analyzing, designing, creating, and maintaining database applications.
- ER/Studio Repository: (Repository) Productivity gains and enforcement of organizational standards can be achieved with strong collaboration capabilities offered by ER/Studio Repository, a version-control application. ER/Studio DA and ER/Studio Business Architect users connect to the Repository to get a data model revision, check in changes, compare and merge local copies of the data model to a version in the Repository, and run various usage reports. ER/Studio Viewer and ER/Studio Team Server users have read-only access to the Repository.
- ER/Studio Viewer: (Viewer) ER/Studio Viewer provides an interactive environment for accessing ER Studio Data Architect data models, offering the same sophisticated navigational features of ER Studio Data Architect to members of the modeling team that want advanced viewing, navigation and printing functionality but do not require the ability to edit the models.
- ER/Studio Team Server: ER/Studio TS allows easy distribution of data governance repository metadata to the enterprise. ER/Studio TS lets you make use of datasource definitions stored centrally, as well as letting you search, browse, and report on the information contained on ER/Studio TS.
- Universal Data Models: Standard and Industry data model templates for ER Studio Data Architect that reduce the development time of database-related projects by an average of 60 percent, as well as improve the quality of existing and new data models.

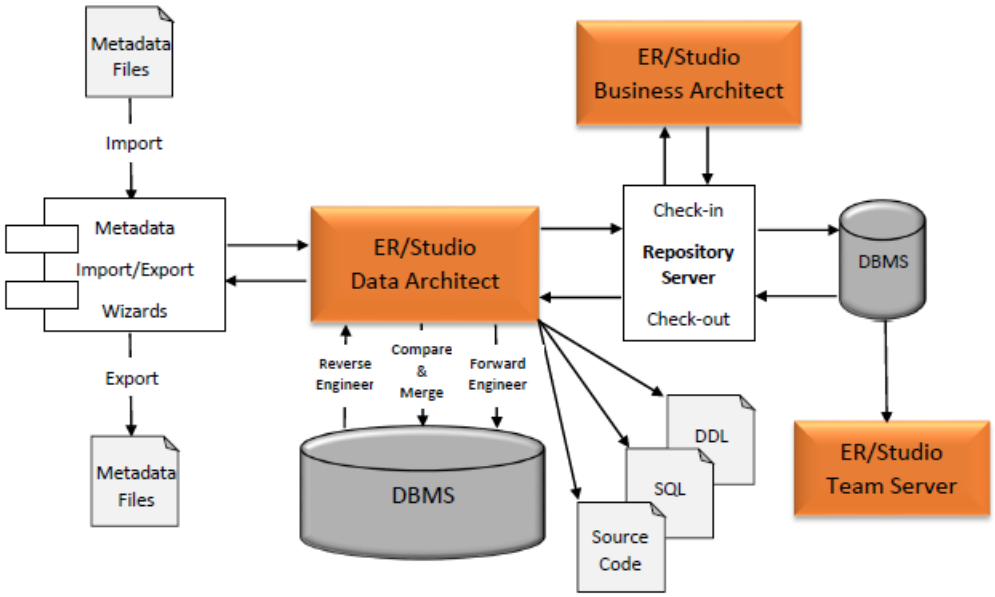
- **MetaWizard**: Integrates metadata across modeling tools, business intelligence, ETL platforms and industry-standard exchange formats (XMI, XML and XSD). This tool enables ER Studio Data Architect to integrate with more than seventy other applications by sharing metadata through an import-export capability.

In addition to the products above, Embarcadero also offers the following complementary applications:

- **ER/Studio Business Architect**: A business modeling tool for creating a graphical representation of your business from the core concepts that describe the business, to the processes that detail how your business operates. The product allows you to easily model your business processes and how those processes use data. The product's Conceptual Modeling provides an intuitive way to outline subject areas and concepts that can then drive the creation of detailed data models. ER/Studio Business Architect is now bundled with ER/Studio Repository.

Application Design

The following diagram shows the functional relationship among ER/Studio components and with the software components of a typical development system.



See Also

- ER/Studio Team Server Wiki
- Customizing Standard Model Features for New Models
- Application Interface
- Configuring and Customizing
- Data Modeling Fundamentals

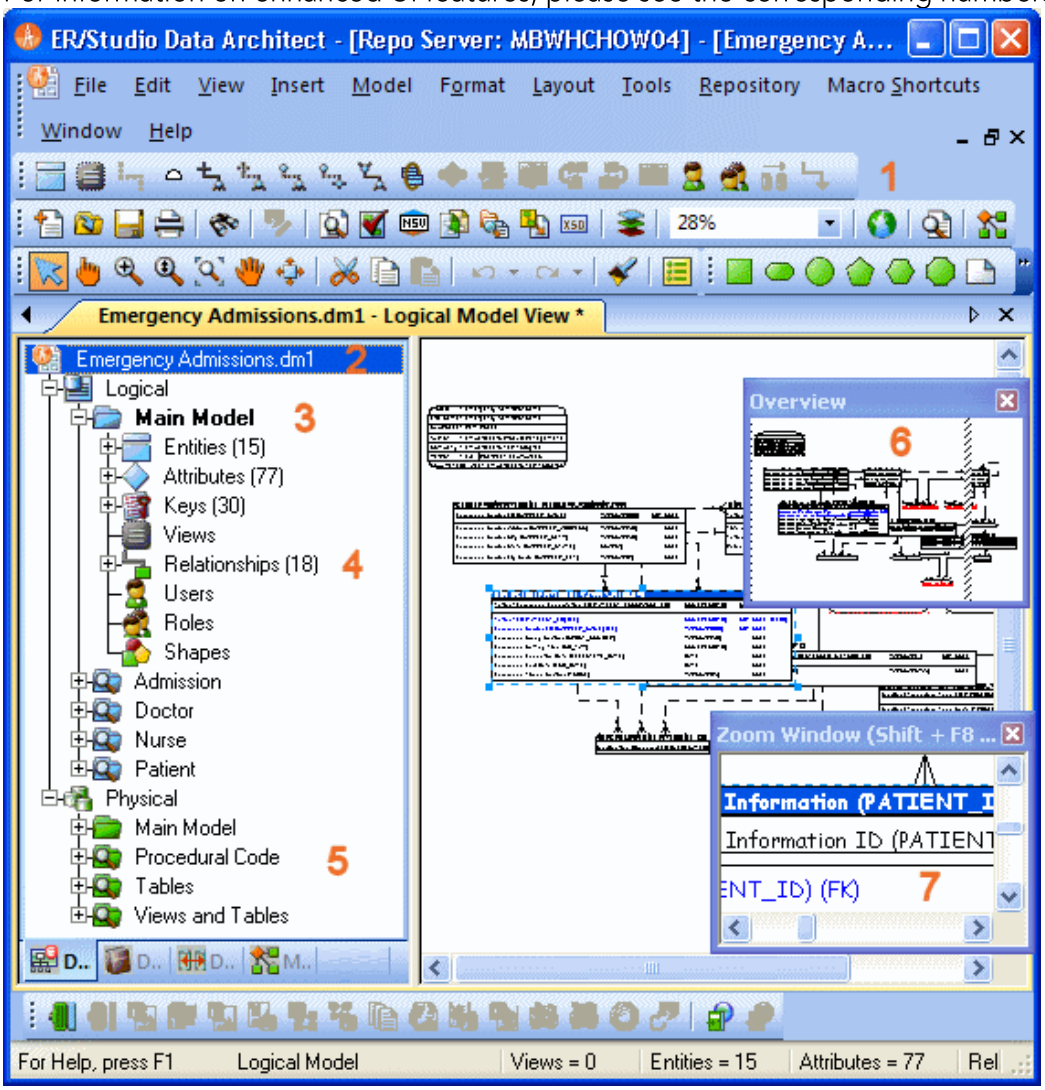
Application Interface

The application interface is divided into several tabbed windows that let you navigate and customize your workspace. The application includes context-sensitive toolbars, menus, intuitive icons, keyboard commands, and other resources to help you work efficiently. ER Studio Data Architect user interface has a standard Windows look and feel.

For a detailed description of the internal data structure of the ER Studio Data Architect application and the Repository, you can look at Metadata models (DM1 files), usually found in the following directories:

- *Windows XP*: C:\Documents and Settings\All Users\Application Data\Embarcadero\ERStudioDA_X.X\Sample Models
- *Windows 8, Windows 7, and Windows Vista*: C:\ProgramData\Embarcadero\ERStudioDA_X.X\Sample Models

For information on enhanced UI features, please see the corresponding numbers below:



Interface Features

ER Studio Data Architect has an enhanced user interface that is similar to Microsoft Windows with intuitive icons.

1. You can dock toolbars anywhere in the ER Studio Data Architect user interface.
2. The name of the active model, with or without the full path to the file, can be displayed on the title bar of the main application window. This is an option you can set in Tools > Options > Application > Application Defaults.
3. ER Studio Data Architect displays the active model in bold type, so that you can easily see which model or submodel you are working on.
4. The *Data Model* tab of the Data Model Explorer displays indexes, relationships, and attributes as separate nodes.
5. The *Data Model* tab of the Data Model Explorer displays schema objects like packages, procedures, functions, materialized views, auxiliary tables, and triggers as separate nodes.
6. The *Overview Window* lets you navigate large Data Models.
7. The *Zoom Window* helps you focus on the details of a specific area of a large, reduced diagram.

The Data Model Explorer is comprised of four tabs that offer easy access to important functionality and also includes an Explorer tree that lets you efficiently navigate the objects of your data models. The ER Studio Data Architect interface to the Data Model is divided into two main areas. On the left is the Data Model Explorer and on the right is the Data Model Window. This convention is carried out on other tabs of the user interface, such as the Data Lineage tab, where you'll find an explorer on the left and a graphical representation on the right. The Data Model Window also provides a complete workspace for creating physical and logical data models and smaller, movable windows to help you navigate large data models.
This section also covers the following items:

- Data Model Explorer
- Data Model Window
- Data Lineage Window
- Pop-up Windows
- Zoom Window
- Overview Window
- Menus
- Toolbars
- Keyboard Commands
- Status Bar

See Also

- Overview
- Automating ERStudio DA

Data Model Explorer

The Data Model Explorer helps you navigate data models and their objects, reuse design elements, and create new objects. The Data Model Explorer has four tabs that offer easy access to important functionality, enabling you to efficiently manage your Data Models.

- Data Model: Expand and collapse the Data Model Explorer nodes as needed. Double-click a node to start an Object Editor.
- Data Dictionary: Manage data dictionary objects, including attachments, defaults, rules, reference values, user data types, domains, reusable procedural logic, reusable procedures, and libraries.
- Data Lineage: Document how data is moved and transformed. You start with a data source and move through an intermediate schema or target using as many transformations as you need.
- Macros: Manage sample, system, and user macros. Create, rename and delete folders as well as add a macro and refresh the macro list.

Data Model

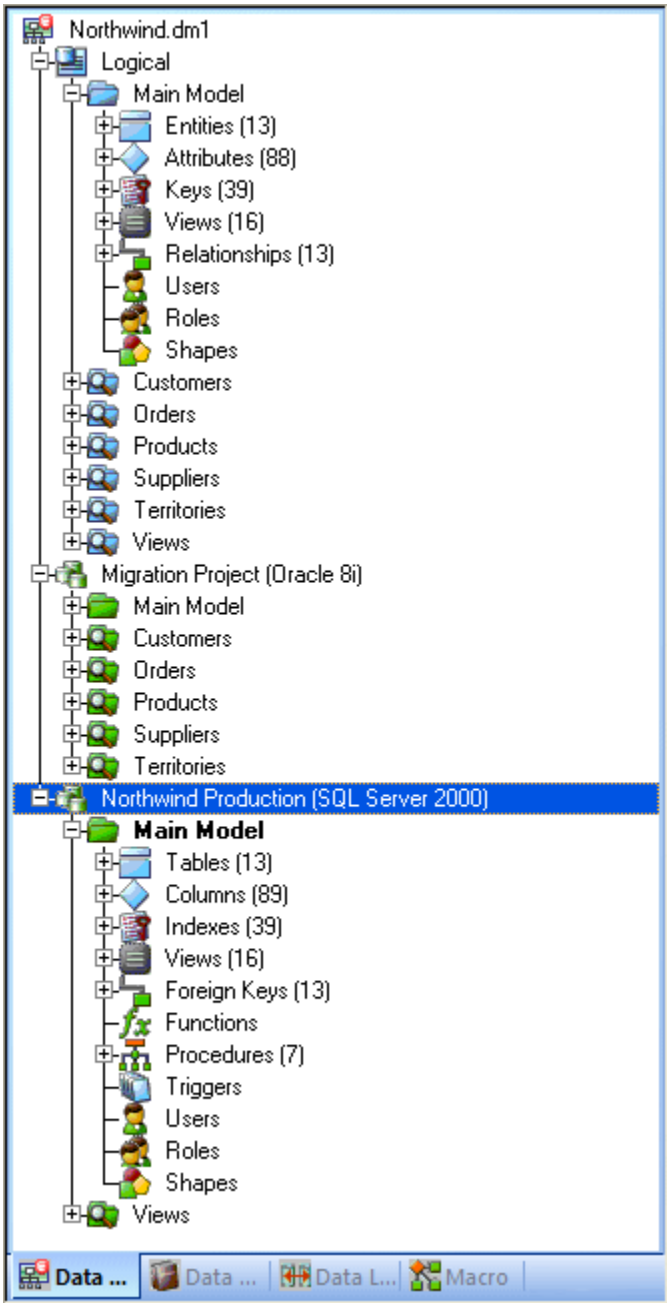
The Data Model tab displays logical elements such as entities and relationships, and physical schema objects such as packages and triggers as separate nodes. Shortcut menus are available on the Data Model tab when you right-click an element. Use the Data Model tab to:

- Create an object
- Edit an object
- Delete an object
- Rename an object

Change how the data model tree displays by choosing *Tools > Options > Application settings*.

Use the Data Model tab also to manage objects that are contained within objects, such as columns, which are contained within tables.

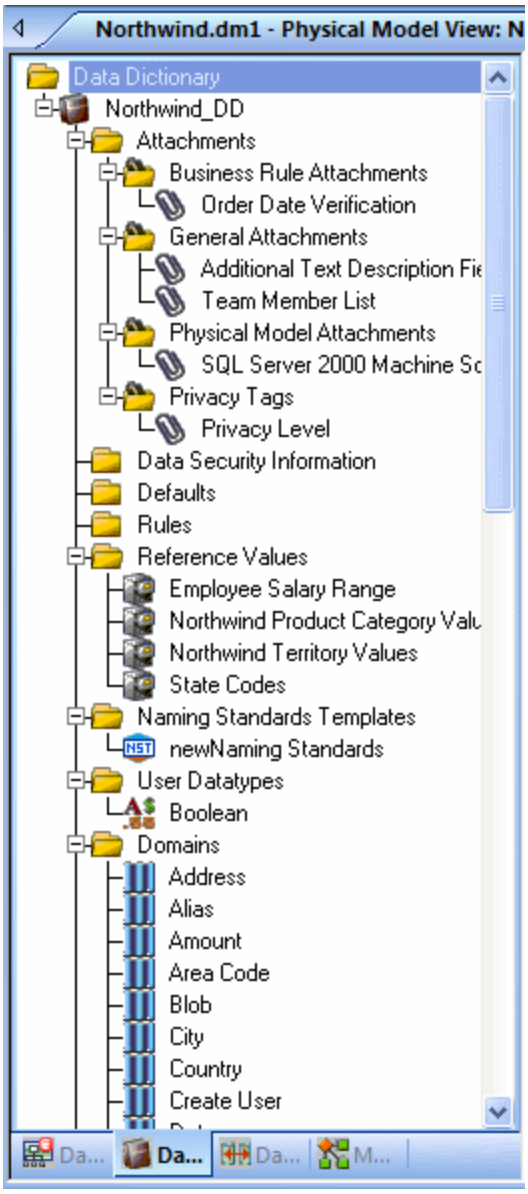
- ER/Studio DA displays the active model in bold type, so you can easily recognize which model or submodel on which you are working.



Data Dictionary

Use the Data Dictionary tab of the Data Model Explorer to manage data dictionary objects, including attachments, defaults, rules, reference values, user data types, domains, reusable procedural logic, reusable procedures, and libraries. The shortcut menu available when you right-click an object lets you add, edit, or delete these Data Dictionary objects.

The Reusable Procedural Logic folder lets you organize several procedural logic items such as reusable triggers, reusable procedures, and libraries.

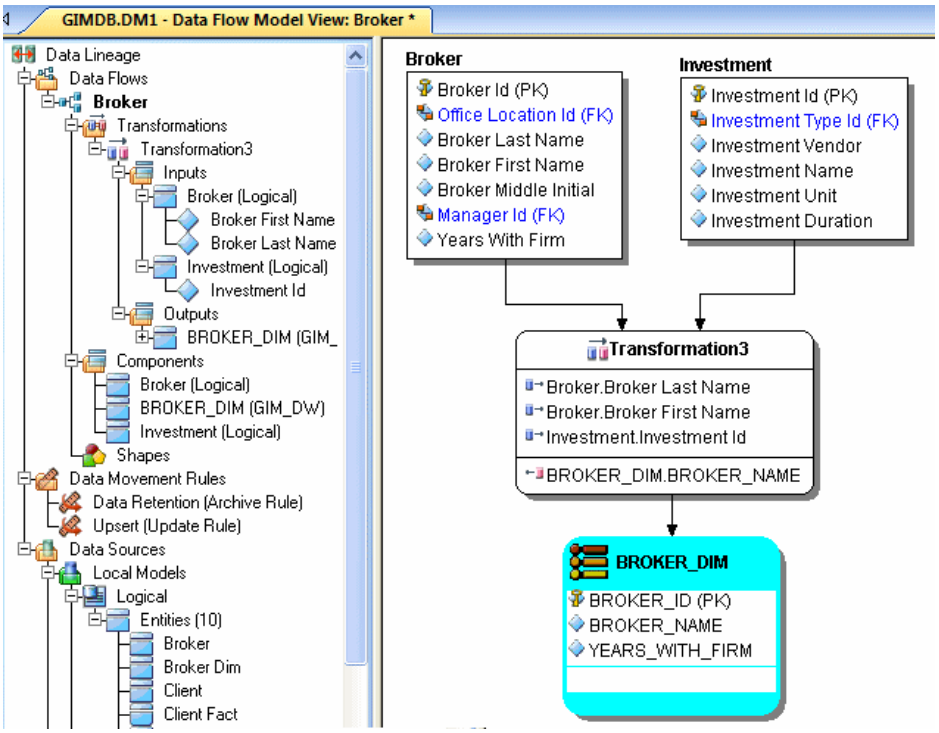


Data Lineage

The Data Lineage tab provides a visual, drag-and-drop interface to enable you to document how data moves between the target and source systems along with why and when the data is moved.

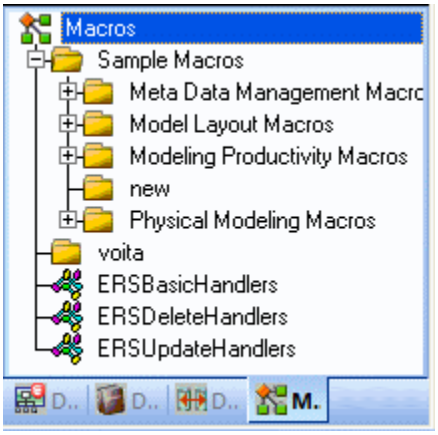
Data lineage documents the Extraction, Transformation, and Load (ETL) of data movement and the relevant computations required when moving data between disparate systems. You can create data movement rules that can dictate, for example, when the data should be archived or a specific value range the data must fall within in order to be moved. You may also need to massage the data to, for example, compute the total sales for a particular product before moving the data to the target. This is sometimes referred to as *source and target* mapping. For example, your organization's data warehouse may have data fed in from multiple sources such as CRM, Payroll, General Ledger, Accounting, or Product/Inventory system. The data warehouse will likely need on-line data marts receiving data from one of those systems to produce reports

for Sales Executives, HR Executives or marketing teams who produce performance reports about of various aspects of your business



Macros

The Macro tab of the Data Model Explorer lets you add, edit, rename, delete, and run macros. You can also create folders in which to store macros for easier reference. ER Studio Data Architect installs several sample macros in the Sample Macros folder. These macros are described in more detail in Using the SAX Basic Macro Editor of this documentation.



See Also

- Application Interface
- Data Model Window

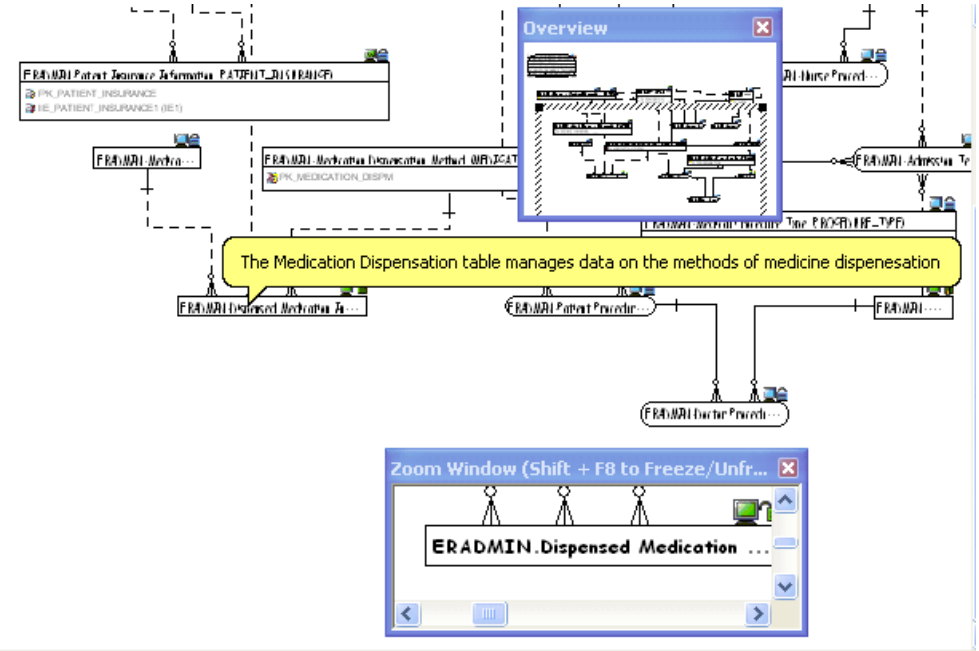
Data Model Window

The Data Model window displays the logical and physical data model. You can access functionality through the menus, shortcut menus, and tool bar buttons. The Data Model window lets you move diagram objects, copy and paste them in a new location, resize the diagram objects, and change their colors.

The Data Model Window also displays physical schema objects like Procedures, Functions, Packages, Auxiliary Tables, Tablespaces, Materialized Views, and Synonyms. This presents a visual representation of object dependencies for impact analysis.

If you right-click a relationship, ER Studio Data Architect opens a shortcut menu that lets you modify the relationship between entities or tables.

In the Data Model Window, ER/Studio DA displays useful information about each object in your Data Model in the Pop-up Windows, Zoom Window, and Overview Window.

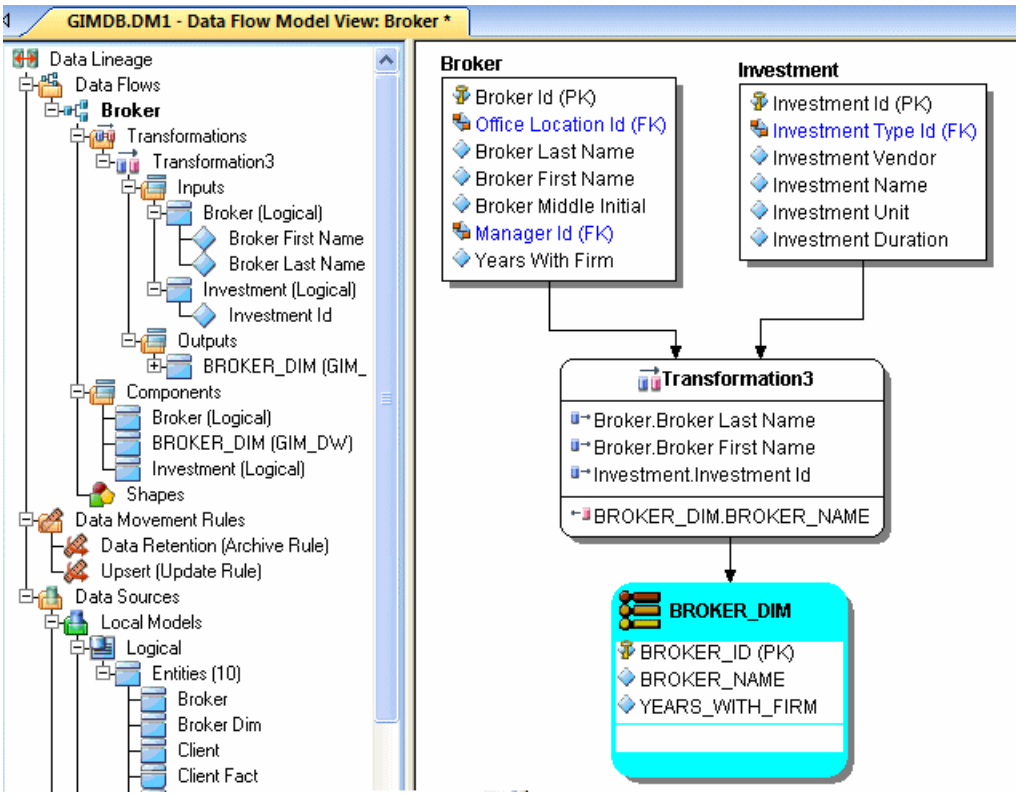


See Also

- Overview
- Data Model Explorer

Data Lineage Window

The Data Lineage window provides a view of the selected data flow, including the transformations and their inputs and outputs. You can use the Data Lineage window in much the same way as you do in the Data Model window. You can arrange the inputs and outputs by dragging them. You can add shapes and colors to enhance the readability of your data lineage diagram. For more information, see Documenting Data Extraction, Transformation, and Load.

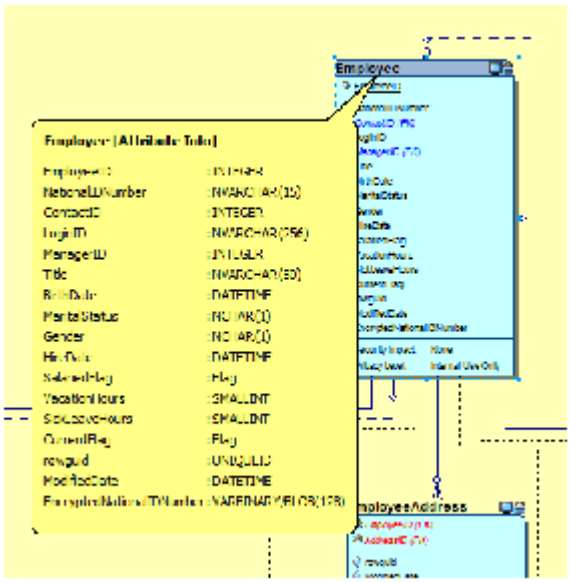


See Also

- Overview
- Application Interface
- Automating ERStudio DA

Pop-up Windows

Pop-up windows can display brief definitions or the DDL of each object in your Data Model. When you hover the mouse pointer over any object briefly, ER Studio Data Architect displays a pop-up window. The popup in the following example above displays the object definition.



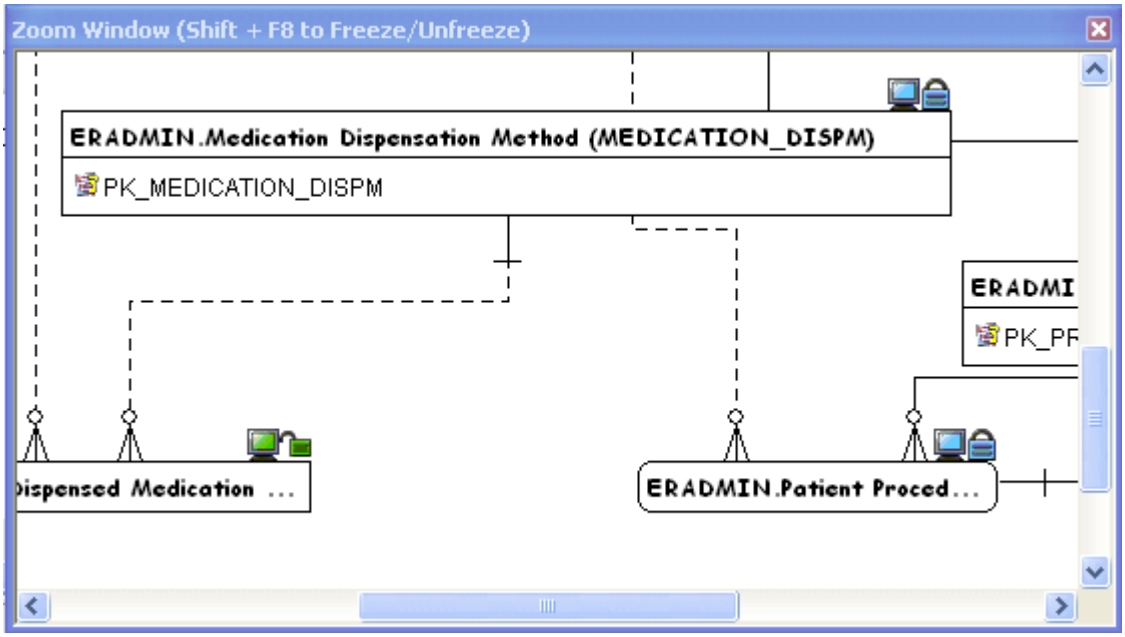
To display the entity definition as in the screenshot above, choose View > Cursor Popup Help Options > Entity Help and then select *Attribute Definition*.

See Also

- Overview
- Introduction

Zoom Window

The Zoom window helps you focus on the details of a specific area of a large, reduced diagram. This feature is only available in the Data Model Window. You can open and close the Zoom Window as needed using the View menu or by pressing F8. You can also move the window by dragging it by its title bar.

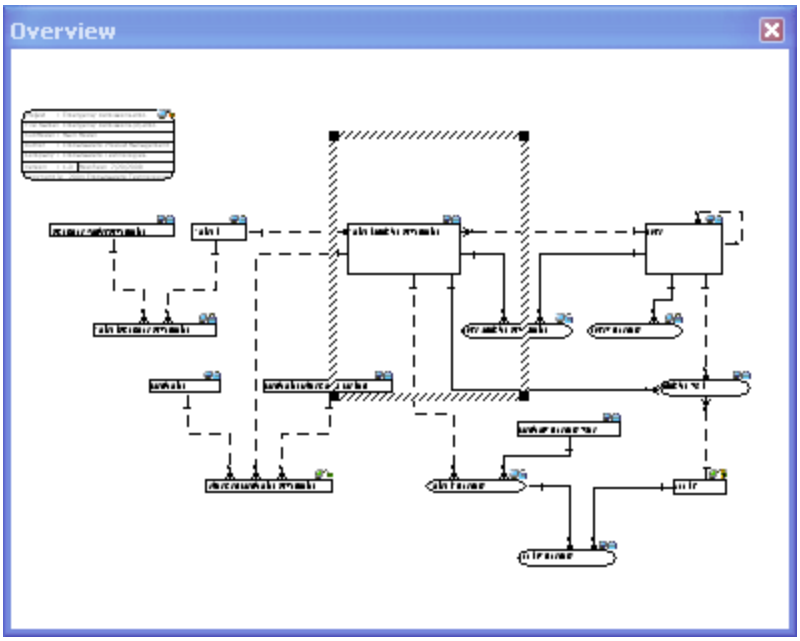


See Also

- Overview
- Overview Window

Overview Window

The Overview window lets you navigate large Data Models. The Overview window provides a thumbnail view of the entire model. You can open and close the Overview window as needed using the View menu or by pressing F9. When you zoom in or out, the Overview window places a border around the portion of the model displayed in the Data Model Window.



See Also

- Overview
- Zoom Window

Menus

You can access the entire ER Studio Data Architect feature set from the application menus. You can also access selected features from shortcut menus. ER Studio Data Architect application menus are across the top of the user interface and shortcut menus are available when you right-click an object or diagram. All menus are context-sensitive and change depending on the content of the Data Model Window.

Functionality available in application menus or shortcut menus is also available on Toolbars.

The following discusses the menus available:

- Application Menus
- Accessing Shortcut Menus

Application Menus

ER Studio Data Architect application menus provide access to all functionality. The menus are context-sensitive and change depending on the content of the Data Model Window. The application menu differs for logical and physical models.

Accessing Shortcut Menus

ER Studio Data Architect shortcut menus offer an easy method of accessing object functionality. Shortcut menus offer the same functionality you can access from the Toolbars or **Menus**. To access the shortcut menu, right-click an object or model.

See Also

- Overview
- Toolbars
- Data Model Window

Toolbars

ER Studio Data Architect toolbars are context-sensitive and change to reflect the element of the application you are using. Toolbar buttons offer quick access to common features of ER/Studio DA. All functionality accessible from toolbar buttons is also accessible from Menus and Accessing Shortcut Menus. Toolbars change depending on if you are working with logical or physical models.

ER Studio Data Architect lets you move toolbars anywhere on the workspace. You can also dock toolbars anywhere on the perimeters of the workspace. You can also specify which toolbars you want displayed on the workspace.

Moving Toolbars

You can dock and undock toolbars, moving them to any place with the application window.

Undock a Toolbar

1. On a docked toolbar, click and hold the toolbar handle .
2. Drag the toolbar to a new location.

Dock a Toolbar

On an undocked toolbar, double-click the title bar of the toolbar.

Displaying Toolbars

Click View and then click to select or deselect the toolbars you want to display or hide.

See Also

- Overview
- Menus

Keyboard Commands

Keyboard commands, function keys, and shortcuts offer alternative ways to access ER Studio Data Architect functionality without using the mouse. You can use keyboard commands to navigate diagrams and perform basic functions.

Basic Functions

- *PAGE UP*: Scrolls up the Data Model Window.
- *PAGE DOWN*: Scrolls down the Data Model Window.
- *UP ARROW*: Scrolls up one line.
- *DOWN ARROW*: Scrolls down one line.
- *RIGHT ARROW*: Scrolls over one position to the right.
- *LEFT ARROW*: Scrolls over one position to the left.
- *MINUS SIGN*: Decreases zoom level by one step.
- *SHIFT+F8*: Lets you freeze the Zoom window so that you can pan around the diagram and still maintain focus on a specific diagram object. The same command unfreezes the Zoom window.

Function Keys

The function keys let you access certain features or dialog boxes.

- *F1*: Opens the online Help.
- *F4*: Opens the Find Entity/View dialog.
- *F5*: Redraws the diagram.
- *F7*: Activates or deactivate shadowing.
- *F8*: Opens or closes a zoom window.

Shortcuts

Shortcuts let you access important functionality through key combinations.

- *CTRL+A*: Selects all entities in the Data Model Window.
- *CTRL+Shift+A*: Selects all relationship lines in the Data Model Window. One relationship line needs to be selected first.
- *CTRL+H*: Straightens the selected relationship lines horizontally. The lines will straighten based on the docking point against the parent entity.

- *CTRL+L*: Straightens the selected relationship lines vertically. The lines will straighten based on the docking point against the parent entity.

Hot Keys

You can use the following hot keys to navigate through your models in the Data Model Explorer:

- *Page Up*: Scrolls up one screen size.
- *Page Down*: Scrolls down one screen size.
- *CTRL+Home*: Scrolls to the upper left hand page of the model.
- *Up Arrow*: Scrolls up one line.
- *Down Arrow*: Scrolls down one line.

Full Undo Redo

The Undo and Redo commands restore your model to the last stage of edits. Find them under the Edit menu or use the CTRL+Z and CTRL+Y hot keys.

See Also

- Overview
- Automating ERStudio DA

Status Bar

ER Studio Data Architect provides statistics pertaining to your logical and physical model in the status bar at the bottom of the application. The table below describes the statistics available on the Status bar:

Diagram Mode	Statistic	Definition
Logical	Views	Total number of views in the current model or submodel
	Entities	Total number of entities in the current model or submodel
	Attributes	Total number of attributes in the current model or submodel
	Relationships	Total number of relationships established in the current model or submodel
Physical	*Tables*	*Total number of tables in the current model or submodel*
	Views	*Total number of views in the current model or submodel*
	Columns	*Total number of columns in the current model or submodel*
	Foreign Keys	*Total number of foreign keys in the current model or submodel*

See Also

- Overview
- Menus

Configuring and Customizing

There are many options available to customize how ER Studio Data Architect behaves and what it displays in the Data Model Window. Some options, such as the Diagram and Object Display Options, and Model Options are saved in the.dm1 file and affect how the model in that file displays or behaves immediately, while the Tools Options and Cursor Popup Help Options affect how the application and any new models created will display or behave.

- Customizing the Display of Diagrams and Objects
- Defining Model Options for the Selected Model
- Customizing Standard Model Features for New Models
- Specifying the Location of Shared Data
- Changing Cursor Popup Help Options

See Also

- Overview
- Application Interface

Customizing the Display of Diagrams and Objects

Diagram and Object Display options affect how the diagram objects appear only for the selected model or submodel. You can choose different display options for each physical and logical model and submodel. By accessing the Diagram and Object Display Options from the Data Lineage Tab, you can also customize the appearance of transformation objects. The diagram and object display options you choose are stored in the.dm1 file along with your diagram and apply only to the current diagram. Diagram display options include setting page boundaries and snapping to grid. Object-specific display options include specifying the information displayed about each object on the diagram. You can specify the content displayed for entities and views. You can also customize display settings for relationships and schema objects.

You can change the entity/table display level to show attributes/columns, primary keys, all keys, tables, definitions or notes. You can also set display options for certain objects to display object datatype, domain, null options, alternate keys, and owner.

For views, you can change the display level to show columns, primary keys, all keys, definitions, and notes. You can also set display options for certain objects to display object owner and parent name.

By default, ER Studio Data Architect automatically sizes the entities and views to display all the content selected. However, you can manually resize any entity or view by right-clicking the object in the Data Model Window, grabbing a handle, and then dragging to resize the object.


Note: You can make global changes to the display of any new diagrams and objects by choosing Tools > Options > Display and making your choices there. For more information, see Defining Model Options for the Selected Model.

1. To change the display of a specific model or submodel, *from the Data Model Explorer on the Data Model Explorer, select the model or submodel.*
   To change the display of objects on the Data Lineage tab, click the *Data Lineage* tab.
2. Choose View > Diagram and Object Display Options.
3. Complete the tabs of the *Diagram and Object Display Options* dialog and then click *OK* to implement the changes.

The following describe options that require additional explanation:

Diagram tab | Entity/Table tab | Relationship tab
View tab | Schema Objects tab | Transformation tab
Drawing Shapes tab | Security Objects tab | Apply To tab

Diagram tab

- *Show Page Boundaries*: If selected, displays the page boundaries according to the print options, such as the printer selected, and page size, scale, and orientation chosen in the Print and Print Setup dialogs.
- *Auto-center Diagram over Pages*: If Show Page Boundaries has been selected, this option is available. Selecting this option arranges the diagram elements so the diagram always appears centered across the pages required to display or print the diagram.
- *Display and Enable Snap to Grid*: If selected, displays grid lines in the diagram background, and align diagram object along their nearest grid line. ER Studio Data Architect lets you set a grid to the background of your diagram. You can use the grid to align diagram objects and to create a regular or uniform appearance. If you set a grid in the background of your diagram, you can set ER Studio Data Architect to snap diagram objects to the nearest grid line.
- *Grid Step Size*: Specifies the horizontal and vertical size of each grid cell.
- *Grid Steps Per Line*: Specifies a horizontal grid spacing per line.
- *Units*: Specifies the unit of measurement for grid cells. You can select inches and centimeters.

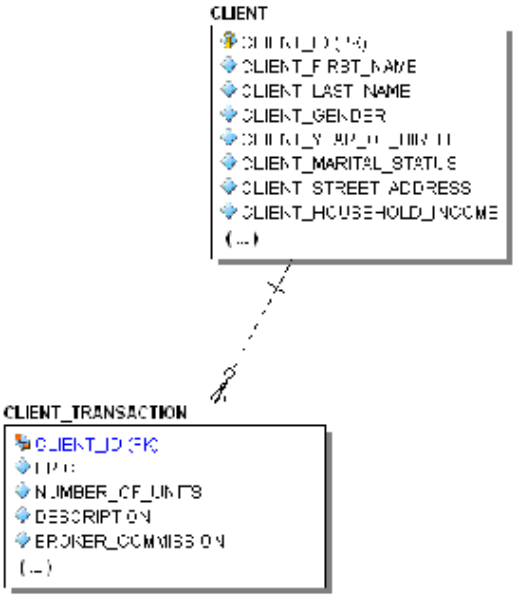
Entity/Table tab

Display Level: The display level options selected dictate which options will be available in the Available options area.

WARNING: If the Display Level in the Diagram and Object Display Options editor does not match the Attribute Order designation in the Model Options Model > Model Options > Attribute Order, drag and drop errors and on-screen editing errors can occur. If the true attribute sequence, as designated in the Model Options, and the displayed sequence, as designated in the Diagram and Object Display Options, are not synchronized, then ER Studio Data Architect cannot make the appropriate changes to the entity.

- *Logical Attribute Order*: When selected the Primary Key of the table displays at the top. The other columns display in the order designated in the Entity Editor for the corresponding entity.
- *Physical Attribute Ordering*: When selected the attributes are ordered by their Column Sequence ID. The sequence ID of a column can be changed by moving it up or down in the list of columns on the Columns tab of the Table Editor.

- *Definition*: If selected, displays the entity or table definition as defined on the Definition tab of the Entity Editor or Table Editor.
- *Note*: If selected, displays the entity or table note as defined on the Notes tab of the Entity Editor or Table Editor.
- *Display Table Names*: If selected, displays entity and table names.
- *Display Column Names*: If selected, displays the attribute and column names.
- *Hidden Attributes Marker*: This option become available when attributes or keys of an entity in a submodel has been hidden. When selected, an ellipsis displays in submodel entities where all attributes are not currently displayed. Attributes or keys of an entity can be hidden by deselecting the object on the Attributes tab of the Submodel Editor.



Relationship tab

- *Relationship Style*: The style chosen here applies to any new relationships created in the model or submodel. Override the relationship style by right-clicking the relationship and choosing one of the format options available.

View tab

- *Definition*: If selected, displays the view definition as defined on the Definition tab of the View Editor.
- *Note*: If selected, displays the view note as defined on the Notes tab of the View Editor.
- *Name Compartment*: If selected, displays the view name within the view object; otherwise, the view name displays just above the object.
- *Wrap Name*: If selected, wraps the view name when the view width is not large enough to accommodate the entire name.

Schema Objects tab

For physical models only, controls the display of schema objects such as packages, procedures, functions, materialized views, auxiliary tables, and triggers.

Transformation tab


Note: Available only if the Data Lineage tab is active when View > Diagram and Object Display Options is chosen.

Transformation display options control the information displayed for transformation objects in a data flow on the Data Lineage window.

- *Transformation Type*: If selected, displays the transformation type as entered or selected in the Transformation Editor. For more information, see Creating and Editing Data Lineage Data Flows.
- *Input and Output Columns*: If selected, displays the names of the input and output columns on the transformation object.
- *Code Definition*: If selected, displays the transformation code entered in the Code section of the Definition tab for the Transformation Editor.
- *Business Definition*: If selected, displays the business description entered in the Business section of the Definition tab for the Transformation Editor.

Drawing Shapes tab

Verb and inverse verb phrases are defined on the Phrases tab of the Relationship Editor. If you choose to display these phrases, in the Data Model Window you can click and drag them for optimal positioning.

Security Objects tab

Security objects, such as users and roles are not automatically displayed in data models that have been reverse-engineered; in this case, you must enable their display here.

Apply To tab

You can selectively apply the object display options set on the other tabs to individual elements of your model.

See Also

- Overview
- Application Interface
- Creating and Working With Data Models

Defining Model Options for the Selected Model

Model Options for the selected model define naming rules, model notation, default datatype, datatype mappings, rolename prefixes, and the attributes order. Use Model Options also to synchronize entity and table names, and attribute and column names.

The preferences set here are maintained in the diagram.*dm1* file.

To set Model Options

1. Right-click the target model and then click *Model Options*.
   Depending on the target model selected, the Logical Model Options or Physical Model Options dialog displays.
2. Complete the changes on the *General Options* and *Name Handling* tabs and then click *OK*.


Note: The options that appear depend on whether the selected model is a logical or a physical model type.

The following describe options that require additional explanation.

- General Options tab
- Name Handling Options tab

General Options tab

- *Max Table Name Length*: Read-only, platform dependent value.
- *FK Column Definition/Note Synchronization*:
- *None*: The Note and Definition text from the parent PK columns is propagated when the relationship is created or a new column is added to the primary key. If there are any changes to the parent PK columns then the changes will not be synchronized with the child columns.
- *Partial*: The Note and Definition text from the parent PK column is propagated when the relationship is created or a new column is added to the primary key and the text is kept synchronized with the parent PK columns as long as the text in the parent equals the text in the child. If the text in the foreign key attributes is changed in the child, the tie between the parent and child will be broken.
- *PK Indexes*: Select Manual Sequence to manually change the sequencing of primary key index columns using the Index Editor. Otherwise, the sequence of the index cannot be manually changed.
- *Attribute Order*: Select Logical Order to position all primary key columns before non-key columns. Select Physical Order to position the keys to mimic the order in the physical

model, meaning that primary keys can appear before or after non-key columns. Select Logical Order to prevent primary keys from being reordered in the Entity Editor. If Logical Order is selected after selecting Physical Order, the primary keys in each entity are re-sequenced so they appear first in the attributes list.

Note: Set the default column sequence for new models on the Options Editor > Application tab.

WARNING: If the Display Level in the Diagram and Object Display Options editor does not match the Attribute Order designation in the Model Options, drag and drop errors and on-screen editing errors can occur. If the true attribute sequence, as designated in the Model Options and the displayed sequence, as designated in the Diagram and Object Display Options, are not synchronized, then ER Studio Data Architect cannot make the appropriate changes to the entity.

- *View Parser*: Determines which parser is used to validate the SQL produced when creating a View. This allows you to maintain and parse platform-specific syntax in the logical model. If you use the same parser for both the logical model and the targeted platform, then after generating a physical model you won't need to rewrite the View SQL for it.
- *Auto Rolename Prefix*: Specify a string to add to the beginning of all rolenames.
- *Datatype Mapping*: If you have customized a datatype mapping for the selected database platform, from the Datatype Mapping list you can choose an alternate mapping or the system mapping, which is the default.

Note: Custom datatype mappings are created using the Datatype Mapping Editor Tools > Datatype Mapping Editor.

Name Handling Options tab

Sets the synchronization level between entity and table names.

- *Complete Synchronization*: Matches table and column names with their associated entity and column names. Both names are always synchronized.
- *Partial Synchronization*: Automatically synchronizes table and column names with their associated entity and attribute names unless you edit the table name, which breaks the synchronization so that the names can be different. If names are the same, editing the entity name will update the table name.
- *No Synchronization*: Prompts you to manually specify both sets of names. Entity and table names are separate. Also, the rolenames link is broken so that in a foreign key relationship child and parent attributes can have different names.
- *Enable Logical Name Editing*: Enables corresponding objects to be directly edited from the table or entity editor. For example, attribute names can be edited from the Table Editor and column names can be edited from the Entity Editor.

See Also

- Configuring and Customizing
- Data Modeling Fundamentals

Customizing Standard Model Features for New Models

Use the Tools Options to customize the work environment for all new models created, including those that are reverse-engineered, although some options apply immediately to affect the display of the current model. See also, Customizing the Display of Diagrams and Objects and Defining Model Options for the Selected Model.

Define the Model Options for All New Models

1. Choose Tools > Options.
2. Complete your changes and then click *OK*.

Click a link below for information on options that require additional explanation:

Application tab	Logical tab	Physical tab
Name Handling tab	Display tab	Directories tab
Tools tab	Diagram tab	View tab
Schema Objects tab	Object Types tab	Object Names tab
Automation Options tab	Data Dictionary tab	Comments tab
Where Used tab	ERX File Import tab	Undo tab
Repository Options tab		

Application tab

- *Show Status Bar*: Displays the status bar at the bottom of the application. For a selected physical model, the status bar displays the name of the database platform and the total number of Views, Tables, Columns, and Foreign Keys. For a logical model, the status bar display the model type (Logical) and the total number of Views, Entities, Attributes, and Relationships.
- *Unification Options:* Unification is the act of combining columns with the same name to create one single column. It normally occurs when you propagate a foreign key into a table that has an existing column of the same name.
- *Compare Names Only*: Unifies overlapping foreign keys that have the same name, even if they originate from different entities.
- *Compare Names and Datatypes*: Unifies overlapping foreign keys that have the same name and that also originated from the same entity.
- *Compare Originating Parents*: Unifies overlapping foreign keys that originate from the same originating parent entity.

- *Prompt to Resolve FK Columns*: Launches the Duplicate Attribute Editor, where you must choose how to resolve duplicate foreign keys names by creating a rolename to resolve the name conflict.
- *Submodel Drag and Drop*: To prevent you from inadvertently moving a submodel onto another submodel, you can choose to have a warning presented when this is attempted, or disable this type of move altogether.

Logical tab

- *Case Shift*: Choose whether to retain case as typed into the editors or convert the letters as you type in names to upper or lower case.
- *FK Column Definition/Note Synchronization*:
- *None*: The Note and Definition text from the parent PK columns will be propagated when the relationship is created or a new column is added to the primary key. If there are any changes to the parent PK columns the changes will not be synchronized with the child columns.
- *Partial*: The Note and Definition text from the parent PK column will be propagated when the relationship is created or a new column is added to the primary key, and the text will be kept synchronized with the parent PK columns as long as the text in the parent equals the text in the child. If the text in the foreign key attribute is changed aside from the parents, the tie between the two will be broken.
- *PK Indexes*: Selecting Manual Sequence enables manual changes to the sequence of primary key index columns using the Index Editor. Otherwise, you cannot manually change the sequencing of the index.

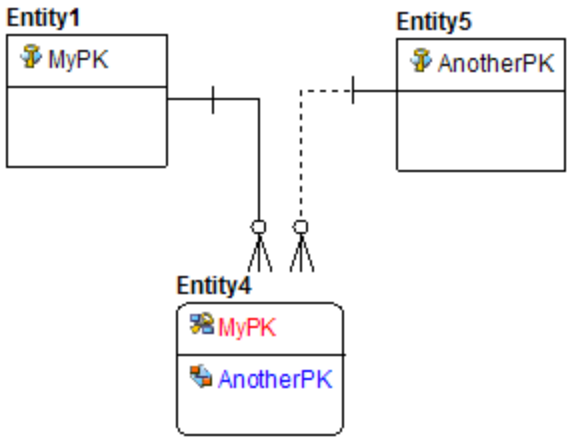
Physical tab

- *Max Table Name Length*: This field is read-only. Its value conforms to the rules of the selected database platform.
- *Auto PK Options*: If you choose Yes, when automatically creating Primary Key indexes, ER Studio Data Architect automatically clusters the indexes, re-ordering the data blocks in the same order as the index. This can increase access speed.
- *FK Column Definition/Note Synchronization*:
- *None*: The Note and Definition text from the parent PK columns will be propagated when the relationship is created or a new column is added to the primary key. If there are any changes to the parent PK columns, the changes will not be synchronized to the child columns.
- *Partial*: The Note and Definition text from the parent PK column will be propagated when the relationship is created or a new column is added to the primary key, and the text will be kept synchronized with the parent PK columns as long as the text in the parent equals the text in the child. If the text in the foreign key attributes is changed aside from the parents, the tie between the two will be broken.

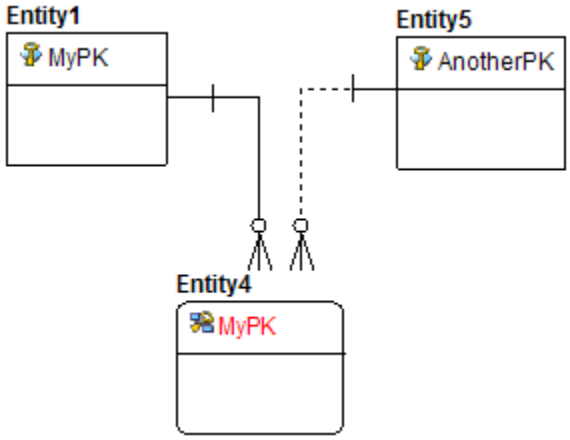
Name Handling tab

These settings determine how entity names are synchronized with table names, how column names are synchronized with attribute names, and the reverse, between Logical and Physical Models. This is useful in the event that you want changes in entity names to reflect in table names, but do not want synchronization in the other direction. For example, you might fine-tune names during the deployment of a Physical Model but do not want name changes automatically reflected back to the Logical Model.

- *Allow Unification When Editing Names*: When set, allows you to change the role names of foreign attributes or columns to the same names as other attributes or columns in the same Entity or Table. You can also specify unification between logical and physical names in the Entity Editor > Attributes tab on a selective basis. Duplicate names will be unified when your edits are committed. This option affects names in the Entity Editor and Table Editor, on-screen name editing, and names in the Explorer tree. For example, assume you edit on-screen the FK in Entity4 below from *AnotherPK* to *MyPK*.



Here is the unified result:



To avoid object name conflicts, you can enclose all names in brackets or quotations or enclose only names that conflict with SQL keywords. If the database platform supports object names enclosed in brackets or quotes, the options to enclose object names are available in the General Options area of the DDL tab Options and the DDL Generation Wizard. If you select

Enclose Names That Conflict with Keywords, then any column names that contain SQL keywords will be automatically enclosed in brackets.

Display tab

- *Page Boundaries*: Displays the page boundaries, enabling you to determine where objects should be placed in the diagram for printing purposes.
- *Turn on Grid And Snap*: Displays a grid on the model background and places new model objects next to grid lines. Lets you evenly space the model objects.
- *Cardinality*: Displays the cardinality of the relationships between entities or tables.
- *Verb Phrases*: Displays defined verb phrases next to the relationship lines in the diagram.
- *Rolename Notation*: Displays full IDEF1X rolename notation.
- *Display Mode*: Lets you select a display mode and then select Display Options. The options are:
- *Attribute (Model Order)*: Displays the entities and all their attributes in the default order according to the model type. For example, in the logical model, the attributes appear as they normally would in a logical model, primary keys first.
- *Attribute (Logical Order)*: Displays the entities and all their attributes in Logical Order, regardless of whether you are viewing the logical or physical model. When the attributes are in Logical Order, ER Studio Data Architect sequences the attributes so that the primary keys are always on top.
- *Attribute (Physical Order)*: Displays the entities and all their attributes in Physical Order, regardless of whether you are viewing the logical or physical model. When the attributes are in Physical Order, ER Studio Data Architect sequences the attributes to reflect their order in the physical model, without regard to whether the primary keys are on top or not.
- *Image Export Settings*: Sets image file options for HTML Reports. There are other options such as file format and dpi that you can set in the *File > Export* dialog.
- *Export model and object background color*: If not selected, the diagram is rendered in black and white.
- *Use legacy image export*: ER Studio Data Architect versions 9.0 and above incorporate improved image creation. Since the new image creation method does not use video memory, it can more efficiently create large images that are memory intensive. In some rare cases, memory warnings that did not occur in versions prior to 8.5 may be generated in the current version. These can be resolved by selecting this option and the image will be generated using the older image generation functionality.

Directories tab

Defines the directories in which to store models, database scripts, reports, macros, quick launch files and datatype mappings.

Tools tab

- *ISQL Path*: Lets you type the file name and location of the SQL tool which will launch to manage the SQL ER Studio Data Architect outputs. You can replace the default *uisql.exe* editor with any ISQL editor. For example, you can use DBArtisan or SQL Server Enterprise Manager's Query Analyzer. The default path for DBArtisan is C:\Program Files\Embarcadero\DBAXXX\DBArtXXX.exe
- *Startup Directory*: Lets you enter a file name and location of the default save directory for the SQL application.

Diagram tab


Note: Options selected here apply immediately to the open data model.

- *Entity*: Select the entity information to display when you hover over an entity.
- *Schema Object Display*: Select the schema object information to display when you hover over a schema object.
- *Loiter Time*: The amount of time in seconds the popup displays when you hover over an object in the Data Model Window.

View tab

- *Duplicate View Column Name Resolution*
- *Prompt*: Prompts you to supply column aliases to differentiate between columns whenever duplicate column names exist.
- *Don't propagate*: ER Studio Data Architect Eliminates duplicate column names from the SELECT clause of the SQL statement generated to create the view.
- *Attach prefix and/or suffix*: Appends the supplied prefix or suffix to the column names to differentiate between duplicate column names. The entity name combined with an underscore character (EntityName_) is the designated prefix; a sequential number is the designated suffix (01).
- *View Column Propagation*
- *Propagate non-keys only*: Propagates only non-key columns from base tables to views.
- *Propagate all columns*: Propagates all columns to the view.
- *Don't propagate any columns*: Does not allow the propagation of any columns to a view. You must select the columns to propagate in the View Editor.
- *When adding new columns to a view...:* If you select Yes, columns added to a view are propagated to the views of any child views.
- *Display SQL Validation Dialog*: Prompts you to validate any updates made to the SQL code for the view.

- *Column Sorting*: Sets the column sorting hierarchy for the view, which will also order the DDL SELECT statement accordingly.

Schema Objects tab

- *Show triggers*: If selected, creates triggers in the DDL of the parent and child according to the parent and child actions specified.

For example, in the Microsoft SQL Server 2005 platform, selecting parent actions update, cascade and delete, and set null produces DDL in the parent table resembling the following:

REFERENCES dbo.child_name(foreign_key_name) ON UPDATE CASCADE ON DELETE SET NULL

Object Types tab

The object types selected here control the options that are by default selected in the Generate Other Object Types area of the General Options tab on page 3 of the DDL Generation Wizard. The options selected by default can be deselected within the DDL Generation Wizard. For more information, see Generating a Script File or Database.

Object Names tab

The object names chosen here control the names of triggers and procedures generated by the DDL Generation Wizard. You can choose to generate the system triggers and default table stored procedures on the Generate Other Object Types area of the General Options tab on page 3 of the DDL Generation Wizard. For more information, see Generating a Script File or Database.

Automation Options tab

Using the Automation Interface you can customize the functions called when select database objects are created, deleted or updated. In order to enable these customized functions, you must select them here. For more information, see [Automation Objects Programmer's Guide].

Data Dictionary tab

- *Show data dictionary name when displaying dictionary objects*: When selected, displays Data Dictionary objects using the selected format. For example, with the second option selected, the tid user datatype, in the *Pubs.dm1* sample model displays as *pubsDD.tid* in the Datatype column of the Entity Editor for the *Royalty Schedule* entity.
- *Migrate domain attachment bindings during drag n drop*: When selected, attachments bound to a domain are propagated to the attribute or column when populating an entity or table with a domain during a drag-and-drop operation, or when selecting the domain in the Domain Name list while editing an attribute or column in the Entity Editor or Table Editor.
- *Attribute/Column Definition Updating*: If Disallow Updating is selected, domain definitions cannot be modified from the entity/table editor.

Comments tab

- *Show Comment Notifications*: When selected, a notification icon will appear to the left of the object in the data model explorer tree when you change or add to the comments on a model object you are viewing. You will also see the comments icon of another use when you and the other user synchronize the model with ER/Studio Repository. You can add comments to most objects. For more information, see Adding and Editing Object Comments.

Where Used tab

The option chosen on this tab dictates which editor opens when you double-click an object on the Where Used tab of the Entity Editor or Table Editor.

- *Edit Mapping*: Choosing this option opens the user-defined mappings editor to edit the mapping properties of the object that was double-clicked.
- *Switch To*: Choosing this option opens the editor of the object that was double-clicked.

ERX File Import tab

- *Validate Views*: If selected, tables and columns in the view are checked for proper syntax. Views failing the validation are marked as invalid.
- *Validate Stored Procedures*: If selected, stored procedures are checked for proper syntax. Stored procedures failing the validation are marked as invalid.
- *Import Foreign Key Indexes*: Imports FK indexes that ERWin automatically generates for all FKs.
- *Use Logical Index Names when Physical Index Names are not defined*: If not selected, and the physical index name is blank, a unique index name is generated for any unnamed indexes. If selected and the physical name is blank, the corresponding name from the logical model is used. If, as a result of selecting this option, duplicate names are generated in the physical model, they can be isolated using the Model Validation Wizard.

Undo tab

For the operations you select here, you will be prompted for confirmation before executing the operation.

Repository Options tab


Note: This feature is not supported by the Developer Edition of ER Studio Data Architect.

This tab only displays if you have licensed the ER/Studio Repository.

- *Server Machine*: Displays the host name of the ER/Studio Repository. Click the Repository list to view and select another Repository. Click Refresh to update the

Repository list with the host names of all Repositories found on the network. The *Previously Used* folder gives a selection of saved servers that you have used before.

- *Refresh*: Click to refresh the list of Repository servers available to you on your network.
- *Active File Directory*: The directory to where files checked out or retrieved from the Repository are saved. The *Active File Directory* should be a directory on the local machine as opposed to a shared directory on the network; a shared network directory can result in Repository operation errors.

WARNING: Keep the Active File Directory on a local machine. A shared network directory can result in Repository operation errors.

- *Hide Repository Status Icons*: If selected, does not show the Repository Status icons, such as the lock, in the Data Model and Data Lineage explorers, and Data Model and Data Lineage windows. This option is useful when you want to print the Diagram without the icons displayed.
- *Show Repository Status Dialog*: Indicate where you want to display or hide the Repository Status dialog box, which shows timing information for each step taken during a Repository operation.
- *View SQL Alerts*: If selected, the Repository Merge operation will display an alert during check-in to warn the user when multiple users have modified data used by a view. When the SQL alert is displayed, there will be an option to not show such messages. Using the *View SQL Alerts* option, you can re-enable these messages.
- *Enable Repo Events Polling*: Repository Events Polling returns information about Repository operations so that the Diagram or Objects' status are updated with the most current information available. If you enable Repo Events Polling, you can then determine the time interval at which the polling occurs. The default is enabled.
- *Repo Events Polling Interval*: The number of seconds between polls to synchronize the status of local diagrams and objects with the Repository.
- *Hide General Information Data*: *The General Information in the Get Diagram dialog is obtained via a query against the database. Depending on the size of the database the query can take a while.* The General Information Data is obtained via a query against the database and depending on the size of the database, the query can take a while. Updating the statistics on the tables and indexes can improve query performance. For performance reasons, you may want to select this option which prevents the Repository status information from displaying in the *General Information* area of the *Get From Repository* dialog. Updating the statistics on the tables and indexes can improve query performance.
- *Display Synchronize Comments Warning*: When selected, this option will prompt the user to save any changes to the file before synchronizing the comments with the Repository. When not selected, the system automatically saves the file before synchronization.

- *Diagram Name Display Format*: Determines how the diagram name displays in Repository dialogs such as Get from Repository. If you choose, *Diagram Name (Filename.dm1)* the diagram name displayed might be Demo1 (Model.dm1) where Demo1 is the name field of the *Diagram Properties* dialog, which is accessed by choosing File > Diagram Properties.

See Also

- Defining Model Options for the Selected Model
- Customizing the Display of Diagrams and Objects
- Generating a Script File or Database

Specifying the Location of Shared Data

1. Choose Tools > Options, and then click the **Directories** tab.
2. In the **File Path** dialog, browse to an existing directory you want to store your shared files.
    - **Note:** ER Studio Data Architect will create a directory that does not exist but not a directory path. If your path has multiple levels it will need to be previously created.
3. In the **Options** dialog, click **OK**. When closing the dialog ER Studio Data Architect creates and/or searches in the changed directory.
4. If you are moving any of the following directories, you will need to copy the contents of the `DataTypeMapping`, `Macros` or `Sample Models` directories to the new location.
5. To effect the changes, please restart ER Studio Data Architect.

See Also

- Changing the Diagram File Name
- Administrator Guide

Changing Cursor Popup Help Options

Sets the popup information displayed when you hover the cursor over a diagram object such as an entity, table, schema object, or relationship. Help set for entities and relationships will be applied to both the logical and the physical model.

Choose View > Cursor Popup Help Options, and then select an entity display option.

See Also

- Customizing the Display of Diagrams and Objects
- Defining Model Options for the Selected Model

Data Modeling Fundamentals

ER Studio Data Architect is an entity-relationship modeling tool. The following sections outline data modeling concepts and especially entity-relationship modeling concepts, upon which ER Studio Data Architect is based. The design process, from creating a logical model to creating the physical model, is also explained in the following sections.

- Data Modeling Concepts: Describes data modeling concepts, the development of the relational model, entity-relationship modeling, and the IDEF1X methodology.
- Developing a Data Model: Describes the steps involved in creating the logical design; including naming objects; documenting the design; and normalizing the model; as well as creating the physical design, from transforming the physical design from the logical model to denormalizing the physical design and implementing the database.

See Also

- Configuring and Customizing
- Creating and Working With Data Models

Data Modeling Concepts

A data model represents the things of significance to your enterprise and the relationships between them. At its core, a data model depicts the underlying structure of an enterprise's data and the business rules governing it. A data model is comprised of two parts, a logical design and a physical design.

The following describes the main components of an ER/Studio diagram:

- The Logical Model
- The Physical Design
- Data Model Design Principals
- The Purpose of a Data Model
- The Relational Model
- The Entity-Relationship Model
- The Dimensional Model

The Logical Model

A logical model is developed before the physical model. It addresses the business and functional requirements of systems development. The logical design allows you to determine the organization of the data that you need to store in a database before you create the database; it serves as a blueprint.

The Physical Design

The physical design addresses the technical implementation of a data model, and shows how the data is stored in the database. For example, you can specify the datatype to be used by each column in the table, and determine how tables will be stored in the database.

Data Model Design Principals

To design the most effective data model possible, you should focus on the logical design before developing the physical design. Both the logical and physical design processes are complex, so it is best to separate rather than to mix the two. A sound logical design should streamline the physical design process by clearly defining data structures and the relationships between them.

The Purpose of a Data Model

A data model can be useful for other things in addition to creating databases, although creating a database is generally its primary purpose. In systems development, the goal is to create an effective database application that can support some or all of your enterprise. However, a data model of your business can help you define operational aspects of your business that you might otherwise overlook. Also, a well-defined data model that accurately represents your business,

can be helpful in orienting employees to goals and operations. The data model can also serve as an invaluable communications tool for both internal and external constituents.

The Relational Model

Most early data models were developed to help systems analysts make business data conform to a physical database or machine architecture. Hierarchical and network models often ran most efficiently on particular systems. In the early 1970s E. F. Codd developed the relational data model, based on relational algebra. The relational model stressed data independence, where data independence is defined as independence of data from the underlying physical structure in which it is stored. Thus, systems that supported relational data models let users easily migrate data to larger or newer systems with little regard to the physical differences between storage devices.

The power of the relational model lies in its simplicity. In the relational model, data is organized in tables of rows and columns. Each table represents one type of data. Each row, or tuple, represents one item of data of that type. Each column, or domain, represents one type of information about the type of data stored in the table.

Columns (Domains, Attributes)

Rows (Tuples, Instances)

EmpNo	LName	FName	SSN
001	Brown	John	002-34-9789
002	White	Ann	989-45-2323
003	Black	Leslie	451-00-4576

Sample Relational Table

The Entity-Relationship Model

Peter Chen introduced entity-relationship modeling during the late 1970s. Along with a number of other theorists, such as Hammer and McLeod with their Semantic Data Model, Chen introduced a new way of thinking about data. Chen's ideas stressed that a data model should represent the reality of a business, without regard for how that model might be implemented in a manual or automated system. Though it can seem ridiculous today, these ideas were revolutionary at the time, and were instrumental in freeing individuals from the constraints of a hierarchical business model. The ability to model a business 'on paper' let business planners test out and debug ideas before implementing them, thus saving money, other resources, and aggravation.

The basic idea behind entity-relationship modeling is this: everything in a business can be generalized into an abstract or archetypal ideal, which we call an entity. These entities have certain characteristics or attributes. These entities also are related to one another through actions that each entity performs on one or more of the other entities. We call these actions, relationships.



Simple Chen E-R Diagram

In determining the relationship between each of the entities in a data model, you will define a number of business rules. Business rules define your business and how you run it. For instance, you know that to be useful, each employee ID, project ID, and department ID must be unique. While you could use the social security number as the employee ID, you might want to keep the ID short and more easily sorted. More complex issues to consider are questions such as, "How are we to connect employees, projects, and departments so that we minimize data redundancy and maximize our reporting capabilities?" The answers to this and other questions form the rules by which you run your business.

Attribute definitions can be used along with relationships to determine or enforce business rules. You can define a particular set of valid values for any attribute of any entity. For example, you can define a valid range of salaries for a subset of employees for an attribute called Salary. This set of values is known as a domain.

The Dimensional Model

Dimensional modeling is generally agreed to be the most useful for representing end-user access to data. Where a single entity-relationship diagram for an enterprise represents every possible business process, a dimensional diagram represents a single business process in a fact table. ER Studio Data Architect can identify individual business process components and create the DM diagram for you.

You can create a dimensional diagram by starting with a logical model, by reverse-engineering, by importing an SQL or ERX file, or by adding a new physical model.

See Also

- Developing a Data Model
- Best Practices

Developing a Data Model

The following overview outlines the ER Studio Data Architect data model development process, which is an iterative process:

- Starting With a Basic Data Model
- Creating a Basic Data Model
- Reverse Engineering a Database
- Importing a File to Create a New Database
- Ensuring Data Integrity
- Automating Data Model Creation and Maintenance
- Fine Tuning the Data Model
- Securing the Data Model
- Preparing for Review and Implementation
- Maintaining the Data Model

Starting With a Basic Data Model

You can start from scratch or reverse engineer an existing database.

Creating a Basic Data Model

1. Create a new data model.
2. Add entities.
3. Define entity attributes.
4. Establish relationships between entities.
5. Assign keys.

Reverse Engineering a Database

Reverse engineer a database.

Importing a File to Create a New Database

Import data from a variety of sources to create a new database. See Importing a Model from an ERX File and Importing a Model from a SQL File for further details.

Ensuring Data Integrity

1. Ensure object names conform to business standards by creating and applying naming conventions.

2. Ensure data values are within a specified range or that they match a value defined in a list by defining reference values.
3. Ensure data input in tables or column is valid by add rules.

Automating Data Model Creation and Maintenance

1. Accelerate entity creation and updates by creating domains.
2. Assist user data entry by assigning default values.
3. Create macros to perform repetitive tasks.
4. Define default values for attributes, domains, and user datatypes.
5. Add reusable procedural logic.

Fine Tuning the Data Model

1. Validate the model.
2. Customize the datatype mappings.
3. Create user-defined datatypes for domains and attributes.
4. Generate a physical model.
5. Normalize to reduce data redundancy.
6. Denormalize to improve performance.

Securing the Data Model

1. Check the data model in to the Repository.
2. Prevent the data model from being updated or viewed by unauthorized persons by creating Users, Groups. and Roles and apply them to the data model. See Creating and Editing Database Users and Creating and Editing Database Roles for more information.
3. Prevent the database from being updated or viewed by unauthorized persons by creating Users and Roles and apply them to the data model. See Creating and Editing Database Users and Creating and Editing Database Roles for more information.
4. Specify data security information.

Preparing for Review and Implementation

1. Customize the model display.
2. Provide additional documentation for data model reviewers by creating attachments.
3. Document data lineage.
4. Create reports for distribution.

5. Make it easy for users to access the data by providing a dimensional mode.
6. Generate SQL to implement the database.

Maintaining the Data Model

1. Ensure the model is updated from the correct sources by documenting data lineage.
2. As the physical database is updated, synchronize the physical database model with the physical database.
3. As the physical database model is updated, synchronize the logical database with the physical database.
4. As updates are required, check out portions of the data model from the Repository.

See Also

- Data Modeling Concepts
- Customizing the Data Model

Using ER Studio DA

This section includes the following topics:

- Creating and Working With Data Models
- Best Practices
- Common Tasks
- Developing the Logical Model
- Developing the Physical Model
- Connecting to Database Sources and Targets
- Working with the Data Dictionary
- Documenting Data Extraction, Transformation, and Load
- Saving and Using Quick Launch Settings
- Generating RTF and HTML Model Reports
- Exporting the Data Model
- Printing the Data Model
- Exporting an Image of the Data Model

See Also

- Administrator Guide
- Working with ERStudio Repository

Creating and Working With Data Models

Using the data modeling features of ER Studio Data Architect you can create automatic and customized data model diagrams. ER Studio Data Architect provides functionality such as automatic layouts, custom colors and fonts for your objects, and relationship line control.

ER Studio Data Architect offers five preset auto layouts: circular, hierarchical, orthogonal, symmetric, and tree. You can use a preset layout to automatically generate your data model into a specific diagram type. These automatic layouts dictate how ER Studio Data Architect displays relationships and objects. Although these layouts create a specific diagram with specific rules that control relationships and object placement, you can still change certain aspects of the display, overriding the automatic layout.

You can set color and font settings for almost any object in your diagram. You can create changes that affect all objects in your diagram. If you make a change in color or font to the entire diagram, you can then select individual objects to customize differently.

ER Studio Data Architect gives you control over relationship lines. You can control the color, bending, straightening, verb phrases, and docking points of relationship lines.

If you are working with large data models, ER Studio Data Architect offers features that help you zoom and navigate your models easily. The Overview Window and Zoom Window display thumbnail views of your entire data model. Utilities accessible from the Diagram toolbar, let you zoom, navigate relationships, and fit your entire data model into the Data Model Window.

There a several ways to create a data model. This section describes these methods. Subsequent discussions describe how you can customize your data model.

- Creating a New, Blank Data Model
- Reverse Engineering an Existing Database
- Importing a Model
- Generating a Physical Data Model
- Using the Compare and Merge Utility
- Changing the Database Platform of a Model

See Also

- Best Practices
- Common Tasks

Creating a New, Blank Data Model

1. Choose File > New.
2. Select *Draw a new data model*, and then click *OK*.
3. Choose File > Save, navigate to the appropriate directory, enter a file name, and then click *Save*.

See Also

- Generating a Physical Data Model
- Changing the Database Platform of a Model

Reverse Engineering an Existing Database

Use the reverse engineering function to create a logical and physical model by extracting information from an existing data source. You can specify a data source to reverse engineer using an ODBC or native connection. Using the Reverse Engineer Wizard and the native connection, you can specify exactly what you want to reverse engineer, selecting from a particular database objects such as: tables, views, indexes, defaults, rules, user datatypes, owners, objects, views, and dependencies. Using the Reverse Engineer Wizard and an ODBC connection allows you to reverse engineer only tables and views. The information extracted for tables includes constraints, primary keys, foreign keys, and comments. Using the Quick Launch feature, you can save a set of reverse-engineering options to a file and retrieve them the next time you need to perform a similar reverse-engineering operation.

1. Choose File > New.
2. Select *Reverse-engineer an existing database*, and then click *Login*.
3. Enter your database connection and login information.
4. The *Reverse Engineer Wizard* displays.

Here you can select specific objects and information to include in the reverse engineered model. The Reverse Engineer Wizard will walk you through the rest of the process.

The following describe fields and options on the pages of the wizard that require additional explanation:

- Page 1
- Page 2
- Page 3
- Page 4
- Page 5
- Notes
- Specifying Application Options for Reverse Engineering

Page 1

- Connection Type:
    - *ODBC*: Using an ODBC connection, you can only reverse engineer tables and views. To add a new ODBC Data Source or edit an existing one, click Setup to open the ODBC Data Source Administrator dialog. For more information, see Configuring ODBC Data Source and Target Connections.
    - *Native/Direct*: Using a native or direct connection, you can reverse engineer all supported objects for the selected platform. For more information, see Using Native or Direct Database Connections.

    - *ER/Studio Team Server Data Source*: Using an ER/Studio Team Server data source, you can reverse engineer a database/schema whose connection details are retrieved from an ER/Studio Team Server server. You can reverse engineer all supported objects for the selected platform. If you are not logged in to ER/Studio TS, you will be prompted to log in.

- **Database Type**: Using a native connection, ER Studio Data Architect can reverse engineer the following databases:
    - IBM DB2 for OS/390
    - IBM DB2 for LUW
    - MS SQL Server
    - Oracle
    - Sybase ASE

This control is disabled when **ER/Studio Team Server Data Source** is selected.

- *Data Source*: This field differs depending on whether you selected an ODBC or a native connection.
    - For an ODBC or ER/Studio Team Server connection, select a data source from the list. Note that for ER/Studio TS data sources, the list may include data sources for platform versions not supported by ER Studio Data Architectta Architect.
    - For a native connection, enter the data source connection string.
- *User Name* and *Password*: These fields let you provide credentials.
- *Create ER/Studio Team Server Data Source from settings*: These controls are disabled if *ER/Studio Team Server Data Source* is selected. Selecting the check and providing a *Name* creates a data source definition on the ER/Studio Team Server server as soon as you click *Next*, using the settings you provided.
- *Quick Launch*: In the last step of the wizard, you can save settings to a file for later reuse. If you have previously saved such a file, you can load it step by clicking the... button next to Select Settings File on page 1 of the wizard. Then click Go to run the import exactly as before, or click Next to modify the settings. The Wizard Quick Launch data is saved as an.rvo file.

The default location for these files is:

    - **Windows XP**: C:\Documents and Settings\<*user*>\Application Data\Embarcadero\ERStudio\XML
    - **Windows 8, Windows 7, and Windows Vista**: C:\Users\<*user*>\AppData\Roaming\Embarcadero\ERStudio\XML
      To change the default directory for Quick Launch files, choose **Tools > Options > Directories**.

- AppWave ION: My Documents\ERStudio Data Architect XE\XML.

Page 2

- *Database List*: If the database platform supports databases, browse and locate one or more databases to reverse engineer.
- *Owner List*: If the database platform supports owners, browse and locate one or more owners to reverse engineer.
- *Include*: Select the objects to include in the data model.

Page 3

- *Capacity Planning*: Selecting any of these options returns information about the database that can help you plan for storage requirements. The options available are database dependent.
- *Object* tabs: Displays a tab for every object type selected in the Include list on the previous page. Click the object tab, review the available objects, and then use the arrows to move the objects you want to reverse engineer to the Selected Objects area.

Page 4

- *Infer Referential Integrity*: ER Studio Data Architect can infer referential Integrity when none is declared in the database. Click one or more options to create relationships between entities in the reverse-engineered diagram.
- *Infer Primary Keys*: If selected, ER Studio Data Architect infers primary keys from unique indexes on a table. If more than one unique index exists on a table, ER Studio Data Architect chooses the index with the fewest columns.
- *Infer Foreign Keys from Indexes*: If selected, ER Studio Data Architect infers foreign keys from indexes. When inferring foreign keys from indexes, ER Studio Data Architect looks for indexes whose columns match the names, datatype properties, and column sequences of a primary key. If the "child" index is a primary key index, it must contain more columns than the "parent" primary key. In this case, an identifying relationship is created.
- *Infer Foreign Keys from Names*: If selected and your database contains foreign keys, ER Studio Data Architect infers foreign keys from names. When inferring foreign keys from names, ER Studio Data Architect looks for columns matching the names and datatype properties of a primary key. In this case, a non-identifying relationship is created. ER Studio Data Architect cannot infer relationships where the child column has a role name, instead create a Non-Identifying Relationship and then designate the correct rolenamed column using the **Edit Rolenames** function; right-click relationship and then select Edit Rolenames.
- *Infer Domains*: If selected, ER Studio Data Architect infers domains from the columns in the database. ER Studio Data Architect creates a domain for each unique combination of a column name and its associated datatype properties. Duplicate domains with an underscore and number suffix indicate that columns with the same name but different

datatypes were found in the database. This can alert you of how standardized the columns are in a database. You can use macros to consolidate domains and preserve the bindings from the inferred domains.

- *View Dependencies*: ER Studio Data Architect can ensure all objects referenced by those selected for reverse engineering are also included.
- *Reverse Engineer View Dependencies*: If selected, ER Studio Data Architect includes referenced view dependencies.
- *Reverse Engineer Other Dependencies*: If selected, ER Studio Data Architect includes referenced dependencies such as procedures and triggers. Dependent objects that are not otherwise selected will be included. For example, a database contains proc1, proc2, and proc3; and proc3 references proc1 and proc2. During reverse engineering, if you selected only proc3 and this option, proc1 and proc2 are also included.
- *Select the Initial Layout Option*: Select an initial layout for the data model
- *Circular Layout and Tree Layout*: Provide best performance when reverse engineering large databases. Reverse engineering a large database to create a hierarchical model can be quite time consuming.
  For more information on layout, see Changing Data Model Layout.
- *View Parser*: Select a platform-specific syntax interpreter.

Page 5

- *Summary of Selected Objects*: Select an object type to display in the Summary of Selected Objects grid and then review the object type, owner, and object name of all the objects you selected to reverse engineer.
- *Create custom datatypes for unsupported types*: If the selected mappings file contains new datatypes because you added them or they were added by a previous reverse-engineering process, then those datatype mappings will be used.

Notes

- You can reverse engineer diagrams created with DT/Designer to create ER Studio Data Architect models.
- Reverse-engineered diagrams do not display users and roles by default. To enable Users and Roles, choose View > Diagram and Object Display Options > Security Objects, and then select *Display All Security Objects*.
- To create a logical model only from the database, choose Tools > Options > Application and then select Logical Model Only in the Reverse Engineer area.
- In order to reverse engineer a database you must have both CONNECT and RESOURCE roles assigned to your user ID for the database you want to access.
- Objects in SQL 2005 are owned by schemas. A schema is similar to a user, and can be treated the same way; however, schemas are not explicitly represented in the explorer tree, nor is there a CREATE SCHEMA DDL function. In the Reverse Engineer Wizard and

Compare and Merge Utility, the "owner" field in the object editors of ER Studio Data Architect represent the schema name for an SQL Server 2005 physical model. The field will be the schema list.

- NUMBER datatype: NUMBER or NUMBER(*) you must set the width and scale to *. A width of 0 will be converted to * when generating DDL.
- Width and scale information of unsupported and unmapped datatypes is not retrieved in the reverse-engineering process. If you need this information, create a custom datatype manually through the Datatype Mapping Editor before reverse-engineering.

Specifying Application Options for Reverse Engineering

Using the Application tab of the Options Editor, you can specify whether reverse engineering always produces both logical and physical models or produces logical models only. You can also specify the column sequence for new models that you reverse engineer. Once applied, this customization will apply to all your reverse-engineering projects.

1. Click Tools > Options, and then click the *Application tab*.
2. Specify your reverse-engineering preferences in the *Reverse Engineer* and *Column Order: Reverse Engineer* areas of the *Application* tab.
3. Click *OK* to apply your changes to future reverse-engineering projects.

See Also

- Changing Data Model Layout
- Importing a Model

Importing a Model

ER Studio Data Architect allows you to import models from several other applications. This section describes the following import methods:

- Importing a Model from External Metadata
- Importing a Model from an ERX File
- Importing a Model from a SQL File

**Note:** For the best performance when reverse engineering large databases, select Circular or Tree as the initial layout of your model.

See Also

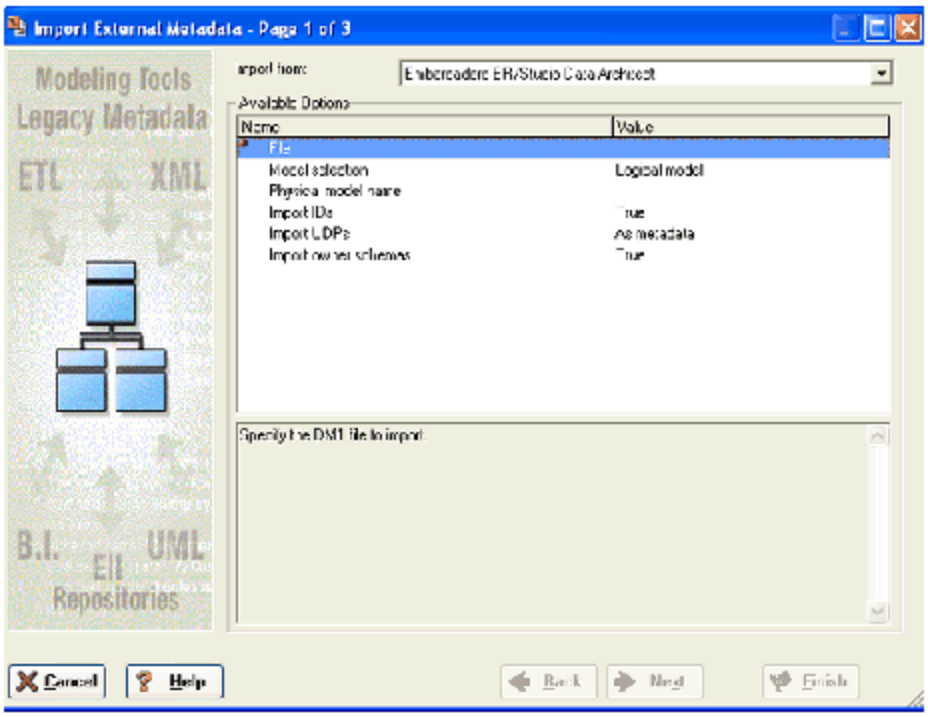
- Reverse Engineering an Existing Database
- Creating a New, Blank Data Model

Importing a Model from External Metadata

MetaWizard, a separate product available from Embarcadero, extends ER Studio Data Architect"s functionality by providing an import bridge that allows you to import metadata from applications such as Microsoft Excel, CA AllFusion Erwin, IBM Rational Data Architect, Oracle Designer, and Sybase PowerDesigner, and also allows you to import metadata from XML structures.

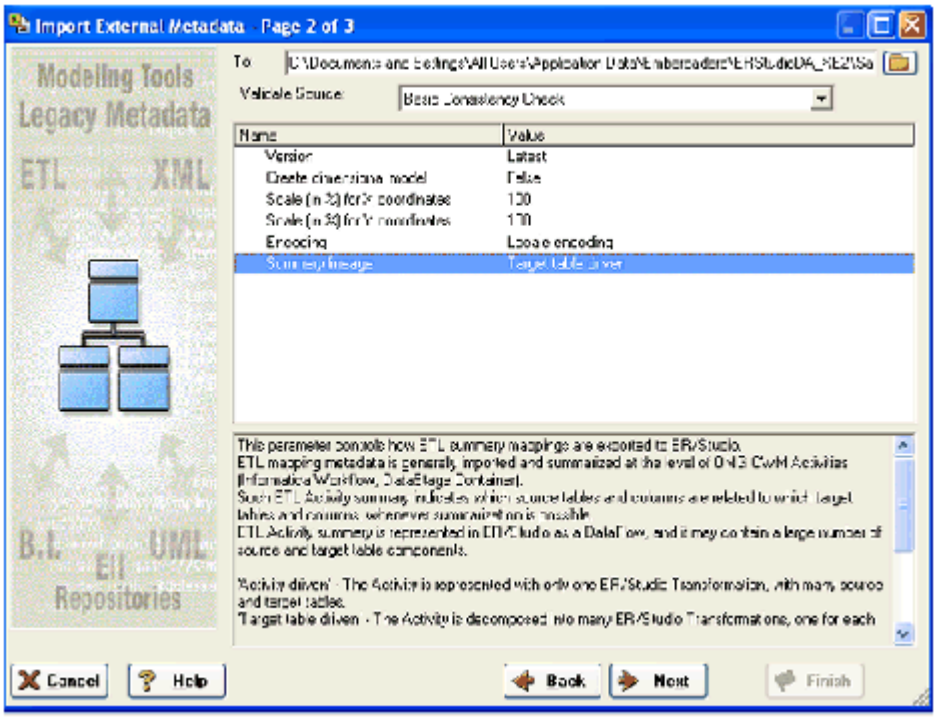
1. Click File > New.
2. Select *Import Model from* and then from the list, click *External Metadata*.

   Tip: Alternatively, with a model already opened, you can select File > Import File > From External Metadata. In this case, the new model will be imported into the open diagram.
3. Click *Import*.
4. From the *Import from* list in the *Import External Metadata* wizard, select the XML type of the file to be imported or the application that created the file.



5. In the *Available Options* area, click the *Value* field in the *File* row, click the ellipses (...) and then select the file to be imported.
6. Optionally, set the other available options, such as *Model selection*, *Physical model name*, and *Import owner schemas* which vary depending on the datasource type selected in the *Import From* list.

7. To change a type-specific option, click its value.
8. Click *Next*.



9. In the *Name* column, click on each of the parameters to see descriptions and help for the parameter. Click on the corresponding cell in the *Value* column to change its value.
10. To start the import, click *Next*.
    The import log displays.
11. If the import was successful, click *Finish* to create the model.
12. When the *Import Status Dialog* displays *Finished Import* click *Close*.
13. Save the new model.

Note:

- The options are self-explanatory and additional details appear in the area beneath the options list.
- You can find additional information on the Meta Integration Model Bridges at http://www.metaintegration.net/Partners/Embarcadero.html.
- The Import File From External Metadata command generates a data model based on metadata in standard XML and common proprietary formats. For a current list of supported platforms, valid BridgeIDs, and corresponding bridge names, see:
- *Windows XP*: *C:\Program Files\Embarcadero\ERStudioX.X\MetaIntegration\MIRModelBridges.xml*

- *Windows 8, Windows 7, and Windows Vista*: *C:\ProgramData\Embarcadero\ERStudioX.X\MetaIntegration\MIRModelBridges.xml*
    - The.pdf version of this file is available at [1].

See Also

- Importing a Model from an ERX File
- Importing a Model from an SQL File

Importing a Model from a SQL File

1. Choose File > New.
2. Select *Import Model from* and then from the list, choose *SQL File*.
3. Click *Import*.
4. Complete the *Import Database SQL File* dialog, click *OK*.
   The import log displays.
5. Optional. To save or print the log, click *Save to File* or *Print*.
6. If import was successful, click *Finish* to create the model.
7. When the *Import Status Dialog* displays *Finished Import*, click *Close*.
8. Save the new model.

The following describe fields and options in the wizard that require additional explanation.

- Quick Launch: In the last step of the wizard, you can save settings to a file for later reuse. If you have previously saved such a file, you can load it by clicking the... button next to Select Settings File on page 2 of the Import Model wizard. Then click Go to run the import exactly as before, or click Next to modify the settings. The Wizard Quick Launch data is saved as an.rvo file. The default location for these files is:
- Windows XP: C:\Documents and Settings\<*user*>\Application Data\Embarcadero\ERStudio\XML
- Windows 8, Windows 7, and Windows Vista: C:\Users\<*user*>\AppData\Roaming\Embarcadero\ERStudio\XML
- AppWave ION: My Documents\ERStudio Data Architect XE\XML
  To change the default directory for Quick Launch files, choose **Tools > Options > Directories**.
- Select a Database SQL File: Type the database file name or browse to the file location and then click Open.
- Select the target database platform: From the list, select the target database platform.
- Infer Referential Integrity: ER Studio Data Architect can infer referential Integrity when none is declared in the database. Click the options that follow to create relationships between entities in your diagram.
  - Infer Primary Keys: If selected, ER Studio Data Architect infers primary keys from unique indexes on a table. If more than one unique index exists on a table, ER Studio Data Architect chooses the index with the fewest columns.
  - Infer Foreign Keys from Indexes: If selected, ER Studio Data Architect infers foreign keys from indexes. When inferring foreign keys from indexes, ER Studio Data Architect looks for indexes whose columns match the names, datatype

properties, and column sequences of a primary key. If the "child" index is a primary key index, it must contain more columns than the "parent" primary key. In this case, an identifying relationship is created.

  - **Infer Foreign Keys from Names**: If selected and your database contains foreign keys, ER Studio Data Architect infers foreign keys from names. When inferring foreign keys from names, ER Studio Data Architect looks for columns matching the names and datatype properties of a primary key. In this case, a non-identifying relationship is created. ER Studio Data Architect cannot infer relationships where the child column has a role name, instead create a non-identifying relationship and then designate the correct role-named column using the Edit Rolename function; right-click the relationship and then click Edit Rolename.
  - **Infer Domains**: If selected, ER Studio Data Architect infers domains from the columns in the database. ER Studio Data Architect creates a domain for each unique combination of a column name and its associated datatype properties. Duplicate domains with an underscore and number suffix indicate that columns with the same name but different datatypes were found in the database. This can alert you of how standardized the columns are in a database. You can use macros to consolidate domains and preserve the bindings from the inferred domains.

- **Select the Initial Layout Option**: Select an initial layout for the data model:
  - **Circular Layout** or **Tree Layout**: If selected, provides best performance when reverse engineering large databases. For more information, see Changing Data Model Layout.
- **View Parser**: Select a platform-specific syntax interpreter.
- **Summary of Selected Objects**: Select an object type to display in the Summary of Selected Objects grid and then review the object type, owner, and object name of all the objects you selected for the reverse engineer.
- **NUMBER datatype**: NUMBER or NUMBER(*) you must set the width and scale to *. A width of 0 will be converted to * when generating DDL.

See Also

- Importing a Model from an ERX File
- Importing a Model

Importing a Model from an ERX File

ER Studio Data Architect lets you create a data model by importing ERX files created with ERwin or Meta Integration's MIMB. ER Studio Data Architect can import ERX files created with ERwin, versions 2.6 through 3.52.

1. Click File > New.
2. Select *Import Model from* and then from the list, choose *ERX File*.
3. Click *Import*.
4. Navigate to the appropriate directory, select the.erx file, and then click *Open*. The import log displays.
5. Optional. To save or print the log, click *Save to File* or *Print*.
6. If import was successful, click *Finish* to create the model.
7. When the *ERX Import Status* displays *Finished Import*, click *Close*.
8. Save the new model.

See Also

- Importing a Model from a SQL File
- Creating a New, Blank Data Model
- Importing a Model from External Metadata

Generating a Physical Data Model

The Generate Physical Model Wizard automatically creates a relational or dimensional physical model based on a logical model, freeing you to focus on performance during the physical model design phase. The wizard can assist you by ensuring conformance with the basic naming and syntax rules of the target database platform; implementing referential integrity, such as the enforcement of logical keys through indexes or table constraints; and replacing non-specific (many-to-many) relationships with alternative schema elements more appropriate to the relational or dimensional model being generated.

Once you have created a logical data model you can generate a physical data model for many different platforms that is based on the logical data model selected.

1. In the *Data Model Explorer*, right-click the *Main* model or a submodel of the logical data model, and then click *Generate Physical Model*.
2. Complete the *Generate Physical Model Wizard* and then click *Finish* to complete the wizard.

The following describe options that require additional explanation:

- *What is the target database platform*: ER Studio Data Architect still refers to IBM platforms by their old names.

Name used in ER Studio Data Architect	Now known as
IBM DB2 AS/400	IBM DB2 iSeries
IBM DB2 UDB	IBM DB2 LUW
IBM DB2 UDB for OS/390	IBM DB2 z/OS

- *Indexes*: The storage and index parameters vary depending on the target database platform selected on page 1. PK indexes will be generated for each entity having primary key columns. For platforms that support clustered PK indexes, a clustered option exists on page 2 of the wizard.
- *Naming standards* are applied on page 3 of the wizard using either an XML file or a Naming Standards Template from the Data Dictionary. Naming standards can replace or remove spaces, perform uppercase/lowercase conversion, and replace invalid SQL characters.
- *Quick Launch Settings and Objects*: All fields and settings from pages 1 through 5 of the wizard will be restored when you click Load File or Use Settings. With the Settings Only option, only those general options that are likely to be useful with a different model are restored, while Settings Only loads information that is generic and can be used across models. The Settings and Objects option will, for example, load SQL path data specific to the model.

Note: For information on defining storage for tables and indexes, see Defining Table Storage and Defining Index Storage.

Creating a New Physical Model

ER Studio Data Architect lets you create a new physical model by reverse-engineering a live database or importing an SQL or ERX file.

1. On the *Data Model* tab, right-click the logical model icon and then select *Add New Physical Model*.
2. Complete the *Add New Physical Model* wizard and then click *Finish*.

The following describes the options that are available from the first page of the wizard:

- *Reverse engineer from a database*: The options on this tab and subsequent steps are the same as those for the Reverse Engineering Wizard. For information on these options, see [Reverse Engineering an Existing Database].
- *Load from a SQL File*: The options on this tab and subsequent steps are the same as those for the Import Model Utility for importing an ERX File. For more information on these options, see Importing a Model from a SQL File.
- *Load from an ERX File*: The options on this tab and subsequent steps are the same as those for the Import Model Utility for importing an ERX File. For more information on these options, see Importing a Model from an ERX File.
- *Load from an ER Studio Data Architect File*: Enter the file name, including the path, select the physical model to import, and then click Finish.
- *Load from a DT/Designer File*: Enter the file name, including the path, select the physical model to import, and then click Finish.

See Also

- Reverse Engineering an Existing Database
- Physical Storage Considerations

Using the Compare and Merge Utility

This section contains the following topics:

- What Can the Compare/Merge Utility Do?
- Comparing Models and Submodels
- Forcing a Match
- Updating an Offline DBMS
- Updating a Model or Submodel
- Resolving and Reporting on Model Differences

What Can the Compare/Merge Utility Do?

The Compare and Merge utility allows you to reconcile differences between models in the same file or between a model and a database. For example, you have designed and created a model and then created a database based on that data model. A data modeler alters the model and adds a column *Gender* to the Employee table and simultaneously a DBA alters the Department table in the database and adds a column *Name* to the table. The model and the database are no longer in synch. Round-trip engineering using the Compare and Merge Utility can now help you reconcile the model with the database.

The Compare and Merge Utility compares, merges, and synchronizes models from a variety of comparison targets. It can perform different kinds of comparisons. You can use the information from these comparisons to perform merges or to create detailed reports.

The Compare and Merge Utility compares a source and a target model. The active diagram, the one you are viewing and editing, will always be the source, and from it you select a valid target with which to compare and subsequently merges if desired, or to create alter sql to update offline databases.

Note: Targets of logical and physical ER Studio Data Architect data models can be Repository-based *data models* including Named Releases, which are read-only archived snapshots of diagrams managed within the Repository.

You can compare and merge a *logical* model to:

- A physical model in the same.dm1 file.
- A logical model of another ER Studio Data Architect data model.
- A physical model of another ER Studio Data Architect data model.

You can compare and merge a *physical* model to:

- The logical model of the same data model.

- Another physical model of the same data model that shares the same DBMS platform (and version).
- A logical model of another ER Studio Data Architect data model.
- A physical model of another ER Studio Data Architect data model sharing the same DBMS platform (and version).
- A live database.
- An SQL file.

When comparing two physical models, make sure they are the same platform and version (i.e. Oracle 11g to Oracle 11g). if they are not the same, then you will only be able to compare to the logical. If you want to compare Oracle to SQL Server, you would need to compare to the logical model from the reverse engineered physical database, then push the changes from the logical model to the original physical model.

You can compare and merge a *submodel* to:

- A submodel of the same data model.
- The physical model of the same data model.
- An SQL file.

The broad spectrum of valid targets allows for various information-merging strategies. For example, information in the Compare and Merge Utility can be:

- Merged from the current model to a valid target.
- Merged from the valid target to the current model.
- Bi-directionally merged simultaneously between source and valid target.


Note: Objects in SQL 2005 are owned by schemas. A schema can behave like a user, but does not have to be a user. A schema is not explicitly represented in the Data Model Explorer, nor is there a CREATE SCHEMA DDL function. In the Reverse Engineering and Compare/Merge Wizards, the "owner" field in the object editors of ER Studio Data Architect represents the schema name for an SQL Server 2005 physical model. The field will be the schema list.

A subset of the functionality on the Compare and Merge Utility is used to bring new data source into the diagram for purposes of importing new data sources from the Data Lineage tab.

Comparing Models and Submodels

Using the Compare and Merge Utility, you can compare two models, two submodels, or a model with a submodel and merge any differences found, if required. You can also report on the differences and if you want, synchronize your source model with another model or update the source or target models selectively.

Note: When comparing or merging with a physical data model or SQL imported data model that was not initially created by reverse-engineering an existing database, you must synchronize constraint names between the data model and the database in order to generate correct ALTER SQL. The Compare and Merge Utility may not generate the SQL correctly unless the target model (physical or SQL import) has the same constraint names as the database.

1. Choose File > Open and display the source data model or submodel.

   Tip: You can save time by selecting diagram objects on the data model before opening the utility. Objects that are pre-selected in this manner will be already in the object tree of the utility.

2. In the *Data Model Explorer*, click the model or submodel to be the comparison source.

3. Choose Model > Compare and Merge Utility.

   Tip: The number of pages and options in the Compare/Merge utility vary depending on what kind of compare/merge you perform and the source and target models selected.

4. The *Compare and Merge Utility Wizard* will walk you through the rest of the process.

The following describes the options that require additional explanation:

- Page 1
- Page 2
- Page 3
- Page 4 - Select Specific Model Objects
- Page 5 - Results

Page 1

- *Compare against a Repository based DM1 file*: Compares a data model with another data model or Named Release managed within the ER/Studio Repository. When you select this option, ER Studio Data Architect opens the Repository Operation Status dialog box and the Get From Repository dialog box where you can choose the diagram you want to compare with. This process connects to the current Repository Server defined in the Repository Options (Repository > Repository Options). The Model Compare and Merge wizard automatically gets the diagram from the Repository.
- *Compare against an SQL file* This option is available when the current model selected is a Physical data model. The selected SQL file must be of the same database platform as the current model. When you select this option, ER Studio Data Architect imports the SQL file and compares it to the current model.
- *Compare against a live database:* If you select this option, a page appears where you can select the database and connection type. The connection type can be:
- ODBC

- Native/Direct Connection.
- ER/Studio Team Server data source

If you select the ODBC or Native/Direct option, you also have the option to create a new data source definition on ER/Studio TS, using the settings you provide on this wizard.

For information about ODBC and native/direct connections, including troubleshooting information, see *Connecting to Database Sources and Targets* in the Repository Installation and Administration Guide.For information on ER/Studio TS data sources, see Logging In and Out of the Repository and ER/Studio TS.

- **Comparison Quick Launch**: The Compare Quick Launch data is saved as an *.rvo file. For information on using the Quick Launch option in the wizard, see Saving and Using Quick Launch Settings.

Page 2

The models available on this page depend on the target type chosen on the previous page. Click one model or submodel to compare against the source model.

Page 3

- Comparison Options

Please select the off-line ALTER SQL generation options: This option is available only when comparing a physical data model to an SQL file or to a live database and you have selected to generate SQL on the first page of the utility and have changed at least one of the default ignore resolutions. When the target model is a live database you have the option to generate ALTER SQL or to update the target model when the merge is executed. If you choose to create the ALTER SQL, an additional page of the Compare and Merge Utility becomes available where you can choose where to save the script file, along with Rebuild and Pre/Post SQL Options.


Note: If physical models were not reverse-engineered in the first place, you may have to synchronize constraint names between models and the database in order to generate correct ALTER SQL. It won't run correctly unless the target model (physical or SQL import) has the same constraint names as the database.

- Objects

Selecting or clearing a higher-order objects selects or clears all related subobjects. For example, if you clear Column Constraints, the Name and Value subobjects will also be cleared.


Note: For Oracle databases, deselecting the Disabled checkbox under the Relationship or Indexes node causes the utility to ignore the disabled attribute of Primary and Foreign Keys.

Source Column Mapping Properties: These options refer to the mappings created on the Data Lineage tab of the Table Editor. For more information, see Documenting Data Extraction, Transformation, and Load.

- General Options

- **Merge only selected objects**: When selected, only objects that are selected in the model prior to invoking the Compare Wizard will be selected in the Object Selection page of the wizard (page 4). This is an easy way to compare just a portion of the model.
- **Match Native to Foreign Attribute**:
- **Show Base Tables Of Denormalization Mappings**: When comparing two physical models, shows the base tables of denormalization maps in the results page; the last page of the wizard. Base tables are the pre-map state tables that are included in a denormalization map. These tables can be seen in the model by expanding the Denormalization Mapping node in the Data Model Explorer. These objects are not actually visible in the Data Model Window, but can be made visible by undoing the denormalization map.
  In the results page of the Compare and Merge Wizard, ignored columns are italicized. By displaying the ignored columns in the results screen, you can preserve changes on the target side as you are merging denormalization maps from one physical model to another.
  Denormalization objects are merged in order; the base objects are merged to the target first, and then the denormalization mapping is done. By setting resolutions to Ignore, you can preserve changes after the denormalization mapping is created.
- **Options for new Tables and Columns**: Exclude table and column definitions, notes, and attachments. These options are for comparing all model types against each other. When objects do not exist in the target model and new objects are created via the merge, the definitions, notes, and attachments will not be included for those new objects. This option can be useful when separate definitions, notes, and attachments are used between models.
- **Compare Primary Key As Unique Index**: When comparing against a database, ER Studio Data Architect normally treats PK indexes in the model as PK constraints. When this option is enabled, it treats PK indexes as indexes instead of constraints. Use this option if the PKs in the database are controlled by unique indexes instead of being controlled by constraints.

Page 4 - Select Specific Model Objects

- **Please select the specific model objects for your comparison**: The tabs available on this page are dependant on the database platform of the model selected for comparison. The list of database objects on each tab is populated by the objects available in the source and target models. On these tabs you can limit the comparison to specific objects only or choose to compare all object types.

**Note:** If you right-click anywhere on the object tree, you can select or deselect all objects using the short-cut menu. You can also display or hide object owner names for those objects with owners.

Page 5 - Results

- Current and Target Model Display Grid: Between the Source and Target models is a Resolution column. The default merge decision is Ignore. You can click the Resolution column to view a list of possible resolutions. If you want to change the resolution, click the list and then click the new resolution. When you change the default resolution of an object, the resolutions of their dependent properties and objects are automatically updated. You can also click the category folders, like the Tables Resolution column to change all the resolutions for all the underlying objects in that object category. And, you can use the CTRL key to select multiple items, and then right click to enable the list of possible resolutions.
- Force Match: In some cases the Compare and Merge utility fails to match objects because certain criteria are not met but you may want them matched anyway. In those cases you can force the system to match those objects. Select the object in the Current Model tree, scroll down and right click the target object in the Target Model tree, and select Match Objects. For more information, see Forcing a Match.
- Reverse-Engineer parent tables: Reverse engineers parent tables of selected objects so referential constraints are valid.
- SQL DIfference: To enable the SQL Difference utility, select any difference that is a long text field, such as a Definition, Note, or DDL, and then click SQL Difference to view the differences between the SQL of the models. This functionality allows you to only view the differences; you can resolve the differences on the Results page of the Compare and Merge Utility.

Forcing a Match

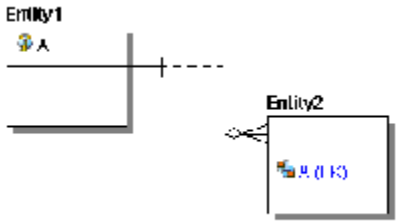
In some cases the Compare and Merge utility fails to match objects because certain criteria are not met but you may want them matched anyway. The following describes a case where the Compare and Merge utility is forced to match objects that differ only slightly.

Create a new diagram with one entity called *Entity1* with a primary key of A.

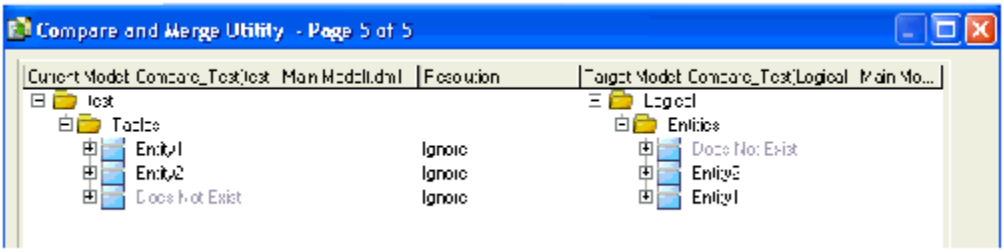
1. Assign *JimB* as the owner of Entity1.

   Tip: The Owner property is defined on the top right-hand side of the Entity Editor or Table Editor.

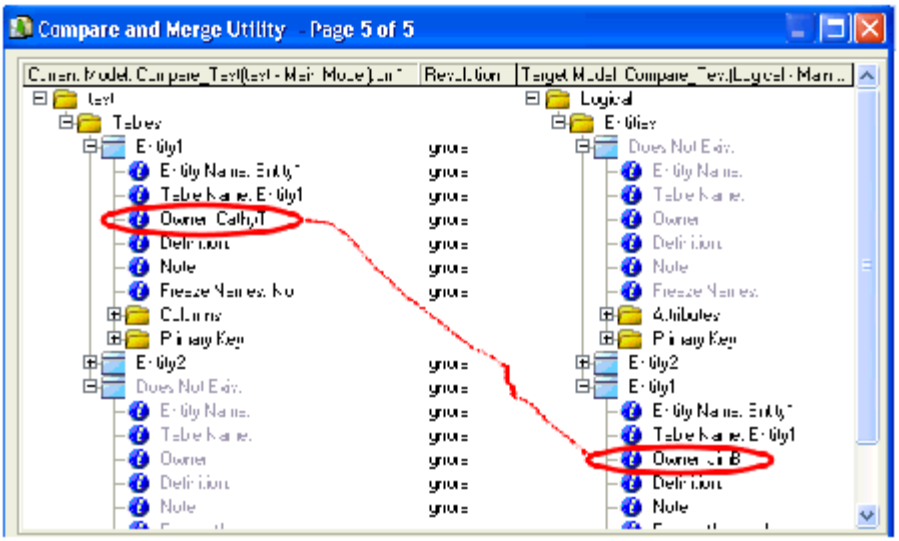
2. Create another entity, Entity2 and then create a non-identifying, mandatory relationship between Entity1 and Entity2.



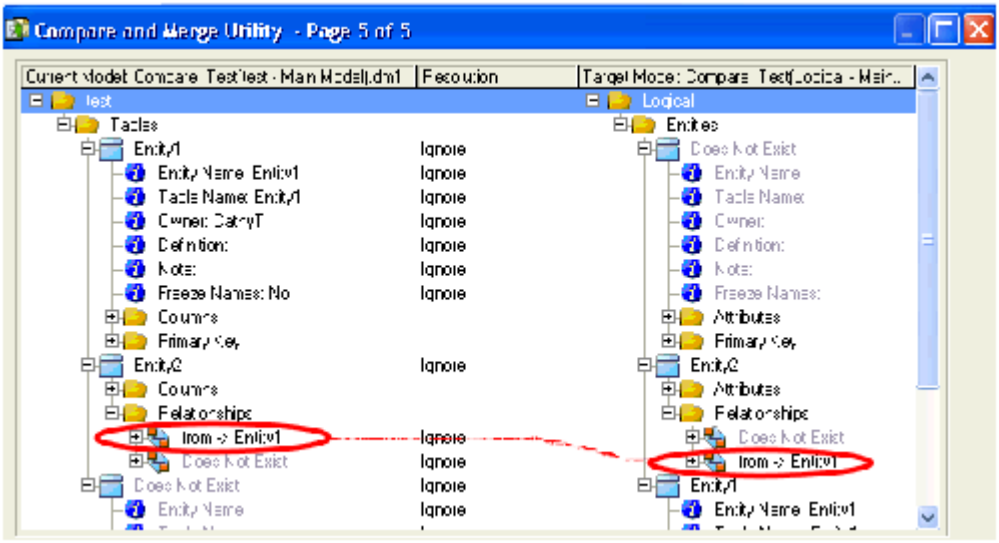
3. Generate a physical model from the logical.
    - In the Data Model Explorer, right-click the logical model and select Generate Physical Model.
4. Assign *CathyT* as the owner of Entity1 in the physical model.
5. In the Data Model explorer, right click the physical model and select **Compare and Merge Utility...**.
6. Click through the Compare and Merge Utility by clicking *Next* on each page until Page 5 of 5 appears.



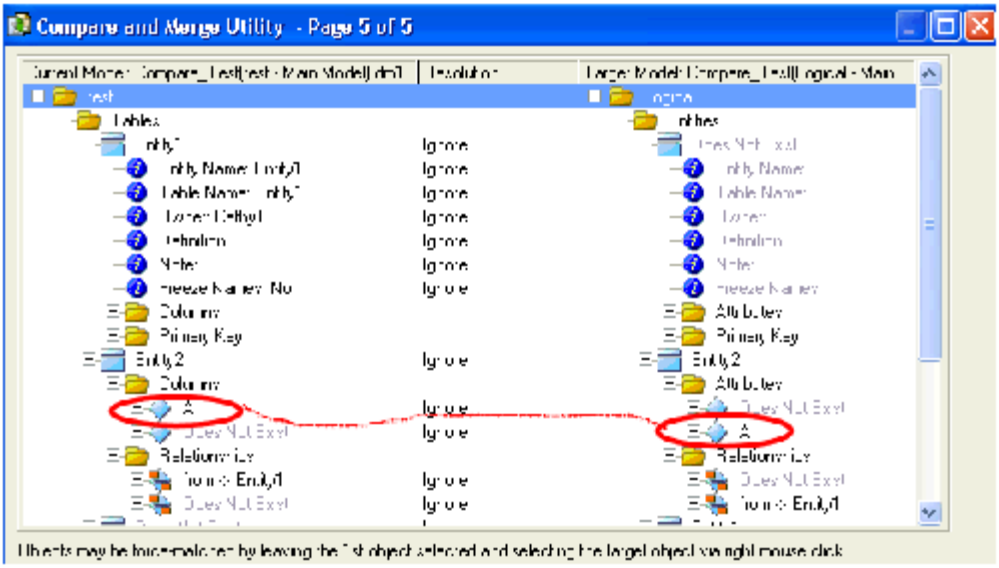
Notice that Entity1 and Entity2 in the physical model do not align with their counterparts in the logical model. This is because Entity1 has a different owner in the logical model than in the physical model



7. Expand Entity2 in the Current and Target Model trees and then expand the Relationships in Entity2

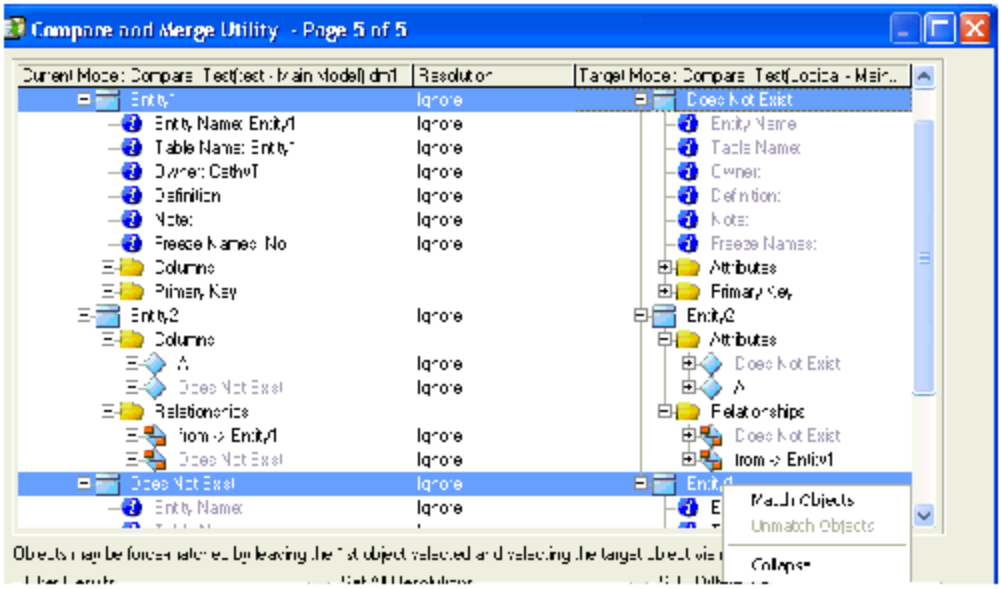


Notice that in Entity2, the relationships are not matched either, because the parent tables are not matched. This means that the foreign key columns are not matched either.



Because you know that Entity1 is the same in both models, you may want to match Entity1 in the Compare and Merge Utility and then update the logical model later.

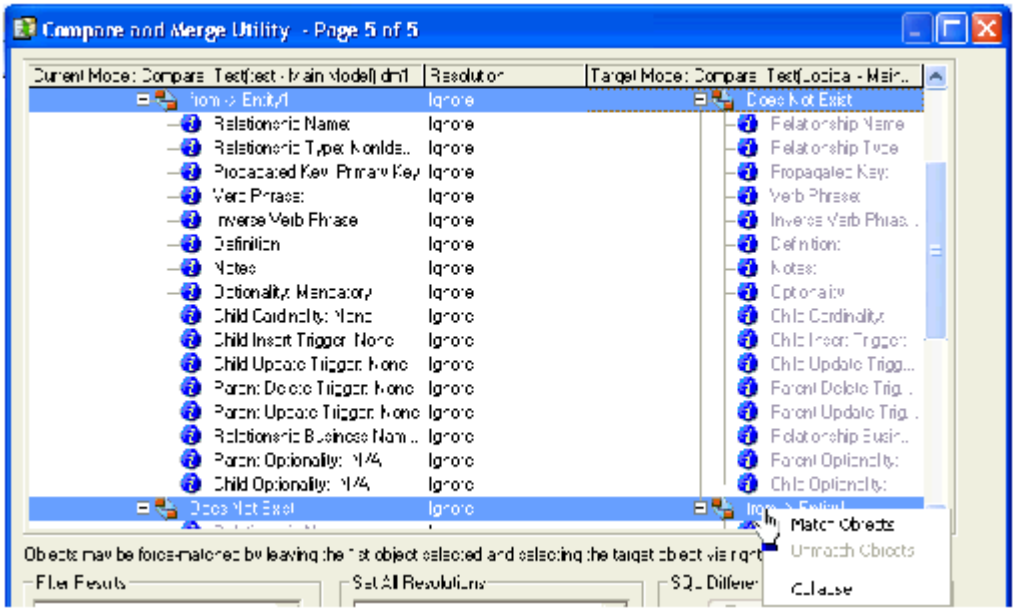
8. To force match Entity1 to its counterpart, in the Current model tree, select Entity1, scroll down to find Entity1 in the Target model tree, right click it and select *Match Objects*.



9. Now that the parent tables are matched, you can also match the relationships and the foreign key columns.

Note:

- Relationships must be matched before you can match columns.
- Fully expand the relationship or column before force matching them.



- You can unmatch objects by selecting one of the matched objects, right-clicking the object it is matched to, and then selecting *Unmatch Objects*.

Updating an Offline DBMS

As a result of comparing a physical model with either another SQL file or a live database, you can create an ALTER SQL script that you can run later to update your DBMS.

1. In the *Data Model Explorer*, click a physical model and then choose Model > Compare and Merge Utility.
2. On the first page of the utility, choose to compare the physical model with either an SQL file or a live database and then choose *Generate.sql for changes merged to target model*.
3. Proceed through the successive pages of the utility.
    - For information on these pages, see Comparing Models and Submodels.
4. To resolve the differences found on the results page, click in the *Resolution* column next to the object to resolve. You can choose to Ignore the differences. If the object exists in both the source and target model you can select either *Merge into Target* or *Merge into Current* ; if the object does not exist in the source or target model you can select *Delete from Target* or *Delete from Current* respectively, and then click *Next*.
    - You must chose to merge at least one of the changes found into the target sql or live database to generate ALTER SQL.
5. On the last page of the utility, provide a file name for the sql script and then choose the *Rebuild Options* and Pre/Post SQL Options you want.
6. Click *Finish*.
7. Run the ALTER SQL at your leisure to update the offline DBMS.

Updating a Model or Submodel

As a result of comparing two models or submodels, you can update a model in the current diagram or in another.DM1 file. The model updated can be either the target or source model. You can also update the source and the target model by merging some changes into the target and others into the source model. Changes are not applied until you click Finish.


Note: To synchronize two submodels where the target model does not have relationships, you must first use the Compare and Merge utility to bring across any relationships from the source to the target submodel and then use the Submodel Synchronize utility to synchronize the submodels.

1. In the Data Model Explorer, right-click a model and then click **Compare and Merge Utility**.
2. On the first page of the utility, choose to compare the model with either another model in the open file or with a model in another file.
3. Proceed through the successive pages of the utility until the last page.
4. For information on these pages, see Comparing Models and Submodels.

5. To resolve the differences found on the results page, click in the *Resolution* column next to the object to resolve. You can choose to Ignore the differences. If the object exists in both the source and target model you can select either *Merge into Target* or *Merge into Current* ; if the object does not exist in the source or target model you can select *Delete from Target* or *Delete from Current* respectively.
    - The default resolution is Ignore. Unless you change the resolution, the models being compared will not be updated when you click Finish.
6. *If you are satisfied with the resolutions you have designated*, click *Finish*. The current and target models are updated as requested.

If you do not want to change the models right now, click *Cancel*. You can run the **Compare and Merge Utility** later and implement the changes at that time.

Resolving and Reporting on Model Differences

The last page of the Compare and Merge Utility presents the differences found during the model comparison. You can modify the default display using the options at the bottom of this page in the utility.

1. Follow the procedure outlined in Comparing Models and Submodels.
2. Proceed through the successive pages of the utility. For information on these pages, see Comparing Models and Submodels.
3. To resolve the differences found on the results page, click in the *Resolution* column next to the object to resolve. You can choose to Ignore the differences. If the object exists in both the source and target model you can select either *Merge into Target* or *Merge into Current* ; if the object does not exist in the source or target model you can select *Delete from Target* or *Delete from Current* respectively.
4. The default resolution is Ignore. Unless you change the resolution, the models being compared will not be updated when you click Finish.
5. Create a report of the differences and your proposed resolutions by clicking *Filter Report on Results*.
6. On the *Merge Report* dialog, choose to create an HTML or an RTF report.
7. Type the *Report Title* name and then type the *Report Filename* path or browse to locate the path.
8. Click *OK* to create the report.
9. *If you are satisfied with the resolutions you designated*, click *Finish*. The current and target models are updated as requested.

If you do not want to change the models right now, click *Cancel*. You can run the **Compare and Merge Utility** later and implement the changes at that time.

See Also

- Synchronizing Physical and Logical Models
- Saving and Using Quick Launch Settings
- Documenting Data Extraction, Transformation, and Load

Changing the Database Platform of a Model

ER Studio Data Architect lets you change the database platform of your physical model. When initially designing your database, you selected a database platform. Your choice of database platform can have a profound effect on development and administration issues. For example, the supported datatypes can differ drastically between database platforms, although ER Studio Data Architect does convert them. In addition, indexing strategy, along with data storage and query optimization can differ greatly between different databases for the exact same physical model.

If you decide later in the design process that you need to change your database platform, you can use the Change Database Platform command.

- ER Studio Data Architect supports all major database platforms. Also, it facilitates the translation between different database platforms with its datatype and DDL conversion facilities.

For a list of supported database platforms, see Supported Database Platforms.

Note: If you change to a database platform that does not support the schema objects in the selected model, you will lose those objects. For example, packages are supported in Oracle but not in IBM DB2 for OS/390. In this case, changing the Oracle model results in package definitions being lost.

1. In the **Data Model Explorer**, click the physical model you want to change to a different database platform.
2. Choose Database > Change Database Platform.
3. Select the database platform and version and then click *OK*.

Notes

- ER Studio Data Architect supports all major database platforms. Also, it facilitates the translation between different database platforms with its datatype and DDL conversion facilities.
- For a list of supported database platforms, see Supported Database Platforms.
- If you change to a database platform that does not support the schema objects in the selected model, you will lose those objects. For example, packages are supported in Oracle but not in IBM DB2 for OS/390, so in this case changing the Oracle model to a DB2 model results in the package definitions being lost.

See Also

- Reverse Engineering an Existing Database
- Creating and Editing DataBases

Common Tasks

This section describes the following tasks:

- Using Toolbar Tools
- Creating a Data Model Object
- Moving Objects
- Copying Objects
- Resizing Objects
- Changing Model Alignment, Distribution, and Layout
- Finding an Entity, Table or View
- Locating an Object in the Data Model Window
- Editing an Object
- Adding and Editing Object Comments
- Editing the Name of an Object
- Deleting an Object
- Renaming an Object
- Searching for and Changing Object or Object Attribute Names
- Creating and Editing Database Views
- Creating and Editing Submodels
- Establishing Database Security
- Comparing Models
- Customizing the Data Model
- Checking Out a Concurrent License for Offline Usage

See Also

- Best Practices
- Creating and Working With Data Models

Using Toolbar Tools

For many procedures, instead of clicking a menu and selecting an option, you can click a tool on one of the toolbars to accomplish the same task.

For example, to insert an entity you can click the Entity tool on the Data Model Window. You can obtain the **Entity tool** using either of the following methods:

Choose Insert > Entity.

OR

Click the **Entity** tool .

When you pause the cursor over any tool, the name of the tool displays as follows: 

See Also

- Overview
- Creating a Data Model Object

Creating a Data Model Object

The Modeling toolbar provides easy access to tools you can use to drag and drop entities and establish the relationships between them, and also create views. The tools available on the Modeling Toolbar depend on whether the data model you selected in the Data Model Explorer is logical or physical, relational or dimensional, and the designated platform for the physical models. Not all the Modeling tools create objects that can be represented in the Data Model Window; some tools like the User and Role tools launch editors that guide you through the process of creating an object of that type. Users and Roles are visible in the Data Model Explorer.

1. From the *Modeling Toolbar*, select an object tool.
   
   The cursor changes to an icon representing the object tool.
2. In the *Data Model Window*, click where you want to place the object.

Some object tools, like the entity or view tools, create an object in the Data Model Window; others, such as the function or user tools, bring up an editor where you specify the object attributes.

See Also

- Developing the Logical Model
- Developing the Physical Model

Moving Objects

ER Studio Data Architect lets you move data model objects to and from the Data Model Explorer and Data Model Window. ER Studio Data Architect provides easy visual maintenance of your entity-relationship models, by letting you move, copy and paste, drag and drop, resize, and align all diagram objects. You can move individual or groups of diagram objects anywhere in your data model even if your model is in a specific diagram auto layout.

You can also move entire submodels to other models in the Data Model Explorer. ER Studio Data Architect also lets you move objects between logical and physical models.

Using ER Studio Data Architect you can also resize any diagram object. When you add entities, tables, or views to your diagram, ER Studio Data Architect automatically determines the proper dimensions of the object to fit the text that it displays. You can manually resize these diagram objects to control the presentation of your data models. If you need to hide attributes in an entity or view, you can resize the entity or view to be smaller than the width of the longest attribute, therefore hiding text.

This section covers moving objects in the Data Model Window, in the Data Model Explorer, and from the Data Model Explorer to the Data Model Window. It is comprised of the following topics:

- Moving Objects in the Data Model Window
- Moving Data Model Objects in the Data Model Explorer

Moving Objects in the Data Model Window

To make your data model more readable, you can move entities, tables, and the relationship lines that join these objects. You can move individual or groups of objects anywhere in your data model even if you have chosen a specific auto layout for the model.

- Clicking an Auto-Layout option repositions the objects according to the rules of the Auto-Layout organization selected. For more information, see Changing Data Model Layout.
- Moving Data Model Objects
- Moving Relationship Docking Points

Moving Data Model Objects

1. On the *MainToolbar*, click the *Selection* tool.
2. In the *Data Model Window*, select and then drag the object to the desired location.
3. Select multiple entities or tables by dragging the selection tool across them or pressing and holding CTRL while you click each object. Once the objects are selected, the cursor changes to 

- The relationship lines joining the objects automatically adjust as the objects are moved.

Moving Relationship Docking Points

1. On the *MainToolbar*, click the *Selection* tool.
2. In the *Data Model Window*, click the relationship line you want to change.
3. Move the cursor over the docking point of the relationship until the cursor icon changes to 
4. Click and drag the relationship docking point to the desired location on the same entity or table.
5. You cannot change the relationship between entities or tables by moving the relationship line. To change entity or table relationships, see Working with Relationships.

Moving Data Model Objects in the Data Model Explorer

- Moving Objects to Different Nodes
- Moving Submodels

Moving Objects to Different Nodes

You can move objects, such as entities and attributes between the main logical model and its submodels within the Data Model Explorer.

1. On the *Main* toolbar, click the  *Selection* tool.
2. In the *Data Model Explorer*, select the objects you want to move.
3. Drag to the objects to a new target node and then release the left mouse button.
4. If the move is permitted, your cursor will change to

Moving Submodels

To assist you in analyzing your data model, you can move submodels from one model to another. You can also move submodels to nest them within other submodels.

1. On the *Main* toolbar, click the *Selection* tool.
2. In the *Data Model Explorer*, click the submodel you want to move and then drag it to another model or submodel.
3. When the submodel selection is over an item it can be moved under, the cursor changes to 
4. Release the mouse button.

See Also

- Copying Objects
- Resizing Objects
- Changing Data Model Layout

Copying Objects

You can create copies of objects such as entities, tables, users, roles, shapes, functions, procedures, and schemas within the data model or between different data models.To copy and paste between different data models, you must have both models open in ER Studio Data Architect. To resolve any referential integrity issues that arise from copying and pasting diagram objects, ER Studio Data Architect applies several basic rules when copying:

- *Relationships*: ER Studio Data Architect only copies relationships that are within the set of model objects chosen. In other words, any relationships connected to entities outside of the selected set will not be duplicated. One important effect of this rule is that copied entities lose inherited foreign keys if you do not also include their parent entities.
- *Entity Types*: Because ER Studio Data Architect can drop relationships due to the rule above, ER Studio Data Architect determines whether entities are independent or dependent based on their status within the selected set.
- *Unique Names*: To preserve unique names, ER Studio Data Architect compares the names of objects being pasted with the names of objects already located in the target mode. If there are duplicates, ER Studio Data Architect serializes object names as "ObjectNameN," where N is a sequential number attached to the base name.

This section describes the following ways to copy objects:

- Copying an Object in the Data Model Window
- Copying Multiple Objects
- Copying an Attribute
- Copying an Object from the Data Model Explorer to the Data Model Window
- Copying Objects Between Models and Submodels

Copying an Object in the Data Model Window

1. On the *Main* toolbar, click the selection tool.
2. In either the *Data Model Explorer* or the *Data Model Window*, click the object to copy and then press CTRL+C.

A copy of the object is created with the same attributes as the original. The name of the new object will be the name of the original object appended by 1, such as Entity1. If you create another copy of the object, the new object would be named Entity2. The copied object can be pasted into another model or submodel in the same file or in another open diagram.

Copying Multiple Objects

1. On the *Main* toolbar, click the selection tool.
2. *To select objects that are close in proximity to each other*, in the *Data Model Window*, click and drag the mouse to draw a lasso around the objects and then release the mouse

button. In this way, you can also select the relationships associated with the selected objects.

To select objects that are not close to each other, in either the *Data Model Explorer* or the *Data Model Window*, press and hold CTRL while clicking each object.

To select all entities or tables in a model or submodel, in the *Data Model Explorer*, click the model or submodel, expand the object, and then press CTRL+A.

A copy of the object is created with the same attributes as the original. The name of the new object will be the name of the original object appended by 1, such as Entity1. If you create another copy of the object, the new object would be named Entity2.

Copying an Attribute

1. On the *Main* toolbar, click the attribute selection tool.
2. In the *Data Model Window*, select the attribute to copy, press CTRL+C, click the target entity, and then press CTRL+V.
3. You can select multiple objects by right-clicking in the Data Model Window and then dragging the mouse to draw a lasso around the diagram objects and then releasing the mouse button. You can also select multiple objects by pressing and holding CTRL while selecting each object individually.

A copy of the object is created with the same attributes as the original. The name of the new object will be the name of the original object appended by 1, such as Entity1. If you create another copy of the object, the new object would be named Entity2.

Copying an Object from the Data Model Explorer to the Data Model Window

1. On the *Main* toolbar, click the selection tool.
2. In the *Data Model Explorer*, click the object you want to copy and drag it to the *Data Model Window*.
3. You can select multiple objects by pressing and holding CTRL while selecting each object individually.

A copy of the object is created with the same attributes as the original. The name of the new object will be the name of the original object appended by 1, such as Entity1. If you create another copy of the object, the new object would be named Entity2.


Note: You can select multiple objects by pressing and holding CTRL while selecting each object individually.

Copying Objects Between Models and Submodels

The following table describes the possible copy operations within the Data Model Explorer.

Data Model Objects	Copy Data Model Object From... to...			
	Logical to Physical	Physical to Logical	Main logical or submodel to logical submodel	Main physical or submodel to physical Submodel
Attributes	no	n/a	yes	n/a
Columns	n/a	no	n/a	yes
Entities	yes	yes	yes	n/a
Functions	n/a	no	n/a	yes
Procedures	n/a	no	n/a	yes
Roles	yes	no	yes	no
Schemas	n/a	no	n/a	yes
Tables	n/a	yes		yes
Users	no	no	yes	yes
Views	yes	yes	no	no

You can copy objects such as attributes, columns, entities, views, users, roles, and shapes from a logical model to a physical model and between models and submodels. You can copy functions, procedures, and schemas from one physical model to another and from one physical submodel to another physical submodel.

1. On the *Main* toolbar, click the *Select* tool.
2. In the *Data Model Explorer*, click the object to copy and drag it to the target node.
3. You can select multiple objects of the same type by pressing and holding CTRL while selecting each object individually.

The cursor changes to this symbol when the mouse is positioned over a valid target for pasting an attribute or column to another entity or table respectively.

The cursor changes to this symbol when the mouse is positioned over an invalid target for pasting an object.

.The cursor changes to this symbol when the mouse is positioned over a valid target for pasting an entity, table, or view to a submodel within the same model or to a different model in the Data Model Explorer.

See Also

- Moving Objects
- Resizing Objects

Resizing Objects

When you add entities, tables or views to your data model, ER Studio Data Architect automatically determines the optimal dimensions of the object to fit the text and display the text of the object. However, you can resize data model objects such as entities, shapes, tables, text blocks, and views to control the presentation of your data models. For example, if you need to hide attributes in an entity or view, you can resize the entity or view to be smaller than the width of the longest attribute, therefore hiding text.

- You cannot resize the Title Block of a data model.
- Manually Resize Data Model Objects
- Automatically Resize Data Model Objects to Fit Text

Manually Resize Data Model Objects

You can manually resize entities, shapes, tables, text blocks, and views to control the presentation of your data models.

1. In the *Data Model Window*, select the object you want to resize.
2. Place the cursor over one of the handles, and then click and drag to resize the object.

Automatically Resize Data Model Objects to Fit Text

ER Studio Data Architect can automatically resize entities, tables, and views to display the entire object name and displayed contents.

1. In the Data Model Window, select the object you want to resize.
2. Choose Format > Resize Entity/View.

The object is resized to accommodate the full width of the text.

If you want to adjust the object size manually, grab a sizing handle and reposition the object boundaries.

- You can also resize the selected object to fit the text by pressing *CTRL+R* or by right-clicking the object and then selecting Resize.

See Also

- Copying Objects
- Moving Objects

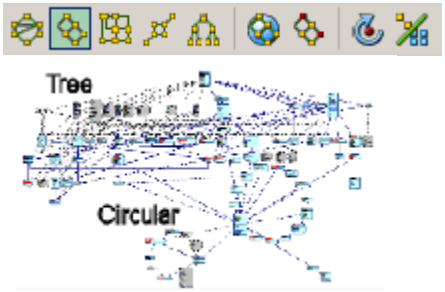
Changing Model Alignment, Distribution, and Layout



To align a group of objects in a straight line, select the objects and then click one of the alignment buttons.



To distribute a group of objects, select the objects and then click an alignment button.



To rearrange a data model or task diagram using a different layout type, display the model, and then click one of the auto-layout buttons on the Layout & Alignment Toolbar. *To apply a layout type to a selected items only*, you can select multiple objects (entities, views, and shapes) and apply a layout format to that selection only. All relationships, view relationships or lines that belong to the selected objects will be also affected by the layout. This allows you to choose a different layout style to draw attention to specific parts of the model, such as objects recently added or to show a special relationship that exists between certain entities. If you do not have any objects selected, the layout style chosen will apply to the entire model.

Note: To change the layout defaults, choose Layout > Layout Properties. Changes affect the current project file only. For more information on changing Layout Properties, see Changing Data Model Layout.

See Also

- Changing Data Model Layout
- Resizing Objects

Finding an Entity, Table or View

1. Choose Edit > Find Entity/View.
2. Enter the first letter or first few letters of the entity or view you want to find.
   The first matching entity or view found will be highlighted in the list.
3. If necessary, scroll through the list to find the object you are looking for or type more letters to further limit the scope of the search.
4. Once you have found the object you are searching for, double-click the entry.
   ER Studio Data Architect highlights the object in the *Data Model Explorer* and the *Data Model Window*.
5. You can also use the Universal Naming Utility to find strings in various objects, enabling you to for example find all entities with an attribute named city. For more information on the Universal Naming Utility, see Searching for and Changing Object or Object Attribute Names.

See Also

- Searching for and Changing Object or Object Attribute Names
- Locating an Object in the Data Model Window

Locating an Object in the Data Model Window

You can use this procedure to locate objects such as entities, tables, attributes, columns, indexes, procedures, keys, views, relationships, foreign keys, functions, procedures, and shapes. Some objects, such as users and roles, are not represented in the Data Model Window and so cannot be found using this method.

Find and click an object in the *Data Model Explorer*.
ER Studio Data Architect highlights the object in the Data Model Window.

- When searching for a foreign key or relationship, ER Studio Data Architect focuses the Data Model Window on the parent and the relationship or foreign key is highlighted so you can follow the line to the child table.

See Also

- Resizing Objects
- Finding an Entity, Table or View

Editing an Object

On the Data Model Explorer or Data Model Window, double-click the object that you want to edit.
An object-specific editor displays where you can change the properties of the object selected.

- Alternatively, you can open the editor by selecting the object and then choosing Edit > Edit object type.

See Also

- Deleting an Object
- Renaming an Object

Adding and Editing Object Comments

You can add comments to many database objects for your own documentation needs or for communicating your ideas when collaborating with others to develop a model. Repository users can share comments between ER Studio Data Architect and ER/Studio Team Server, so if a ER/Studio TS user makes a comment on a model, submodel, or database object, ER/Studio users can view these comments and vice versa. Comments are stamped with the date and the name of the logged in user.

1. On the Data Model Explorer or Data Model Window, right-click the object you want to add comments to and then select Comments.
    - Comments can be added to objects in the Repository without needing to check out the object.

    The Comments editor displays where you can view or delete comments previously recorded or add new comments.
    Comments can be added to objects in the Repository without needing to check out the object.
2. Enter the comment in the bottom area of the *Comment* editor and then click *Add*.
3. If you want the Repository to be updated immediately with the new comments, select *Synchronize All Comments In Repository Immediately*.
4. Add more comments if desired and then click *Save*.

Note: If you are adding comments to a Repository diagram, you can add and commit comments to an object without needing to check out or check in the diagram. Click the *Synchronize Comments* button in the Repository toolbar and your comments are added to the diagram, with your Repository user name and the comment date and time.

You can add comments to most ER Studio Data Architect objects, including the following object types:

• Alias	• Attachment Binding	• Attachment Types	• Attachments
• Attribute	• Attribute	• Auxiliary Table	• Column
• Data Lineage Column	• Data Lineage Source	• Data Lineage Table	• Data Movement Column
• Data Security	• Database	• Diagram	• Dictionary
• Dictionary Default	• Dictionary Rule	• Domain	• Entity
• Function	• Logical Model	• Logical Submodel	• Package and Object Type
• Physical Model	• Physical Submodel	• Procedure	• Reference Value
• Relationship	• Security	• Security Binding	• Stogroup
• Synonym	• Table	• Tablespace	• User Data Type
• View	• View Column	• Visual Data Lineage Objects	

- For ER/Studio Repository users, when logged in to the Repository, any comments added are stamped with the Repository user name and date.
- Comments are stamped with the user name and comment creation date and time.
- You can delete the comment by clicking the X for the comment in the Comments dialog.
- Once logged in to the Repository, you can share your comments with other ER Studio Data Architect users by storing them with the diagram file in the Repository.
  To upload your comments to the Repository, choose Repository > Update Comments.
  To download existing or new comments in the open diagram, choose Repository > Synchronize Comments. Access the comment editor via the comments tool on the modeling toolbar, though the context menu, or by choosing Edit > Comments.
- You can add a comment to one object at a time. If more than one object is selected the *Comments* option will not be available.

See Also

- Editing an Object
- Editing the Name of an Object

Editing the Name of an Object

On the Data Model Explorer or Data Model Window, double-click the object that you want to edit and then change the name of the object in the editor.

**Note:** Alternatively, shift-click the object to change and the object name becomes an editable field. Click Backspace to erase the name, enter a new name, and then click anywhere outside the object to effect the change.

See Also

- Deleting an Object
- Editing an Object

Deleting an Object

1. On the Data Model Explorer or Data Model Window, right-click the object to delete, and then click *Delete* from the shortcut menu.
2. Click *Yes*.

Notes

- When you right-click a submodel object, there is no Delete option. Instead, there is an option to Remove. When you remove an object from a submodel, the object remains in the Main model unless you select Delete from Model from the dialog that appears.
- To select multiple entities, press and hold CTRL while clicking the entities.
- To delete an attribute, select the attribute in the Data Model Explorer or double-click the parent entity or table in the Data Model Window to bring up the entity or table editor. In the editor, you can select the attribute to delete and then click Delete.
- When deleting a relationship, you can either delete the relationship entirely or, by selecting Make Foreign Keys Native in the dialog that displays, you can choose to convert the foreign key relationship to an attribute of the child entity or table.

See Also

- Renaming an Object
- Editing an Object

Renaming an Object

1. On the Data Model Explorer, right-click the object that you want to rename.
2. From the shortcut menu, click *Modify Name*.
3. Enter the new name and press *Enter*.
4. To rename an attribute, double-click the object or its parent in the Data Model Window to bring up the entity or table editor. In the editor, you can click the attribute and then click *Edit*.

See Also

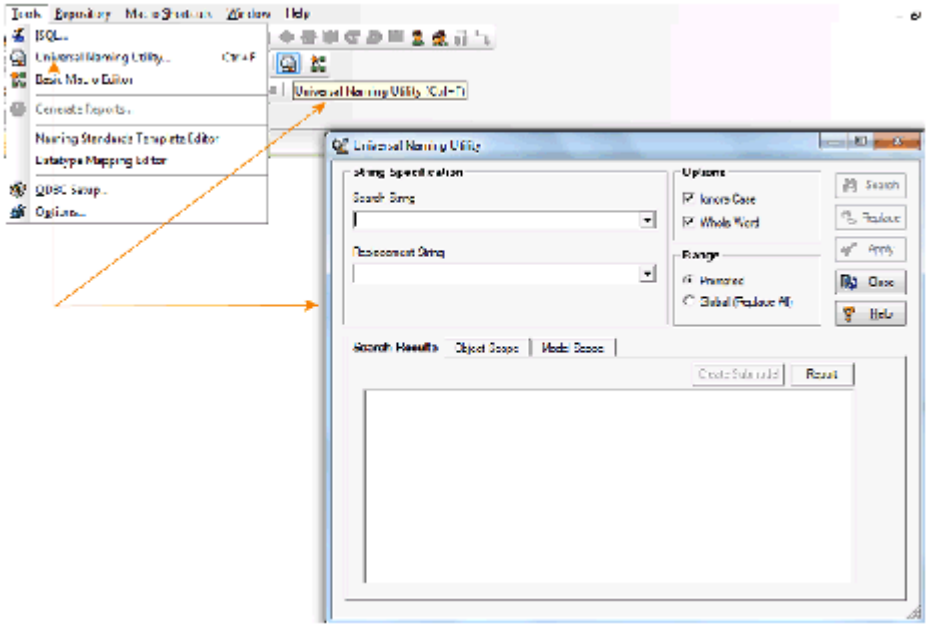
- Deleting an Object
- Editing an Object

Searching for and Changing Object or Object Attribute Names

Using the Universal Naming Utility you can globally search, and replace if desired, names, strings, and attachment value overrides for bound attachments for both your logical and physical models. You define your search, then confine the search to specific objects within your models, or confine the search to specific models only. You can also narrow the search to include only specified objects or object properties or to include only certain models. The results can include model objects such as attachment bindings and table storage parameters.

On the *Search Results* tab of the Universal Naming Utility (UNU,) you can double-click on an object and open the editor for that object. In this process, you are asked to save any changes you made in the UNU; the UNU closes; the object is located and selected in Data Model Data Dictionary, or Data Lineage tree; and the object editor opens with the object selected.

1. Choose Tools > Universal Naming Utility.



2. Complete the *String Specifications*.
3. You can limit the scope of the search or search and replace using the Object Scope and Model Scope tabs.
4. *To search for a string*, click *Search*.
   To replace a string, click *Replace*. Examine the *Search Results* and then click *Apply* to implement the changes.
   - Changes are not implemented until you click Apply.
5. *To create a report of the search results in RTF or HTML format*, click *Report*.

To create a submodel of selected results, select objects in the results window and then click *Create Submodel*.

Tip: RTF-formatted reports are ideal for print distribution, while HTML-formatted reports are better-suited for browser viewing.

The following describe options that require additional explanation.

Search Results tab

Whole Word: Searches for whole words in the search string. For example, if your search string is the word "data," check the Whole Word check box if you do not want results such as the word "database."

Object Scope tab

Lets you narrow your search to specific objects and to specific string usages within the selected objects.

Model Scope tab

Lets you restrict the search to specific models within the diagram.

See Also

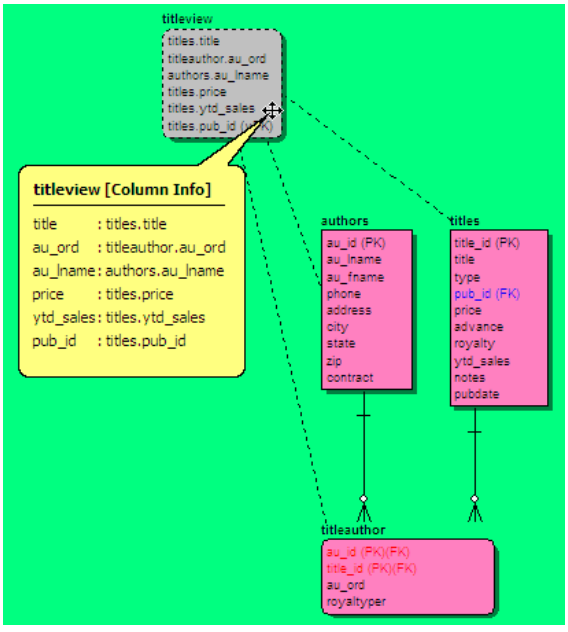
- Creating and Editing Submodels
- Resizing Objects

Creating and Editing Database Views

A database view, also known as a materialized view, is a dynamic virtual table that reflects changes made to the parent entity or table data. In SQL, a view is the set of columns or attributes resulting from a *SELECT* statement. It is not part of the physical schema of a relational database and does not incur any extra storage overhead.

Some of the uses for views include:

- Presenting a subset of the data included in a table.
- Joining and simplifying tables into a single virtual table.
- Hiding the complexity of data.
- Depending on the database platform, providing an additional layer of security by restricting user access to specific columns and/or rows.
- Acting as aggregated tables, where aggregated data such as sum and average, are calculated and presented as part of the view.
- Limiting the amount of exposure of sensitive data in the tables.
- Creating a layer of abstraction to simplify complex data, transparently partitioning the actual underlying table.
- Facilitating database normalization beyond second normal form.
- Making it easier to decompose the underlying table and then join the parts without losing any information.
- Some database platforms offer more properties than others that can be applied to physical view definitions.



In the View Editor you can select specific attributes from entities on the Entities tab or specific columns from tables on the Tables tab to include in the view. In the View Editor, you can reorganize the column order in the view and add notes and descriptions to your view. You can also add Pre and/or Post SQL that generate immediately prior to or after the CREATE OBJECT statement.

1. In the *Data Model Explorer*, expand the node of the model to which you want to create a view.
2. Right-click the *Views* node and then click *New View*.
3. Complete the tabs of the *View Editor* and then click *OK* to implement the changes to the view.

The following will help you complete the View Editor:

Notes	Entity/Table tab	Attribute/Column tab
Column Alias dialog	Where tab	Group By tab
Having tab	DDL tab	SQL Validation dialog
Options tab	Definition tab	Notes tab
Where Used tab	Dependencies tab	PreSQL & PostSQL tab
Permissions tab	Security Information tab	Attachment Bindings tab

Notes

- You can reference the same table twice in a view, doing so automatically creates an alias for the parent table that is replicated more than once in the view.
- You can also create a view by selecting the tables in the Data Model Window that you want to include in the view and then choosing Insert > View. Clicking anywhere in the Data Model Window will create a new view that is comprised of the tables you selected. Double-click the view to edit its properties.
- After you have created the view, you can edit it by right-clicking the view and then clicking Edit View from the shortcut menu.
- The status bar at the bottom of the screen shows the total number of views in the selected model or submodel.
- In the Diagram Explorer tree you can see the views created both under the Views node and also by expanding the relevant entity or table node.
- Views that use the attributes/columns of the current entity/table are displayed on the Dependencies tab of the entity or table editor.

The following describes options in the View Editor that require additional explanation.

Entity/Table tab

- *Logical Only*: In a logical model, if selected, the view will not be implemented when generating a physical model and will be ignored when comparing the logical model with a physical model.
- *Physical Only*: In a physical model, if selected, the view will be ignored when comparing the physical model with a logical model.

Attribute/Column tab

- *View Columns/Attributes > Column Expression*: This is a standard SQL expression that creates a computed column, combining existing data in order to produce data that you do not need to store but want to have available. For example, in the Northwind model, the Product Sales for 1997 view uses the following column expression to calculate the value of sales using the unity price, quantity sold and the applicable discount.
*Sum(CONVERT(money,("Order Details".UnitPrice*Quantity*(1-Discount)/100))*100*
A column expression can only be added when no column is specified and requires an alias.
- *Select distinct*: If selected, adds a *SELECT DISTINCT* statement to the View DDL, which selects and shows only unique rows.
- *Include view column list in DDL*: If selected, includes the columns selected on this tab in the DDL created for the View. For example, when this option is not selected the following DDL is created.

CREATE VIEW View1 AS

SELECT En.p p, En.a a, En.b b, Ent.k k, Ent.z z, Ent.y y, Ent.p f

FROM Entity1 En, Entity2 Ent

WHERE Ent.p = En.p

When *Include view column list in DDL* is selected, the following DDL is created.

CREATE VIEW View1

(p, a, b, k, z, y, f) AS

SELECT En.p, En.a, En.b, Ent.k, Ent.z, Ent.y, Ent.p

FROM Entity1 En, Entity2 Ent

WHERE Ent.p = En.p

Column Alias dialog

This dialog appears if you try to add columns with duplicate names to a view. To use the columns in the view, you must assign an alias to each duplicate column name. If an alias is not assigned to the duplicate column name, an SQL syntax error will result when generating the view. Note that a column expression can only be added when no column is specified. So you may have a column with an alias, or an expression with an alias. The expression requires an alias.

Where tab

Define a WHERE clause of the view, such as *WHERE [city] = "Monteray"*, which selects rows matching the stated criteria.

Group By tab

Select columns and tables to include in a GROUP BY clause for the view, such as
SELECT Product, SUM (Sales) GROUP BY Product
which allows you to easily find the sum for each individual group of column values.

Having tab

Enter a HAVING clause for the view, such as,
SELECT Product, SUM (Sales) GROUP BY Product HAVING SUM (Sales) > 10000
which allows you to test for result conditions.

DDL tab

Displays and allows you to edit the *CREATE VIEW* statement needed to build the view. In physical models, ER Studio Data Architect uses the platform-specific parser of the database platform of the selected model to generate the view. If you are adding a view to a logical data model, the default parser AINSI SQL is used unless you change the View Parser in the Logical Model Options to another parser. For more information, see Defining Model Options for the Selected Model.

SQL Validation dialog

If any errors are detected, they are displayed in the Validation Results area. You can then exit the SQL Validating dialog, correct the errors on the DDL tab and then rerun the validation check.

Options tab

When selected, the With Check Option and With Read Only options add a *WITH CHECK OPTION* and *WITH READY ONLY* clause, respectively to the DDL statement.

Definition tab

Enter or edit a definition for the view. If the target database supports it, ER Studio Data Architect adds this definition as a view comment when generating SQL code.

Notes tab

Add notes about the view that you can format using HTML tags that are processed when creating an HTML Report of the model.

Where Used tab

Shows the models and submodels where the view is used according to definitions you set.

Dependencies tab

Create, edit or delete any scripted procedures that are bound to the view. If any procedures or functions are bound the view, the folder is expandable. Double-click or right-click the view folder to access the view editor. For information on creating procedures see Create an SQL Procedure and for information on creating functions, see Add a Stored SQL.

PreSQL & PostSQL tab

Enter SQL to be applied before or after the *CREATE OBJECT* statement. The PreSQL and PostSQL scripts entered here are included in the script when you generate the physical database.

Permissions tab

Sets access roles and user permissions for the view. Keep in mind that the more specific the permissions are, the more time you may have to spend maintaining and updating them. The Roles and Users must be previously assigned to appear in the View Editor. For more information, see Creating and Editing Database Roles and Creating and Editing Database Users.

Security Information tab

Displays the current security information as defined in the Data Dictionary. You can add or edit security reporting information in by double-clicking the field you want to change. To set the model-wide default value for a property, go to the Data Dictionary tab and expand the Data Security Information Node. For more information, see Enforcing Security Using Data Security Types and Properties.

Attachment Bindings tab

Binds an external piece of information or attachment to the database view. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect displays the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary and must be applied to the view before they will display on this tab. For more information, see Attaching External Documents to the Data Model.

See Also

- Common Tasks
- Creating and Editing Submodels

Defining the Default View Settings for New Models

1. Choose Tools > Options, and then click the *Display* tab.
2. Complete your setting selections in the *View Display Settings* area.
3. Click the *View* tab.
4. Complete your setting selections and then click *OK*.
5. You can override the settings here for a specific model or submodel using the options in View > Diagram and Object Display Options > View.

The following describe options that require additional explanation:

View Reverse-Engineer/Import > Import Invalid Views: Lets you import invalid views when importing a data model or reverse-engineering a database.

Duplicate View Column Name Resolution

- *Prompt*: Prompts for column aliases to differentiate between columns whenever duplicate column names exist.
- *Don't propagate*: Eliminates duplicate column names from the SELECT clause of the SQL statement generated to create the view.
- *Attach prefix and/or suffix*: Appends a prefix or suffix to the column names to differentiate between duplicate column names. The prefix is comprised of the entity name and an underscore character. The suffix is comprised of sequential numbering.

View Column Propagation > Don't propagate any columns: Does not automatically propagate any columns to a view. You can select the columns to include in the View Editor.

Column Sorting: Sets column sorting hierarchy.

- The Column Sorting setting determines the column order in the DDL SELECT statement.

See Also

- Common Tasks
- Creating and Editing Submodels

Customize the View Settings for a Specific Model

Use the View tab of the Diagram and Object Display Options to control which components of the views in a specific model are displayed.

- The settings here override display settings configured in Tools > Options > Display.

1. Choose View > Diagram and Object Display Options and then click the *Views* tab.
2. Complete your settings selections and then click the *Apply To* tab.
3. Select the models and submodels to apply the new settings to and then click *OK*.

See Also

- Overriding Color and Font Settings for a Specific Object
- Creating and Editing Materialized Views

Creating and Editing Submodels

ER Studio Data Architect lets you create independent views of all or part of your logical or physical models called submodels. Submodels can focus attention on specific characteristics of a model and can facilitate faster data access. Any changes to submodel objects, except object deletions, automatically propagate to the main model. When deleting a submodel object, you have the option to apply the change to the main model or to only delete the object from the submodel. You can create any number of submodels within your data model and can nest any number of submodels within a submodel.

In addition to creating submodels of models, with ER Studio Data Architect you can derive a submodel from a submodel and then display the submodel relationships by nesting submodel folders in the Data Model Explorer.

ER Studio Data Architect automatically includes relationships between objects if related objects are part of the submodel. You can edit the submodel to include or exclude specific relationships. When you add a relationship, ER Studio Data Architect automatically includes the corresponding parent table in the submodel definition.

1. On the Data Model Window or Data Model Explorer, select one or more objects to include in the submodel.

   **Tip:** CTRL-click to select multiple objects.

2. Choose Model > Create Submodel.
3. In the *Create* Submodel dialog, enter a name for the submodel.
4. From the list of objects, *CTRL-click* to select the objects to add to the submodel and then press the right arrow to add the selected objects to the submodel.
5. Click *OK* to create the submodel.

Once you are finished creating a submodel, ER Studio Data Architect adds the submodel to the *Data Model Explorer* and displays the model in the Data Model Window.

The following describe options that require additional explanation:

Edit Submodel tab Attributes tab

Definition tab Attachment Bindings tab

Security Information tab Notes

Edit Submodel tab

- Click *Select Related Objects* to add parent and child entities and dependent views or procedural logic.
- If you select *Inherit Objects Added to Nested Submodels*, any objects you later add to submodels nested within the model or submodel will automatically be also added to the model.

Attributes tab

Lets you choose which attributes of the entities selected on the *Edit Submodel* tab to include in the submodel. If you deselect an attribute, ER Studio Data Architect can display an elipsis to indicate that not all attributes are displayed. For more information, see the Entity or Table tab description in Customizing the Display of Diagrams and Objects.

Definition tab

Enter or edit a definition for the submodel.

Attachment Bindings tab

Bind an external piece of information, or attachment to the submodel. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the *Value* field of the target attachment. ER Studio Data Architect opens the *Value Override Editor* or a list depending on the attachment datatype. Attachments are created in the *Attachments* folder of the *Data Dictionary*. For more information, see Attaching External Documents to the Data Model.

Security Information tab

Displays the current security information as defined in the Data Dictionary. You can add or edit security reporting information in by double-clicking the field you want to change. To set the model-wide default value for a property, go to the Data Dictionary tab and expand the Data Security Information node. For more information, see Enforcing Security Using Data Security Types and Properties.

Notes

- If you select a submodel to add to the new submodel, then all objects belonging to the source submodel will be added to the new submodel.
- Deleting an object from a submodel does not delete the object from the Main model unless you select the Delete from Model option in the Delete dialog.
- Once you have created a submodel, you can add objects to it by right-clicking an object in the main model or in another submodel and then selecting Apply to Submodel(s) from the shortcut menu. In the dialog that appears you can select which submodels to add the object to.}}

See Also

- Using the Submodel Synchronization Utility
- Synchronizing Submodels

Synchronizing Submodels

Use the Submodel Synchronization Utility to synchronize object positioning, color and font settings, display level, members and hierarchies of submodels within the same file or across disparate files. You can synchronize submodels between logical and physical models by adding, removing, and updating submodel items or you can synchronize submodels of the local model with a local ER Studio Data Architect file or a file in the ER/Studio Repository. Submodels in source and target models are matched by Submodel name.

- You can synchronize submodels between logical and physical models by adding, removing, and updating submodel items.

The following are examples of when synchronizing submodels can be useful:

- Example 1
- Example 2
- Example 3

Example 1

You have existing logical and physical models in a **.dm1* file, each with 20 to 30 submodels. The physical model represents the current development environment. You use the Add New Physical Model function to add a physical model of the test environment. The physical model of the test environment is close but does not exactly match the existing physical model that is the development benchmark environment. To match the test and development physical models, you can use the submodel synchronization utility to add or transfer submodels from the development model to the test model within the same or across disparate **.dm1* files.

Example 2

You have a logical and physical model within the same **.dm1*. For reporting purposes, the same submodels are used in the logical and the physical models. You make substantial changes to the logical model, adding and removing objects or adding new submodels for new projects. You can use the Submodel Synchronization Utility to synchronize or promote these changes to the physical model.

Example 3

Many people manage their submodels in the logical model and want to be able to move the structure to a physical or project model that was spun off from the Enterprise logical model. Members and display settings change often; this utility automates the process of synchronizing the submodels.

- To synchronize two submodels where the target model does not have relationships, you must first use the Compare and Merge utility to bring across any relationships from the source to the target submodel and then use the Submodel Synchronize utility to synchronize the submodels.

See Also

- Using the Submodel Synchronization Utility
- Creating and Editing Submodels

Using the Submodel Synchronization Utility

1. In the *Data Model Explorer*, select the source submodel.
2. Choose *Model* > *Submodel Synch Utility*.
3. Complete the pages of the *Submodel Synchronization Utility* and then click *Finish* to implement the changes to the view.

The following describe options that require additional explanation:

Page 1 Page 2

Page 3 Page 4

Notes

Page 1

For information on using the Quicklaunch utility, see Saving and Using Quick Launch Settings.

Page 2

Your changes to the source model can be merged into one target model. Select the target from the list or go back to the first page and use the file browser controls to load the **.dm1* for the target file. The submodels to be merged can be selected on page 4.

Page 3

- *Synchronize submodel display level*: If selected, the settings specified for the submodel on the Diagram and Object Display Options dialog will be inherited by the target submodel.
- *Synchronize submodel hierarchies*: If selected, synchronizes nested submodels.

Page 4

For each operation, select the submodels you want to synchronize with the target model. The available operations (update, delete) depend on the submodel chosen. If this is a frequently updated submodel, you may want to save all the wizard settings in a Quick Launch file. The file will load your selections when you next use the utility if you select it upon launching the utility. You can load the quick launch file and make changes to it and then save the changes if you want.

Notes

- To make changes to both the source and target models, run the utility twice, reversing the source and target the second time you run the utility.
- The available operations (update, delete) will depend on the submodel chosen and the differences that exist between the source and target submodels. For example, if the source contains updates not reflected in the target, then the update operation will be available.

See Also

- Synchronizing Submodels
- Creating and Editing Submodels

Copying a Database Object to a Submodel (Apply to Submodel)

You can replicate database objects by copying the object and then pasting it to each submodel individually or you can use the Apply to Submodel function to paste an object into multiple submodels within the same model simultaneously.

1. In either the *Data Model Explorer* or the *Data Model Window*, right-click the object you want to copy to one or more submodels in the active model and then click *Apply to Submodel*.
2. In the *Apply to Submodel* dialog, individually select the submodels you want to copy the object to, or select the model to copy the object to the main model and all submodels of the active model.
3. In the Apply to Submodel dialog, clicking Place New Object Near Existing Objects prevents the new object from being placed randomly in the Data Model Window. This can be important when working with large models and submodels.

See Also

- Using the Submodel Synchronization Utility
- Creating and Editing Submodels

Establishing Database Security

Define and establish database security to, for example, inform data model reviewers and database administrators of the security required for data model objects when these objects are implemented on a DBMS.

This section includes the following topics:

- Creating and Editing Database Users
- Creating and Editing Database Roles
- Associating Database Users with New Database Roles
- Associating Database Roles with New Database Users
- Changing Database Roles Associated with Existing Database Users
- Changing Database Users Associated with Existing Database Roles
- Granting and Revoking Permissions to Modify Database Objects

See Also

- Best Practices
- Enforcing Security Using Data Security Types and Properties
- Establishing Security for the Repository

Creating and Editing Database Users

Granting database access, authorities, and privileges to users and groups is one of the most important ways to ensure data security.

In conjunction with database roles, users and the permissions you grant them enable you to manage the permissions logical model users have to insert, select, update, delete, alter, or reference entities, tables, and views in the logical database. In addition to being supported by tables and views in the physical data model, depending on the database platform, users also can be applied to specific physical models to manage system privileges such as those required to backup databases and logs, or to create databases, defaults, procedures, rules, tables, and views. The system privileges available are database dependent and display on page 2 of the User Wizard and User Editor.

Database users are supported by the logical data model and are also supported by physical data models for the following platforms:

- IBM DB2 for LUW 5.x, 6.x, 7.x, 8.x, 9.x. 10.x
- IBM DB2 for OS/390 5.x and 6.x, and DB2 for z/OS 7.x, 8.x, 9.x, and 10.x
- Microsoft SQL Server 4.x, 6.x, 7.x, 2000, 2005, 2012, 2014
- Oracle 7.x, 8.x, 9i, 10g, 11g, 12c
- Sybase ASE 11.0, 11.5, 11.9, 12.0, 12.5, 15
- The User Wizard and User Editor share the same options, except for Attachment Bindings options which are present only in the editor.

1. In the *Data Model Window*, expand the *Main Model* of the Logical model, or of a physical model for which database users are supported, right-click the *Users* node, and then click *New Database User*.
2. In the *User Wizard*, assign the permissions desired, and then click *Finish*.
3. Options and permissions available are platform-dependant.

The following describe options that require additional explanation:

General page/tab	System Privileges page/tab
Object Privileges page/tab	Dependencies page/tab
Definition page/tab	See DDL tab
Attachment Bindings tab	Notes

General page/tab

- *User Name*: The restrictions for the number of character permitted when defining user and group names is dependent on the database platform.
- *Is Group*: Members of a user group automatically inherit group privileges. Privileges granted to a user override the privileges associated with any groups the user is a member of, and are retained if the user is later removed from the groups. Explicit privileges granted to a user must be explicitly revoked.
- *Password*: When selected, specifies that the user is a local user who must specify a password when accessing the database.
- *Externally*: When selected, specifies that the user is an external user who must be authorized by an external service, such as an operating system or third-party service, to access the database.
- *Globally*: When selected, specifies that the user is a global user who must be authorized to use the database by the enterprise directory service or at login.
- *Add*: When clicked, the Select Roles dialog appears where you can select the roles to assign to the user. Hold CTRL-Shift to select multiple roles. If you have not previously defined a role, the list will be empty.
- *Tablespace Quota*: On some physical models, you can define tablespace usage parameters (in units of 10KB). This will limit a user's ability to grow a tablespace, but should not be used instead of designed range limits.

System Privileges page/tab

Clicking the column name selects the entire column. Control-click to select multiple objects. The permissions available are dependent on the physical platform selected.

Object Privileges page/tab

Set access privileges for users for objects such as tables and views. Some database platforms also allow detailed access to database objects such as procedures, packages, materialized views, sequences, and functions. Clicking the column name selects the entire column. Control-click to select multiple objects individually.

Dependencies page/tab

If the object associated with the permissions assigned has dependencies, these will be displayed here. Dependencies are SQL code such as functions, triggers, and procedures. For more information, see Creating and Editing Functions, Creating and Editing Triggers, and Creating and Editing Procedures.

Definition page/tab

Describe why you are defining the limitations of the user. Enter or edit a definition for the user. If the target database supports it, ER Studio Data Architect adds this definition as a comment when generating SQL code.

DDL tab

View the *CREATE USER* statement needed to build the user, or the *GRANT... ON CONNECT* statement required to establish user privileges when connecting to the database. In physical models, ER Studio Data Architect uses the platform-specific parser of the selected database platform of the model to generate the view. When adding a view to a logical data model, the default parser AINSI SQL is used unless you change the View Parser in the Logical Model Options to another parser. For more information, see *Defining Model Options for the Selected Model*.

Attachment Bindings tab

Bind an external piece of information, or attachment to the user. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary and must be applied to the database user before they will display on this tab. For more information, see Attaching External Documents to the Data Model.

Notes

- In order to view the users in the Data Model Window, ensure that *Display All Security Objects* is selected in View > Diagram and Object Display Options > Security Objects. Reverse-engineered diagrams do not display security objects by default; however, models created from scratch do.
- Once you have created a user, you can assign that user a particular role in the Role Editor or Role Wizard. For more information, see Associating Database Users with New Database Roles and Associating Database Roles with New Database Users.
- You can change the permissions assigned to a user in the User Editor or on the Permissions tab of the Entity Editor or Table Editor. For more information, see Creating and Editing Entities and Tables.
- Database users created in the logical model are propagated to the physical model when you generate a physical model that supports database users.
- Depending on the database platform, database users can provide security for entities, tables, functions, materialized views, packages, procedures, and sequences.

See Also

- Creating and Editing Database Roles
- Associating Database Users with New Database Roles
- Associating Database Roles with New Database Users

Creating and Editing Database Roles

A role is a named collection of privileges. In a business setting, roles are defined according to job competency, authority and responsibility. The ultimate intent of creating database roles is to achieve optimal security administration based on the role each individual plays within the organization.

Advantages of effective role creation and application include:

- Optimally assigning roles to user privileges.
- Identifying users who operate outside the normal pattern.
- Detecting and eliminating redundant or superfluous roles or user privileges.
- Keeping role definitions and user privileges up-to-date.
- Eliminating potential security loopholes and minimizing consequent risks.

By implementing database roles for specific user groups, you can save time by assigning roles to users instead of assigning permissions to each object for each individual user. Database roles enable you to manage the permissions logical model users have to insert, select, update, delete, alter, or reference entities, tables, and views in the logical database. In addition to being supported by tables and views in the physical data model, depending on the database platform, roles also can be applied to select physical models to manage system privileges such as those required to backup databases and logs, or to create databases, defaults, procedures, rules. tables, and views. The system privileges available are database dependent and are listed on page 2 of the Role Wizard and Role Editor.

Database roles are supported in the logical model and by the following physical database platforms:

- Microsoft SQL Server 4.x, 6.x, 7.x, 2000-2014
- Oracle 7.x, 8.x, 9i, 10g, 11g and 12c

1. In the *Data Model Window*, expand the logical *Main Model* or the *Main Model* of a physical model for which database roles are supported, right-click the *Roles* node, and then click *New Database Role*.
2. In the *Role Wizard*, assign the permissions desired and then click *Finish*.
3. The options and permissions available are platform-dependant.

The following describe options that require additional explanation:

General page/tab
System Privileges page/tab
Object Privileges page/tab
Dependencies page/tab
Definition page/tab
DDL tab
Attachment Bindings tab
Notes

General page/tab

- *Role Name*: The restrictions for the number of character permitted when defining role names is dependant on the database platform.

The following options specify the specific method used to authorize a user before the role can be enables with the *SET ROLE* statement.

- *Password*: When selected, specifies that the user is a local user who must specify the password when enabling the role.
- *Externally*: When selected, specifies that the user is an external user who must be authorized by an external service (such as an operating system or third-party service) to enable the role.
- *Globally*: When selected, specifies that the user a global user who must be authorized to use the role by the enterprise directory service before the role is enabled or at login.
- *Not Identified*: When selected, indicates that this role is authorized by the database and no password is required to enable the role.
- *Add*: When selected, the Select User dialog appears where you can choose the users to assign to the role. CTRL-click to select multiple users. If you have not previously defined a user, the list will be empty.

System Privileges page/tab

Clicking the column name selects the entire column. CTRL-click to select multiple objects individually. The permissions available are dependent on the physical platform selected.

Object Privileges page/tab

Set access privileges for users for objects such as tables and views. Some database platforms also allow detailed access to database objects such as procedures, packages, materialized views, sequences, and functions. Clicking the column name selects the entire column. Control-click to select multiple objects individually.

Dependencies page/tab

If the object associated with the permissions assigned have dependencies, these will be displayed here. Dependencies are SQL code such as functions, triggers, and procedures. For

more information, see Creating and Editing Functions, Creating and Editing Triggers, and Creating and Editing Procedures.

Definition page/tab

Enter or edit a definition for the user. Usually used to describe why you are defining the limitations of the user. If the target database supports it, ER Studio Data Architect adds this definition as a comment when generating SQL code.

DDL tab

View the *CREATE ROLE* statement needed to build the user, or the *GRANT... ON CONNECT* statement required to establish user privileges when connecting to the database. In physical models, ER Studio Data Architect uses the platform-specific parser of the model's selected database platform to generate the view. If you are adding a view to a logical data model, the default parser AINSI SQL is used unless you change the View Parser in the Logical Model Options to another parser. For more information, see *Defining Model Options for the Selected Model*.

Attachment Bindings tab

Bind an external piece of information, or attachment to the role. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary and must be applied to the role before they will display on this tab. For more information, see Attaching External Documents to the Data Model.

Notes

- To view the roles you create in the Data Model Window, ensure that Display All Security Objects is selected in View > Diagram and Object Display Options > Security Objects. Reverse-engineered diagrams do not display security objects by default; however, models that you create from scratch display security objects by default.
- Once you have created a role, you can assign that role to a particular user in the User Editor or User Wizard.
- You can change the permissions assigned to a role in the Role Editor or on the Permissions tab of the Entity Editor or Table Editor. For more information, see Creating and Editing Entities and Creating and Editing Tables.
- Database roles created in the logical model are propagated to the physical model when you generate a physical model that supports database roles.
- Depending on the database platform, database roles can provide security for entities, tables, functions, materialized views, packages, procedures, and sequences.

See Also

- Creating and Editing Database Users
- Associating Database Users with New Database Roles
- Associating Database Roles with New Database Users

Associating Database Roles with New Database Users

Once you have a database role defined, you can assign that role to any new users created. Once defined, the permissions granted to users directly or by applying roles can be edited on the Permissions tab of the Entity Editor.

1. In the *Data Model Window*, expand the logical or physical *Main Model*, right-click the *Users* node, and then click *New Database User*.
2. On the first page of the wizard, click *Add* in the *Roles* area and then choose the roles you want to apply.
3. Click *Finish*.

Now, when you edit the database user, if you have selected Show inherited permissions on the Object Privileges tab of the User Editor, you will see blue check marks denoting the granted permissions afforded to the user by the applied roles.

- Generally, permissions granted or revoked through the application of a role are overridden by permissions assigned to the user in the User Editor, or permissions assigned to the user on the Permissions tab of the Entity/Table Editor. Check your DBMS documentation for verification.

See Also

- Creating and Editing Database Users
- Creating and Editing Database Roles
- Changing Database Users Associated with Existing Database Roles

Associating Database Users with New Database Roles

Once you have a database user defined, you can assign that user a database role. Once defined, the permissions granted to a particular user directly or by applying roles can be edited on the Permissions tab of the Entity Editor or Table Editor.

1. In the *Data Model Window*, expand the logical or physical *Main Model*, right-click the *Roles* node, and then click *New Database Role*.
2. On the first page of the wizard, in the *Roles* area click *Add* and choose the roles you want to apply.
3. Click *Finish*.

Now, when you edit the database user, if you have selected Show inherited permissions on the Object Privileges tab of the User Editor, you will see blue check marks denoting the granted permissions afforded to the user by the applied roles.

- Permissions granted or revoked through the application of a role are overridden by permissions afforded the user in the User Editor, or on the Permissions tab of the Entity/Table Editor.

See Also

- Creating and Editing Database Users
- Creating and Editing Database Roles
- Associating Database Roles with New Database Users

Changing Database Roles Associated with Existing Database Users

Once you have a database role defined, you can assign that role to any users created. Once defined, the permissions associated with a role can be edited on the Permissions tab of the Entity/Table Editor.

1. In the *Data Model Window*, expand the logical or physical *Main Model*, expand the *Users* node, right-click the user you want to edit, and then click *Edit Database User*.
2. To add a new role to the user, in the *Roles* area of the *General* tab, click *Add* and select the roles you want to apply.
   To drop a role from the user, in the *Roles* area of the *General* tab, click *Drop* and select the roles you want to remove.
3. Click *Finish*.

Now, when you edit the database user, if you have selected Show inherited permissions on the Object Privileges tab of the User Editor, you will see blue check marks denoting the granted permissions afforded to the user by the applied roles. Permissions revoked by rules do not display with crosses.


Note: Generally, permissions granted or revoked through the application of a role are overridden by permissions afforded the user in the User Editor, or on the Permissions tab of the Entity/Table Editor. Check your DBMS documentation for verification.

See Also

- Creating and Editing Database Users
- Creating and Editing Database Roles
- Changing Database Users Associated with Existing Database Roles

Changing Database Users Associated with Existing Database Roles

Once you have a database user defined, you can assign a role to that user. Once defined, the permissions granted to users directly or by applying roles can be edited on the Permissions tab of the Entity Editor.

1. In the *Data Model Explorer*, expand the logical or physical *Main Model*, expand the *Roles* node, right-click the role you want to change, and then click *Edit Database Roles*.
2. Click the *General* tab.
3. To associate the role with a user, in the *Users* area of the *General* tab, click *Add* and choose the users you want to associate with the role.
   To disassociate a role from a user, in the *Users* area of the *General* tab, in the *Users* area click *Drop* and choose the users you want to disassociate from the role.
4. Click *Finish*.

Now, when you edit the database user, if you have selected Show inherited permissions on the Object Privileges tab of the User Editor, you will see blue check marks denoting the granted permissions afforded to the user by the applied roles. Permissions revoked by rules do not display with crosses.

- Permissions granted by applying roles are overridden by permissions granted to the user in the User Editor, or on the Permissions tab of the Entity/Table Editor.

See Also

- Creating and Editing Database Users
- Creating and Editing Database Roles
- Associating Database Roles with New Database Users

Granting and Revoking Permissions to Modify Database Objects

Once you have created users and roles you can grant or revoke their permissions to insert, select, update, delete, reference, and alter entities, tables, attributes, columns, and views.

1. In the *Data Model Explorer*, double-click an object to invoke the editor.
2. Click the *Permissions* tab.
3. Select a role or user.
4. Select an operation or click the column header to select the entire range of operations.
5. You can select multiple operations by holding CTRL+Alt down while clicking the operations.
6. Click *Grant* or *Revoke*.
7. Click *OK* to exit the editor

- Keep in mind that the more specific permissions are, the more time you may have to spend maintaining and updating them.

See Also

- Creating and Editing Database Users
- Creating and Editing Database Roles
- Changing Database Users Associated with Existing Database Roles

Comparing Models

For information on comparing data models, see Using the Compare and Merge Utility.

See Also

- Copying Objects Between Models and Submodels
- Connecting to Database Sources and Targets

Customizing the Data Model

This section describes customizations you can make to the Data Model. It is comprised of the following topics:

- Setting Diagram Properties
- Setting Data Model Properties
- Creating Entity Relational or Dimensional Models
- Dimensional Table Types Indepth
- Changing the Model Notation
- Setting the Relationship Notation
- Manipulating Relationship Lines
- Changing the Model Background
- Change the Background and Outline Colors of Existing Models and Views
- Setting Default Diagram Colors and Fonts
- Customizing the Layout
- Locking Data Model Objects to a Drawing Shape
- Creating and Editing Shapes
- Adding and Editing Lines
- Adding and Editing Title Blocks
- Adding Text Blocks
- Overriding Color and Font Settings for a Specific Object
- Change Data Model Display Preferences
- Changing Entity/Table Display Preferences
- Changing Data Model Layout
- Enforcing Data Integrity Using Check Constraints

See Also

- Best Practices
- Creating and Working With Data Models

Setting Diagram Properties

In the Diagram Properties dialog you can document basic information about your data model. Data entered on this dialog is used when generating reports about the model and is also used to populate the Title Block.

1. Choose File > Diagram Properties
2. Make your changes to the *Information*, *Description*, and *Attachment Bindings* tabs, and then click *OK*.

The following describe options that require additional explanation:

Attachment Bindings tab

Bind an external piece of information, or attachment to the domain. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary and must be applied to the diagram before they will display on this tab. For more information, see Attaching External Documents to the Data Model.

See Also

- Attaching External Documents to the Data Model
- Setting Data Model Properties
- Setting the Relationship Notation

Setting Data Model Properties

The Model Properties Editor lets you modify the Security, Attachment, PreSQL and PostSQL properties for the model. It also displays the name handling options that were applied when the physical model was generated. The editor varies slightly between the logical and physical models.

1. Right-click the model or data flow and then click *Model Properties*.
2. *For Logical Models*, make your changes on the *Security Information* and *Attachment Bindings* tabs and then click *OK*.

- *For Data Flows*, make your changes on the *Security Information* and *Attachment Bindings* tabs and then click *OK*.
- *For Physical Models* make your changes on the *Security Information*, *Attachment Bindings*, *PreSQL & PostSQL* tabs and then click *OK*.

The following describe options that require additional explanation:

- Security Information tab
- Attachment Bindings tab
- PreSQL & PostSQL tab
- Name Handling tab

Security Information tab

Displays the current security information as defined in the Data Dictionary. You can add or edit security reporting information in by double-clicking the field you want to change. To set the model-wide default value for a property, go to the Data Dictionary tab and expand the Data Security Information Node. For more information, see Enforcing Security Using Data Security Types and Properties.

Attachment Bindings tab

Bind an external piece of information, or attachment to the model. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary and must be applied to the data model before they will display on this tab. For more information, see Attaching External Documents to the Data Model.

PreSQL & PostSQL tab

Enter SQL to be applied before or after the CREATE OBJECT statement. If you select Generate, the PreSQL and PostSQL scripts entered here are included in the script when you generate the physical database.

Name Handling tab

The Name Handling tab is read-only and displays the name handling options that were selected in the General Options on page 3 of the Generate Physical Model Wizard when the physical data model was generated. If a naming standards template was used when generating the model, the General options in the Generate Physical Model Wizard are not available, and the settings displayed here should be disregarded.

See Also

- Setting Diagram Properties
- Setting the Relationship Notation

Creating Entity Relational or Dimensional Models

You can generate a DM diagram by starting with a logical model, by reverse-engineering, by importing an SQL or an ERX file, or by adding a new physical model.The Model Notation controls whether the data model design technique used is Entity Relational (ER) or Dimensional (DM). You can use either technique depending on your requirements.

- What is a Dimensional Model?
- Dimensional Model Table Type Determination

What is a Dimensional Model?

A Dimensional Model is a specific type of 'entity/relationship' design used for designing systems which help optimize queries used in decision support databases (e.g. Data Warehouses, Data Marts etc).

Dimensional modeling is often used for data warehouses, providing for high-performance end-user access to data. Where a single ER diagram for an enterprise represents every possible business process, a DM diagram represents a single business process in a fact table. ER Studio Data Architect can identify individual business process components and create the DM diagram for you.

ER Studio Data Architect supports Dimensional Modeling as a new 'type' of Physical Model. Designating a model as Dimensional Modeling occurs:

- During Generate Physical Model time (as a choice in the Wizard)
- As a notational choice in the Physical Model to toggle the Notation to 'Dimensional' which will update any existing Physical Model to Dimensional status

Where entity relationship modeling is used to define the structure and association of data (most often implemented in relational databases), dimensional modeling not only covers that aspect (i.e. structure) but moves deeper into designing the SEMANTICS of the data itself. What happens to the data in the decision support system (Is it fixed data or does it change? And if it changes, to what extent?) The purpose of capturing this level of metadata in the design is so that when you leverage software systems to query the 'measures' in the database, you will more easily know what is the 'truth' behind the lineage/history of the data.

Dimensional Model Table Type Determination

Tables must be assigned to a specific dimensional model table type. This is one of the most critical aspects of offering a dimensional modeling 'notation' in ER Studio Data Architect. Initially, a choice of table types will be available from the Dimensional Modeling Toolbar when creating new tables. Users can choose a Fact, Dimension or Snowflake table to drop it on the workspace. Two things can influence the Dimensional Table Type:

- User Over-ride (see Table Editor info below):
- Automatic Assignment by ER/Studio: The dimensional modeling rules discussed in detail below will automatically be enforced by ER Studio Data Architect.

Example:

1. User drops down a Dimension 'A'

2. User Drops down another Dimension 'B'

3. User draws relationship from DimA to DimB

4. DimB automatically changes its type to a 'Fact.'

Generally speaking, this dimensional metadata now managed in the DM1 is to be leveraged externally by other 'consumers' such as reporting tools like Business Objects, Cube Views, Cognos et al in an effort to ensure that people querying data from these reporting engines know what type of data the table representsï¿½i.e. a Facts, Dimensions, etc. Interfacing ER/Studio with these large B.I. and ETL products is helpful because ERStudio can pre-populate your Repositories with this important metadata.

Dimensional Model Table Types

- *Undefined*: An Undefined Table is a table managed in the Dimensional Model that has been intentionally set by the designer (over riding the default type of 'Dimension') in order to flag it that a decision must be made about what it is to become (Dimension, Snowflake etc).
- *Fact*: A Fact Table is the focus of a Dimensional Model's intended query to determine specific business performance measurement on a selected 'topic' of data. The fact table is an aggregate collection of Dimensional Table Foreign Keys (in addition to other measurement attributes) assembled to gather specific numeric or additive data the business is interested in reviewing historical performance of.
- *Dimension*: A Dimension Table contains the very specific and descriptive data on a particular aspect of a query which the Fact Table assembles. An example would be a Dimension called 'Product' which is a table containing product specific information fed into the Fact. Other Dimensions provide other 'descriptors' such as 'Region' When fed into the Fact, queries can be derived for Product sold within a specific Region.
- *Snowflake*: Outriggers (sometimes called Snowflakes due to the structure they visually look like in a Dimensional Model as extensions to Dimensions) act as 'qualifier' for Dimensions. Snowflaking essentially 'normalizes' a Dimension table by removing specific attributes in a Dimension (oftentimes these are 'types') and implementing them as separate tables acting as lookup parents of the Dimension. An example would be a Dimension called 'Marketing Brand' which at one time contained attributes called Finance Category and Marketing Category. In order to normalize this data, those two attributes were removed from the dimension and established as their own tables and provided with a PK which are then propagated via non identifying relationship to the child dimension, Marketing Brand.

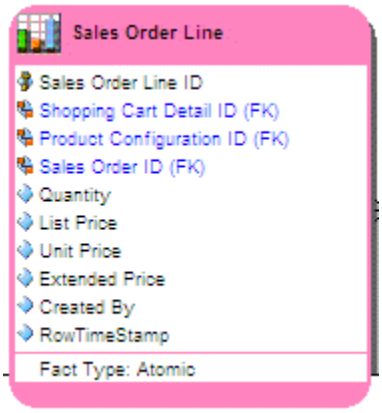
These table types are further discussed below.

See Also

- Dimensional Table Types Indepth
- Setting the Relationship Notation

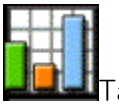
Dimensional Table Types Indepth

In a dimensional model, each table is assigned a Table Model type, which is distinguished by an icon displayed in the upper left of the entity or table box. You can further define a table in the Table Editor by assigning it a Table Type that displays at the bottom of the table. The Table Type does not change how the data is handled but provides information to the reader about the data in the table:



ER Studio Data Architect supports the following Dimensional table types:

Fact

Tables with one or more foreign keys and no children.

Fact tables contain individual records, the data for which the database is being designed. Fact tables are the central tables in a star schema.

ER Studio Data Architect allows you to further define fact tables as one of the following types:

- *Aggregate*: Contains information that is more coarsely granulated than the detail data contained in other tables. For example, in a retail, transactional database, you can have a Sales_Receipts fact table that contains information about individual sales, and a Sales_Monthly aggregate table that aggregates information from the Sales_Receipts table to present information such as monthly sales totals for software, hardware, and services. Aggregates provide quick access to data during reporting; like indexes they improve performance.
- *Atomic*: Contains detail information, such as the Sales_Receipts table in the discussion above.
- *Cumulative*: Contains cumulative information such as how long it took to complete the sale by accumulating the time it took between the date of sale and the shipping date for each line item of the sales receipt.

- *Snapshot*: Contains time-related information that detail specific steps in the life on the entity. For example, snapshot information for a sale could contain information such as when the order was created, committed, shipped, delivered, and paid for.

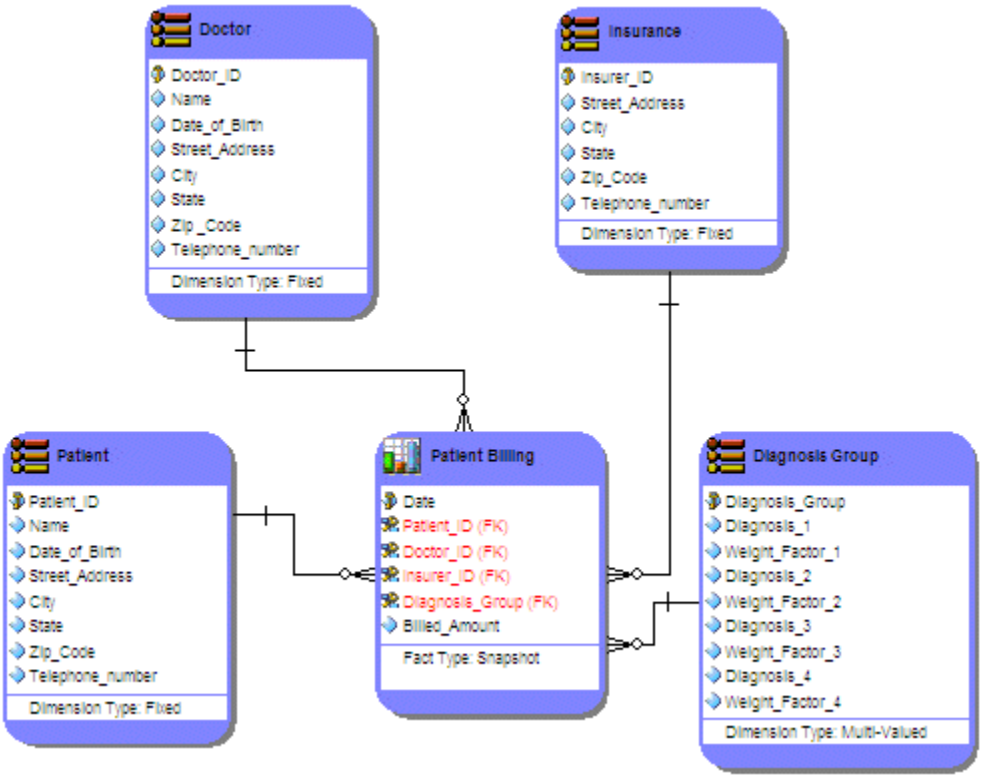
Dimension

Parents of fact tables.

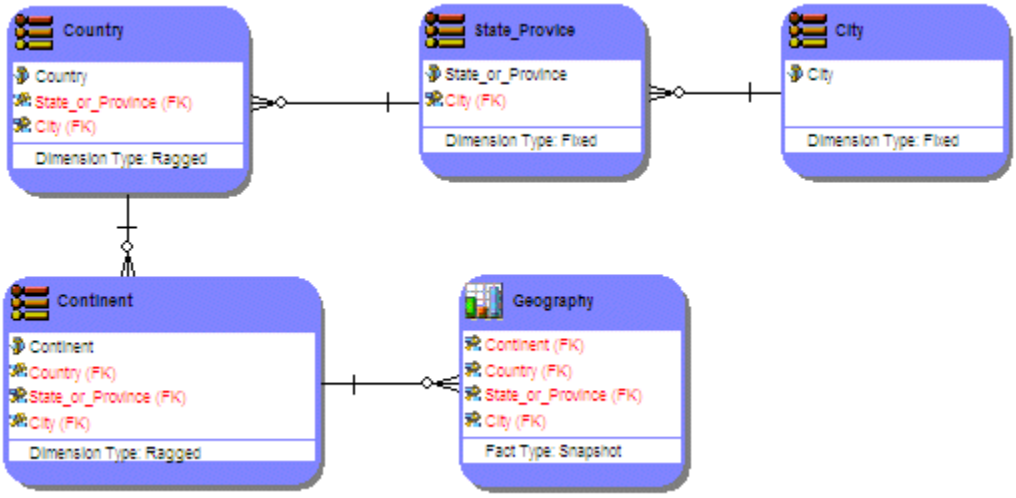
Dimension tables contain groups of related data, such as dates, hours, minutes, and seconds, represented by one key such as time in a fact table.

ER Studio Data Architect allows you to further define dimension tables as one of the following types:

- *Fixed Dimension*: The values in the table are not expected to change.
- *Degenerate*: A degenerate dimension is derived from the fact table. Degenerate dimensions are useful when the grain of a fact table represents transactional level data and you want to maintain system-specific identifiers such as order numbers and invoice numbers without forcing them to be included in their own dimensions. Degenerate dimensions can provide a direct reference to a transactional system without the overhead of maintaining a separate dimension table. For example, in the sample model, Orders.dm1, you could create a degenerate dimension that references the Commission Credit ID, Payment Detail ID, and Sales Order ID from the Commission Credit, Payment Detail, and Sales Order Line ID fact tables respectively,
- *Multi-Valued*: A multi-valued dimension can help you model a situation where there are multiple values for an attribute or column. For example, a health care bill has a line item of Diagnosis, for which there could be multiple values. Best practice modeling dictates that there should be a single value for each line item. To model this multi-valued situation, you could create a multi-valued table that would capture the diagnosis information and weighs each diagnosis, so that the total adds up to one. This weighting factor allows you to create reports that do not double count the Billed Amount in the fact table.



- *Ragged*: In a ragged dimension, the logical parent of at least one member is missing from the level immediately above the member. Ragged dimensions allow for hierarchies of indeterminate depth, such as organizational charts and parts explosions. For example, in an organization some employees can be part of a team which is managed by a team leader, while other employees report directly to the department manager. Another example of a ragged dimension is illustrated in the model below which shows that cities such as: Washington, D.C.; Vatican City; and Monte Carlo do not belong to states.



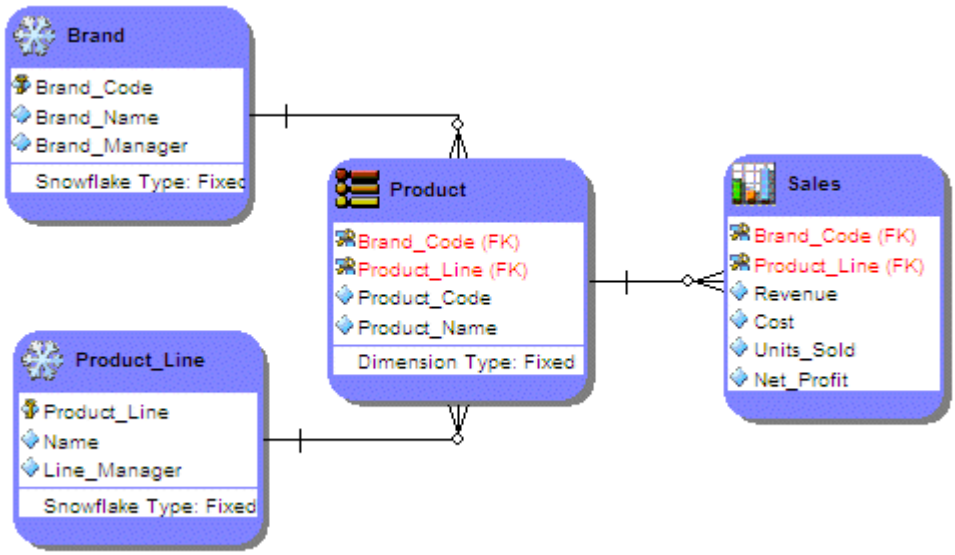
- *Shrunken*: A shrunken table is a version of the fact table that it is attached to but with fewer attributes in order to draw attention to those specific attributes.

- *Slowly Changing Type 0, 1, 2, 3, 6*: Slowly changing dimensions capture information such as the customerï¿½s income level or income-to-debt ratio which can change overtime. The most commonly used methods to maintain these changes are:
- *Type 0*: No effort has been made to deal with the changes.
- *Type 1*: Overwrites the existing dimension member values. Does not track the change history.
- *Type 2*: Creates a new row with the current dimension member. Tag the original row as expired.
- *Type 3*: Used for dimensions that change with a predictable rhythm, such as every year. Every dimension row has a *current* category attribute that can be overwritten as well as attributes for each annual designation, such as 2008 category and 2009 category.
- *Type 6*: Combines the approaches of types 1, 2, and 3. For changes to this type of table, the row is overwritten but the table can contain an additional pair of date columns to indicate the date range at which a particular row in the dimension applies, or the table can contain a revision number.

Snowflake

Parents of dimension tables.

Snowflake tables present a more-normalized format where elements of dimension tables are listed. The following example shows two snowflake tables which are parents of a dimensional table. There are many other good examples or snowflake tables in the sample model, Orders.dm1.



Bridge

Implements a many-to-many relationship. Bridge tables support multi-valued dimensions or complex hierarchies. The Bridge is also known as a helper table or an associative table. It is the only way to implement two or more one-to-many relationships or many-to many relationships.

Hierarchy Navigation

Used to support complex hierarchies, such as organizational charts.

Undefined

All other tables. Assign this type to flag tables for which you have not yet determined the appropriate type, such as a table with a many-to-many relationship, or a table that is a parent to both a fact table and a dimension table.

See Also

- Changing the Model Notation
- Dimensional Model Table Types

Changing the Model Notation

1. Click the target model and then Choose Model > Model Notation.
2. From the list, select *Relational* or *Dimensional*.

The model is redrawn to conform with the model notation selected.

Notes

- When generating a physical model, by reverse-engineering or importing an external data source to create the model, you can choose the model notation.
- You can change the notation of a physical model in the Physical Model Options dialog. Access this dialog by right-clicking the target physical model, and then from the shortcut menu selecting Model Options.
- You can change the table type of an object in the dimensional model using the Dimensional tab of the Entity Editor or Table Editor. Access this editor by double-clicking an entity or table.

See Also

- Reverse Engineering an Existing Database
- Importing a Model
- Generating a Physical Data Model

Setting the Relationship Notation

ER Studio Data Architect can represent the relationships in your data models in IDEF1X notation or in one of three varieties of IE notation.

IDEF1X versus IE Relationship Notation

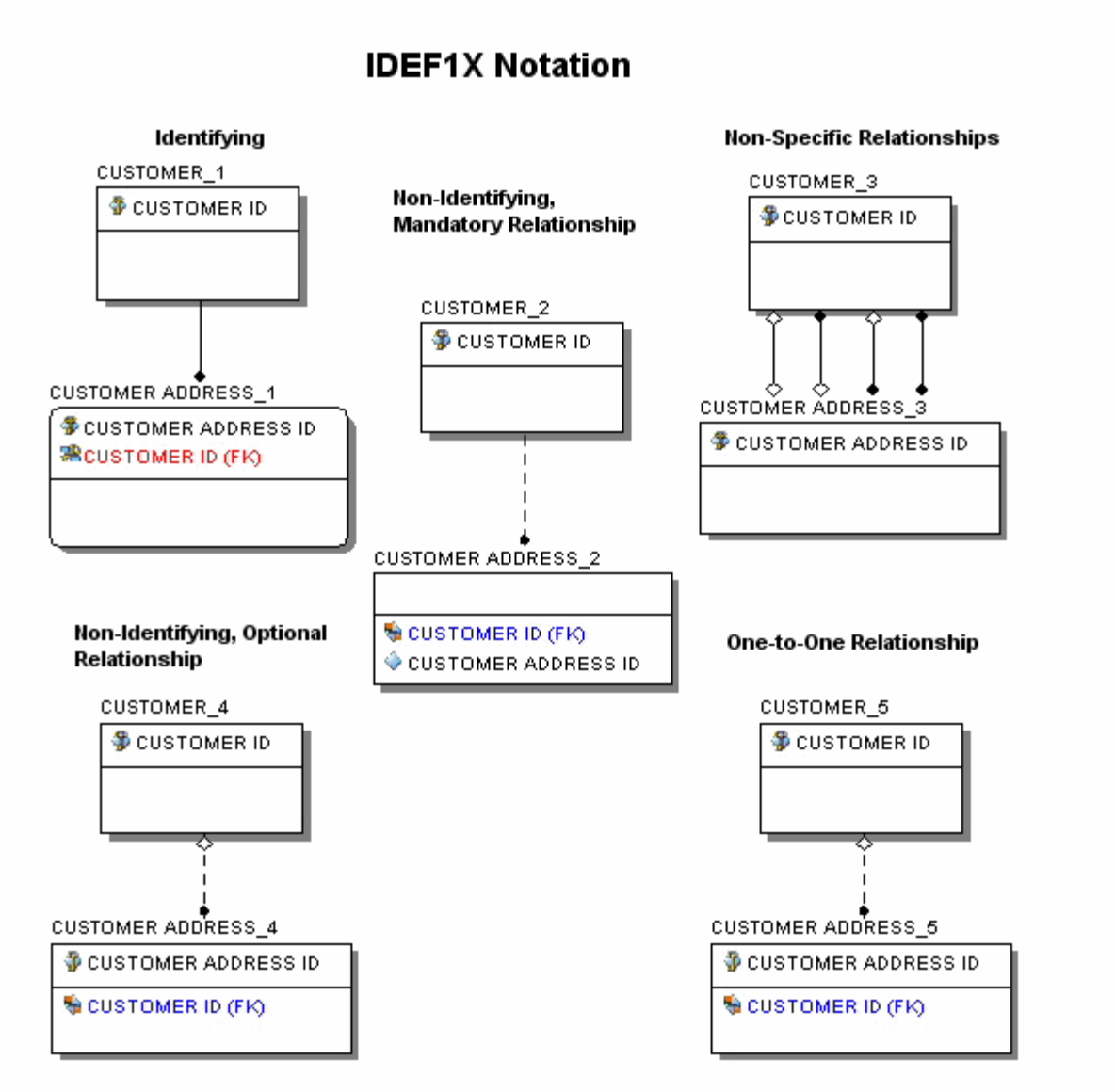
IDEF1X is based on the relational data model designed by E. F. Codd and on the entity-relationship model designed by Peter Chen, but IDEF1X models differ from the models on which IDEF1X is based. A model created using Peter Chen's or E.F.Codd's designs will appear different compared to a model created using ER Studio Data Architect and the IDEF1X notation, and vice versa.

IDEF1X makes it easier to visualize primary and foreign keys. IDEF1X also offers the power (and danger) of using identifying relationships to cascade primary keys through a chain of entities.

- IDEF1X Notation
- IE (James Marting) Notation
- IE (Crow's Feet) Notation
- FItered ID (Hide Foreign Keys) Notation

IDEF1X Notation

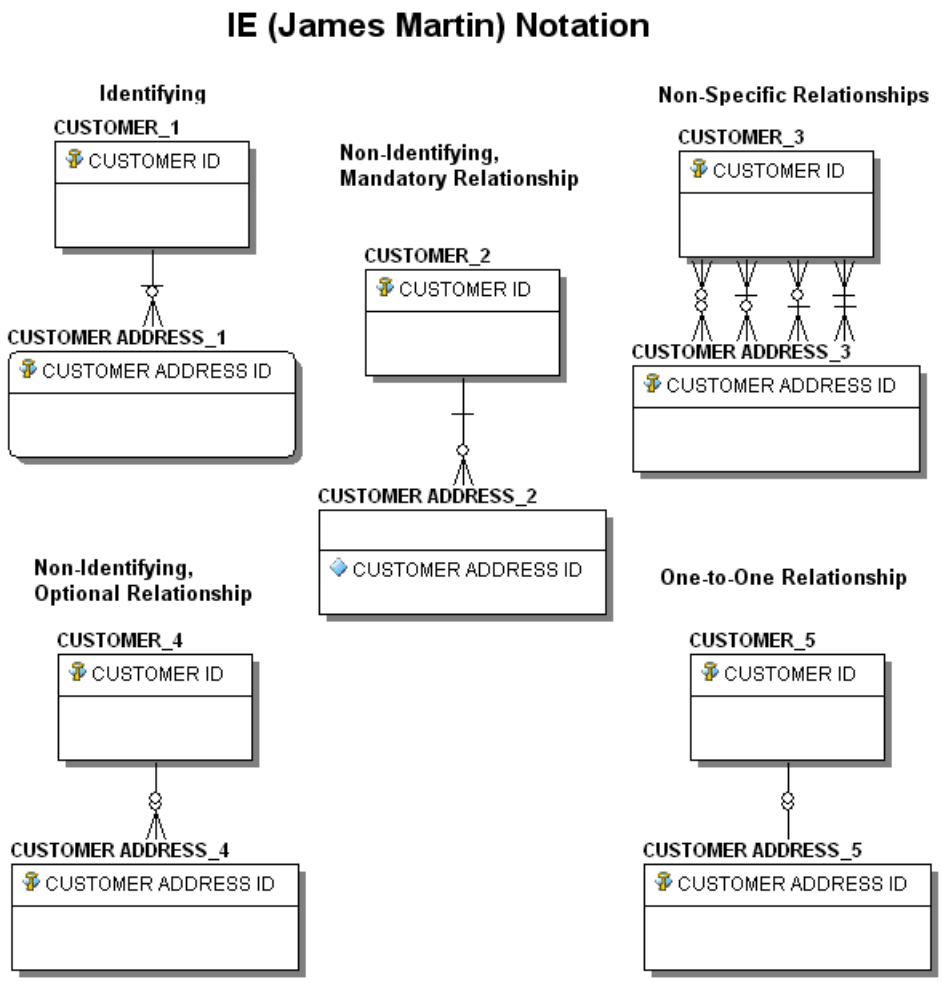
The following illustrates how relationships are presented in ER Studio Data Architect using the IDEF1X notation:



IE is considered to be semantically stronger than IDEF1X, supporting more abstract approaches to data modeling. IE is more flexible in supporting subtypes, permitting roll up (generalization) and roll down (specialization) transformations. In addition, primary keys are less prominent in the IE notation; therefore, its practitioners are less likely to think about logical key choice prematurely.

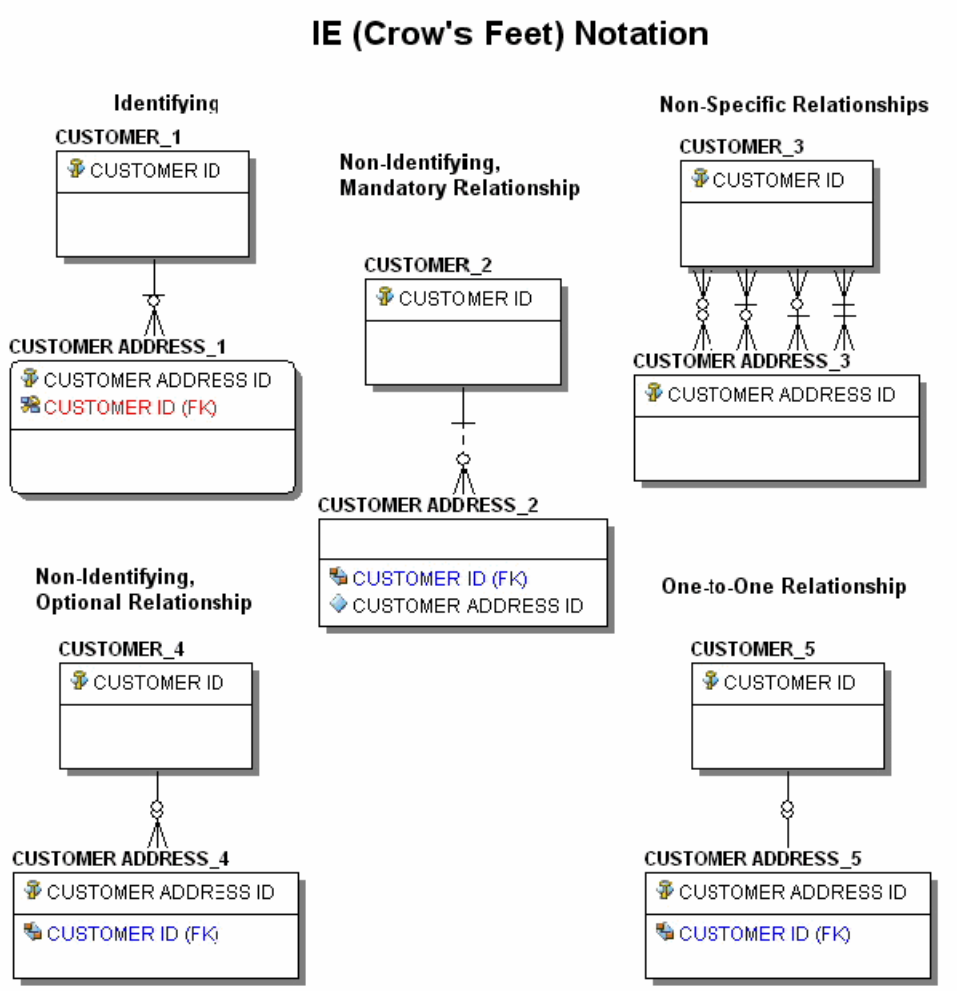
IE (James Marting) Notation

The following illustrates how relationships are presented in ER Studio Data Architect using the IE (James Martin) notation.



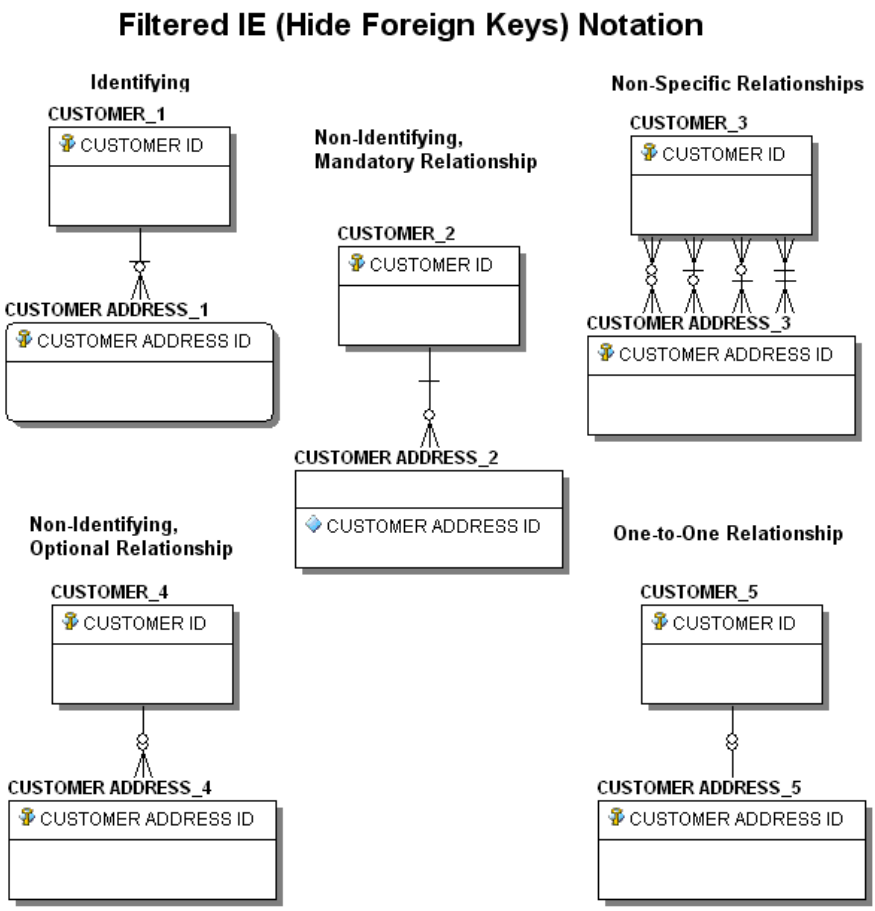
IE (Crow's Feet) Notation

The following illustrates how relationships are presented in ER Studio Data Architect using the IE (Crow's Feet) notation.



FItered ID (Hide Foreign Keys) Notation

The following illustrates how relationships are presented in ER Studio Data Architect using the FItered ID (Hide Foreign Keys) notation.



Changing the Relationship Notation for all Diagram Models

1. Choose View > Diagram and Object Display Options.
2. Click the *Relationship* tab.
3. In *Display Preferences*, choose the desired notation and then click *OK*.

Changing the Relationship Notation for a Selected Model

1. Right-click the target model and then select *Model Options*.
2. In the *Notation* area, choose the desired notation and then click *OK*.

See Also

- Manipulating Relationship Lines
- Creating Entity Relational or Dimensional Models
- Changing the Model Notation

Manipulating Relationship Lines

Relationship lines represent relationships between entities or between attributes. You can customize the appearance of any relationship line in any model, even when using auto layout. For example, if you use an Orthogonal layout, where all relationships bend at a 90 degree angle, you can change any individual bend to an N bend or straighten the relationship line.

You can also set the bend attributes of any relationship in a model. There are two kinds of bends in ER Studio Data Architect, N bends and Orthogonal bends. N Bends are relationship lines that bend at any angle while Orthogonal bends only bend at a 90 degree angle. Both Hierarchical and Orthogonal auto layouts use only Orthogonal relationship lines. All other auto layouts use straight relationship lines. You can also create straight relationship lines that are vertical or horizontal.

When you use the auto layouts, ER Studio Data Architect organizes the data model objects according to the specifications of that particular layout. Each layout has different specifications for the way relationships bend and how relationships dock or intersect with entities and attributes. You can move the docking positions of relationship lines on parent or child entities.

You can also change the color of all or individual relationships in the data model. You can specify a default relationship color on the Options Editor - Application tab. Use the Colors and Fonts Editor to change all or an individual relationship in your model.

- You can overwrite relationship defaults after applying an automatic layout by changing relationship lines in a diagram or by changing the line disposition in a diagram.
- You can use the Options Editor, Diagram and Object Display Options Editor, and the Format Menu to change relationship disposition and colors. Use the Options Editor - Application tab to make global changes to all your data models. Use the View Menu to make changes to the current data model and the Format Menu to make changes to the selected objects. Changes made in the Format Menu will override changes made first in the Options Editor.

This section is comprised of the following topics:

- Changing the Style of Relationship Lines
- Create an N Bend
- Straightening an N Bend
- Change the Disposition of Relationship Lines

Changing the Style of Relationship Lines

To make your diagram easier to read, you may want to change the style of relationship lines from elbowed to straight or vice-versa.

- When changing from elbowed to straight relationship lines, ER Studio Data Architect repositions the docking points to as close as possible to their original positions.

- To change the style of relationship lines in all new data models, choose Tools > Options > Display and in the *Model Display* area, select the *Line Drawing Mode* you want, either *Elbowed Lines* or *Straight Lines*. Relationships in all new data models you create will now default to the Line Drawing Mode selected.
- To change the style of all relationship lines in the current data model, choose View > Diagram and Object Display Options > Relationship and in the *Display Preferences* area, select the *Relationship Style* you want, *Elbowed* or *Straight*.
  The relationships in the current data model are redrawn to conform with the Relationship Style selected.
- To change the style of a selected relationship line, right-click the relationship line, click Layout Relationships, and then click Elbowed or Straight. Alternatively, after selecting the relationship line, on the Format > Layout Relationships list, click *Elbowed* or *Straight*.
  The selected relationship is redrawn to conform with the relationship style selected.
- To straighten a relationship line, right-click the relationship line and then click *Straighten Relationship Line*.
- Alternatively, you can straighten a horizontal relationship by selecting the relationship line and then pressing CTRL+H. Pressing CTRL+L straightens a vertical relationship.
- Changes are cascading, meaning that changes made in the Format menu will override changes made in the Options Editor.

Create an N Bend

To make your diagram easier to read, you may want to create N bends, which are like elbows, on one or more relationship lines.

- If you are using an automatic layout, ER Studio Data Architect lets you create N bends only in the following layouts: Circular Layout, Symmetric Layout, and Tree Layout.

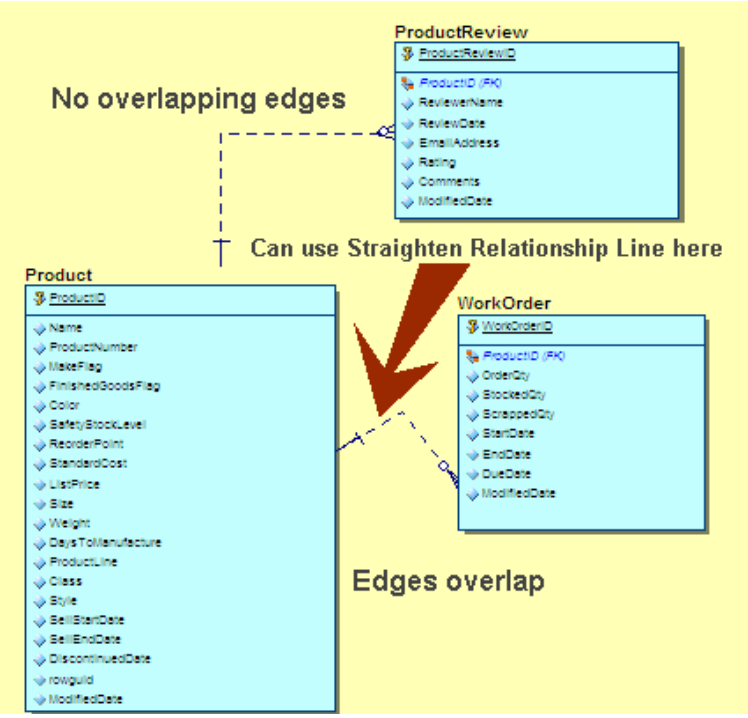
1. Click the relationship line that you want to bend.
2. Move the cursor over the line at the point on the relationship where you want to create the bend.
3. The pointer should change to 
4. Click and drag to position the bend.
5. To create more than one bend point on the relationship, repeat steps 2 and 3.

Straightening an N Bend

Once you have created an N Bend, you can use the following methods to straighten the line.

- To remove a bend, click the relationship line you want to straighten and then choose Format > Layout Relationship > Straight

- To reset the line to its default position, click the relationship line you want to straighten and then choose Format > Line Disposition > Reset to Default Docking Side(s).
- To straighten a line between two entities that have overlapping edges, as shown in the following illustration, right-click the line and then click *Straighten Relationship Line*.



Change the Disposition of Relationship Lines

To make your diagram easier to read, you may want to change the positioning or disposition of the relationship lines.

1. Click the relationship line and then choose Format > Line Disposition.

- You can select all the relationship lines in the data model by selecting one and pressing CRTL+SHIFT+A.
- From the list select *Center*, *Vertical*, *Horizontal* or *Reset to Natural Docking Side(s)*.
- You can reset the disposition of a relationship line to the natural docking side by right-clicking the relationship line and then from the list, selecting *Reset to Default Docking Position*.

The selected relationship is redrawn to conform with the relationship line disposition selected.

See Also

- Overview
- Creating and Editing Relationships

Changing the Model Background

To make your model more visually appealing, you can change the background color displayed to complement the colors you choose for other object models in the Data Model Window.

1. Choose Tools > Options.
    - *To change the background of new logical models*, select the *Logical* tab.
    - *To change the background of new physical models*, select the *Physical* tab.
2. In the *Main Model Display* area, click *Colors & Fonts >Background Color*.
3. *To choose one of the Basic colors* or one of the existing *Custom Colors*, select a color box and then click *OK*.
   To choose a color not already displayed in this dialog, click *Define Custom Colors*, define the new color, click *Add to Custom Colors*. (This adds a new color option to the available *Custom Colors*.) Now you can select the new color box and then click *OK*.
4. Click *OK* again to exit the *Options* editor.
5. You can export the model background by selecting Export model and object background color in *Model > Options > Display*.

See Also

- Change Data Model Display Preferences
- Overriding Color and Font Settings for a Specific Object

Change the Background and Outline Colors of Existing Models and Views

1. In the *Data Model Explorer*, click a data model or a view of a data model.
2. Choose Format > Colors & Fonts > Colors.
   To set the background color of a model, from the list select *Model Background* and then click *Set Color*.
   To set the background color of all views in the model, from the list, select *View Background* and then click *Set Color*.
   To set the foreground color of all views in the model (the dotted line around the view box), from the list select *View Foreground* and then click *Set Color*.
3. *To choose one of the Basic colors* or one of the existing *Custom Colors*, select a color box and then click *OK*.
   To choose a color not already displayed in this dialog, click *Define Custom Colors*, define the new color, and click *Add to Custom Colors*. (This adds a new color option to the available *Custom Colors*.) Then you can select the new color box and click *OK*.
4. To apply your changes and exit the *Colors & Fonts* editor, click *OK* again.

Notes

- When exporting the model, you can also export the model background by selecting *Export model and object background color* in Model > Options > Display.
- You can specify background and foreground colors for an individual view by right-clicking the view in the Data Model Window and then clicking *View Background/Outline Color*.
- You can specify that all views in the data model share the same background and foreground (outline) colors by choosing Tools > Options > View and then in the *View* area, selecting *Same background color for all views*.

See Also

- Overriding Color and Font Settings for a Specific Object
- Customizing Standard Model Features for New Models

Setting Default Diagram Colors and Fonts

You can create color schemes and change fonts to help manage your workspace. The Colors and Fonts Editor lets you customize the data model display by customizing the background and foreground colors of the model, entities, and database objects and by customizing the font for individual components of the data model or standardizing on a specific font for all displayed text.

1. Choose Format > Colors & Fonts.
2. Make the changes as required and then click *OK* to implement the changes.

Notes

- Override the default colors and fonts for specific objects or for instances of a specific object. For more information, see Overriding Color and Font Settings for a Specific Object.
- Set the default background color of the model in the Main Model Display area of the Logical or Physical tabs of the Options editor, choose Tools > Options.

See Also

- Changing the Model Background
- Overriding Color and Font Settings for a Specific Object

Overriding Color and Font Settings for a Specific Object

To make your model more attractive and easier to read, you can change the background, outline, and font colors along with the font type entities/tables, attributes/columns, views, and relationships in your data model. Use specific colors and/or fonts to help you organize your data model.

- Set the default background color of the model in the Main Model Display area of the Logical or Physical tabs of the Options editor, choose Tools > Options.

This section is comprised of the following topics:

- Change Color and Font Settings for an Entity, Table, Attribute, Column or View
- Change Entity/Table Background and Outline Colors
- Change the Background and Outline Colors of Views in an Open Data Model
- Changing the Color of Relationships

Change Color and Font Settings for an Entity, Table, Attribute, Column or View

Use the *Color and Fonts* editor to do the following:

- Change the font, font style, size, and effects (strikeout and underline) for one or more selected components of an entity or table, such as: entity name, entity attributes, primary keys, role names, alternate keys, and inversion entries.
- Change the font, font style, size, and effects (strikeout and underline) for selected components of a view column.
- Apply font and color settings to attribute categories such as: non-inherited primary keys, inherited primary keys, non-inherited non-keys, inherited non-keys, primary key index, unique indexes, non-unique indexes, attachments, triggers, and data security.
- Change the background and outline color of entities, tables and views.

1. In the *Data Model Window*, right-click the object you want to change or the object containing the object you want to change.
2. *For entities, tables, columns, and attributes*, choose Entity Color and Font Settings > Entity/Attributes Colors and Fonts.
   For views and view columns, choose View Color and Font Settings > View/View Columns Colors and Fonts.
3. In the *Colors and Fonts* editor make the changes desired.

4. You can apply the same changes to multiple attributes or columns by pressing and holding CTRL while clicking the objects.
5. To apply color and font setting changes to all occurrences of the selected object in the model, select *Apply to all submodels* ; otherwise, the changes will affect only the occurrence of the entity selected.
6. Use the preview area to preview your changes and then click *OK* to implement the changes.

Change Entity/Table Background and Outline Colors

1. In the *Data Model Window*, right-click the entity/table you want to change.
2. Choose Entity Color and Font Settings > Entity Background/Outline Color.
3. In the *Entity Color* editor make the changes desired.
4. To apply color changes to all occurrences of the selected entity in the model, select *Apply to all submodels* ; otherwise, the changes will affect only the occurrence of the entity selected.

Change the Background and Outline Colors of Views in an Open Data Model

1. In the *Data Model Explorer*, expand the *Views* node of a logical or physical mode, and then right-click the view you want to change.
2. Choose View Color and Font Settings > Colors & Fonts > Colors.
3. *To set the background color of all views in the model*, from the list, select *View Background* and then click *Set Color*.
   To set the foreground color of all views in the model (the dotted line around the view box), from the list, select *View Foreground* and then click *Set Color*.
4. *To choose one of the Basic colors* or one of the existing *Custom Colors*, select a color box and then click *OK*.
   To choose a color not already displayed in this dialog, click *Define Custom Colors*, define the new color, click *Add to Custom Colors*. (This adds a new color option to the available *Custom Colors*.) Then you can select the new color box and click *OK*.
5. To apply your changes and exit the *Colors & Fonts* editor, click *OK* again.

Notes

- When exporting the model, you can also export the view background by selecting Export model and object background color in Model > Options > Display.
- You can specify background and foreground colors for an individual view by right-clicking the view in the Data Model Window and then selecting View Background/Outline Color. This setting can be overridden by selecting Same background color for all views in Tools > Options > View. If Same background color for all views is enabled, the View Background/Outline Color option will be unavailable in the View Color and Font Settings short-cut menu.

- You can specify that all views in the data model share the same background and foreground (outline) colors by choosing Tools > Options > View and then in the View area, selecting Same background color for all views. This setting will override any setting made for individual views in Tools > Options > View.

Changing the Color of Relationships

1. In the *Data Model Window*, right-click the relationship for which you want to change the color.
2. Choose Relationship Color > Edit Settings.
3. You can change the color of all or individual relationships in your diagram. You can specify a default relationship color on the Options Editor - Application tab. Even if you set a default color for your relationships, you can still use the Colors and Fonts Editor to change all or individual relationships in your diagram.

See Also

- Changing the Model Background
- Customizing the Display of Diagrams and Objects
- Customizing Standard Model Features for New Models

Change Data Model Display Preferences

The Display tab of the Options Editor lets you specify the default Model display settings for any new models you create.

1. Choose Tools > Options > Display.
2. In the *Model Display* area, select the options desired.

Notes

- You can change the display settings of the current model or submodel by choosing View > Diagram and Object Display Options and then making your changes within the editor that appears.
- Selecting Rolename Notation displays the full IDEF1X rolename notation.

See Also

- Overriding Color and Font Settings for a Specific Object
- Setting Diagram Properties
- Setting Data Model Properties

Changing Entity/Table Display Preferences

The Display tab of the Options Editor lets you specify the default Entity display settings for any new models you create.

1. Choose Tools > Options > Display.
2. In the *Entity Display Settings* area, from the list select the *Display Mode*, and then select the *Display Options* desired.
3. Click *OK* to implement the changes.
4. You can change the display settings of the current model or submodel by choosing View > Diagram and Object Display Options and then making your changes within the editor that appears.

See Also

- Overriding Color and Font Settings for a Specific Object
- Changing the Model Background

Customizing the Layout

Using the Layout Properties Editor, customize layout properties common to all layouts and customize the appearance of a specific layout method.

- Any changes made in the Layout Properties Editor apply only to the model currently selected.

1. In the *Data Model Explorer*, click the model you want to layout using a customized layout.
2. Choose Layout > Layout Properties Editor.
3. Click a layout tab.
4. Complete the changes as desired.
5. For help on a specific option, click the option and then press F1. Contextual help for this option appears.
6. Click *OK* to apply the changes and exit the editor.

The following describe options that require additional explanation:

- All tabs
- Disconnected tab
- Circular tab
- Hierarchical tab
- Orthogonal tab
- Symmetric tab
- Tree tab

All tabs

- *Apply*: Applies your changes to all selected diagrams without closing the dialog box.
- *Defaults*: Resets all values of the selected property sheet to ER Studio Data Architect default values.
- *Layout*: Applies your changes to all selected diagrams without closing the dialog box, and immediately displays any changes to the active model in the Data Model Window.
- *Reset*: Resets all values of the selected property sheet to the values displayed when you opened the Layout Editor.

Disconnected tab

- *Tiling*: Offers three ways of tiling disconnected entities: to grid, to rows, and to columns.

- *Tile to Grid*: All disconnected entities are spaced evenly apart in a grid.
- *Tile to Rows*: Maximizes space efficiency by placing connected entities in rows running left to right, leaving just enough vertical space for the tallest entity in the row.
- *Tile to Columns*: Maximizes space efficiency by placing connected entities in columns running top to bottom, leaving just enough horizontal space for the widest entity in the column.

Circular tab

- *Biconnectivity*: A graph is considered biconnected if within that graph, when any entity is deleted the graph remains completely connected.
- *Cluster By*: The final clustering phase must be either Biconnectivity or Degree.
- *Degree*: Specifies the minimum number of relationship lines that must branch off a diagram symbol before necessitating the creation of a new cluster. For example, if the data model Auto Layout Degree value is five (the default), any entity in that diagram with five or more relationship lines seeds a new cluster.

Hierarchical tab

- *Edge Routing*: Specifies the nature and look of the relationship lines in the selected data model.
- *Incremental Layout*: Specifies the retention of familiar component shapes in the Data Model Window. For example, if you produce a data model in one layout style, paste into the window of a data model laid out in a different style, and then join them to make a connected diagram, selecting the Incremental Layout check box preserves the different layout styles as you continue to make modifications to the data modelEdge RoutingV.
- *Layout Quality*: Specifies the speed of the algorithm used to produce the layout.
- *Draft*: Provides the lowest quality, but fastest level of layout.
- *Proof*: Provides the highest quality, slowest level.
- *Orientation*: Lets you specify one of four ways to orient the data model. Changing data model orientation does not automatically change port specifications, connectors, entity shapes, or constraints, some or all of which can rely on specific sides of entities, making for some complicated side effects of changing the orientation of complex data models. Selecting Rotation Specifications includes port specifications, connectors, entity shapes, or constraints when re-orienting the data model.
- *Undirected Layout*: When a diagram is undirected, the source and target entities of any relationship are treated the same. In a directed diagram, the direction of the relationship is significant. Networks are often represented in undirected layouts, while processes are often represented in directed layouts.

Orthogonal tab

- *Compaction Quality*: Specifies the quality of the process that removes unused white space in the target diagram. The primary difference between Draft, Default, and Proof quality is the number of times ER Studio Data Architect runs a compaction algorithm on the diagram. The more often the algorithm runs, the more efficient the use of white space, and the more resource-intensive the process.
- *Edge Spacing*: Specifies the horizontal and vertical distance between parallel relationship lines.
- *Incremental Layout*: Specifies the retention of familiar component shapes in the Data Model Window. For example, if you produce a data model in one layout style, paste into the window of a data model laid out in a different style, and then join them to make a connected diagram, selecting the Incremental Layout check box preserves the different layout styles as you continue to make modifications to the data modelEdge RoutingV.
- *Node Spacing*: Specifies the horizontal and vertical distance between entities.

Symmetric tab

- *Layout Quality*: Specifies the speed of the algorithm used to produce the layout.
- *Draft*: Provides the lowest quality, but fastest level of layout.
- *Proof*: Provides the highest quality, slowest level.

Tree tab

- *Undirected Layout*: When a diagram is undirected, the source and target entities of any relationship are treated the same. In a directed diagram, the direction of the relationship is significant. Networks are often represented in undirected layouts, while processes are often represented in directed layouts.

See Also

- Overriding Color and Font Settings for a Specific Object
- Changing the Model Background

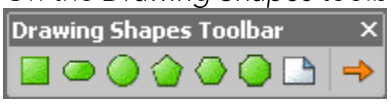
Creating and Editing Shapes

Drawing shapes are supported in the data model window of the logical model and in physical models for all database platforms supported by ER Studio Data Architect. You can also use shapes on the data lineage diagram window.

Drawing shapes have no fixed meaning in your diagrams, so you can determine how they are used and interpreted. There are three, primary uses for drawing shapes:

- *Unique Representation*: Drawing shapes can represent schema objects in your model that are not commonly represented by regular ER notation. For example, using ER Studio Data Architect's drawing shapes, you can add objects in a physical model to represent Java classes. Such objects can ultimately have crucial meaning to your database design.
- *Notation*: The note drawing shape enables you to document special circumstances that exist in your diagrams. Because the contents of the note drawing shape are displayed in the Data Model Window, they are more easily seen than entity notes or table notes, which are buried in the entities and tables themselves.
- *Containment*: Enabling containment embeds entities and tables within the drawing shapes, essentially locking the tables and entities to the drawing shapes. When you move the drawing shape, the entities and tables follow, maintaining their positions within the drawing shape. Containment is particularly useful when you want to isolate entities and tables, making those entities and tables more obvious. Similarly, you can use containment to keep objects together that share a certain importance so can you move these shared objects in unison when you move the drawing shape.

1. On the *Drawing Shapes* toolbar, click the icon for the shape you want to insert.



2. On the Data Model Window or Data Lineage Window, click the mouse where you want to add the shape.
3. Double-click the shape to launch the *Shape Editor*, where for example, you can add text and change the color of the shape.

The following help you complete the options in the *Shape Editor*:

- Shape Text tab
- Colors and Fonts tab
- Attachment Bindings tab
- See Also

Shape Text tab

You can adjust the text positioning choices either with the Horizontal Justification and Vertical Justification drop-down menus or by clicking one of the nine squares in the adjacent grid.

With Fit Shape selected, the shape is automatically resized to fit the shape text. With this option selected, Wrap Text is disabled, and the object cannot be resized by dragging its outline handles.

Colors and Fonts tab

Here you can change the shape type (e.g. oval to rectangle).

Attachment Bindings tab

Bind an external piece of information, or attachment to the shape. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary. For more information, see Attaching External Documents to the Data Model.

See Also

- Overriding Color and Font Settings for a Specific Object
- Changing the Model Background
- Creating Entity Relational or Dimensional Models
- Adding and Editing Lines
- Locking Data Model Objects to a Drawing Shape

Locking Data Model Objects to a Drawing Shape

1. In the *Data Model Window*, place a shape close to the data model object you want to contain.
2. Click the shape and then use the sizing handles to resize the shape so that it encompass the objects you want to move with the shape.
3. Right-click the shape and choose Containment > Enable.
4. You can remove the containment restriction by right-clicking the object and then choosing Containment > Disable.

See Also

- Overriding Color and Font Settings for a Specific Object
- Changing the Model Background

Adding and Editing Lines

You can use lines to establish a visual connection between two entities without propagating foreign keys as would occur when creating a relationship. You can also use lines to illustrate a connection between data lineage objects.

1. On the *Drawing Shapes* toolbar, click the *Line* tool.
2. Click the object where the line is to start and then click the object where the line is to end.
3. Double-click the line to launch the *Line Editor* and then click *OK* to exit the editor.

The following help you complete the options in the *Line Editor*:

- Properties tab
- Attachment Bindings tab
- Notes

Properties tab

Here you can change the color or weight of the line and end-point styles

- Define the relationship between the source and target objects of the line using the Verb Phrase and Inverse Verb Phrase definitions in the Line Editor.

Attachment Bindings tab

Bind an external piece of information, or attachment to the line. You can also remove an attachment from a line, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary. For more information, see Attaching External Documents to the Data Model.

Notes

- Edit the line by right-clicking the line and then clicking Layout Line, Straighten or Format.
- Change the properties of a line in the Line Editor by right-click the line you want to change and then selecting Format.
- Turn on and off the display of verb phrases in the Display tab of the Options editor (Tools > Options).
- Change the default line type from elbowed to straight lines in the Line Drawing Mode area of the Display tab in Tools > Options.

- Choose whether or not to display the verb phrase of the line in the Model Display area of the Display tab in Tools > Options.
- When you hover over a line, ER Studio Data Architect displays the name of the parent and child of the line if you have chosen View > Cursor Help Popup Options > Display Relationship Help.

See Also

- Adding and Editing Title Blocks
- Adding Text Blocks
- Changing the Model Background

Adding and Editing Title Blocks

The Title Block provides important identification information about your model or submodel that by default includes the file name and information from the Diagram Properties, such as project name, author name, company name, diagram version number, copyright information, the diagramï¿½s creation date and last modification date. You can change many of the field values in the title block without affecting the values in the Diagram Properties editor.

1. Click the model or submodel to which you want to add a title block.
2. Click Insert >Title Block.
3. In the *Data Model Window*, click where you want to place the title block, and then right-click to deselect the *Title Block* creator.
4. Use the selection tool to reposition the title block if desired.
5. Double-click the *Title Block* to launch the Title Block editor, make the changes desired, and then click *OK* to affect the changes and exit the editor.
6. Repeat steps 1-4 for each model or submodel as desired.

Notes

- The File Name and Submodel or Model name fields in the Title Block editor are read-only, as are the creation and modification dates.
- For information on Diagram Properties, see Setting Diagram Properties.

See Also

- Adding Text Blocks
- Changing the Model Background

Adding Text Blocks

Use text blocks to insert useful information into the diagram. You can add multiple text blocks at specific points throughout the diagram to add clarity and help with organization. When you no longer need to use a text block in your diagram, you can delete the text block.

1. Choose Insert > Text Block.
2. In the *Data Model Window*, click where you want to place the text block, enter the text you want to display, and then right-click to deselect the *Text Block* creator.

Notes

- To resize the text box, select the text box, grab a sizing handle and drag to the desired position.
- Change the font attributes by right-clicking the Text Block and then clicking Colors & Fonts.

See Also

- Adding and Editing Lines
- Adding and Editing Title Blocks

Changing Data Model Layout

ER Studio Data Architect offers two ways to change the layout of your data models:

- Auto Layout that redraws your data model according to preset styles.
- The Layout Properties Editor that allows you to further customize the preset styles by setting parameters such as spacing, packing, tiling, and labeling conventions.

The following will help you change the layout of a data model:

- Redraw a Data Model Using Auto Layout
- Notes

Redraw a Data Model Using Auto Layout

1. Open a data model.
2. From the *Alignment* toolbar, select an auto layout option.



ER Studio Data Architect redraws the data model in the Data Model Window according to the rules of the preset style chosen.

Notes

The following describes the auto layout preset styles shown above, in order from left to right:

- *Circular Layout*: The circular layout reveals the dependencies and relationships between the data model objects.Emphasizes group structure by rearranging the data model into one or more circular patterns. This layout groups related entities into clusters and each cluster is grouped into circles according to the logical interconnection of the clusters and in a manner that minimizes the number of lines that cross the middle of the data model.
- *Hierarchical Layout*: Organizes entities in a hierarchical pattern based on the direction of the relationship between the entities. Hierarchical diagrams can display circular relationships, where the relationship path starts and ends on the same entity.
  Note: The Hierarchical Layout is the only type that represents data flows in a consistent direction.
- *Orthogonal Layout*: Useful for data models where inherent hierarchical structures are unimportant but the relationship orientation and ease of navigation is important. The orthogonal layout rearranges the data model into square-line, rectangular patterns that use only horizontal and vertical line routing. Orthogonal Layouts have few crossings and allow minimal stretching of entities that have a high number of incident relationships. Diagrams with this layout never overlap between entities or between entities and non-incident relationships.

- *Symmetric Layout*: This layout provides a symmetrical pattern centered around a single entity where peripheral entities move to the edge of the diagram. Symmetric Layouts naturally expose ring and star diagrams, producing clear data model diagrams that highlight the natural symmetry of many data models. Symmetric Layouts produce congruent data models of isomorphic graphs (graphs with the same structure), distribute entities uniformly, and create diagrams with few relationship crossings.
- *Tree Layout*: Use a tree layout for data models that contain a unique predecessor relationship. The tree layout rearranges your diagram into a tree pattern with parent and child entities, producing a data model that contains a root entity and only one unique path from the root entity to any other entity. Tree layouts can have branches and siblings where parent to child relationships are inherent.
- *Global Layout*: Rearranges your data model according to the style chosen for the target data model.
- *Incremental Layout*: Rearranges your data model according to the style chosen for the target data model without disturbing styles previously applied to pre-existing objects.

See Also

- Customizing the Layout
- Setting the Relationship Notation
- Manipulating Relationship Lines

Enforcing Data Integrity Using Check Constraints

The Check constraint evaluates the data as it is entered and only if the check evaluates to true is the entry allowed. The constraint entered for an entity is used when creating the physical model and can be seen in the table's DDL.

1. Double-click the entity or table you want to create a check constraint for.
2. On the *Rule/Constraints* tab of the Entity or Table Editor, name the constraint and then enter the constraint in the text box.

The constraint is added to the CREATE TABLE statement in the Table's DDL in the format:

CONSTRAINT (constraint_name) CHECK (constraint_text)

where

constraint_name is the name of the constraint

constraint_text is the text entered in the text box

- Constraints are created in the Data Dictionary, in the Rules folder. For more information, see Promoting Data Integrity Through Rules.

See Also

- Promoting Data Integrity Through Rules
- Enhancing Data Integrity Using Defaults

Checking Out a Concurrent License for Offline Usage

In an environment where the Embarcadero License Center (ELC) is used to manage concurrent ER Studio Data Architect licenses you will need to check out the license in order to use it while disconnected from your network. Depending on the settings specified by your ELC Administrator, you can check out a license up to a maximum of 30 days (720 hours).

1. Select *Help > Checkout License*.
2. In the *Checkout License from Server* dialog that appears, enter the number of hours for which you want to use the license offline, and then click *OK*..
   Now, when you are disconnected from the ELC server, your copy of ER Studio Data Architect will be licensed.
3. When you next connect to the network, you can return your license to the license pool by selecting Help > Checkin License.

Your license will then be available for other users to use when you quit ER Studio Data Architect.

See Also

- Licensing
- Administrator Guide

Developing the Logical Model

A logical model is developed before the physical model. It addresses the business and functional requirements of systems development. The logical design allows you to determine the organization of the data that you need to store in a database before you create the database; it serves as a blueprint.

To design the most effective data model possible, you should focus on the logical design before developing the physical design. Both the logical and physical design processes are complex so it is best to separate rather than to mix the two. A sound logical design should streamline the physical design process by clearly defining data structures and the relationships between them.

A data model can be extremely useful for other things besides creating databases, though that is generally its primary purpose. In systems development, the goal is to create an effective database application that can support some or all of your enterprise. However, a data model of your business can help you define operational aspects of your business that you might otherwise overlook. Also, a well-defined data model that accurately represents your business can be helpful in orienting employees to goals and operations. The data model can also serve as an invaluable communications tool for both internal and external constituents.

For more information, see the following topics:

- Logical Design Concepts
- Creating and Editing Entities and Tables
- Creating and Editing Keys
- Associate Similar Objects Using User-Defined Mappings
- Working with Relationships
- Working with Universal Mappings
- Validating the Model
- Creating and Editing Database Users
- Creating and Editing Database Roles
- Creating and Editing SQL Procedures
- Customizing Datatype Mappings

See Also

- Developing the Physical Model
- Common Tasks

Logical Design Concepts

To avoid bugs, rewrites, and, in some instances, failed projects, ensure critical design issues have been fully identified and addressed. In order to realize the benefits of data modeling, you must commit to certain practices that require a more thorough design cycle in order to realize a more efficient development cycle. A sound design can streamline the development process and improve the quality of your data models and ultimately improve the performance of your database.

Because most people involved in data modeling are database administrators or developers, there is a tendency to approach data modeling as a straightforward database design exercise. While you ultimately implement a data model as a database design, you should not dismiss the opportunity to re-examine fundamental assumptions about your business and the rules that govern it. At the design stage the focus should be on business problems, not the technical details. In the process of developing the logical model, you might discover a simpler approach to implementing a database design that better accommodates growth and change.

The following topics provide guidelines for the logical design process that should help you achieve better results:

- Object Name Rules describes how to name the entities and attributes in the logical design.
- Logical Design Documentation describes the importance of providing definitions for entities, attributes, and relationships.
- Normalization briefly describes the importance of normalizing your logical design before transforming it to a physical design.

See Also

- Developing the Physical Model
- Common Tasks
- Validating the Model

Logical Design Documentation

To ensure your ideas are understandable, document the meaning of your entities, relationships and attributes. A well-documented data model is easier to maintain and will probably find greater use in your organization.

Provide a definition for each entity in the logical model, even when their meanings seem obvious. Sometimes, this can reveal an incomplete design or logical inconsistencies. In addition to defining the entities, define all attributes in your logical design. Detailed documentation is invaluable to database developers and administrators who need to understand the design at its most basic level.

In addition to defining entities and attributes, you should also define the relationships in the logical model. Since relationships are used to enforce business rules, it is especially important to document their meaning. Relationships can be defined using verb phrases, names, and definitions. Verb phrases can clarify the business rules that govern the interaction between a pair of entities, while a name and definition can provide additional details about the relationship.

See Also

- Normalization
- Object Name Rules

Object Name Rules

Naming entities and attributes deserves serious attention. Long after you implement a system, the names you give to entities and attributes persist. You should not sacrifice clarity for brevity. Names should be descriptive yet general enough to represent all members. Several naming rules should be followed in order to ensure that your design is as clear as possible. Consider the following naming rules:

- *Singular Form*: Entity and attribute names should be singular (for example, Customer, not Customers), even though they can have many members.
- *Name Length*: The IDEF1X methodology permits name lengths of up to 65 characters, which is more than adequate for naming objects properly. However, most databases support maximum table and column name lengths of only 18 or 30 characters. While you can enter longer names, the Validate Logical or Physical Models function will check length violations before DDL generation.
- *Case*: When naming your objects, you have the choice of using upper, lower or mixed case. Perhaps the easiest format for people to read and understand is mixed case, capitalizing the first letter of every word (for example, CustomerAddress).

See Also

- Normalization
- Logical Design Documentation

Normalization

Normalization helps eliminate redundancy and streamline your logical design. A normalized database design is well-organized, even elegant. It is very important to complete the normalization process before transforming the logical design to a physical design; doing so allows a more efficient implementation of the physical design on an actual database.

Normalization is the process of separating data into multiple, related tables. A normalized database design is characterized by multiple entities with relatively few attributes in each; an unnormalized design is characterized by few entities with relatively many attributes in each. Normalizing the logical design helps eliminate certain types of redundancy and incompleteness in the model.

Part of the normalization procedure entails moving certain sets of repeating data from one entity to its own entity, then creating a relationship between the two. For example, consider the development of a data model for a university to store information about alumni. One entity can hold information about alumni, like their names, degrees they have earned, and the year they graduated. If some alumni have earned more than one degree from the university, then degree information becomes a set of repeating data. In the normalization process, the degree information is removed from the entity and placed in a separate entity. Next, a relationship is created between the first entity and the new entity so that information in each entity is associated with information in the other.

While working with logical data models, a data modeler should understand the rules (or forms) of normalization to produce a good data model.

The following defines the normalization forms:

- *Eliminate Repeating Groups (1NF)*: A separate table exists for each set of related attributes and each table has a primary key.
- *Eliminate Redundant Data (2NF)*: Attributes depending on only part of a multi-valued key is removed it to a separate table.
- *Eliminate Columns Not Dependent On Key (3NF)*: Attributes not contributing to a description of the key, are removed to a separate table.
- *BCNF. Boyce-Codd Normal Form (BCNF)*: Non-trivial dependencies between candidate key attributes are separated out into distinct tables.
- *Isolate Independent Multiple Relationships (4NF)*: No table contains two or more 1:n or n:m relationships that are not directly related.
- *Isolate Semantically Related Multiple Relationships (5NF)*: There may be practical constraints on information that justify separating logically related many-to-many relationships.
- *Isolate Semantically Related Multiple Relationships (ONF)*: A model limited to only simple (elemental) facts, as expressed in Object Role Model notation.
- *Domain-Key Normal Form (DKNF)*: A model free from all modification anomalies.

- All normal forms are additive, so if a model is in 3rd normal form, it is also in 2nd and 1st normal form.

See Also

- Logical Design Documentation
- Object Name Rules

Creating and Editing Entities and Tables

Entities are logical model objects and represent real objects, such as people, places or things, that are relevant to your enterprise. Entities store data about particular objects that users can query or modify. In their physical implementation, entities become tables, which are the focal point of any database.

The table is the basic unit of a physical data model. Embarcadero ER Studio Data Architect facilitates the smooth conversion of databases into physical models for deployment on an SQL database platform. You can populate your data model with tables by importing or reverse-engineering an existing database, generating a new physical model from a logical model, or by adding individual tables as needed. You can make changes to a table at any time using the Table Editor. You can also edit table Columns and Indexes using the Column Editor and Index Editor. respectively.

When you have finished work on an entity or table you can click the ellipsis at the top left-hand corner of the Entity Editor or Table Editor to switch to another entity or table respectively.

Because the entity and table are so closely related, the options available in their editors are often common to both objects.

This section is comprised of the following topics:

Creating and Editing Entities

Creating and Editing Tables

Creating and Editing Attributes and Columns

See Also

- Developing the Physical Model
- Common Tasks
- Creating and Editing Keys

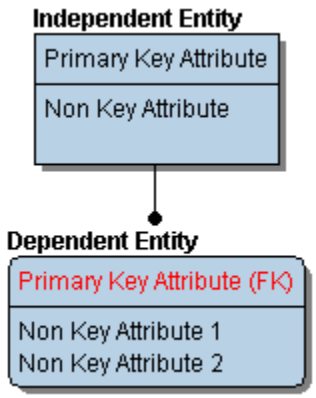
Creating and Editing Entities

Entities are the starting point for a data model. The relevant properties or characteristics of an entity are represented by attributes. Entities are basically an aggregation of attributes, and therefore we recommend that you are clear on the scope and purpose of each entity before worrying about its attributes. Once you have thought carefully about an entity's purpose, then its attributes should become apparent.

Supertypes and subtypes are one of the more advanced topics in entity-relationship modeling. For more information, see Creating Subtype Cluster Relationships.

ER Studio Data Architect represents entities in the Data Model Window as boxes. In the default display mode, entity names appear above the entity boxes with the primary key and non-key attributes showing inside the entity box, separated by a line.

ER Studio Data Architect entity boxes have either square or rounded corners. Entity boxes with square corners are independent entities and those with rounded corners are dependent entities. In IDEF1X, an entity becomes dependent when it becomes the child entity in any identifying relationship. Based on this simple rule, ER Studio Data Architect automatically maintains entity type. An entity is always created as an independent entity because it starts with no relationships attached to it. An entity remains independent until it becomes a child in any identifying relationship.



1. **To create a new entity**, in the *Data Model Explorer* expand the Logical *Main Model*, right-click the *Entities* node, and then click *New Entity*. ===

1. **To define or edit entity details**, double-click the entity to launch the *Entity Editor*.

1. Define the entity as required and then click *OK*.

The following helps you complete the options of the *Entity Editor*:

All tabs | Attributes/Columns tab | Keys tab
Relationships tab | Definition tab | Note tab
Where Used tab | User-Defined Mappings tab | Constraints tab
Dependencies tab | Permissions tab | Naming Standards tab
Data Lineage tab | Security Information tab | Attachment Bindings tab
Notes

All tabs

Logical Only: When selected, the entity will not be implemented as a table when generating a physical model, and will be ignored when comparing the logical model with a physical model.

Attributes/Columns tab

Add, edit, or delete attributes/columns. You can also change the list order of the attributes/columns, which changes their corresponding placement in the entity/table. You can also add an attribute/column to the primary key for the entity/table. The tabs at the bottom of the Attributes/Columns tab apply only to the attribute/column selected. For more information, see Creating and Editing Attributes and Columns.

Keys tab

Add, edit, or delete alternate keys and inversion entries. Provides access to the Key Editor. For more information, see Creating and Editing Keys.

Relationships tab

View any Relationships, if any, that exist between the entity/table and any other entities./tables. Information about the relationships include the relationship type and the Foreign Keys. On this tab, you can double-click a related entity/table to open the entity/table editor for that entity/table. For more information, see Creating and Editing Relationships.

Definition tab

Enter or edit a definition for the entity/table.

Note tab

The notes added here are included as part of the HTML report when you generate a report for the model. You can format the notes using standard HTML tags. For information about reports, see Generating RTF and HTML Model Reports.

Where Used tab

Shows which logical and physical models and submodels implement the entity. Also shows any existing user-defined mappings (UDM) for this entity/table. UDMs tie data together that can be conceptually related but not necessarily with concrete transactional relationships. UDMs are not used when comparing data models or when generating SQL.

ER/Studio Team Server data sources related to the current model are listed under the *Related Data Sources* node. A catalog and/or owner, if available, are listed as properties underneath the data source. Related data sources are only visible if you are logged in to ER/Studio TS. Similarly, related data sources are only visible for models in a Repository diagram and is synchronized in Connect.

Use the Definition tab of the User-Defined Mapping editor to describe the relationship between entities that is created using user-defined mappings. For more information, see Associate Similar Objects Using User-Defined Mappings.

User-Defined Mappings tab

Shows any existing user-defined mappings (UDM) for this entity/table. UDMs tie data together that can be conceptually related but not necessarily with concrete transactional relationships. UDMs are not used when comparing data models or when generating SQL. Use the Note tab to describe the relationship between entities that is created using user-defined mappings. For more information, see Associate Similar Objects Using User-Defined Mappings.

Constraints tab

Add, modify, or delete a constraint. Check constraints are used to ensure the validity of data in a database and to provide data integrity. For more information, see Enforcing Data Integrity Using Check Constraints.

Dependencies tab

Displays views that use the attributes/columns of current entity/table. For information on Views, see Creating and Editing Database Views.

Permissions tab

Sets access roles and user permissions for the entity/table. Keep in mind that the more specific permissions are, the more time you may have to spend maintaining and updating them. The Roles and Users must be previously assigned to appear in the Entity/Table Editor. For more information, see Creating and Editing Database Roles and Creating and Editing Database

Users. For more information, see Granting and Revoking Permissions to Modify Database Objects.

- *Column Permissions*: Sets access roles and user permissions for the attribute/column. Keep in mind that the more specific permissions are, the more time you may have to spend maintaining and updating them.

Naming Standards tab

Controls User and Object Permissions to modify the affected item. When a Naming Standards Template is bound to an object, then that template is used instead of the Naming Standards Template selected in the Naming Standards Utility or the Generate Physical Model utility. This allows you to apply different naming standards to different objects when portions of a model require different versions of a standard. For example, some objects already exist in the database and you do not want to apply a newer naming standard to them. The *Freeze Names* option when selected prevents any naming standards applied from changing the name of the entity selected. For more information, see Enforcing Naming Standards Using Naming Standards Templates.

Data Lineage tab

Maps the flow of data from source to target entities and attributes, or tables and columns, in the model. The Data Lineage tab appears in the entity and table editors. At the attribute level, you can add transformation logic for source/target attribute mapping. If a Data Movement Rule has been bound to an entity, you can override the bound value from this tab by double-clicking the row containing the Dave Movement Rule and then entering the new value in the Value Override Editor. For more information, see Data Lineage Workflow.

Security Information tab

Displays the current security information as defined in the Data Dictionary. You can add or edit security reporting information in by double-clicking the field you want to change. To set the model-wide default value for a property, go to the Data Dictionary tab and expand the Data Security Information Node. For more information, see Enforcing Security Using Data Security Types and Properties.

Attachment Bindings tab

Bind an external piece of information, or attachment to the entity. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary and must be applied to the entity before they will display on this tab. For more information, see Attaching External Documents to the Data Model.

Notes

- When you create an entity, it automatically appears as an independent entity with square corners. As relationships are drawn associating the entity with others, ER Studio Data Architect automatically maintains its status as an independent or dependent entity.
- You can name entities as you add them by using on-screen editing. Either type a name for an entity immediately after adding it to the diagram, or press the SHIFT key, click the default entity name to enable the entity name field, and then type a name. Click the diagram background to save the name and deactivate on-screen editing. For information on entity naming rules and conventions, see [[Best Practices]
- The status bar at the bottom right of the application window displays the number of entities in the selected model.
- You can specify naming rules, such as maximum length, case, and synchronization level between entities and tables on the *Logical and Name Handling* tabs of the *Options Editor*, which you can access by choosing Tools > Options > Logical.
- You can specify the display order of the attributes and primary keys of any entity on the Display tab of the Options Editor, which you can access by choosing Tools > Options > Logical.
- You can customize the information that pops up when you hover the mouse over an entity by selecting an option from the list that appears when you choose View > Cursor Popup Help Options > Entity Help.
- By default, ER Studio Data Architect automatically fits entities and views to display all content selected. However, you can resize any entity to any size by selecting the entity and then dragging the size handles. You can make global changes to how entities display by right-clicking the model, and selecting Diagram and Object Display Options. Then you can choose the display options desired in the Entity and View tabs of the Diagram and Object Display Options editor.
- For information on arranging entities in the Data Model Window, see Changing Data Model Layout.
- For information on changing the colors of entities, see Overriding Color and Font Settings for a Specific Object.

See Also

- Overview
- Customizing the Data Model

Creating and Editing Tables

The table is the basic unit of a physical data model. ER Studio Data Architect facilitates the smooth conversion of databases into physical models for deployment on an SQL database platform. You can populate your data model with tables by importing or reverse-engineering an existing database, generating a new physical model from a logical model, or by adding individual tables as needed. You can make changes to a table at any time using the Table Editor. You can also edit table Columns and Indexes using the Column Editor and Index Editor. respectively.

ER Studio Data Architect lets you add as many tables to your physical model as you need. When you add a table to the diagram, it creates as an independent table with square corners in the Data Model Explorer. You can create relationships between tables and create views as needed. As you draw relationships and foreign Keys, associating the table with others, ER Studio Data Architect automatically maintains the table status as an independent or dependent (child).

1. In the *Data Model Explorer*, expand a Physical model or submodel.
2. From the *Data Modeling* toolbar, click the *Table* tool, position the cursor over the desired location in the *Data Model Window*, and then left-click.
3. Right-click to revert to the *Selection* tool.
4. To define table details, double-click the table to launch the *Table Editor*.
5. Define the table as required and then click *OK*.

The following helps you complete the options of the *Table Editor*:

Columns tab	Dimensional tab	DDL tab
Indexes tab	Foreign Keys tab	Definition tab
Note tab	Where Used tab	User-Defined Mappings tab
Storage tab	Dimensions tab	Properties tab
Partition Columns tab	Distribute Columns tab	Distribution tab
Organization tab	Partitions tab	Overflow tab
Constraints tab	Dependencies tab	Capacity Panning tab
Permissions tab	PreSQL & Post SQL tab	Naming Standards tab
Compare Options tab	Data Lineage tab	Security Information tab
Attachment Bindings tab	Notes	

Columns tab

Add, edit, or delete columns. You can also change the order of the columns in the list which changes their corresponding column placement in the entity's table. You can also add an column to the primary key for the table. The tabs at the bottom of the Columns tab apply only to the column selected and are the same as those available for the Attributes tab in the Entity Editor. For more information, see Creating and Editing Columns.

Dimensional tab

This tab is available for Dimensional Models only. Lets you set the dimensional type of the table and the data type.

- *Override Automatic Designation*: If selected, the table's type will not be changed when the Automatic Table Type Identification process is run on a related table.
- *Run Automatic Table Type Identification*: If selected, when you change the Dimensional Model Table Type and then click OK to close the Table Editor, ER Studio Data Architect automatically changes the type of other tables to reflect the change, using the same logic as it does when creating a new dimensional model from a logical model. For example, if you change a table's type from Dimension to Fact, its parent's type can change from Snowflake to Dimension.

DDL tab

Displays the DDL to create the table. The DDL shown here includes code to reproduce the table including all its properties, as defined on the tabs of the Table Editor. You can edit the code if you like. You can customize the DDL that appears to include only include the options you want. For more information, see Customizing Table DDL.

Indexes tab

Add, edit, or delete indexes. Provides access to the Index Editor. For more information, see Creating and Editing Indexes.

Foreign Keys tab

View any relationships that exist between the selected table and other tables in the model. Information included about the relationships include the relationship type and the Foreign Keys. For more information, see Creating and Editing Keys.

Definition tab

Enter or edit a definition for the table. If the target database supports it, ER Studio Data Architect adds this definition as a table comment when generating SQL code.

Note tab

The notes added here are included as part of the HTML report when you generate a report for the model. You can format the notes using standard HTML tags.

Where Used tab

Shows which logical and physical models and submodels implement the table. Also shows the denormalization mapping history of the table. From the Where Used tab you can edit denormalization mappings and user-defined mappings. For more information, see Associate Similar Objects Using User-Defined Mappings and Viewing and Editing Denormalization Mappings.

User-Defined Mappings tab

Shows any user-defined mappings (UDM) for this table. UDMs tie data together that can be conceptually related but not necessarily with concrete transactional relationships. UDMs are not used when comparing data models or when generating SQL. Use the Note tab to describe the relationship between entities that is created using user-defined mappings. For more information, see Associate Similar Objects Using User-Defined Mappings.

Storage tab

Lets you select various storage options depending upon the selected database platform. For more information, see Defining Table Storage.

Dimensions tab

Available for the IBM DB2 for LUW 9.x (and later versions) platform. This tab supports multi-dimensional clustering (MDC) tables, a table type that is advantageous in certain application areas. The MDC table uses block-based indexes and can be physically clustered on more than one key or dimension simultaneously.

Click Add to bring up the Add Dimension dialog where you can choose from a list of available columns in the table to add to a dimension. You can create multiple dimensions for a table.

You can add the columns one at a time by adding a column and then clicking OK and then clicking the Add button again on the Dimensions tab. This creates a single-key dimension.

You can also add the columns as a comma separated list by adding multiple columns on the Add Dimension dialog, changing their order as required and then clicking OK. This creates a multi-key dimension.

You should carefully consider the page and extent size of the table because once the table is created you cannot change the table storage parameters of an MDC table without dropping and recreating the tablespace.

Properties tab

Select the table type storage engine you want. This tab is available only for MySQL.

- *HEAP*: Also know as Memory. Provides in-memory tables. Handles non-transactional tables.
- *INNODB*: Provides transaction-safe tables.
- *MYISAM*: Manages non-transactional tables. Provides high-speed storage and retrieval, as well as full text searching capabilities.

Partition Columns tab

Lets you select partitioning options for the selected table. Partitioning a table can enhance query and I/O performance. For more information, see Table Partition Columns for IBM DB2 for LUW 5.x, 6.x, 7.x, 8.x

Distribute Columns tab

For IBM DB2 for LUW 9.x (and later versions). Lets you select the columns to use for the distribution key, which can enhance query and I/O performance. For more information, see Distributing Columns for IBM DB2 for LUW 9.x (and later versions).

Distribution tab

For the Netezza and GreenPlum platforms. Lets you select the columns to use for the distribution key, which can enhance query and I/O performance. For more information, see Distributing Columns for Netezza and Distributing Columns for GreenPlum.

Organization tab

For the Netezza 6.0 (and later versions) platform. Lets you select up to four columns to use for the organizing key, which can enhance query and I/O performance.For more information, see Organizing Columns for Netezza.

Partitions tab

Lets you select the columns to include in a partition and reorder the selections. Provides access to the Partitions Editor. For more information, see Partitioning a Table.

Overflow tab

Lets you specify how to store the overflow records when a row is updated and no longer fits on the page where it was originally written. For more information, see Define Table Overflow Options.

Constraints tab

Add, modify, or delete a constraint. Check constraints can ensure the validity of data in a database and provide data integrity. For more information, see Enforcing Data Integrity Using Check Constraints.

Dependencies tab

Add, modify, or delete trigger, function, and procedure dependencies that are bound to the table. These dependencies must be met before the data can be loaded into the table. Dependencies are SQL code such as functions, triggers, and procedures. This tab displays views that use the attributes/columns of current entity/table For information on Views, see Creating and Editing Database Views. For more information on other dependencies, see Creating and Editing Functions, Creating and Editing Triggers, and Creating and Editing Procedures.

Capacity Panning tab

Assists in planning the space required to accommodate database growth. For more information, see Planning for and Predicting Database Growth.

Permissions tab

Sets access roles and user permissions for the table. Keep in mind that the more specific permissions are, the more time you may have to spend maintaining and updating them. The Roles and Users must be previously assigned to appear in the View Editor. For more information, see Creating and Editing Database Roles and Creating and Editing Database Users.

PreSQL & Post SQL tab

Lets you enter SQL to be applied before or after the CREATE OBJECT statement. The PreSQL and PostSQL scripts entered here are included in the script when you generate the physical database.

Naming Standards tab

Controls User and Object Permissions to modify the affected item. When a Naming Standards Template is bound to an object, then that template is used instead of the Naming Standards Template selected in the Naming Standards Utility or the Generate Physical Model utility. This allows you to apply different naming standards to different objects when portions of a model require different versions of a standard. For example, some objects already exist in the database and you do not want to apply a newer naming standard to them. The *Freeze Names* option when selected prevents any naming standards applied from changing the name of the table selected.For more information, see Enforcing Naming Standards Using Naming Standards Templates.

Compare Options tab

Select the differences you want the Compare and Merge Utility to ignore.

Data Lineage tab

Maps the flow of data from source to target tables and columns in the model. The Data Lineage tab appears in the entity and table editors. At the column level, you can add transformation logic for source/target column mapping. If a Data Movement Rule has been bound to an entity, you can override the bound value from this tab by double-clicking the row containing the Dave

Movement Rule and then entering the new value in the Value Override Editor. For more information, see Data Lineage Workflow and Documenting Table/Entity Data ETL.

Security Information tab

Displays the current security information as defined in the Data Dictionary. You can add or edit security reporting information in by double-clicking the field you want to change. To set the model-wide default value for a property, go to the Data Dictionary tab and expand the Data Security Information Node. For more information, see Enforcing Security Using Data Security Types and Properties.

Attachment Bindings tab

Bind an external piece of information, or attachment to the table. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary and must be applied to the table before they will display on this tab. For more information, see Attaching External Documents to the Data Model.

Notes

- A new a table automatically appears as an independent table with square corners. As relationships are drawn associating the table with others, ER Studio Data Architect automatically maintains its status as an independent or dependent table.
- You can name tables as you add them by using on-screen editing. Either type a name for a table immediately after adding it to the diagram, or press the SHIFT key, click the default table name and then type a name. Click the diagram background to save the name and deactivate on-screen editing. For information on table naming rules and conventions, see Best Practices.
- The status bar at the bottom right of the application window displays the number of entities in the selected model.
- Specify naming rules, such as maximum length, case, and synchronization level between entities and tables on the Logical and Name Handling tabs of the Options Editor, which you can access by choosing Tools > Options > Logical.
- Specify the display order of the attributes and primary keys of any entity on the Display tab of the Options Editor, which you can access by choosing Tools > Options > Logical.
- You can customize the information that pops up when you hover the mouse over an entity by selecting an option from the list that appears when you choose View > Cursor Popup Help Options > Entity Help.
- By default, ER Studio Data Architect automatically fits entities and views to display all content selected. However, you can resize any entity to any size by selecting the entity and then dragging the size handles. You can make global changes to how entities

display by right-clicking the model, and selecting Diagram and Object Display Options. Then you can choose the display options desired in the Entity and View tabs of the Diagram and Object Display Options editor.

- For information on arranging tables in the Data Model Window, see Changing Data Model Layout.
- For information on changing the colors of tables, see Overriding Color and Font Settings for a Specific Object.

See Also

- Overview
- Customizing the Data Model

Creating and Editing Attributes and Columns

Attributes represent the relevant properties or characteristics of an entity. In the physical model, attributes are represented as table columns.

This section is comprised of the following topics:

- Creating and Editing Attributes
- Creating and Editing Columns

See Also

- Creating and Editing Entities
- Creating and Editing Tables

Creating and Editing Attributes

There are two types of attributes:

- *Identifiers*: Helps to identify an entity instance because it is all or part of the entity's primary key.
- *Descriptor*: A non-key attribute. Following the rules of normalization, if an attribute is not part of the primary key, then its only purpose is to describe each entity instance.

Attribute definitions together with relationships can determine or enforce business rules. You can define a particular set of valid values for any attribute of any entity. For example, you can define a valid range of salaries for a subset of employees for an attribute called Salary. This set of values is known as a domain. For more information, see Ensuring Consistent Domain Definitions Using User Datatypes.

If you define your entities carefully, then defining attributes should be relatively straightforward. When you first add attributes to an entity, be sure to name your attributes appropriately. Attributes can be native to an entity or inherited via an identifying or non-identifying relationship. To promote consistency, ER Studio Data Architect only lets you edit the properties of an entity's native attributes. To change the properties of a foreign key of a child entity, you must edit the attribute in the parent entity. For information on attribute naming rules and conventions, see Best Practices.

1. In the *Data Model Explorer*, expand the Logical *Main Model*, right-click the *Entities* node, and then select the entity to which you want to add attributes.
2. On the *Attributes* tab of the *Entity Editor*, click *Add*.
3. Define the attribute as required and then click *OK*.

The following helps you complete the options of the *Attributes* tab:

Main page	Datatype tab	Rule/Constraint tab
Definition tab	Default tab	Rule/Constraint tab
Definition tab	Notes tab	Where Used tab
User-Defined Mappings tab	Reference Values tab	Naming Standards tab
Compare Options tab	Data Lineage tab	Security Information tab
Attachment Bindings tab	Data Movement Rules tab	Notes

Main page

- *Domain Name*: Lists all defined domains. You can bind any available domains to the attribute. To change the domain name, you must first select Override Bound Data on the Datatype tab.

- *Create Domain*: Lets you create a new domain that is automatically added to the Data Dictionary. Enter a new Domain Name and then select Create Domain. When you click OK to save the changes, the new domain is created in the Data Dictionary.
- *Default Attribute Name*: The column name is by default the attribute name. If you want a different column name, type a new one in the box. ER Studio Data Architect uses this name when generating the physical model.
- *Logical Rolename*: This option is only available when there is an overlapping foreign key in a logical model and the Synchronize Column Rolename with Logical Rolename option is not selected. Use this when the name of the child attribute differs from the parent attribute. In the logical model, ER Studio Data Architect distinguishes between the logical rolename and the column rolename. ER Studio Data Architect uses the column rolename when generating the physical model.
- *Hide Key Attribute*: If selected, hides the key in the child entity. Commonly used to support partial migration of PK columns when you do not want to migrate primary key columns into child tables. ER Studio Data Architect will not generate a foreign key because most database systems require that FKs have the same number of columns as the child. By selecting this option, you will 'remove' the column from the child table so that it won't show up in any child tables further down the FK chain, DDL generations or comparisons to the database. It will display as unavailable when editing the child table, but you can make it available again later.
- *Logical Only*: If selected, the entity will not be implemented as a table when generating a physical model, and will be ignored when comparing the logical model with a physical model.
- *Synchronize Column Rolename with Logical Rolename* Available when the attribute selected is a foreign key. This option affects name modification in the entity or table editor, on-screen name editing, and name modification in the Data Model Explorer. Lets you change the role names of foreign attributes or columns to the same names as other attributes or columns in the same entity or table. Duplicates will be unified when the user ends the edit session.
- *Add to Primary Key?*: When chosen, adds the selected attribute to the Primary Key, which is used to index the entity enabling faster data retrieval.
- *Edit Foreign Key Datatype*: When selected for a child object, you can edit the Datatype values of the foreign key. When this option is selected, changes made to this relationship in the parent are not propagated to the child until you deselect this option. This preserves the foreign key overrides.

Datatype tab

- *Override Bound Data*: Overrides datatypes on an attribute basis (for supported types: SQL Server, Sybase, IBM DB2 for LUW, and IBM DB2 for OS/390). Used to handle any exceptions while maintaining the binding to the same domain. For example, there is a VARCHAR domain on two separate DBMS platforms. The character domain's primary use can be on Oracle. You specified it as VARCHAR (which will be VARCHAR2 in Oracle). Using this option, you can have a DB2 model where it is implemented it as CHAR. You

can customize the domain's implementation to accommodate the specific platform, without losing the bindings between DB2 columns and the domain.

- *Identity Properties*: For numeric datatypes for which nulls are not allowed and for whose datatype supports identity properties, you can specify identity properties by selecting the Identity Column option and then specifying a seed and increment value. The seed and increment values are used to automatically generate sequential numbers for each row in the table beginning with the Seed value and incrementing by the amount specified by Increment. The default seed and increment values are one but can be changed.

Default tab

If a default value has not already been bound to the entity attribute (Data Dictionary > Defaults > Edit Default Definition > Binding Information), you can type a Declarative Default or select Override Bound Data and then choose another value from the Default Binding list. The declarative default will appear in the SQL produced when you select Generate Database from the physical model. If a default value attribute has been previously bound to the entity, you can override the default by selecting another default from the Default Binding list. For more information, see Working with the Data Dictionary. If the attribute selected is a primary key of the entity, you have the option to select the option to Propagate Default to Child Keys.

Rule/Constraint tab

Lets you select a rule binding from the Data Dictionary or manually type in a rule binding. The exact text in the declarative box gets inserted into the physical model DDL. Add, modify, or delete a constraint. Check constraints are used to ensure the validity of data in a database and to provide data integrity. For more information, see Enforcing Data Integrity Using Check Constraints. If the information is grayed out (read-only) it is because a domain is being used and the definition, datatype, rule, constraint or default is inherited from the domain.

Definition tab

Enter or edit a definition for the attribute. If the target database supports it, ER Studio Data Architect adds this definition as an attribute comment when generating SQL code.

Notes tab

The notes added here are included as part of the HTML report when you generate a report for the model. You can format the notes using standard HTML tags.

Where Used tab

Shows which logical and physical models and submodels implement the attribute. Also shows the denormalization mapping history of the table. From the Where Used tab you can edit mappings and user-defined mappings. For more information, see Associate Similar Objects Using User-Defined Mappings.

User-Defined Mappings tab

Shows any existing user-defined mappings (UDM) for this attribute. UDMs tie data together that can be conceptually related but not necessarily with concrete transactional relationships. UDMs are not used when comparing data models or when generating SQL. Use the Notes tab to describe the relationship between entities that is created using user-defined mappings. For more information, see Associate Similar Objects Using User-Defined Mappings.

Reference Values tab

For information, see Defining Valid Attribute Data Using Reference Values.

Naming Standards tab

Controls User and Object Permissions to modify the attribute. When a Naming Standards Template is bound to an object, then that template is used instead of the Naming Standards Template selected in the Naming Standards Utility or the Generate Physical Model utility. This allows you to apply different naming standards to different objects when portions of a model require different versions of a standard. For example, some objects already exist in the database and you do not want to apply a newer naming standard to them. The *Freeze Names* option when selected prevents any naming standards applied from changing the name of the attribute selected.For more information, see Enforcing Naming Standards Using Naming Standards Templates.

Compare Options tab

Lets you select which, if any, properties of the attribute to ignore when comparing this attribute to another using the Compare and Merge Utility. For more information, see Using the Compare and Merge Utility.

Data Lineage tab

Maps the rules from source to target entities and attributes in the model. At the attribute level, you can add transformation logic for source/target attribute mapping. You can map multiple source/targets to one attribute in the physical model. For more information, see Documenting Column/Attribute ETL and Data Lineage Workflow.

Security Information tab

Displays the current security information as defined in the Data Dictionary. You can add or edit security reporting information in by double-clicking the field you want to change. To set the model-wide default value for a property, go to the Data Dictionary tab and expand the Data Security Information Node. For more information, see Enforcing Security Using Data Security Types and Properties.

Attachment Bindings tab

Bind an external piece of information, or attachment to the attribute. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to

the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary and must be applied to the attribute before they will display on this tab. For more information, see Attaching External Documents to the Data Model.

Data Movement Rules tab

Displays the data movement rules that have be assigned to the attribute. Allows you to override the default rules assigned to the table/entity in the Data Movement Rules editor of the Data Dictionary. Here you can apply or remove data movement rules. The Data Movement rule could be something like 'Data for this table is updated after 30 days.' For more information, see Relating Source and Target Tables and Columns.

Notes

- Choosing the correct datatype is a critical data integrity and storage consideration. Select a datatype that is efficient for its intended use but properly accommodates the data it stores. Your choice of database platform dictates your choice of column datatypes because each has its own set of datatypes. For example, some databases have binary datatypes; others do not. Some provide support for blobs and text, but many do not. Finally, only a few databases support user-defined datatypes. ER Studio Data Architect can convert datatypes between different databases; however, some conversions can only be approximations.
- Another important consideration in specifying datatypes is determining the appropriate width, precision and scale, as applicable. You want to store data efficiently, but not at the expense of completeness, accuracy or precision.
- For information on datatype mapping, see Customizing Datatype Mappings. If you want to enforce datatypes across common columns, use Data Dictionary Domains. For more information, see Reusing Attribute Definitions Using Domains.
- When determining attribute and column names, make sure you consider the naming rules of the target database. Some key considerations are name length, avoiding the use of reserved words, and prohibited characters. Most databases restrict name lengths to 30 or 18 characters. All have reserved key words that cannot be used in names unless surrounded by quotes. Certain characters, such as a space, *, +, and % are also prohibited from being used in names.
- You can specify naming rules, the default datatype and whether NULL values are permitted for the attributes created with the default datatype on the Logical tab of the Options Editor by choosing Tools > Options > Logical.
- You can specify the order in which the attributes appear on the Display tab of the Options Editor by choosing Tools > Options > Display.
- You can specify the level of synchronization between the attribute and column names, choose Tools > Options > Name Handling.
- The status bar at the bottom right of the application window displays the number of entities in the selected model.

See Also

- Creating and Editing Database Dependent Objects
- Creating and Editing Columns

Creating and Editing Columns

Attributes represent the relevant properties or characteristics of an entity. In the physical model, attributes are represented as table columns. There are two types of attributes and columns:

- *Identifiers*: Helps to identify a entity/table instance because it is all or part of the entity's/table's primary key.
- *Descriptor*: A non-key attribute/column. Following the rules of normalization, if an attribute/column is not part of the primary key, then its only purpose is to describe each entity instance.

Attribute definitions together with relationships can determine or enforce business rules. You can define a particular set of valid values for any attribute of any entity. For example, you can define a valid range of salaries for a subset of employees for an attribute called Salary. This set of values is known as a domain. For more information, see Ensuring Consistent Domain Definitions Using User Datatypes.

If you define your entities carefully, then defining attributes should be relatively straightforward. When you first add attributes to an entity, be sure to name your attributes appropriately. Attributes can be native to an entity or inherited via an identifying or non-identifying relationship. To promote consistency, ER Studio Data Architect only lets you edit the properties of an entity's native attributes. To change the properties of a foreign key of a child entity, you must edit the attribute in the parent entity. For information on attribute naming rules and conventions, see Best Practices.

When determining column names, make sure you consider the naming rules of the target database. Some key considerations are name length, avoiding the use of reserved words, and prohibited characters. Most databases restrict name lengths to 30 or 18 characters. All have reserved key words that cannot be used in names unless surrounded by quotes. Certain characters, such as a space, *, +, and % are also prohibited from being used in names.

1. In the *Data Modeling Window*, double-click the table to which you want to add column.
2. On the *Columns* tab of the *Table Editor*, click *Add*.
3. Define the attribute as required and then click *OK*.

The following helps you complete the options of the *Columns* tab:

Main page | Datatype tab | Default tab
Definition tab | Notes tab | Where Used tab
User-Defined Mappings tab | Reference Values tab | Naming Standards tab
Compare Options tab | LOB Storage tab | Data Lineage tab
Security Information tab | Attachment Bindings tab | Data Movement Rules tab
Notes

Main page

- *Domain Name*: Lists all defined domains. You can bind any available domains to the column.
- *Attribute Name*: The attribute name is by default the column name. If you want a different column name, type a new one in the box. ER Studio Data Architect uses this name when generating the physical model.
- *Default Column Name*: The attribute name is by default the column name. If you want a different column name, type a new one in the box. ER Studio Data Architect uses this name when generating the physical model.
- *Logical Rolename*: This option is only available when there is an overlapping foreign key in a logical model and the Synchronize Column Rolename with Logical Rolename option is not selected. Use this when the name of the child attribute differs from the parent attribute. In the logical model, ER Studio Data Architect distinguishes between the logical rolename and the column rolename. ER Studio Data Architect uses the column rolename when generating the physical model.
- *Hide Key Attribute*: If selected, hides the key in the child table. Commonly used to support partial migration of PK columns when you do not want to migrate primary key columns into child tables. ER Studio Data Architect will not generate a foreign key because most database systems require that FKs have the same number of columns as the child. By selecting this option, you will 'remove' the column from the child table so that it won't show up in any child tables further down the FK chain, in DDL generations or in comparisons to the database. It will display as unavailable when editing the child table, but you can make it available again later.
- *Physical Only*: If selected, the table will not be implemented as a table when generating a physical model, and will be ignored when comparing the logical model with a physical model.
- *Synchronize Column Rolename with Logical Rolename:* Available when the attribute selected is a foreign key. This option affects name modification in the entity or table editor, on-screen name editing, and name modification in the Data Model Explorer. Lets you change the role names of foreign attributes or columns to the same names as other attributes or columns in the same entity or table. Duplicates will be unified when the user ends the edit session.

- *Add to Primary Key?*: Add the column as a primary key for indexing.

Datatype tab

- **Datatype list**: Click the list to select the appropriate datatype. The list of datatypes presented is dependent on the database platform selected. This discussion describes datatypes that need further explanation.
- **ROWVERSION datatype**: The *ROWVERSION* datatype exposes automatically generated, unique binary numbers in a database. It is an incrementing number, 8 bytes in size, that can be used to version-stamp table rows. A table can have only one rowversion column. Every time that a row with a *ROWVERSION* column is modified or inserted, the incremented database *ROWVERSION* value is inserted in the rowversion column. This property makes a *ROWVERSION* column a poor candidate for keys, especially primary keys. Any update made to the row changes the *ROWVERSION* value and, therefore, changes the key value.

**Note:** Rowversion is a synonym for timestamp. You can specify *ROWVERSION* as data type in ER Studio Data Architect, but the created column on MS SQL Server 2008 is *TIMESTAMP*.

- **TIMESTAMP datatype**: For DB2 LUW 9.x, and DB2 z/OS 9.x and 10.x. A generate *By Default* option is now available for a *TIMESTAMP* datatype if the column does not allow nulls (*NOT NULL*), and you have selected *Generated Column*.

For Teradata 13.x & 14.x. A PERIOD option is now available for **Timestamp, Time** and **Date**.

The following is an example of the SQL produced when these options are selected:

HireDate TIMESTAMP NOT NULL GENERATED BY DEFAULT FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP

- **INTERVAL datatype**: For Teradata databases, the Interval datatype now includes options to specify the units for the datatype.
- *Allow Nulls*: The *Allow Nulls* control corresponds to the platform-specific nullability option/argument for a column specification.
- **NUMBER datatype**: NUMBER or NUMBER(*) you must set the width and scale to *. A width of 0 will be converted to * when generating DDL.

Against Oracle (8 and higher) platforms, if an *Allow Nulls* value of *No* is selected, providing a *Constraint Name* results in a column definition that specifies an inline, named NOT NULL constraint.

- *Override Bound Data*: Overrides datatypes on an attribute basis (for supported types: SQL Server, Sybase, IBM DB2 for LUW, and IBM DB2 for OS/390). Used to handle any exceptions while maintaining the binding to the same domain. For example, there is a *VARCHAR* domain on two separate DBMS platforms. The character domain's primary use can be on Oracle. You specified it as *VARCHAR* (which will be *VARCHAR2* in Oracle). Using this option, you can have a DB2 model where it is implemented it as *CHAR*. You can customize the domain's implementation to accommodate the specific platform, without losing the bindings between DB2 columns and the domain.

- *Identity Property*: For numeric datatypes for which nulls are not allowed and for whose datatype supports identity properties, you can specify identity properties by selecting the Identity Column option and then specifying a Seed and Increment value. The seed and increment values are used to automatically generate sequential numbers for each row in the table beginning with the Seed value and incrementing by the amount specified by Increment. The default seed and increment values are one but can be changed.
- *Virtual*: Available for Oracle 11g or later platforms. The following describes the requirements for virtual columns as described by Oracle. Note, these restrictions are not enforced by ER/Studio.

Column restrictions

A column can be specified as virtual if the following criteria are met:

- The table containing the column is not index-organized, external, object, cluster or temporary.
- The virtual column cannot be an Oracle supplied datatype (e.g. Spatial Types, XML Types, Any Types and Media Types), a user-defined type, an object types, LONG, LOB, BLOB, RAW, REF or BFILE.

Expression restrictions

To specify a column as virtual, the *Virtual* option must be selected and you must specify an expression, such as (isnull('AW'+[dbo].[ufnLeadingZeros]([CustomerID]),'')) in the expression field to the right of virtual. The expression used on virtual columns must satisfy the following restrictions:

- It cannot refer to another virtual column by name
- It can only refer to other columns in the same table.
- It can refer to a deterministic user-defined function but then the virtual column can't be used as a partitioning key column.
- The result of the expression must be a scalar value.

When the *Virtual* option is selected, you can specify that the datatype of the column is displayed by selecting *Show Datatype*. When *Show Datatype* is not selected, then data type list, *Width*, *Scale*, *Allow Nulls* option and *Identity Property* are disabled. These options are available when *Show Datatype* is selected, however, the *Identity Property* options may not be available if the requirements are not met. See *Identity Property* above.

- *Edit LOB Segment*: If the specified datatype of the column is *BLOB*, *CLOB*, or *NCLOB*, you can click the *Edit LOB Segment* option to define LOB storage. For more information, see Oracle LOB Segment Storage Options.
- *Character Set*: For the Teradata platform, if the specified datatype of the column is CHAR, VARCHAR or CLOB, you can then specify the character set for the datatype. Click the list to choose the appropriate datatype. For the CHAR and VARCHAR datatypes, you

can choose DEFAULT, LATIN, UNICODE, GRAPHIC, KANJI1, KANJISIS. The CLOB datatype supports only LATIN and UNICODE character sets.

- *CASESPECIFIC*: For the Teradata platform, if the specified datatype of the column is CHAR or VARCHAR, you can choose the CASESPECIFIC option, which indicates the system should differentiate between upper-case and lower-case characters.
- *Compress*: For the Teradata platform, if the selected column is not a primary key and the appropriate datatype is selected, such as BIGINT, DATE or CHAR, you can choose to compress the columns to minimize the table size. Teradata does not support the compression of CHAR datatype with a length more than 256 characters. You can either enter the compress statement in the space provided or if the statement is longer, clicking More brings up the Compression Statement Editor where you can enter a more lengthy compression statement. You can compress up to 256 column values. For example, in a customer database you could compress the state names to reduce the table size. The Compression statement could look like,

COMPRESS ('New Hampshire', 'California', 'Alaska',...)

which would replace the state values in the table with a state code and store the state values in the table header.

**Note:** Teradata does not support compress for a column with these data types: INTERVAL, TIME, TIMESTAMP, VARBYTE, VARCHAR, CLOB, BLOB, CHAR(n) with n > 255. Also if the column is a primary key, it cannot be compressed.

Collect Statistics: (*COLLECT STATISTICS)* Collects demographic data for one or more columns of a table, computes a statistical profile of the collected data, and stores the profile in the data dictionary. Collecting full statistics involves scanning the base table, and sorting to determine the number of occurrences for each unique value. The Teradata Optimizer uses this profiling data when it generates its table access and join plans. Full statistics is generally recommended for relevant columns and indexes on tables with less than 100 rows per Access Module Processor (AMP).

This is the SQL generated when *Collect Statistics* has been selected in the *Table Editor* for a column in the table.

COLLECT STATISTICS ON table_1 COLUMN column_1;

This is the SQL generated when *Collect Statistics* has been selected in the *Index Editor* for an index in the table.

COLLECT STATISTICS ON table_1 INDEX unique_1;

Using Sample Collecting sampled full-table statistics is generally faster than collecting full table statistics, but may not produce the best table access and join plans. When you specify this option, Teradata will determine the appropriate percentage to the data to scan. The best choices for USING SAMPLE are columns or indexes that are unique or 95% unique.

This is an example of the SQL generated when *Collect Statistics* and *Using Sample* have been selected in the *Tale Editor* for a column in the table.

COLLECT STATISTICS USING SAMPLE ON table_1 COLUMN column_1;

This is the SQL generated when *Collect Statistics* and *Using Sample* have been selected in the *Index Editor* for an index in the table.

COLLECT STATISTICS USING SAMPLE ON table_1 INDEX unique_1;

- *RowGuidCol*: Lets you return the row global unique identifier column. When using the *ROWGUIDCOL* property to define a globally unique identifier column, consider that a table can have only one *ROWGUIDCOL* column, and that column must be defined using the unique identifier data type.
- *Computed Column*: If selected you can enter a formula to transform the column; for example, sale commission = 10% total sales.
- *Sparse*: Specify sparse if the column can have many *NULLS*. Specifying sparse for such columns can improve retrieval response. A sparse column is stored nulls do not take up any space; however, a sparse column with data in it requires a little more space than usual. This option will be available only if the datatype selected supports the sparse property.
- *Collate*: Lets you specify collations for each character string. ER Studio Data Architect will use the *COLLATE* clause with the *CREATE TABLE* or *ALTER TABLE* statement.
- *For Bit Data*: Lets you use binary SQL to specify a binary character array for character SQL types. IBM database platforms support columns with binary data types by defining *CHAR*, *VARCHAR*, *LONGVARCHAR*, columns with the *FOR BIT DATA* attribute.
- *LOB Unit*: If the datatype is *BLOB*, *CLOB*, or *DBCLOB*, lets you select the LOB Unit measurement. Specify K - for Kilobytes, M for Megabytes or G for Gigabytes.

Default tab

- *Default Binding*: If a default value has not already been bound to the entity attribute (Data Dictionary > Defaults > Edit Default Definition > Binding Information), you can type a Declarative Default or select Override Bound Data and then choose another value from the Default Binding list. The declarative default will appear in the SQL produced when you select Generate Database from the physical model. If a default value attribute has been previously bound to the entity, you can override the default by selecting another default from the Default Binding list. For more information, see Working with the Data Dictionary. If the attribute selected is a primary key of the entity, you have the option to select the option to Propagate Default to Child Keys.
- *Default Name*: Specifies a name for a declarative default when a declarative default is specified below, or when the *Override Bound Data* is selected.If you are using a bound default from the Data Dictionary, you cannot specify a name for an inline default
- *Declarative Default*: Specifies the default column value when a value is not explicitly specified during an insert. *DEFAULT* definitions can be applied to any columns except those defined as timestamp, or those with the *IDENTITY* property. Only a constant value, such as a character string; a system function, such as *SYSTEM_USER()* ; a *NULL* ; or a constraint name can be assigned to a *DEFAULT*.

Rules/Constraints: Lets you select a rule binding from the Data Dictionary or manually type in a rule binding. The exact text in the declarative box gets inserted into the physical model DDL. Add, modify, or delete a constraint. Check constraints are used to ensure the validity of data in a database and to provide data integrity. For more information, see Enforcing Data Integrity Using Check Constraints. If the information is grayed out (read-only) it is because a domain is being used and the definition, datatype, rule, constraint or default is inherited from the domain.

Definition tab

Enter or edit a definition for the column. If the target database supports it, ER Studio Data Architect adds this definition as a column comment when generating SQL code.

Notes tab

The notes added here are included as part of the HTML report when you generate a report for the model. You can format the notes using standard HTML tags.

Where Used tab

Shows which logical and physical models and submodels implement the column or attribute. Also shows the history of the column in terms of Denormalization Mappings. From the Where Used tab you can edit both the mappings and the user-defined mappings. For more information, see Associate Similar Objects Using User-Defined Mappings.

User-Defined Mappings tab

Shows any user-defined mappings (UDM) for this table. UDMs tie data together that can be conceptually related but not necessarily with concrete transactional relationships. UDMs are not used when comparing data models or when generating SQL. Use the Notes tab to describe the relationship between entities that is created using user-defined mappings. For more information, see Associate Similar Objects Using User-Defined Mappings.

Reference Values tab

Reference Values are attributes that define allowed data. They represent look-up table columns or code-value ranges or a rule or constraint applied to columns. For more information, see Defining Valid Attribute Data Using Reference Values.

Naming Standards tab

Controls User and Object Permissions to modify the affected item.When a Naming Standards Template is bound to an object, then that template is used instead of the Naming Standards Template selected in the Naming Standards Utility or the Generate Physical Model utility. This allows you to apply different naming standards to different objects when portions of a model require different versions of a standard. For example, some objects already exist in the database and you do not want to apply a newer naming standard to them. The *Freeze Names* option when selected prevents any naming standards applied from changing the name of the column selected. For more information, see Enforcing Naming Standards Using Naming Standards Templates.

Compare Options tab

Select the differences you want the Compare and Merge Utility to ignore.

LOB Storage tab

For HiRDB only. If the column datatype specified on the Datatype tab is *BLOB*, this tab becomes available where you can specify a large object storage location.

Data Lineage tab

Maps the rules that describe data movement from source data to target entities and attributes in the model. At the attribute level, you add transformation logic for source/target attribute mapping. You can map multiple source/targets to one attribute in the physical model. For more information, see Documenting Data Extraction, Transformation, and Load.

Security Information tab

Displays the current security information as defined in the Data Dictionary. You can add or edit security reporting information in by double-clicking the field you want to change. To set the model-wide default value for a property, go to the Data Dictionary tab and expand the Data Security Information Node. For more information, see Enforcing Security Using Data Security Types and Properties.

Attachment Bindings tab

Bind an external piece of information, or attachment to the column. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary and must be applied to the column before they will display on this tab. For more information, see Attaching External Documents to the Data Model.

Data Movement Rules tab

Displays the data movement rules that have be assigned to the attribute. Allows you to override the default rules assigned to the table/entity in the Data Movement Rules editor of the Data Dictionary. Here you can apply or remove data movement rules. The Data Movement rule could be something like 'Data for this table is updated after 30 days.' For more information, see Relating Source and Target Tables and Columns.

Notes

- Choosing the correct datatype is a critical data integrity and storage consideration. Select a datatype that is efficient for its intended use but properly accommodates the data it stores. Your choice of database platform dictates your choice of column datatypes because each has its own set of datatypes. For example, some databases have binary datatypes; others do not. Some provide support for blobs and text, but

many do not. Finally, only a few databases support user-defined datatypes. ER Studio Data Architect can convert datatypes between different databases; however, some conversions can only be approximations.

- Another important consideration in specifying datatypes is determining the appropriate width, precision and scale, as applicable. You want to store data efficiently, but not at the expense of completeness, accuracy or precision.
- For information on datatype mapping, see Customizing Datatype Mappings. If you want to enforce datatypes across common columns, use Data Dictionary Domains. For more information, see Reusing Attribute Definitions Using Domains.
- When determining attribute and column names, make sure you consider the naming rules of the target database. Some key considerations are name length, avoiding the use of reserved words, and prohibited characters. Most databases restrict name lengths to 30 or 18 characters. All have reserved key words that cannot be used in names unless surrounded by quotes. Certain characters, such as a space, *, +, and % are also prohibited from being used in names.
- You can specify naming rules, the default datatype and whether NULL values are permitted for the attributes created with the default datatype on the Logical tab of the Options Editor. Choose Tools > Options > Logical.
- You can specify the order in which the attributes appear on the Display tab of the Options Editor. Choose Tools > Options > Display.
- You can specify the level of synchronization between the attribute and column names on the *Name Handling* tab of the *Options Editor*. Choose Tools > Options > Name Handling.
- The status bar at the bottom right of the application window displays the number of entities in the selected model.

See Also

- Creating and Editing Database Dependent Objects
- Creating and Editing Attributes

Creating and Editing Keys

The purpose of keys in a database is to enforce unique values in an entity and provide a means of sorting the tables to access entity data faster.

In a relational database, there are four types of keys:

- *Primary Key*: A primary key or unique key is a candidate key to uniquely identify each row in a table. A primary key comprises a single column or set of columns where no two distinct rows in a table can have the same value (or combination of values) in those columns. Depending on its design, a table can have many unique keys but at most one primary key. A unique key must uniquely identify all possible rows that exist in a table and not only the currently existing rows, such as social security numbers. A primary key is a special case of a unique key in that for unique keys the implicit NOT NULL constraint is not automatically enforced, while for primary keys it is. The values in a unique key column may or may not be NULL. Unique keys as well as primary keys can be referenced by foreign keys.
- *Alternate Key*: Alternate keys are the equivalent to unique index or unique constraints in the physical model. You can enforce alternate keys with unique indexes or unique constraints, depending on the database platform
- *Inversion Entry*: An attribute or set of attributes that does not uniquely identify every instance of an entity, but which are frequently used for access. The database implementation of an inversion entry is a non unique index.
- *Foreign Key*: A primary key or non-key attribute that is inherited from another entity.

ER Studio Data Architect can reverse-engineer key definitions that are not primary key or unique constraints, from a database. You can create primary, inversion, or alternate keys using the Key Editor, which is accessible from the Entity Editor.

For more information, see the following:

- Create a Primary Key
- Create an Alternate Key or Inversion Entry
- Create a Foreign Key/Relationship

Create a Primary Key

1. In the *Data Model Explorer*, expand the Logical *Main Model*, expand the *Entities* node, and then double-click the entity to which you want to add keys.
2. *To change an existing attribute to a primary key*, in the *Entity Editor*, click an attribute you want to make a key and then click *Edit*.

To create a primary key based on a new attribute, the *Entity Editor*, click *Add* and define the entity properties as required.

1. On the *Attributes* tab, click *Add to Primary* and then click *OK* and exit the *Entity Editor*.

Create an Alternate Key or Inversion Entry

1. In the *Data Model Explorer*, expand the Logical *Main Model*, right-click the *Entities* node, and then double-click the entity to which you want to add keys.
2. *To change the properties of an existing attribute*, In the *Entity Editor*, click an attribute you want to make a key and then click *Edit*.

To create an alternate key or inversion entry based on a new attribute, in the *Entity Editor*, click *Add* and define the entity properties as required.

1. On the *Keys* tab, click *Add*.
2. In the *Key Editor*, type a name for the key and select the type of key you want to create.
3. From the list of *Available Keys*, which are all the attributes of the entity, double-click the attributes you want to add to the key.
4. You can reorder the attributes by selecting an attribute and then pressing Up or Down until the attribute is in the desired position within the list of Selected Keys.
5. Click *OK* to accept the attribute selections and then click *OK* again to affect the changes to the entity.

Attachment Bindings tab: Bind an external piece of information, or attachment to the key. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary. For more information, see Attaching External Documents to the Data Model.


Note:

- You can prevent a key from being implemented when generating a physical model or SQL, by selecting Logical Only on the Keys tab.
- To modify the properties of a foreign key attribute, you must alter the corresponding primary key column in the originating parent table.
- To preserve domain consistency, ER Studio Data Architect only lets you change the column name of foreign keys. To modify other properties of a foreign key column, you must alter the corresponding primary key column in the originating parent table.

Create a Foreign Key/Relationship

Foreign keys are created when you establish relationships between entities. The primary key of the parent object becomes the foreign key of the child object. Foreign keys are not displayed in the entity boxes in the Data Model Window when using Filtered IE (No FKs) but are displayed using all other notation types. For more information, see Working with Relationships.

Specify FK Attribute Name and Datatype

By default, the foreign keys inherit their names from the primary key of the parent object; however, you can choose to provide another name for the foreign key using the Logical Rolename.

1. Double-click the entity you want to change. ===
2. Click the foreign key in the attributes list and then click *Edit*.
3. Enter a *Logical Rolename* and then select *Synchronize Column Rolename with Logical Rolename*.
4. Click *Save*.

The child now displays the logical rolename of the foreign key.

Similarly, you can specify a different datatype for the foreign key.

1. Double-click the entity and from the *Entity Editor*, select the foreign key from the attributes list and then click *Edit*.
2. Select *Edit Foreign Key Datatype* and then make the necessary changes to the datatype.
3. Click *Save*.

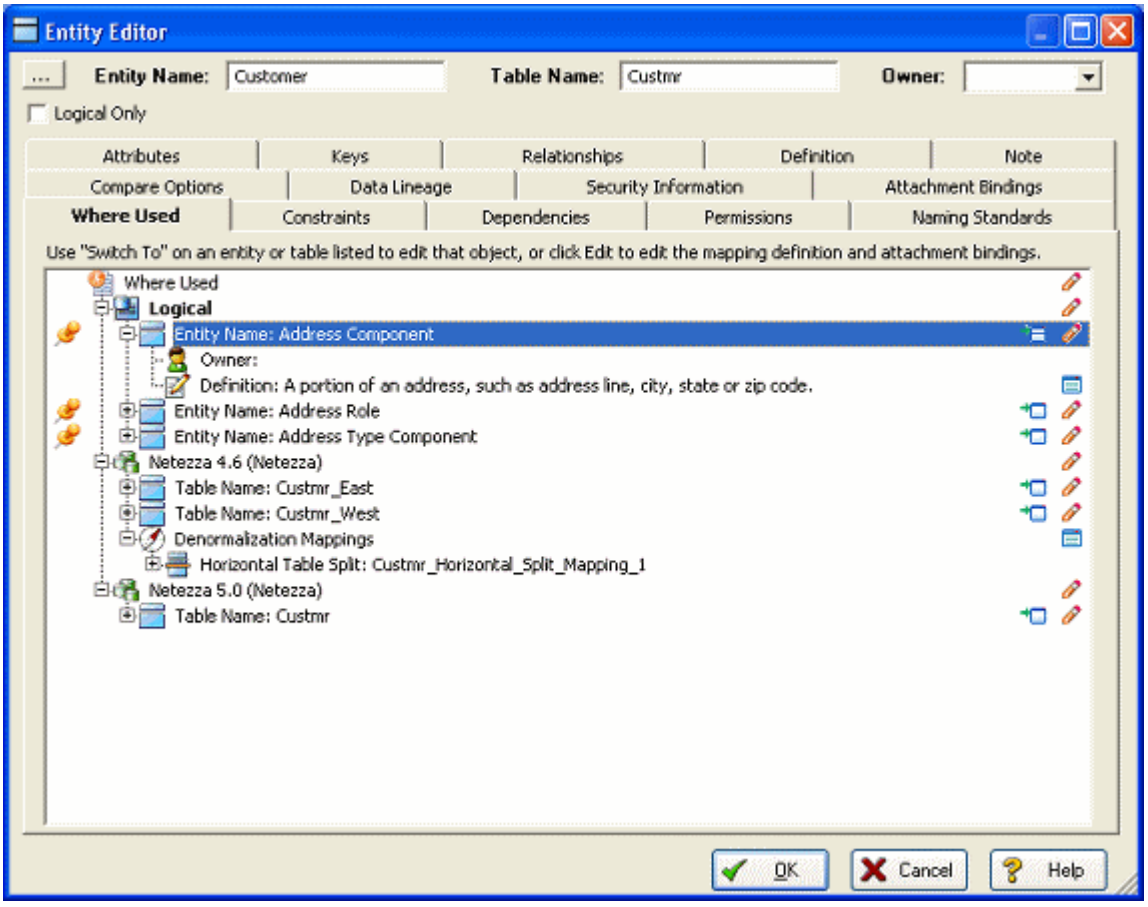
See Also

- Developing the Physical Model
- Common Tasks
- Creating and Editing Entities and Tables

Associate Similar Objects Using User-Defined Mappings

When consolidating various data assets across the enterprise, quite often the logical and physical model are completely unrelated. When new physical models are brought in via the Add New Physical Model wizard or Compare and Merge wizard, it is difficult to "map" existing logical entities to the physical tables. The physical objects may already be denormalized and complex (one-to-many, many-to-one) relationships may exist between the logical and physical objects. In this case, there is a gap between what is logically modeled and what is physically implemented. ER/Studio helps modelers map completely unrelated objects across models within a dm1 file using User-Defined Mappings.

Create user-defined mappings to relate logical entities to physical tables. The UDM can relate any objects of the same type. Once the UDM is created, you can see a graphical tree/node representation of the mapping on the Where Used tab in the Entity Editor or Table Editor. In the screenshot below, the entities mapped to Customer are marked with a little pushpin icon . On the right of the screen are icons that let you edit the mapping  or edit the user-defined mappings , and switch the entity editor to view or edit the details of the object selected. For information purposes you also map a Table to an Entity and edit that mapping from the Where Used tab.



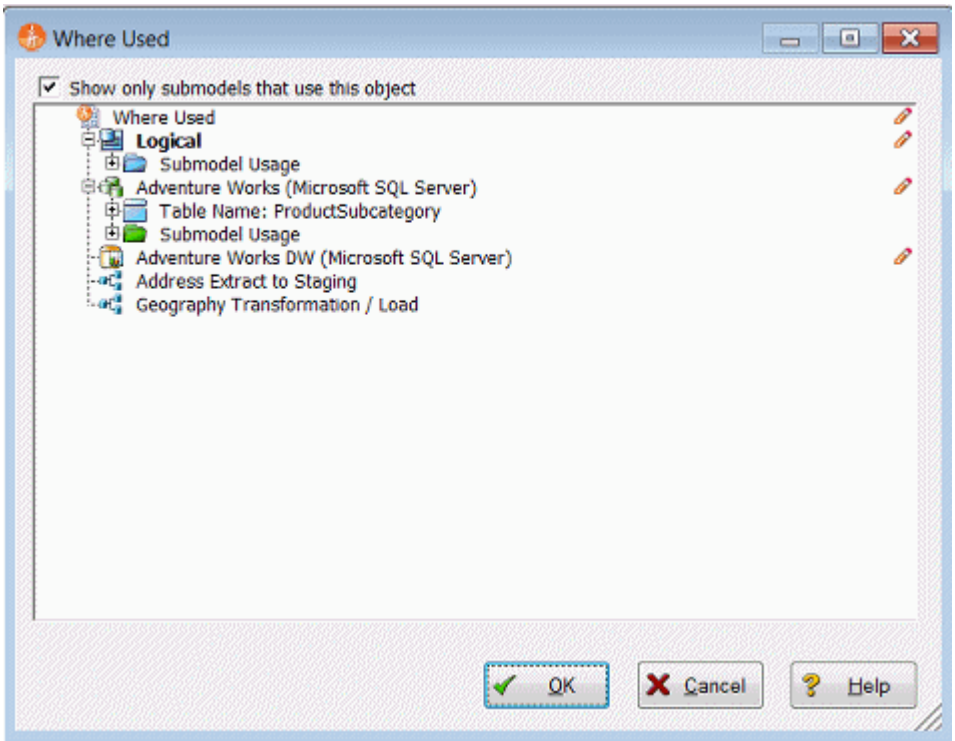
One use of the UDM is to back-construct the relationships shown in the Where Used tab.

For example, frequently a legacy physical model must be imported and added to a mature logical model (Using the Add New Physical model wizard: from the Main menu, Model > Add New Physical Model). You must relate the two models and can use the user-defined mapping to create and document the relationship.

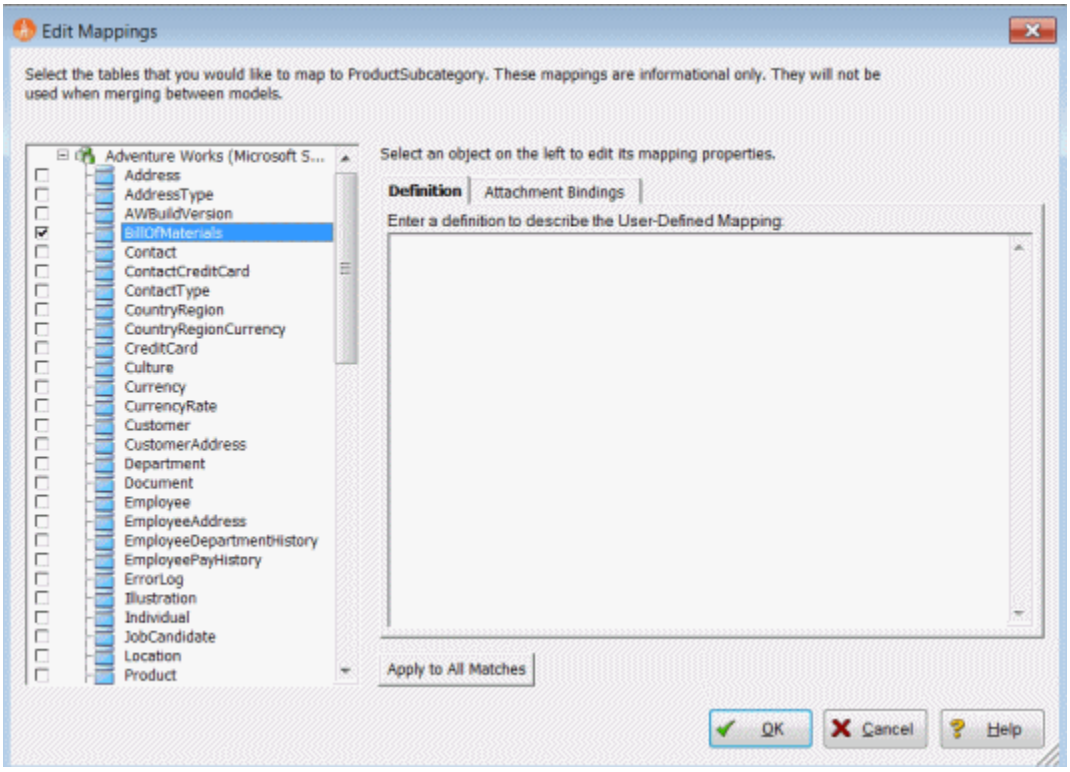
Denormalization information for the imported physical model likely is not available. Therefore, ER Studio Data Architect has no denormalization mapping information to display on the Where Used tab. The UDM system allows you to build the mappings with no restrictions. You can map attributes to columns and entities to tables by right-clicking any of the models listed and selecting the related objects.

Create User-Defined Mappings

1. In the *Data Model Explorer* or *Data Model Window*, right-click the entity or table that you want to associate with other entities or tables.
2. Click the *Where Used* icon 



3. *To associate the object with any entity/table in the model*, in the *Where Used* dialog, double-click the *Where Used* entry (the one with the pencil icon next to it) or right-click the pencil icon and select *Edit Mappings*. Click the target entity/table and the *Definition* and *Attachment Bindings* tabs appear.



The following helps you complete the options of the *Edit Mappings* dialog:

- Definition tab
- Attachment Bindings tab
- Notes

Definition tab

Enter or edit a definition for the user-defined mapping. If the target database supports it, ER Studio Data Architect adds this definition as a table comment when generating SQL code.

Attachment Bindings tab

Bind an external piece of information, or attachment to the user-defined mapping. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary. For more information, see Attaching External Documents to the Data Model.

Notes

- You can map an entity to tables in disparate physical models.

- Once you have created a user-defined mapping, to view or edit the mapping you can open a Table Editor or Entity Editor and then select the Where Used tab. You can also right-click a table in the diagram and select Edit User-Defined Mappings.
- You can also edit the mappings from the Where Used tab in the Table Editor or Entity Editor by right-clicking the model you want to editor and clicking the Edit option. When editing the user-defined mappings, you can also edit the definition and the attachments bound to the user-defined mapping.

See Also

- Developing the Physical Model
- Common Tasks
- Optimizing Query Performance on the Physical Model (Denormalizing the Physical Model)
- Working with Relationships
- Working with Universal Mappings

Working with Universal Mappings

Universal Mappings have been implemented to support enterprise-level modeling in large organizations with many systems and overlapping data and processes. They allow you to map objects between models in the same dm1 file, between models in separate dm1 files, and between an ER Studio Data Architect diagram and a ER/Studio Business Architect workspace in the Repository. These mappings are retained and can be used when comparing and merging. They are automatically applied in the *Compare and Merge Utility* if they exist.

What is a mapping?

A mapping is a link between two objects. There are two types of mappings that serve different purposes.

- *Mappings within a design layer*: This horizontal type of mapping shows reuse within a design layer, such as logical to logical, physical to physical and conceptual to conceptual. The scope of this mapping is limited to mapping objects of the same type, such as entity to entity, table to table, and column to column.
- *Mappings across different design layers:* This vertical type of mapping shows traceability or where used, upstream and downstream across different design layers, such as: conceptual to logical; logical to physical; and logical and physical.

Why are cross-repository mappings important?

- *Impact Analysis*: Understanding where common objects are located. Traceability between conceptual and logical models.
- *Scalability*: A mapping system across dm1s and workspaces reduces the need for large models that have everything in them. You can now perform Where Used (ER Studio Data Architect) or Impact Analysis (ER/Studio BA) that can include information on multiple dm1 or workspace files. Organizing objects into different dm1s or workspaces and then mapping related objects scales much better.
- *Performance*: Smaller dm1 or workspaces improves performance and manageability.
- *Enterprise Modeling*: Facilitates an enterprise view of your data. Typically this starts with identifying subject areas, then conceptual entities and then logical/physical models from the conceptual. Once the hierarchy is in place, you can see upstream from logical/physical objects and downstream from conceptual and process models.

The following topics help you understand how Universal Mappings are implemented in ER/Studio:

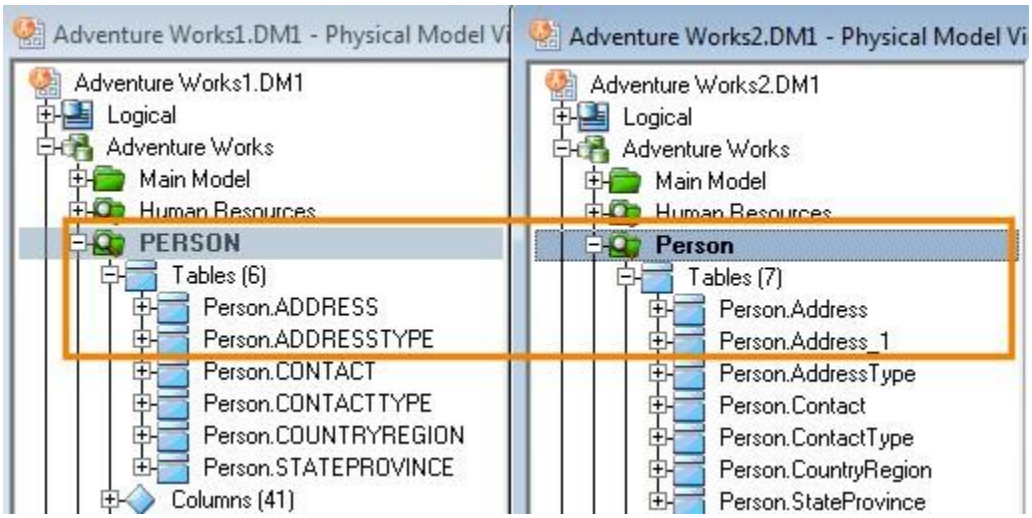
- Creating Universal Mappings through the Compare/Merge Utility
- Creating a Universal Mapping using the Universal Mapping Editor
- Viewing Universal Mappings

Creating Universal Mappings through the Compare/Merge Utility

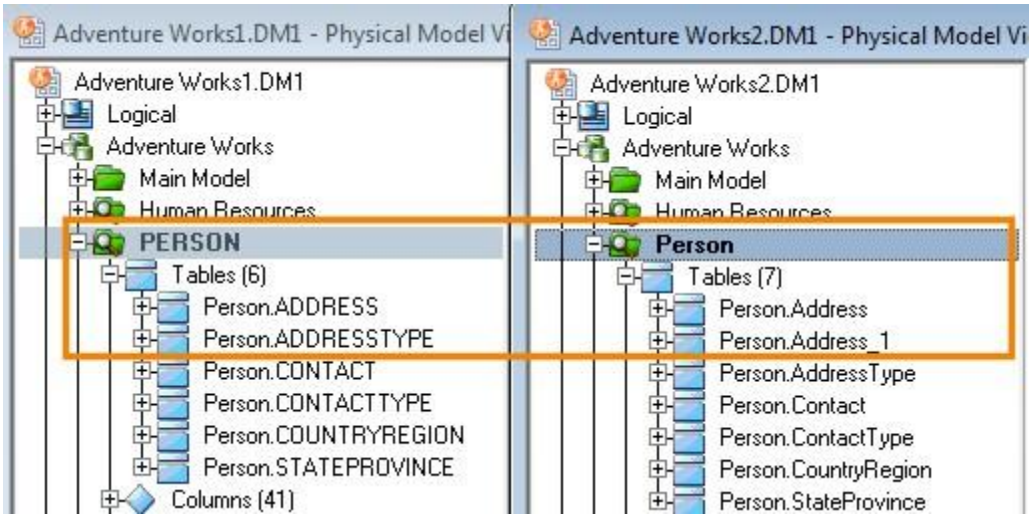
The following illustrates the process of creating and using a *Universal Mapping*:

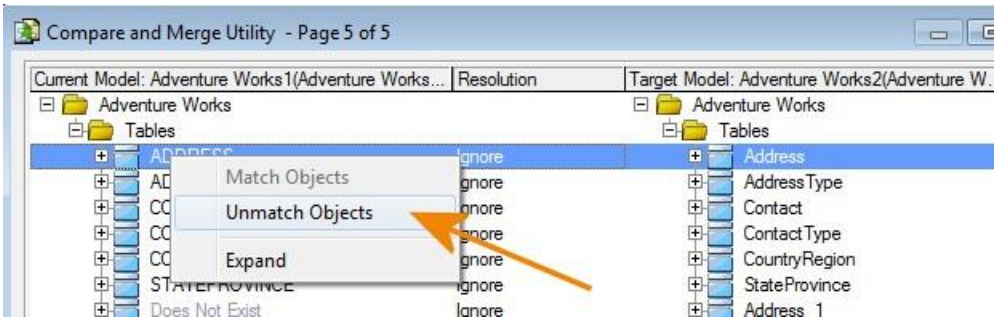
- *User-Defined Mappings* can be used in a similar way when comparing between two models in the same dm1 file. Existing user-defined mappings will now be considered when matching in the compare/merge, and you will have the option to save matches as *User-Defined Mappings* for inter-diagram comparisons.
- The precedence for matches in the Compare/Merge Utility is as follows, from high to low precedence: force matches, universal mappings, name matches, attribute sequence number and datatype matches. In other words, if the datatype matches between two attributes but the universal mapping matches one of those attributes to a different attribute, then the second match will be automatically applied. You can use *Force Match* in the *Compare and Merge Utility* to implement a different match.

In the following image, we see two dm1 files that differ slightly.



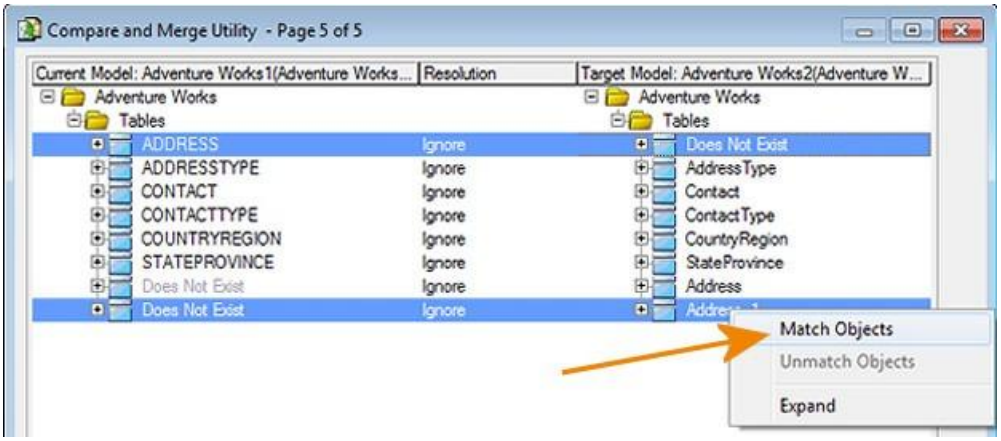
The *Compare and Merge Utility* automatically matches Person.ADDRESS to Person.Address, but if we want to match Person.ADDRESS to Person.Address1, then we can unmatch these attributes and create a different match.



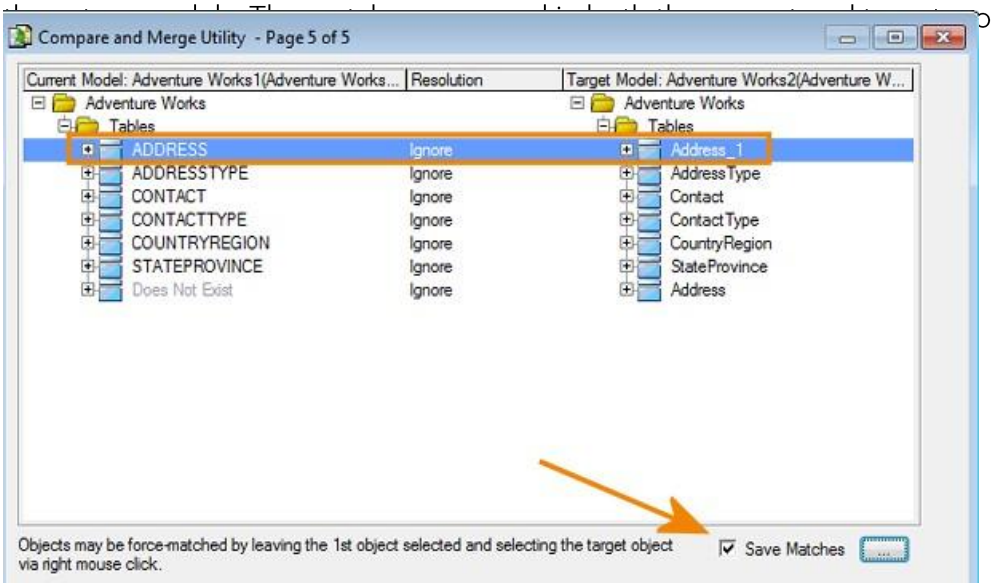


To illustrate the power of Universal Mapping, we can match two objects and then have the matches saved.

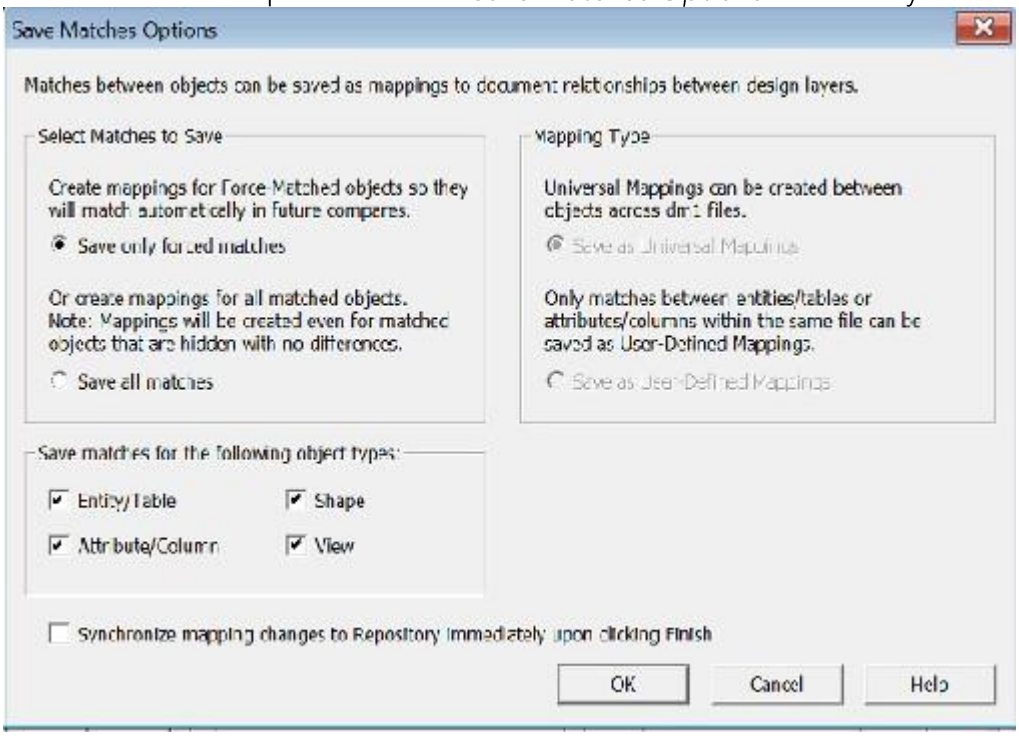
As indicated in the *Compare and Merge Utility*, 'Objects may be force-matched by leaving the 1st object select and selecting the target object via right mouse click.'



Now that the match is made, we can choose to save the match to use the next time we compare models.



You can click the ellipses to see the *Save Matches Options* and make your choices.



A checkbox for each object type supported, Entity/Table, Attribute/Column, View, and Shapes is listed and all are selected by default. Only objects of the types selected are eligible to have matches saved as mappings.

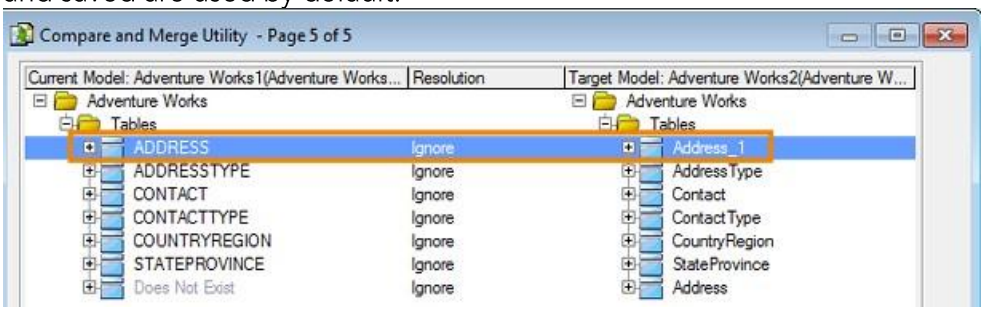
When comparing models in the same diagram, you can choose between *Save as User-Defined Mappings* or *Save as Universal Mappings*.

When comparing models in different diagrams, *Save as Universal Mappings* is selected. This is the only option available.

If you are comparing two diagrams in the Repository, the option to *Synchronize mapping changes to Repository immediately upon clicking Finish* becomes available.

Click *OK* on the *Save Matches Options* and then click *Finish* to complete the compare/merge.

When you run the *Compare and Merge Utility* again, you will see that the matches you forced and saved are used by default.

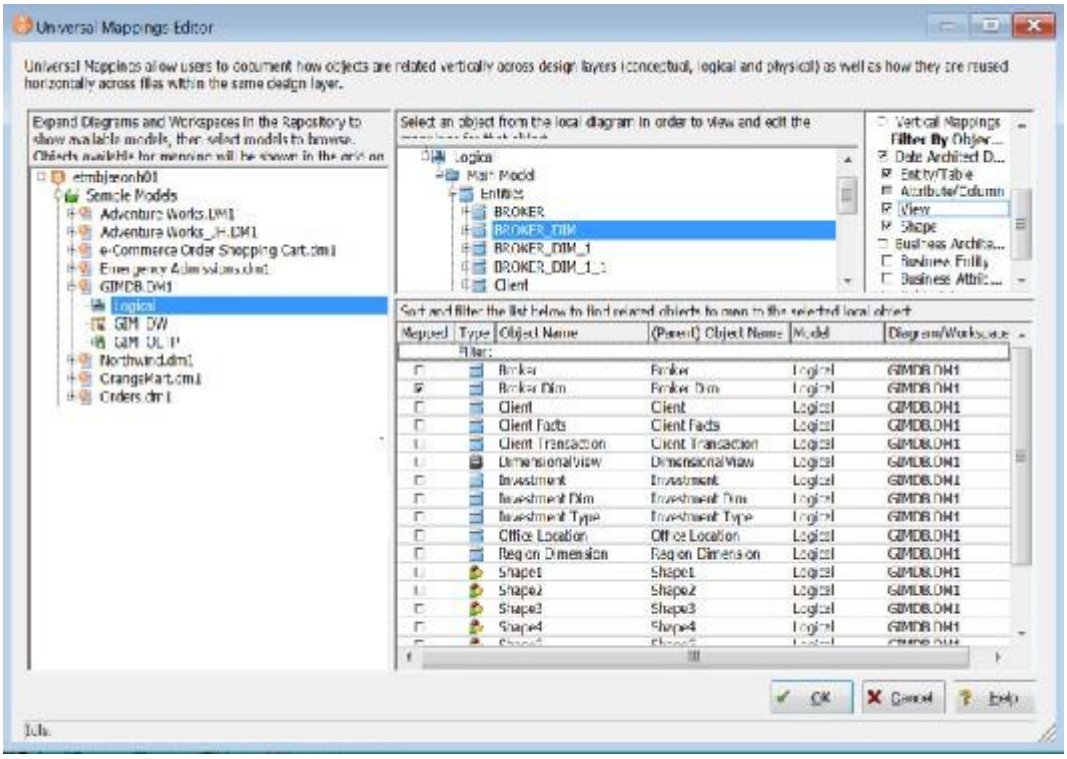


Creating a Universal Mapping using the Universal Mapping Editor

The Universal Mappings Editor provides the following functionality:

- *Local Objects* You can edit the Universal Mappings of a source object without exiting the Universal Mapping Editor.
- *Browse/Filter/Search*: You can view objects in other diagrams and workspaces that can be mapped to the currently selected object. Because the number of objects can potentially be very large, you can narrow down the list by using the filters available and sorting the related objects.
- *Existing Mappings*: You can see which objects are already mapped, as well as the description and any other data associated with the mapping.

1. In the diagram window, right-click the object you want to map or change the mapping for and select *Universal Mappings*.



2. In the Universal Mapping Editor, in the center of the window, as indicated onscreen, *Select an object from the local diagram in order to view and edit the mappings for that object*.

3. On the far left of the screen select a model in the tree. Once you have selected the model, the data for it is retrieved from the Repository.

The list of mappable objects that displays on the bottom right of the editor is generated from your selection in the Repository tree. If you select a diagram node, then all objects in its models will be included in the mappable object list. If you select a Project, then all objects in diagrams in that project will be included.

1. On the far right of the editor, select the filters to use in order to see the objects of those type you can map to.

2. You can map ER Studio Data Architect entities/tables, attributes/columns, views, shapes and ER/Studio Business Architect entities, attributes, and subject areas to each other, including mapping ER Studio Data Architect objects to ER/Studio BA objects.

3. At the bottom of the editor, click a column name to reorder the list of related objects to find the object to map to the local object.

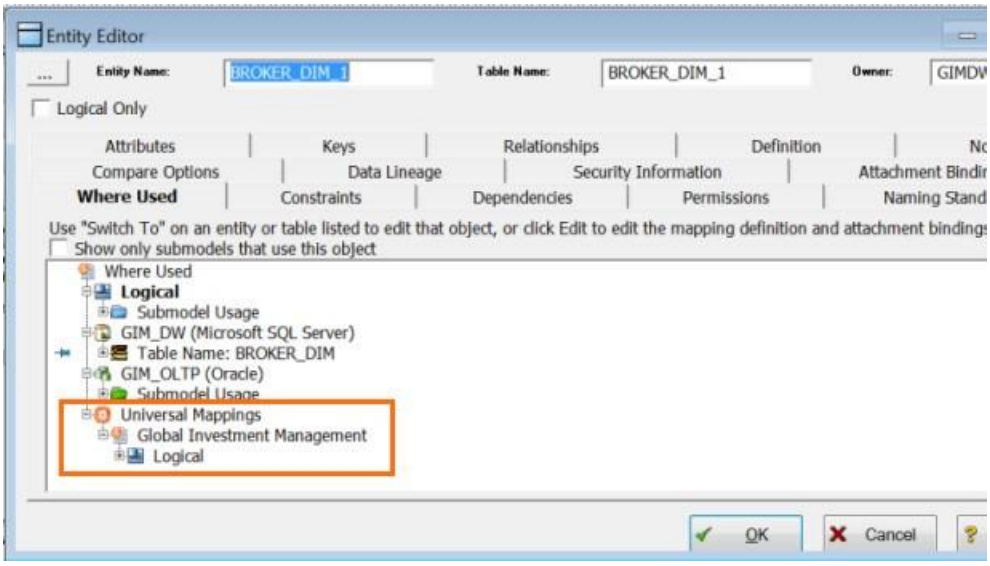
4. Deselect or select the target object's *Mapped* checkbox to remove or add a mapping.

5. Click *OK* to complete the change.

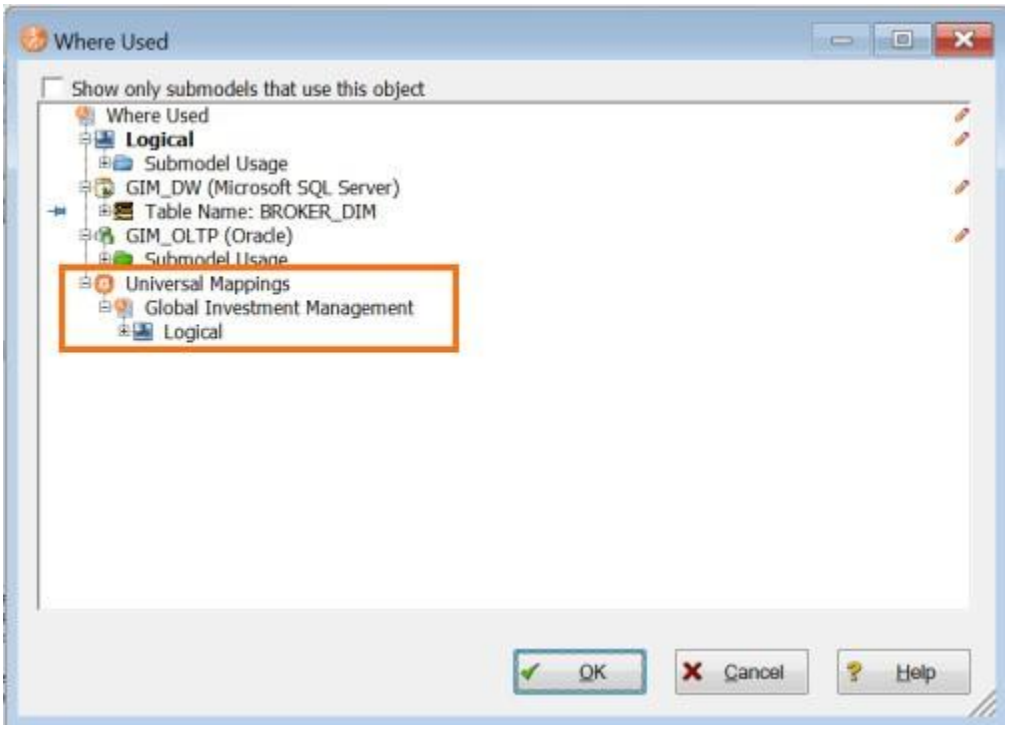
6. Save your diagram and in the case of Repository diagrams, check in the affected object to save the mapping change to the Repository.

Viewing Universal Mappings

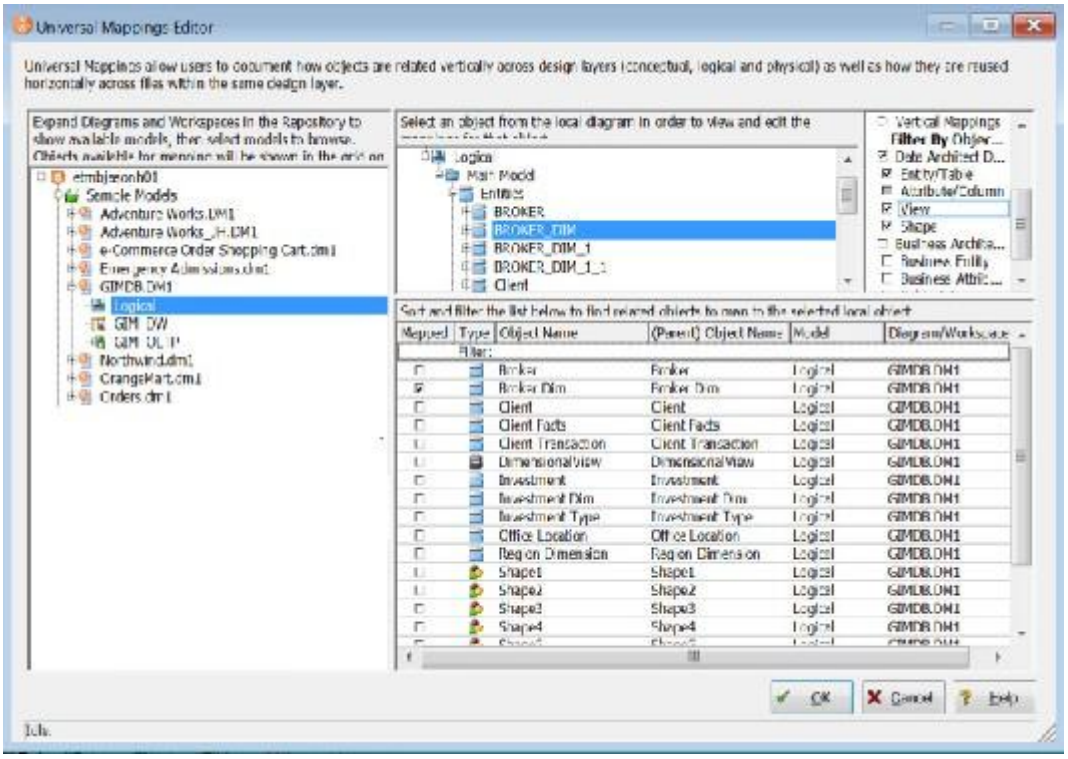
Universal Mappings are displayed in the object's *Where Used* tab.



You can also see the mappings of an object by right-clicking the object and selecting *Where Used*.



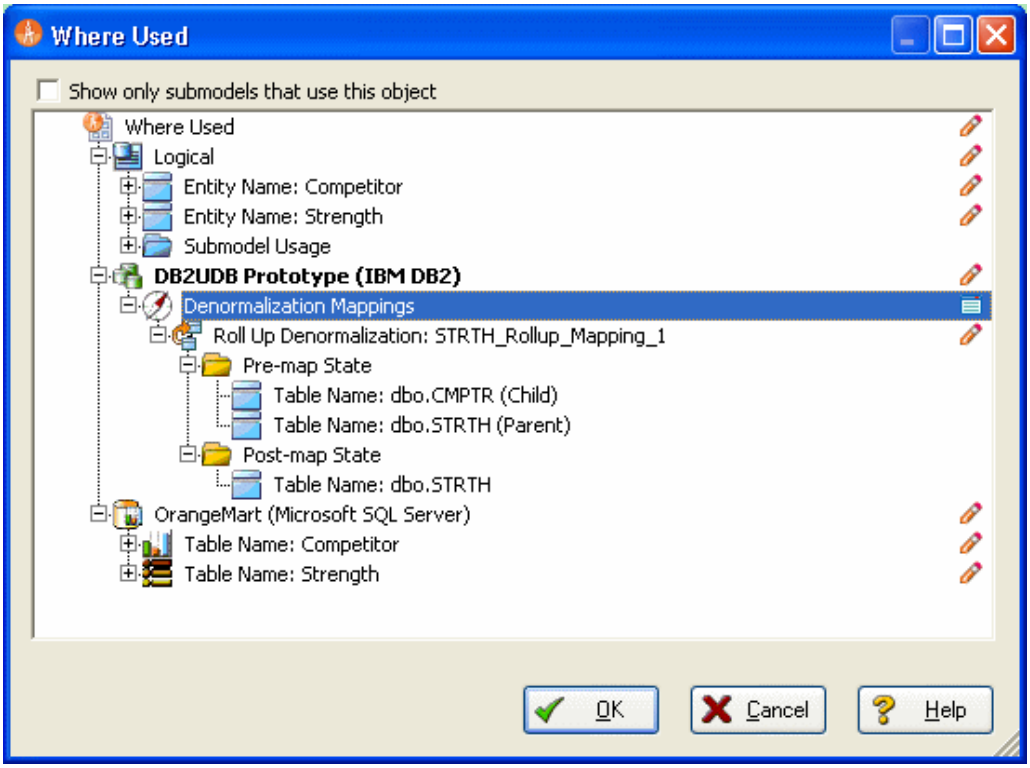
And you can see the mappings of an object in the *Universal Mappings Editor*.



Viewing and Editing Denormalization Mappings

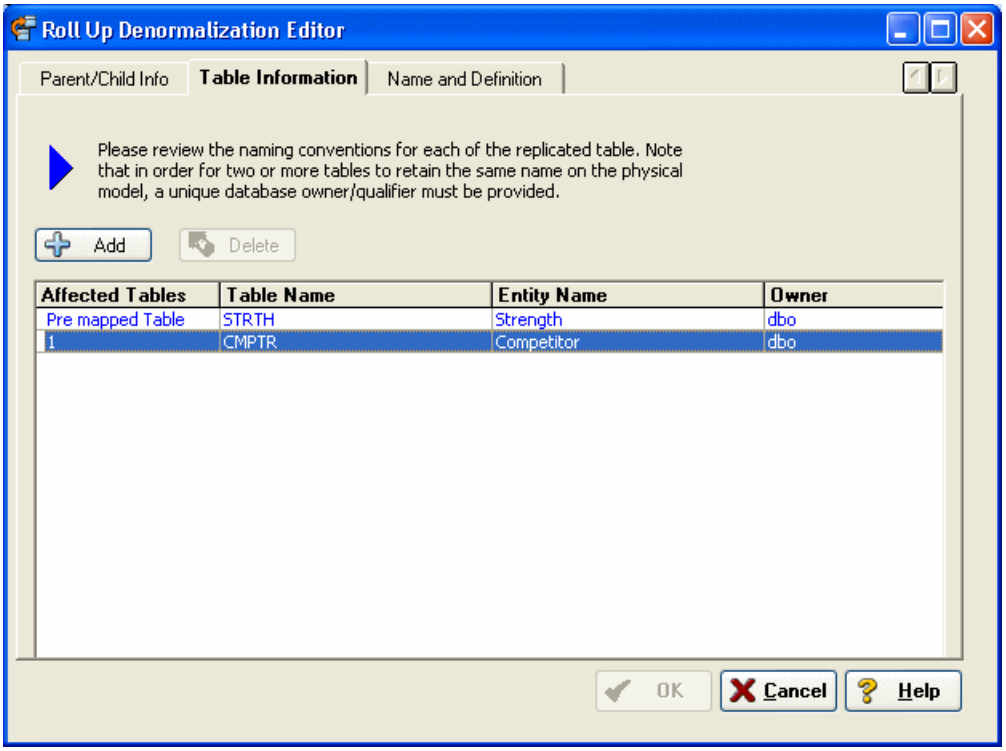
The history of a table's denormalization is stored with the table and can be viewed on the Where Used tab of the Table Editor. Where Used information can also be accessed using the Edit User-Defined Mappings function available on the right-click menu of a table. From the Where Used tab or window you can see how the table was transformed.

1. From the Data Model Window, right-click the table you want to learn about and select *Where Used*.
    - The Where Used tab or window presents a graphical tree/node representation of the denormalization. In the screenshot below, by expanding the Denormalization Mappings entry, you can see what type of denomalization transformed the table. You can also see the pre-map and post-map state of the table. The example below shows that two tables, CMPTR and STRTH were rolled up into STRTH. On the right of the screen is an icon that let you edit the mapping



1. To edit the Denormalization mapping, click the edit mapping icon 

The appropriate Denormalization Editor appears where you can change the mappings.



- Parent/Child Information is for information only.

If, after you perform a denormalization, you want to revert the table to its premap state and the undo operation is no longer available you must perform the reverse denormalization. For example, if you rolled up two tables you must select the table and then perform a roll-down denormalization on it, splitting up the one table into two.

See Also

- Developing the Physical Model
- Common Tasks
- Optimizing Query Performance on the Physical Model (Denormalizing the Physical Model)
- Working with Relationships

Working with Relationships

Relationships help enforce business rules and assertions in a data model. Relationships determine how data are related between two entities and/or views.  **Note:** The dashed / dotted lines indicating those of a relationship with a View, differs from those indicating a non-identifying relationship.

Relationships are implemented as foreign keys in the physical model. The properties of these foreign keys dictate how referential integrity is enforced between tables through constraints and triggers.

In addition to standard relationships ER Studio Data Architect lets you create recursive relationships, those where an object is related to itself; duplicate relationships, where multiple relationships relate to the same objects; view relationships that relate entities or tables to views, and subtype clusters that associate multiple homogenous entities (subtypes) to a supertype entity.

For more information on relationships, see the following topics:

- Creating and Editing Relationships
- Creating and Editing Recursive Relationships
- Creating View Relationships
- Creating Subtype Cluster Relationships
- Resolving Name Duplication in Parent and Child Entities (Unification)

See Also

- Developing the Physical Model
- Common Tasks

Creating and Editing Relationships

1. On the Data Model Explorer, click the Data Model tab.
   To create a new relationship
2. Choose Insert > Relationship and then click the type of relationship you want to insert.
3. On the Data Model Window, click the parent object and then click the child object.
   - The *Duplicate Attribute Editor* displays If both the parent and child objects have the same column or attribute names. Here you can resolve these conflicts by assigning rolenames to foreign keys.
   - The *Recursive Relationship* editor displays when you create a recursive relationship, which is a non-identifying relationship where the parent and child object are the same. Using this editor you can assign rolenames to any duplicate keys to differentiate the native and foreign key. For more information, see Creating and Editing Recursive Relationships.
   - The *Duplicate Relationship* editor displays when you draw a relationship that will result in multiple identifying and/or non-identifying relationships between two entities. Using this editor you can assign rolenames to any overlapping foreign keys; otherwise, ER Studio Data Architect unifies any overlapping Foreign Keys.
   - The *Edit Rolenames* dialog displays if overlapping foreign keys are detected when you are drawing a relationship. On the Edit RoleNames dialog you can specify rolenames for foreign keys that are inherited more than once. When a primary key is propagated to a child table, the name of the attribute is also propagated to the child table. A rolename lets you rename the attribute of the foreign key to avoid confusion and resolve the relationship conflict.
4. Repeat steps 2 and 3 until you have created all necessary relationships.
5. To revert to the *Selection* tool, right-click anywhere on the *Data Model Window*.
   To edit an existing relationship
6. On the *Data Model Window*, double-click the relationship
7. Complete the *Relationship Editor* and then click *OK* to exit the editor.

The following helps you complete the *Relationship Editor*.

All tabs Properties tab

Phrases tab Name tab

Trigger tab Definition tab

Compare Options tab Note tab

See Also

All tabs

- *Parent Key*: Lets you select and propagate an Alternate Key (logical model) or a Unique Index (physical model) to a child entity or table. Use this option if you do not want to propagate the primary key. If there are no alternate keys or unique indexes in the parent table the primary key will be used.
- *Logical Only*: Logical model only. If selected, the relationship will not be implemented when generating a physical model, and will be ignored when comparing the logical model with a physical model.
- *Physical Only*: Physical model only. If selected, the relationship will be ignored when comparing the physical model with a logical model.
- *NoValidate*: This option is available for the Oracle 9i, 10g, 11g and 12c platforms. Specify NoValidate if you do not want the constraint to be applied to the data in the table.

If generation of the FK or PK constraint is enabled, NOVALIDATE will be generated if the option is set for the constraint. For example, if the PK is not disabled, but NoValidate is set, the DDL would be

CREATE TABLE Entity1

(

PKCOL CHAR(10) NOT NULL,

CONSTRAINT PK1 PRIMARY KEY (PKCOL) NOVALIDATE

);

The same syntax will be used for FKs when created inside the CREATE TABLE or by ALTER TABLE.

ALTER TABLE ENTITY2 ADD CONSTRAINT BLIVOT

FOREIGN KEY (PK1)

REFERENCES ENTITY1(PK1) NOVALIDATE

If Disable is set but NoValidate is not set, the DDL for an FK created by ALTER TABLE would be the following, since NOVALIDATE is the default option. The same syntax is used for PKs.

ALTER TABLE ENTITY2 ADD CONSTRAINT BLIVOT

FOREIGN KEY (PK1)

REFERENCES ENTITY1(PK1) DISABLE VALIDATE

- *Do Not Generate*: Physical model only. If selected, the foreign key (FK) columns (if any) will be ignored when generating DDL, as in the Compare and Merge Utility. The FK constraint is not generated but the FK column is generated as a native column. To prevent the FK from being generated, edit the FK column in the Table Editor and select the Hide Key Attribute option. Use this option when relational integrity is enforced in the application rather than by an explicit foreign key in the RDBMS.
- *Not For Replication*: If selected, updates to the parent will not be replicated to the child entity. This prevents any foreign key constraints from being applied when updating the child entity.
- *NO CHECK*:This option is used when creating FK constraints in NCR TeraData databases to disable constraint data checking when performing SQL operations on the data in the table the FK is created on.
- *NOCHECK*: This option is used when creating FK constraints in SQL Server 2005 or 2008 databases to disable checking of existing data against the contraint during the add operation. In addition, the FK constraint is disabled and future inserts or updates to the column are not validated against it.
- *Deferrable*: Physical model only. If selected, the REFERENCES.... DEFERRABLE syntax will appear in the FK DDL for inline (in the create table) or ALTER (out of the create table). The DEFERRABLE statement becomes part of the constraint state logic after the REFERENCES clause.
  In general, the DEFERRABLE parameter indicates whether constraint checking in subsequent transactions can be deferred until the end of the transaction using the SET CONSTRAINT(S) statement. If omitted, the default is NOT DEFERRABLE. If a constraint is deferrable in the model but not deferrable in the database it will need to be dropped and recreated with the deferrable constraints. See Oracle syntax documentation for a description of deferrable constraints.
- *Initially Deferred*: Physical model only. If Deferrable is selected, this option becomes available. When selected, the REFERENCES... DEFERRABLE INITIALLY DEFERRED syntax appears in the DDL for the child table in the relationship. Initially deferred indicates that constraint checking is postponed until the end of the transaction.
- *Disable*: This option is available for the Oracle Platform only. If selected, disables the foreign key of a child entity. The following is an example of the DDL generated for the child entity when the FK is disabled:

CREATE TABLE Entity2(

PK2 CHAR(10) NOT NULL,

a2 CHAR(10),

b2 CHAR(10),

PK1 CHAR(10) NOT NULL,

CONSTRAINT PK2 PRIMARY KEY (PK2),

CONSTRAINT RefEntity11 FOREIGN KEY (PK1)

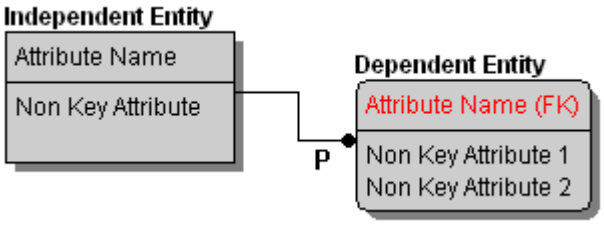
REFERENCES Entity1(PK1) DISABLE

Properties tab

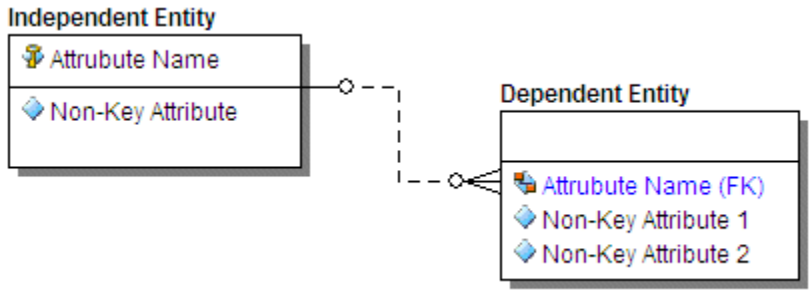
- *Relationship Type > Identifying Relationships* Identifying relationships are always mandatory. A foreign key value must exist in the child entity and the foreign key value must be found in the primary key of the parent. Identifying relationships propagate primary keys as primary keys to child entities, which can result in compound keys.

The use of compound keys is often valid and appropriate; however, they limit the flexibility of your data model. When you use a compound key, the data contained in the child entity can only be identified in the context of its parent. If you ever need to use the data independently, then you will have a problem

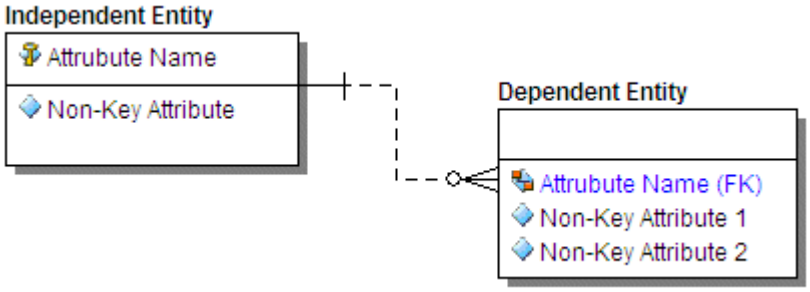
The cardinality of a mandatory relationship must be in the form of one-to-something. When generating a physical model, foreign key columns propagated by a mandatory relationship default to *NOT NULL*.



- *Relationship Type > Non-Identifying, Optional Relationships*: Non-identifying relationships propagate the parent's primary key to the non-key attributes of the child. Since the relationship is optional the foreign key value is not always required in the child entity; however, if a value does exist, then the foreign key value must be found in the primary key of the parent.
  The cardinality of an optional relationship takes the form of zero or one to something. When generating a physical model, foreign key columns propagated by an optional relationship default to *NULL*.

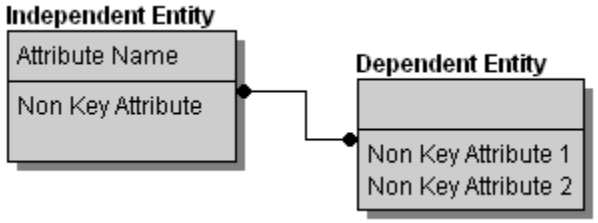


- *Relationship Type > Non-Identifying, Mandatory Relationships*: Non-identifying relationships propagate the parent's primary key to the non-key attributes of the child. A foreign key value must exist in the child entity and the foreign key value must be found in the primary key of the parent.
  The cardinality of a mandatory relationship must be in the form of one to something. When generating a physical model, foreign key columns propagated by a mandatory relationship default to *NOT NULL*.



- *Relationship Type > Non-Specific Relationships*: Non-specific relationships denote many-to-many relationships. Because many-to-many relationships cannot be resolved, non-specific relationships do not propagate any foreign keys. Many-to-many relationships are undesirable and should be removed as you normalize your data model.

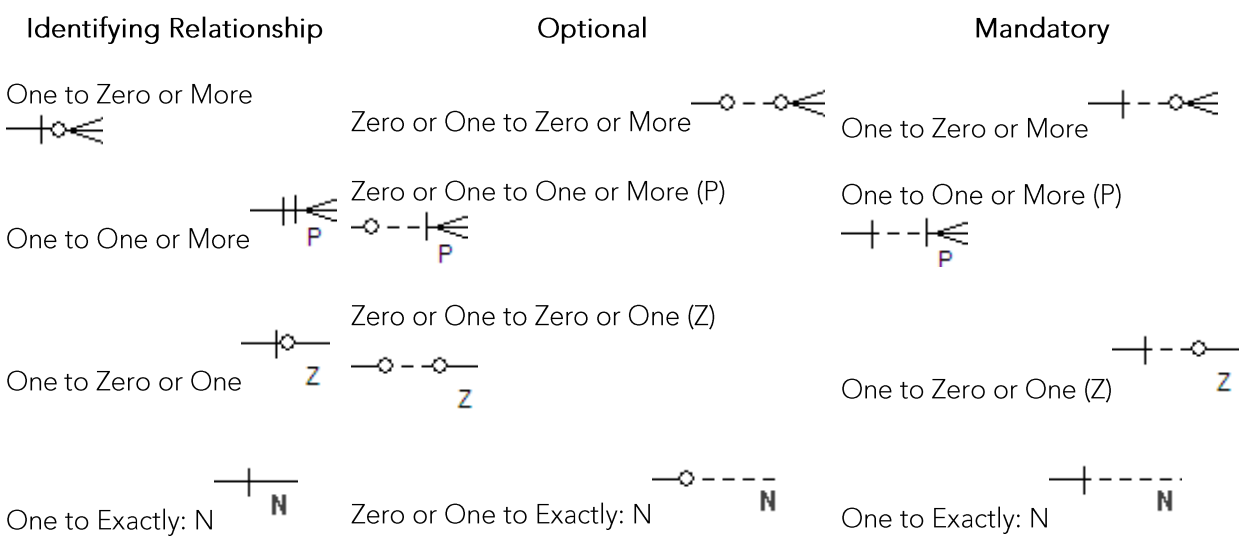
The following illustrates a non-specific relationship where the parent and the child are both optional. In the Relationship Editor you can specify whether the parent and the child are optional or mandatory.



- *Cardinality*: Cardinality displays the ratio of related parent and child instances. The cardinality ratio for the parent depends on whether the relationship is mandatory or optional. Although cardinality is a powerful concept for expressing business rules, you should know that no database can directly enforce cardinality. Enforcing cardinality constraints can be accomplished through procedural database logic or somewhere other than in the database.

The iconic cardinality representations for IE (Crows Feet) notation are illustrated below:

Non-Identifying Relationship



Phrases tab

- *Verb Phrase*: Describes the relationship of the Parent table to the Child table.
- *Inverse Verb Phrase*: Describes the relationship of the Child table to the Parent table.

Name tab

- *Business Name*: Identifies the relationship for business documentation purposes.
- *Constraint Name*: Names the referential constraint the relationship enforces between parent and child tables, if any. ER Studio Data Architect uses this name as the foreign key constraint name when generating DDL.

Trigger tab

Select a trigger to maintain referential integrity for *INSERT*, *UPDATE*, and *DELETE* operations on child and parent objects. Select one of the following triggers for each operation.

- *Restrict*: Verifies the existence of foreign key values in the object's primary key and prevents the insertion, updating, or deleting of data if the values cannot be validated.
- *Cascade*: Propagates any modification of a primary key value to the corresponding foreign key values in the child table.
- *Set Null*: Verifies the existence of the foreign key values in the parent table's primary key. If the values cannot be validated, the trigger sets the foreign key values to null in the child table and lets the data modification operation proceed.

For more information, see Creating and Editing Database Schemas.

Definition tab

Enter or edit a definition for the relationship. If the target database supports it, ER Studio Data Architect adds this definition as a comment when generating SQL code.

Compare Options tab

Allows you to choose which properties of the attribute will be ignored when using the Compare and Merge, thus allowing you to have intentional discrepancies.

Note tab

The notes added here are included as part of the HTML report when you generate a report for the model. You can format the notes using standard HTML tags.

Rolename tab

You can change the rolename of foreign keys to differentiate between child and parent attributes in the event that overlapping foreign keys exist. To change a rolename, double-click any of the rows to invoke the *Foreign Key* dialog. You can also change logical and default column rolenames through the *Columns* tab of the *Table Editor* or by right-clicking the relationship and selecting **Edit RoleName**.

Attachment Bindings tab

Bind an external piece of information, or attachment to the relationship. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary and must be applied to the relationship before they will display on this tab. For more information, see Attaching External Documents to the Data Model.

Notes

- You can view the incoming and outgoing relationships of a table by expanding the relevant table node and then expanding *I* ncoming Relationships or Outgoing Relationships. Expand the relationship node to see the name of the related table.
- You can change the relationship type, specify cardinality, create verb phrases to describe the relationship, create a logical business and constraint name, and define triggers by double-clicking the relationship, and then making your changes in the Relationship Editor.
- You can delete a relationship by right-clicking it in the Data Model Window and then selecting Delete Relationship.
- You can edit a relationship by right-clicking the relationship in the Data Model Window and then selecting Edit Relationship for a logical model or Edit Foreign Key for a physical model.
- Changing relationship types causes ER Studio Data Architect to re-propagate foreign keys using the following rules:
    - *When changing from identifying to non-identifying*, foreign keys change from primary keys to non-key columns in the child object.

  - *When changing from identifying to non-specific, or from non-identifying to non-specific*, the foreign keys are removed from the child object.
  - *When changing from non-identifying to identifying*, the foreign keys change from non-key to primary key columns or attributes in the child object.
  - *When changing from non-specific to identifying*, the child object inherits the parent object's primary keys as primary key columns or attributes.
  - *When changing from non-specific to non-identifying*, the child object inherits the parent table's primary keys as non-key columns or attributes.

- You can configure how ER Studio Data Architect displays relationships for the selected model by choosing View > Diagram and Object Display Options Editor > Relationship and then making your changes.
- You cannot edit a View Relationship (see Creating View Relationships).

See Also

Moving Relationship Docking Points

Setting the Relationship Notation

Manipulating Relationship Lines

Overriding Color and Font Settings for a Specific Object

Creating and Editing Recursive Relationships

A recursive relationship associates an object with itself; the object is both the parent and child. For example, you can have an employee who manages many employees and each employee is managed by one employee. The employee entity has a recursive relationship to itself.

If you have a recursive relationship, you must assign rolenames to the relationship. Use the Recursive Relationship dialog to specify rolenames for a foreign key that is inherited more than once within the same entity. For example, if a recursive relationship is created where one entity is designated as both the parent and child entity, then the primary key is propagated within the entity. ER Studio Data Architect automatically opens the Recursive Relationship dialog box when it detects overlapping foreign keys in an entity.

The steps for adding recursive relationships are slightly different from adding regular relationships; however, you can for edit and delete recursive relationships in the same manner as for regular relationships.

1. Choose Insert > Insert Relationship, and then select a non-identifying relationship type.
2. On the Data Model Window, click the object to which you want to apply the recursive relationship, and then click it again.
3. Complete the *Recursive Relationship* or *Recursive Relationship for Multiple Keys* dialog box, and then click *OK*.
4. Right click anywhere on the Data Model to revert to the *Selection* tool.

Notes

- If you have a recursive relationship, the Recursive Relationship editor appears where you must assign a rolename to the primary key or to at least one attribute of the primary key. The rolename cannot be the same as the attribute or column name. In order for the foreign key to appear in the child, it must be assigned a rolename so it can be differentiated it from the native key.
- If you have a recursive relationship where multiple foreign keys that are inherited more than once within the same entity, the *Recursive Relationship for Multiple Keys* dialog display where you specify rolenames for each foreign keys. For example, if a recursive relationship is created, where one entity is designated as both the parent and child entity, then the primary keys are propagated within the entity.
- Edit the rolenames associated with the relationship by right-clicking the relationship in the Data Model Window, and then selecting *Edit Rolenames*.
- Navigate between the parent and child object by right-clicking the object and then selecting *Navigate to Parent* or *Navigate to Child* from the shortcut menu.

See Also

- Creating and Editing Relationships
- Creating Subtype Cluster Relationships

Creating View Relationships

The steps to adding and deleting view relationships are the same as those of regular relationships. However, view relationships are a special type of relationship used to propagate attributes from an entity or a view to a view. When adding a view relationship, the end point of the relationship must be a view. View relationships cannot be edited, but they can be deleted. To determine which attributes of an entity can be propagated from the parent entity or view to the child view, use the View Editor - Column Tab.

- In order for view relationships to display, you must enable the options to display views and view relationships on the *View* tab of the *Diagram and Object Display Options* dialog. Choose View > Diagram and Object Display Options.

See Also

Creating and Editing Database Views

Creating Subtype Cluster Relationships


Note: Subtype cluster relationships can only be represented in the logical model. When the physical model is generated from a logical model, the subtype relationship is converted to identifying relationship.

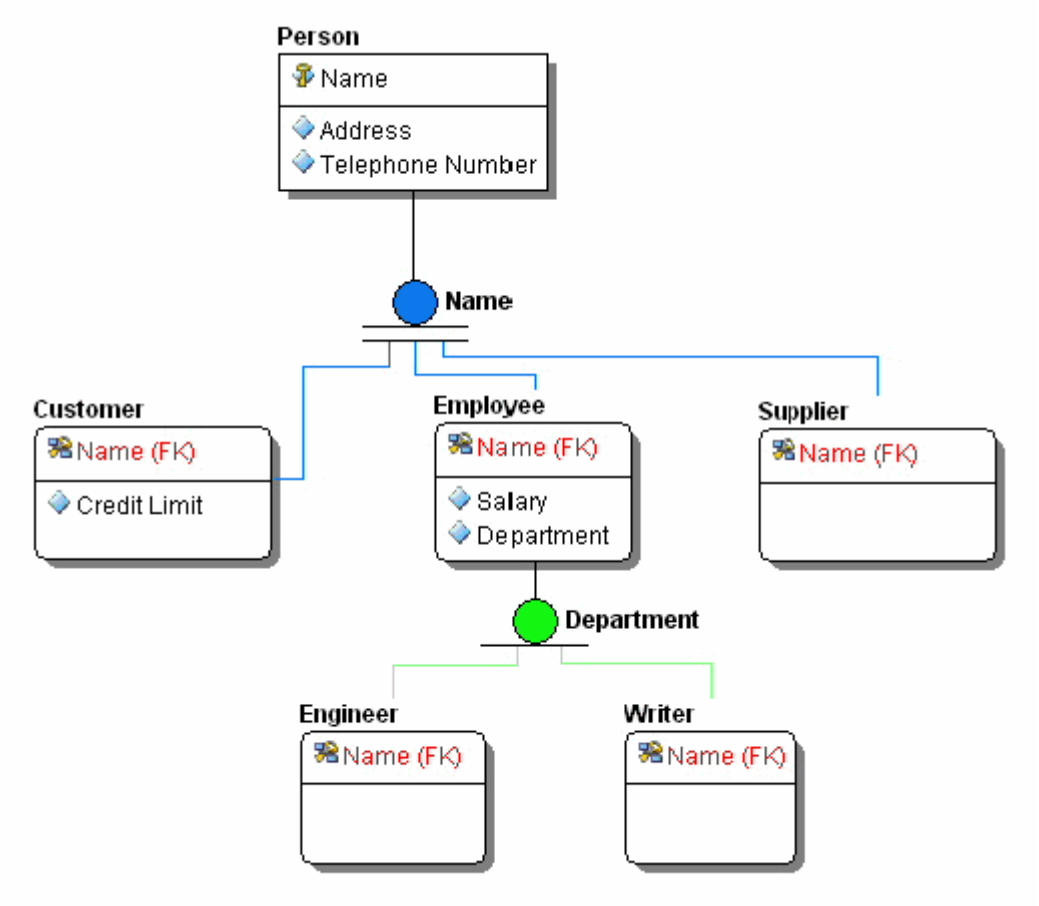
A subtype cluster refers to a group of entities sharing similar characteristics that form a *generalization hierarchy*. In each subtype cluster, there is one parent entity, known as a *supertype*, and one or more *subtype* entities. The supertype generalizes a set of subtypes; the common characteristics of subtype entities are assigned to the supertype while the subtype entities have more specific attributes. The subtypes, also known as category entities, represent homogeneous subsets of the supertype. The attributes and relationships of the supertype entity are propagated to all of its subtypes. Discriminators, an attribute of the subtype, distinguish the entities in the subtype from each other.

Other than recognizing a generalization hierarchy, an important issue with subtyping is to decide how to implement a subtype cluster in the physical model. Sometimes it is best to project subtypes into the supertype, representing the entire cluster as a single, generalized entity. Other times, you may need the granularity of separate subtype entities, choosing to implement the subtypes and supertype one-for-one or rolling the supertype down into the subtypes.

During the logical model phase, you should search for hierarchies that can reveal a more general approach to aggregating attributes into entities. By addressing the general case, you can design a more flexible and extensible approach to storing data. The use of subtyping is a powerful tool in identifying and portraying such generalizations.

Example

To illustrate the use of subtyping, let's take the example of the group of entities - Person, Customer, and Employee. Each entity stores information about employees. Accordingly, Person is the supertype in this example and the remaining entities can be grouped in a subtype. Any person contained in a subtype entity is also represented in Person. In each entity, the primary key is the Name. Because each entity instance for the entities in the subtype are also represented in the supertype, the relationships between the supertype entities and all subtype entities must be an identifying relationship, that is the customer, engineer and writer must all be employees. The relationship between Person, Customer, Employee, and Supplier is complete, because the business is concerned only with these types of persons. The relationship between the Employee, Engineer, and Writer is an incomplete subtype because there can be other employee types, such as quality assurance analysts.



In this example, the Person entity is the supertype. All Person's have a name, address and telephone number. A person can be either an Employee or a Customer. In addition to sharing name, address and telephone number attributes, each of these subtypes has its unique attributes; for example, an Employee has a Profession; and a Customer has a Credit Limit. An entity can be both a supertype and a subtype, as is the case with Employee, which can be further divided into Engineer and Writer.

1. Click the *Logical* model.
2. Choose Insert > Subtype Cluster and then choose *Complete* or *Incomplete*.
3. Click an entity to designate it as the parent (supertype) and then click the entity (subtype) you want to add to the subtype cluster.
   Tip: You can add multiple entities to the subtype cluster by pressing CTRL while clicking the desired entities.
4. Right-click to revert to the *Selection* tool and then double-click the newly created subtype relationship indicator.
5. Complete the *Subtype Cluster Editor* as required and then click *OK*.

The following helps you complete the *Subtype Cluster Editor* options:

- Subtype tab

- Attachment Bindings tab
- Notes

Subtype tab

- *Type*: Select the subtype cluster type, *Complete* or *Incomplete*.
- *Complete*: A subtype cluster is complete when all possible subtype entities are included in the subtype cluster. An example of a complete subtype cluster is one in which *Person* is the supertype and *Male* and *Female* are the subtypes.
- *Incomplete*: Select if all possible subtype entities will not included be in the cluster. For example, a subtype cluster is made up of several entities, *Employee*, *StoreManager*, *SalesPerson* and *StockPerson*, each of which stores information about employees. In this example, *Employee* is the supertype and the remaining entities can be grouped in a subtype. This is probably an incomplete subtype cluster because other employee types cannot be represented in the cluster, such as security guards and cashiers.
- *Membership Type*: Select Inclusive or Exclusive subtype.
- *Inclusive*: Select if each supertype that has subtypes can be more than one subtype at a time. For example, an *EMPLOYEE* and be both an *ENGINEER* and a *WRITER*.
- *Exclusive*: Select if each supertype that has subtypes can be only one subtype at a time. For example, a PERSON cannot be both an *EMPLOYEE* and a *CUSTOMER*.
  In IE notation, an exclusive subtype is represented with an exclusive *OR* symbol. The diagram and reports will show the type of supertype/subtype cluster in the *TYPE* field depending on what you selected, Complete, Inclusive; Complete, Exclusive; Incomplete, Inclusive; Incomplete, Exclusive.
- *Discriminator*: Select a discriminator from the list.A discriminator is an attribute that distinguishes each of the subtype entities from one another. A subtype discriminator can be a primary key or non-key attribute of the supertype. After you define a discriminator, ER Studio Data Architect displays it next to its subtype cluster symbol.

Attachment Bindings tab

Bind an external piece of information, or attachment to the domain. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary. For more information, see Attaching External Documents to the Data Model.

Notes

- You can add an entity to a subtype cluster by choosing Insert > Relationship > Identifying, clicking the subtype cluster symbol next to the supertype you want to associate the entity with, and then clicking the entity you want to add to the subtype cluster.

- You can edit the subtype by right-clicking the subtype cluster symbol and then selecting *Edit Subtype Cluster*.
- You can change the colors of the subtype cluster symbol, by right-clicking it and then selecting *Subtype Cluster Color*.
- You can remove an entity from a subtype by selecting the relationship between the subtype and the entity and then choosing Edit > Remove Relationship.
- You can delete a subtype if you no longer want to include it in your data model. You can also add or remove entities from a subtype cluster in much the same way you add or delete relationships between entities. Delete a subtype by selecting the subtype and then choosing Edit > Delete Subtype Cluster. Deleting a subtype cluster deletes all relationships between the parent entity (supertype) and the child entities (subtypes).
- You must redefine the relationships between the entities after you delete a subtype.
- While subtypes can be specified as complete or incomplete, you'll get a visual indication of it only in IDEF1X. To see the distinction in IE (Crow's feet) you can view it in the Subtype Cluster Editor, or use a naming convention or color as a mnemonic.
- IE is more flexible in supporting subtypes, permitting roll up (generalization) and roll down (specialization) transformations.

See Also

- Creating and Editing Relationships
- Creating and Editing Recursive Relationships

Resolving Name Duplication in Parent and Child Entities (Unification)

The Duplicate Attribute Editor displays when you create or rename primary key attributes, relationships, or foreign keys and it duplicates existing attributes, columns, or foreign keys in child tables or entities. For example, if you have two entities called Parent and Child where Parent has a primary key called Key and Child also has an attribute called Key, when you create a relationship from Parent to Child, then the Duplicate Native Attribute Editor launches where you must decide how to handle the propagated attributes.

- If the child entity has a foreign key with the same name as the parent entity, the Duplicate Attribute Editor launches if you have selected Prompt to Resolve FK Columns on the Application tab of the Options editor. Choose Tools > Options.
- The option, Allow Unification when Editing Names on the Options Editor - Name Handling tab allows you to change the role names of foreign attribute or columns to the same names as other attributes or columns in the same Entity or Table.

The following will help you understand how to use the *Duplicate Attribute Editor*:

- Attribute Name Editing Options
- Examples of Duplicate Name Resolutions

Attribute Name Editing Options

The following describes the options and functionality on the *Duplicate Attribute Editor*:

- *Propagating Attributes to Resolve*: Displays a list of the Attributes/Columns that are duplicates. Select the propagating attribute that you want to resolve then select the way you want to propagate.
- *Replace Native Attribute with propagating Attribute*: If selected, ER Studio Data Architect replaces the attribute in the child with the propagated foreign key attribute. If the relationship is deleted from the main model, ER Studio Data Architect removes the propagated attribute and original native attribute from the child entity.
- *Rolename propagating Attribute*: If selected, you can rename the foreign key so that the native attribute in the child entity can still exist with its original name. After propagation, both attributes will exist in the child entity.
- *Change Native Attribute Name to allow propagating Attribute Name*: If selected, you can rename the original native attribute in the child entity so that the attribute name from the parent entity can be used for the foreign key. After propagation, both attributes will exist in the child entity.
- *Unify Native Attribute with propagating Attribute*: If selected, ER Studio Data Architect unifies the propagating foreign with the native attribute in the child table. If the relationship is later deleted, ER Studio Data Architect leaves the native child attribute.

Examples of Duplicate Name Resolutions

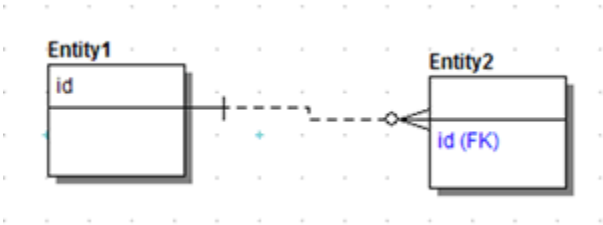


The following examples illustrate how duplicate names are resolved depending on the options chosen in the Duplicate Attribute Editor dialog.

- Replace Native Attribute with Propagating Attribute
- Rolename Propagating Attribute
- Change Native Attribute Name to Allow Propagating Attribute Name
- Unify Native Attribute with Propagating Attribute

Replace Native Attribute with Propagating Attribute

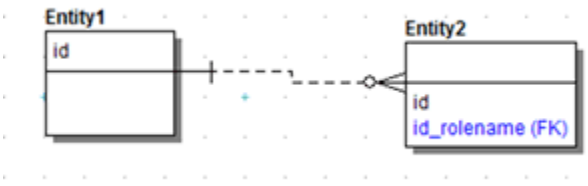
If selected, ER Studio Data Architect replaces the attribute in the child with the propagated foreign key attribute. If the relationship is deleted from the main model, ER Studio Data Architect removes the propagated attribute and original native attribute from the child entity.



Rolename Propagating Attribute

If selected, you can rename the foreign key so that the native attribute in the child entity can still exist with its original name. After propagation, both attributes will exist in the child entity.

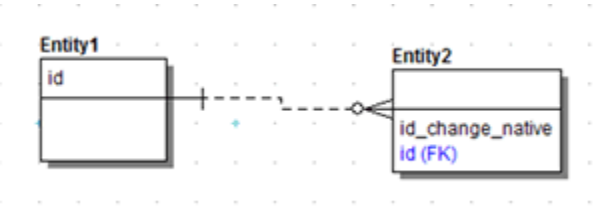
- To resolve the naming conflict, in the Duplication Attribute Editor you can specify the rolename *idrolename* for the foreign key.



Change Native Attribute Name to Allow Propagating Attribute Name

If selected, you can rename the original native attribute in the child entity so that the attribute name from the parent entity can be used for the foreign key.

- To resolve the foreign key, the new child name is specified as id_change_native.



Unify Native Attribute with Propagating Attribute

If selected, ER Studio Data Architect unifies the propagating foreign with the native attribute in the child table.

- ER Studio Data Architect keeps id in the child entity if you delete the relationship.



See Also

- Creating and Editing Relationships
- Creating Subtype Cluster Relationships

Validating the Model

The Model Validation wizard helps enforce your selected standards for data model objects and data dictionary usage. Validating the model also eases the process of model correction and review. More than 50 checks validate either the logical or physical model for problems such as missing object definitions, unused domains, identical unique indexes, and circular relationships. You can choose which validation options to use and sort the report results found by the wizard so that colleagues know exactly where to begin when correcting a model. You can also build validation templates by using the Quicklaunch feature.

Validate the Model

1. In the *Data Model Explorer*, navigate to and then right-click the top-level of the model node.
2. Click *Validate Logical Model* (or Physical model) from the shortcut menu.
3. In the *Model Validation Wizard,* choose your validation rules and options as follows:
4. Click *Run Validation*.
5. Click the *Output* tab to view and sort the results, and then to export the results if desired.

The following help you complete the *Model Validation Wizard* options:

- Validation Options tab
- Object Selection tab
- Notes

Validation Options tab

Select the validation rules you want enforced for the model and for the Data Dictionary.

The following options support Oracle's foreign key disable/enable syntax. The relationship between two entities can be disabled as can the primary key index, allowing a relationship to have a disabled primary key in the parent object and an enabled foreign key in the child object, and vice versa. This may necessary sometimes, however, Oracle will raise an error if a primary key is disabled but the foreign key is enabled (or vice-versa). By selecting the options below, the validation wizard will check for these conditions and raise an error if they exist. Deselecting these options will ignore these conditions.

- Index: Disabled PK With At Least One Enabled FK
- Foreign Key: Enabled FK With A Disabled PK

Object Selection tab

Select the objects to which the rules should be applied.

Notes

- For information on using the Quick Launch feature, see Saving and Using Quick Launch Settings.
- If you have selected the SQL Validation option on the View tab of the Options Editor then whenever you update the sql for a view, the validation dialog will appear to notify you of any syntax errors. Choose Tools > Options > View.
- If you select View SQL Alerts on the Repository tab of the Options Editor then when you check your models into the Repository you'll be notified of any SQL problems. Choose Tools > Options > Repository.

See Also

- Developing the Physical Model
- Common Tasks
- Optimizing Query Performance on the Physical Model (Denormalizing the Physical Model)
- Working with Relationships

Customizing Datatype Mappings

When creating an entity attribute you can assign it a datatype, which defines how the data is stored in the database along with the length and precision. When you want to generate a physical model from the logical model, the datatype of the logical attribute is translated into a physical datatype. The target datatype used in the logical to physical translation depends on the target platform and the datatype mapping. Physical datatypes are platform specific and can differ depending on your DBMS platform. You can view the specific mappings for each supported platform in the Datatype Mapping Editor. You can either use the default system mappings or create customized mappings.

Using the designated mapping, when generating a physical model ER Studio Data Architect knows if, for example, the logical CHAR datatype should be converted into a string, C, or remain unchanged. From the datatype mappings ER Studio Data Architect also knows the length and precision of the physical datatype. ER Studio Data Architect supports many different database platforms and provides standard/default datatype mappings for each platform. For example, using the default mappings for Oracle 10g and 11g, ER Studio Data Architect maps FLOAT to BINARY_FLOAT, whereas for Oracle 9i a FLOAT attribute is mapped by default to DOUBLE PRECISION. You can customize the default datatype mappings to create new mappings that better suit your purposes.

Use the Datatype Mapping Editor to customize the datatype mapping as you move your logical models to physical models.

1. Choose Tools > Datatype Mapping Editor.
2. Display the mapping for the target platform, by choosing a mapping from the *Select Mapping* list or to create a new mapping, click *New Physical Mapping*, enter a name for the mapping, and then click *OK*.
3. Make the changes as required, and then click *OK*.

Notes

- In the Datatype Mapping Editor, you can change the Physical Mapping values for every logical datatype.
- In the Datatype Mapping Editor, you can change the Is Default, Default Length, and Default Precision values only if you have not defined a length, precision, or default for the datatype in the logical model.
- *Is Default* refers to whether the logical type is the default for a given physical type. This is used when reverse-engineering or importing from SQL. Since physical types can be mapped to more than one logical type, when we get a physical type from the database or SQL we need to assign a logical type to it as well. When a physical type is mapped to more than one logical type, the mapping that has Is Default marked as Yes is the one used. A physical type can only have one default logical mapping.

- *Default length*: Can be negative, as -1 is a valid length for certain datatypes on some platforms.
- *Default precision*: Can be negative, as -1 is a valid length for certain datatypes on some platforms. -2 indicates that the precision is not used for this data type.
- Click *New Custom Datatype* to define custom datatypes so the unsupported datatypes will not be assigned a default datatype, such as CHAR (10) when importing, reverse engineering or merging databases that contain those unsupported datatypes.
- Rename a datatype by clicking Rename in the Datatype Mapping Editor.
- Customized datatype mappings are saved as.xml files in the...Users\All Users\Embarcadero\ERStudioDA_X.X\DatatypeMapping directory. For AppWave ION users these files are saved in My Documents\ERStudio Data Architect XE\DatatypeMapping.
- In the Generate Physical Model Wizard, the list of datatype mappings that you can choose from is limited to the system mapping for the physical platform and any customized mappings that were based on this system mapping.

See Also

- Logical Design Concepts
- Working with Universal Mappings

Developing the Physical Model

In order to create a database, ensure you have the client software installed and are using the correct connection strings. For more information, see Connecting to Database Sources and Targets.

If you do not have the client connection software for the platform for which you want to create a database, you can generate a physical model from the logical model (See Generating a Physical Data Model), and then change the database platform to the target database. You can then access the Database Wizard to create the DDL required to create the database.

Generally, a logical data model represents business information and defines business rules, and a physical data model represents the physical implementation of the model in a database. The physical design addresses the technical implementation of a data model, and shows how the data is stored in the database. For example, you can specify the datatype of each column in the table, and determine how tables will be stored in the database.

To design the most effective data model possible, focus on the logical design before developing the physical design. A physical data modeler should have the technical-know-how to create data models from existing databases and to tune the data models with referential integrity, alternate keys, indexes and how to match indexes to SQL code. In systems development, the goal is to create an effective database application that can support some or all of your enterprise.

The physical design focuses on performance and tuning issues for a particular database platform. A highly normalized database design often results in performance problems, particularly with large data volumes and many concurrent users. These problems are often due to complex query patterns, the cost of multi-table joins, and the high I/O intensity of querying or modifying large tables. When addressing performance problems, you should consider the following physical design issues:

- Transforming Model Objects from Logical to Physical: How the elements of a physical model correspond to the elements of the corresponding logical model.
- Optimizing Query Performance on the Physical Model (Denormalizing the Physical Model): How to modify the logical design to increase the efficiency of database operations.
- Creating and Editing Indexes: How to create indexes to increase the efficiency of read operations (queries) without sacrificing the efficiency of write operations (insert, update, delete).
- Physical Storage Considerations: How to determine the physical storage of the tables and indexes in the database to maximize efficiency and allow for growth.

For information on physical data model objects in ER Studio Data Architect, see the following topics:

- Creating and Editing Entities and Tables
- Creating and Editing Attributes and Columns

- Customizing Table DDL
- Creating and Editing Indexes
- Defining Table Storage
- Creating and Editing Database Dependent Objects

See Also

- Creating and Editing Relationships
- Upgrading Diagrams with Denormalization Mappings
- Synchronizing Physical and Logical Models
- Planning for and Predicting Database Growth

Transforming Model Objects from Logical to Physical

Before considering physical design issues, it is important to understand how a physical design maps to its corresponding logical design. When ER Studio Data Architect derives a physical design from a logical one, it transforms logical model objects into their physical model analogs. The table below details how different logical objects are transformed into physical objects. Understanding the transformation lets you better assess the impact of altering the physical design to differ from the logical design.

Logical Model Object	Physical Model Object	Notes
Entity	Table	An entity translates directly into a table. In a dimensional model, tables are automatically assigned types (fact, dimension, snowflake, or undefined) based on an analysis of the schema.
Attribute	Column	An attribute translates directly into a table column.
View	View	Views translate directly between logical and physical models. However, some database platforms offer additional properties that can be applied to physical view definitions.
Relationship	Foreign Key	A relationship becomes a foreign key in the physical design. You can enforce referential integrity through foreign key constraints or through triggers.
Primary Key	Unique Index or Primary Key Constraint	You can enforce primary keys with unique indexes or primary key constraints, depending on the database platform.
Alternate Key	Unique Index or Unique Constraint	You can enforce alternate keys with unique indexes or unique constraints, depending on the database platform.
Inversion Entry Key	Non-unique Index	You can convert inversion entries into non-unique indexes.
Default	Default or Declared Default	Data Dictionary defaults are converted to default objects for Sybase and Microsoft SQL Server. Otherwise, they are converted to declared defaults, if supported by the selected database platform.
Rule	Rule, Check Constraint, Default Value	Data Dictionary rules are converted to rule objects for Sybase and Microsoft SQL Server. Otherwise, they are converted to check constraints, if supported by the selected database platform.
Relationship	Foreign Key	
Definition	Comment	

User Datatype	User Datatype or Base Datatype	Data Dictionary datatypes are converted to user datatypes for Sybase and Microsoft SQL Server. Otherwise, they are converted into their underlying base datatype definitions.
Domain	Column Definition	Domains are translated into the base column definition for each column that refers to a domain.

The table below describes logical model objects and their corresponding physical model objects:

See Also

- Developing the Logical Model
- Physical Storage Considerations

Physical Storage Considerations

Planning the storage of tables and indexes is best done during the physical design stage in order to improve performance and to streamline data administration tasks. When planning physical storage, carefully consider both the placement and size of tables and indexes.

The performance of almost all database applications is I/O bound. To improve I/O throughput, you should physically separate tables that are frequently joined together. You should also separate tables from their indexes. The objective is to have the database read or write data in parallel as much as possible.

Two key concerns of every database administrator are free space management and data fragmentation. If you do not properly plan for the volume and growth of your tables and indexes, these two administrative issues could severely impact system availability and performance. Therefore, when designing your physical model, you should consider the initial extent size and logical partition size.

As a starting point, you should estimate the size of each table and its indexes based on a projected row count. For databases that let you specify initial extent sizes, such as Oracle, you should set the initial extent size to the estimated size in order to avoid data fragmentation as the table grows. By keeping tables within a single extent, you can decrease data access times and avoid reorganizing tables.

Once you have determined the placement and sizes of the tables and indexes in your database, you can estimate the total size requirements of the database. This should help you avoid running out of free space for your database.

The following sections describe how you can optimally store and partition tables and indexes through the options available in the table and index editors for the specific database platforms that support these options.

- Defining Table Storage
- Distributing Columns for IBM DB2 for LUW 9.x (and later versions)
- Distributing Columns for GreenPlum
- Distributing Columns for Netezza
- Organizing Columns for Netezza
- Partitioning a Table
- Defining Index Storage
- Partitioning a Table Index

See Also

Creating and Editing StoGroups

Creating and Editing Tablespaces

Defining Table Storage

Storage parameters affect both how long it takes to access data stored in the database and how efficiently space in the database is used. For heavily accessed tables, place these tables in one location and place the table's indexes in a different location on different physical disk arrays. This will improve performance, because separate threads will be created to access the tables and indexes.

You can define how your tables are stored using the options in the Storage tab of the Table Editor.

You can define how the indexes in your tables are stored using the options in the Index tab of the Table Editor. For more information, see Defining Index Storage.

Also to be considered, is how to store the overflow records when a row is updated and no longer fits on the page where it was originally written. The Overflow tab of the Table Editor is available for index-organized tables on the Oracle 8.x and 9.x platforms. For more information, see Define Table Overflow Options:

1. In the *Data Model Window*, double-click the table for which you want to specify storage options.
2. From the *Table Editor*, select the *Storage* tab.
3. Complete the *Storage* tab options as required and then click *OK* to exit the editor.
4. The table storage options available in the Table Editor depend on the database platform.

Click a link below for information on the storage options for your database platform:

GreenPlum Table Storage Options	HiRDB Table Storage Options
HiRDB LOB Column Storage	IBM DB2 Common Server and IBM DB2 for LUW Table Storage Options
IBM DB2 for OS/390 Table Storage Options	Informix Table Storage Options
Interbase Table Storage Options	Microsoft SQL Server Table Storage Options
Teradata Table Storage Options	Oracle 7 Table Storage Options
Oracle 8.x, 9i, 10g, and 11g Table Storage Options	Oracle LOB Segment Storage Options
PostgreSQL Table Storage Options	Sybase Table Storage Options

GreenPlum Table Storage Options

- *Tablespace* (*TABLESPACE*): Displays the tablespace on which the table is stored.

HiRDB Table Storage Options

- *FIXX*: Specifies a fixed length table, without null data. Select this option to reduce table access time and thus improve performance.
- *Table Storage*: Select to enable the *IN* and *PARTITIONED BY* options. Select *IN* to define the RD area of the table, the space where the table and index information is stored. Select *PARTITIONED BY* to define the partition setting, such as *(RDINDX19) 10, (RDDATA10)*.
- *PCTFREE*: (*PCTFREE*) Specifies the maximum percentage of space in each data block to reserve for future updates. This reserved space helps to avoid row migration and chaining caused by an update operation that extends a data row's length. Tables that will not be updated should have this value set to 0. Specify *PCTFREE* in one of the following formats:

UNUSED-SPACE-PERCENTAGE

(UNUSED-SPACE-PERCENTAGE, FREE-PAGES-PERCENTAGE-PER-SEGMENT)

(, FREE-PAGES-PERCENTAGE-PER-SEGMENT)

- *Lock Mode*: Select the locking mode. Select *Page* to lock all rows on a page when one row is locked for update. Select Row to lock only the row being updated.

HiRDB LOB Column Storage

If the column datatype specified on the Datatype tab is *BLOB*, this tab becomes available where you can specify a large object storage location.

IBM DB2 Common Server and IBM DB2 for LUW Table Storage Options

- *Database*: Displays the database on which the table is stored. Enter the name of the database or choose a database from the list.
- *Tablespace*: Displays the tablespace on which the table is stored. Enter the name of the tablespace or choose another tablespace from the list.
- *Encoding*: Select to enable encoding and then choose the encoding format desired. Creates a *CCSID* statement.
- *Restrict On Drop*: Select to prevent users from accidently dropping the table (removing the table from the database) until the *RESTRICT ON DROP* attribute is removed. Creates a *RESTRICT ON DROP* statement.
- *Volatile (VOLATILE)*: Select to define the table as volatile so that when querying the table, the DB2 optimizer uses the indexes defined on the table even though statistics

can indicate that using the index would result in slower performance. This prevents the optimizer from using table statistics that are out of date that would result in poor performance.

- *Log Data Changes (DATA CAPTURE)*: Select to log *INSERT*, *UPDATE*, and *DELETE* statements.
- *Table Procedures (EDITPROC* and *VALIDPROC)*: Specify edit procedures and validation procedures which can be run against the table. Creates the *EDITPROC* and *VALIDPROC* statements.
- *Audit Option (AUDIT CHANGES* or *AUDIT ALL*): Specify the audit policy to use to monitor table activity. Audited events are determined by the server's associated audit policy.
- *Index Tablespace*: Displays the tablespace on which the table index is stored. Choose another tablespace from the list.
- *LONG Tablespace*: Displays the name of the tablespace on which the long objects (*LOB*, *CLOB*) stored on the table reside. Choose another tablespace from the list if you like.
- *Log for Replication*: Select to keep a record of all changes in the table. Replication enables the data to be copied from a source table to one or more target tables and synchronize the changes in the source to the target.
- *Not Logged Initially*: Select to prevent logging on operations that are performed in the same unit of work in which the table is created; however, other operations against the table in subsequent units of work are logged.

IBM DB2 for OS/390 Table Storage Options

The options available depend on the version of the database platform.

- *Database*: Displays the database on which the table data is stored. Choose another database from the list if you like.
- *Tablespace*: Displays the tablespace on which the table data is stored. Choose another tablespace from the list if you like.
- *Log Data Changes*: Select to log any changes to the table data.
- *Audit Option*: Choose the audit option desired. Record activity related to changes only or record all access activities.
- *Edit Proc*: Enter the name of an edit routine that edits or encodes a row whenever a row is inserted or updated. Typically used to compress the storage of rows to save space, and to encrypt the data.
- *Valid Proc*: Enter the name of the validation routine that validates the data whenever a row is inserted or updated. Typically used to impose limits on the information that can

be entered in a table; for example, allowable salary ranges for job categories in the employee table.

- *Encoding*: Select to enable encoding and then choose the encoding format desired.
- *Restrict On Drop (RESTRICT ON DROP)*: Select to prevent users from accidently dropping the table (removing the table from the database) until the *RESTRICT ON DROP* attribute is removed.
- *Volatile*: Select to define the table as volatile so that when querying the table, the DB2 optimizer uses the indexes defined on the table even though statistics can indicate that using the index would result in slower performance. This prevents the optimizer from using table statistics that are out of date that would result in poor performance.

Informix Table Storage Options

The options available depend on the version of the database.

- *DB Space*: Specifies the name of the database in which the table data is stored.
- *Lock Mode*: Specifies the locking mode. Select *Page* to lock all rows on a page when one row is locked for update. Select *Row* to lock only the row being updated.
- *Initial Extent*: (*INITEXTENT*) Specifies the table's initial number of data blocks. Informix will reserve the data blocks that correspond to the initial extent for that table's rows.
- *Next Extent*: (*NEXT*) Specifies the size of the next extent, in kilobytes. Monitor this value in comparison with the largest available chunk of free space in the tablespace if the free space is less than the next extent size then the table extent cannot be allocated; the table will no longer be able to extend and, therefore, cannot accept additional data.
- *Path Name*: Specifies the full path name of the database in which the table data is stored.

Interbase Table Storage Options

- *External File*: Specifies the file where the table data is stored.

Microsoft SQL Server Table Storage Options

- The options available depend on the version of the database platform.
- *Segment*: Specifies where the table data is stored.
- *File Group*: Specifies the name of the file group where the table data is stored.
- *Partition Scheme*: Specifies the name of the partition scheme created that associates the partitions with their physical location in a file group.

- *Text Image File Group*: Specifies the name of the file group where text or image data for the table is stored. This option becomes available when any of the columns in the table have an image datatype.
- *Partition Keys*: Specifies the partition key. This column guides how data is divided along the partition boundaries. Typically, a partition key is data-based or a numerically ordered data grouping. You can choose from all available columns in the table.

Teradata Table Storage Options

- *Journal Table (WITH JOURNAL TABLE)*: Specifies the name of the journal table in the format *database_name.table_name*. If the database name is not specified, the default database for the current session is assumed. If a default journal table was defined for the database, then this option can override the default. The journal table stores an image of the table before initiating a transaction. If the transaction completes successfully, the journal is deleted; if the transaction fails, the journal data is used to perform a rollback.
- *Before Journal (BEFORE JOURNAL)*: Specifies the number of before change images to be maintained. If the *JOURNAL* keyword is specified without *NO* or *DUAL*, then a single copy of the image is maintained unless *FALLBACK* is in effect or is also specified. If journaling is requested for a table that uses fallback protection, *DUAL* images are maintained automatically.
- *After Journal (AFTER JOURNAL)*: Specifies the number of after-image to be maintained for the table; any existing images are not affected until the table is updated.
- *Freespace*: Specifies percent free space (between 0 and 75) that remains on each cylinder during bulk data load operations on this table. This reserved space helps to avoid row migration and chaining caused by an update operation that extends a data row's length. Tables that won't be updated should have this value set to 0.
- *Datablock Size*: Specifies the maximum data block size for blocks contain multiple rows. Larger block sizes enhance full-table scans by selecting more rows in a single I/O; smaller block sizes are best for transaction-oriented tables to minimize overhead by retrieving only what is needed. The maximum data block size is 127.5 Kilobytes.
- *Duplicate Row Control*: Specifies whether or not duplicate rows are permitted. Specify Set to disallow duplicate rows and Multiset to allow duplicate rows.
- *Checksum (CHECKSUM)*: Checksum detects and logs disk I/O errors. The choices available determine the percentage of data used to compute the checksum for the table. This is an example of the SQL generated with *Checksum > Medium* has been selected on the *Storage* tab of the *Table Editor*.

Checksum = Medium

- *Fallback Protection (FALLBACK)*: Select this option to implement software fault tolerance which copies the primary table to create a duplicate copy.

Oracle 7 Table Storage Options

- *Tablespace*: Specifies the name of the tablespace where the table data is stored.
- *Initial Transactions*: (*INITTRANS*) Specifies the number of DML transactions for which space is initially reserved in the data block header. Oracle stores control information in the data block to indicate which rows in the block contain committed and uncommitted changes. The amount of history that is retained is controlled by this parameter. If too many transactions concurrently modify the same data block in a very short period, Oracle may not have sufficient history information to determine whether a row has been updated by a 'too recent' transaction. Specify a higher value for tables that may experience many transactions updating the same blocks.
- *Max Transactions*: (*MAXTRANS*) Once the space reserved by *INITRANS* is depleted, space for additional transaction entries is allocated out of any available free space. Once allocated, this space effectively becomes a permanent part of the block header. This parameter limits the number of transaction entries that can concurrently use data in a data block and therefore limits the amount of free space that can be allocated for transaction entries in a data block.
- *Percent Free*: (*PCTFREE*) Specifies the maximum percentage of space in each data block to reserve for future updates. This reserved space helps to avoid row migration and chaining caused by an update operation that extends a data row's length. Tables that will not be updated should have this value set to 0.
- *Percent Used*: (*PCTUSED*) Specifies the maximum percentage of available space in each data block before re-adding it to the list of available blocks. When deletes take place and the room available in a block falls below this value, the block is made available for new inserts to take place. Tables that won't be updated should have this value set to 99. The default value is 40% means that blocks are available for insertion when they are less than 40% full.
- *Initial Extent*: (*INITEXTENT*) Specifies the initial number of data blocks that Oracle should reserve. Oracle will reserve the number of data blocks that correspond to the initial extent for that table's rows.
- *Next Extent*: (*NEXT*) Specifies the size in kilobytes of the next extent. Monitor this figure against the largest available chunk of free space in the tablespace. If a table cannot allocate its next extent, it will no longer be able to extend and, therefore, cannot accept additional data.
- *Percent Increase*: (*PCTINCREASE*) Specifies the percentage by which the next extent should grow over the previous extent's size. Be careful when setting Percent Increase because it magnifies how an object grows and, therefore, can materially affect available free space in a tablespace.
- *Min Extents*: (*MINEXTENTS*) Specifies the number of extents to allocate when the segment is created. Controls free space fragmentation by making sure that every used or free extent is at least as large as the value you specify.
- *Max Extents*: (*MAXEXTENTS*) Specifies the maximum number of extents that Oracle can allocate to the materialized view. Once this limit is reached, Oracle prevents further

growth of the cluster and cannot accept additional data. As a result, you should carefully monitor the number extents already allocated to the table with this limit.

Oracle 8.x, 9i, 10g, and 11g Table Storage Options

The options available depend on the version of Oracle used.

- *Heap*: Organizes the table with physical rowids. Oracle Corporation does not recommend that you specify a column of datatype *UROWID* for a heap-organized table.
- *Cache*: (*CACHE*) Select for data that is accessed frequently to specify that blocks retrieved for this table be placed at the most recently used end of the least recently used (*LRU*) list in the buffer cache when a full table scan is performed. This attribute is useful for small lookup tables.
- *Index Organized*: (*ORGANIZATION INDEX*) Groups data rows according to the primary key. It the table is Index Organized you can specify how overflow is handled.For more information, see Define Table Overflow Options. You can also specify that the index be compressed. For more information, see Defining Index Storage.
- *Temporary Table*: (*GLOBAL TEMPORARY TABLE*) Creates a global temporary table whose definition is visible to all sessions but the data is visible only to the session that created it until the session or transaction is committed. The data persists in the temporary table at either the session or transaction level based on how *ON COMMIT* is specified. You cannot use this option for tables with relationships.
- *DELETE ROWS*: Deletes the temporary table with the user ends the transaction by issuing a commit statement.
- *PRESERVE ROWS*: Retains the table changes until the session is ended.
- *Enable Table Compression*: (*COMPRESS*) Select to enable compression, which can reduce disk and buffer cache requirements, while improving query performance in addition to using fewer data blocks thereby reducing disk space requirements. This is most useful for fact tables in a data warehouse environment.
- *Enable Row Movement*: (*Enable Row Movement*) Select to allow Oracle to change ROWIDs to condense table rows and make it easier to reorganize tables. However, this option also allows Oracle to move a row to a discontinuous segment which can cause performance problems. This option must be enabled in order for you to use Oracle features such as ALTER TABLE SHRINK, flashback table, and table reorganization.
- *Tablespace*: Specifies the name of the tablespace on which the table is stored.
- *No Logging*: (*NOLOGGING*) Select if you do not want the DLL operations to be logged in the redo file. This can reduce index creation and updates by up to 30%.
- *Parallel*: (*PARALLEL*) Selects Oracle's parallel query option, allowing for parallel processes to scan the table.You can achieve substantial performance gains by using Oracle's parallel query option.

- *Degrees*: Specifies the number of query server processes that should be used in the operation. Usually this would be the number of CPUs on the Oracle server -1.
- *Instances*: Specifies how you want the parallel query partitioned between the parallel servers.
- *Pct Free*: (*PCTFREE*) Specifies the maximum percentage of space in each data block to reserve for future updates. This reserved space helps to avoid row migration and chaining caused by an update operation that extends a data row's length. Tables that will not be updated should have this value set to 0.
- *Pct Used*: (*PCTUSED*) Specifies the maximum percentage of available space in each data block before re-adding it to the list of available blocks. When deletes take place and the room available in a block falls below this value, the block is made available for new inserts to take place. Tables that will not be updated should have this value set to 99. The default value is 40% means that blocks are available for insertion when they are less than 40% full.
- *Initial Trans*: (*INITTRANS*) Specifies the number of DML transactions for which space is initially reserved in the data block header. Oracle stores control information in the data block to indicate which rows in the block contain committed and uncommitted changes. The amount of history that is retained is controlled by this parameter. If too many transactions concurrently modify the same data block in a very short period, Oracle may not have sufficient history information to determine whether a row has been updated by a *too recent* transaction. Initial Transactions limit the minimum number of concurrent transactions that can update a data block to avoid the overhead of allocating a transaction entry dynamically. Specify a higher value for tables that may experience many transactions updating the same blocks.
- *Max Trans*: (*MAXTRANS*) Specifies the maximum number of concurrent transactions that can update a data block to avoid performance problems. Once the space reserved by *INITRANS* is depleted, space for additional transaction entries is allocated out of any available free space. Once allocated, this space effectively becomes a permanent part of the block header. This parameter limits the number of transaction entries that can concurrently use data in a data block and therefore limits the amount of free space that can be allocated for transaction entries in a data block.
- *Initial Extent*: (*INITEXTENT*) Specifies the initial number of data blocks that Oracle should reserve. Oracle will reserve the number of data blocks that correspond to the initial extent for that table's rows.
- *Next Extent*: (*NEXT*) Specifies the size in kilobytes of the next extent. Monitor this figure against the largest available chunk of free space in the tablespace. If a table cannot allocate its next extent, it will no longer be able to extend and, therefore, cannot accept additional data.
- *Pct Increase*: (*PCTINCREASE*) Specifies the percentage by which the next extent should grow over the previous extent's size. Be careful when setting Percent Increase because it magnifies how an object grows and, therefore, can materially affect available free space in a tablespace.

- *Min Extents*: (*MINEXTENT*) Specifies the number of extents to allocate when the segment is created. Controls free space fragmentation by making sure that every used or free extent is at least as large as the value you specify.
- *Max Extents*: (*MAXEXTENT*) Species the maximum number of extents that Oracle can allocate to the materialized view. Once this limit is reached, Oracle prevents further growth of the cluster and cannot accept additional data. Carefully monitor the number extents already allocated to the table with this limit.
- *Free Lists*: (*FREELISTS*) Specifies the number of free lists to apply to the table. The default and minimum value is 1. Free lists can help manage the allocation of data blocks when concurrent processes are issued against the index. Oracle uses the free list to determine which data block to use when an *INSERT* operation occurs. Oracle allows table and indexes to be defined with multiple free lists. All tables and index free lists should be set to the high-water mark of concurrent *INSERT* or *UPDATE* activity. For example, if the table has up to 20 end users performing *INSERT* s at any time, then the index should have *FREELISTS=20*.Too low a value for free lists will cause poor Oracle performance. An increase in *FREELISTS* or *FREELIST GROUPS* can alleviate segment header contention.
- *Free List Groups*: (*FREELIST GROUPS*) Applicable only if you are using Oracle with the Parallel Server option in parallel mode. Specifies the number of free list groups which allow the table to have several segment headers. This enables multiple tasks to insert into the index; thereby alleviating segment header contention. Free list groups should be set to the number of Oracle Parallel Server instances that access the index. For partitioned objects and cases of segment header contention, free list groups my be set for non-RAC systems.
- *Buffer Pool*: (*BUFFER_POOL*) Specifies the memory structure that is used for caching data blocks in memory, providing faster data access.
- *DEFAULT* caches data blocks in the default buffer pool.
- *KEEP* retains the object in memory to avoid I/O conflicts.
- *RECYCLE* removes data blocks from memory as soon as they are no longer in use, thereby saving cache space.

Oracle LOB Segment Storage Options

If a datatype is defined as *LOB*, *CLOB* or *NLOB* on the *Datatype* tab of the *Table Editor*, the *Edit LOG Segment* option is available, which when clicked opens the *LOG Segment* editor. The following describes the options available on the *LOG Segment* editor:

- *Segment Name*: Specifies the name of the segment on which the *LOB* data is stored.
- *Tablespace*: Specifies the name of the tablespace on which the *LOB* data is stored.
- *Blocks Accessed at one time*: (*CHUNK*) Determines the granularity of allocation for out of row LOBs.

- *Percent Space Used for New Version*: (*PCTVERSION*) Controls the mechanism that stores the previous image within the data block. The default value for PCTVersion is 10, reserving 10% of the LOB storage space for UNDO operations.
- *Enable LOB Storage In Row*: An LOB can be stored either along the row to which it belongs (in row storage) or in the lob segment (out of row storage). The maximum size for in row LOBs is 3964 bytes.
- *LOB Cache*: (CACHE) Select for data that is accessed frequently to specify that blocks retrieved for this column be placed at the most recently used end of the least recently used (*LRU*) list in the buffer cache when a full table scan is performed.
- *Logging*: (*LOGGING*) Select if you want the DLL operations to be logged in the redo file. This can increase index creation and updates by up to 30%.
- *Initial Extent*: (INITEXTENT) Specifies the initial number of data blocks that Oracle should reserve. Oracle will reserve the number of data blocks that correspond to the initial extent for that table's rows.
- *Next Extent*: (NEXT) Specifies the size in kilobytes of the next extent. Monitor this figure against the largest available chunk of free space in the tablespace. If a table cannot allocate its next extent, it will no longer be able to extend and, therefore, cannot accept additional data.
- *Pct Increase*: (PCTINCREASE) Specifies the percentage by which the next extent should grow over the previous extent's size. Be careful when setting Percent Increase because it magnifies how an object grows and, therefore, can materially affect available free space in a tablespace.
- *Min Extents*: (MINEXTENT) Specifies the number of extents to allocate when the segment is created. Controls free space fragmentation by making sure that every used or free extent is at least as large as the value you specify.
- *Max Extents*: (MAXEXTENT) Species the maximum number of extents that Oracle can allocate to the materialized view. Once this limit is reached, Oracle prevents further growth of the cluster and cannot accept additional data. Carefully monitor the number extents already allocated to the table with this limit.
- *Free Lists*: (FREELISTS) Specifies the number of free lists to apply to the table. The default and minimum value is 1. Free lists can help manage the allocation of data blocks when concurrent processes are issued against the index. Oracle uses the free list to determine which data block to use when an INSERT operation occurs. Oracle allows table and indexes to be defined with multiple free lists. All tables and index free lists should be set to the high-water mark of concurrent INSERT or UPDATE activity. For example, if the table has up to 20 end users performing INSERTs at any time, then the index should have FREELISTS=20.Too low a value for free lists will cause poor Oracle performance. An increase in FREELISTS or FREELIST GROUPS can alleviate segment header contention.
- *Free List Groups*: (FREELIST GROUPS) Applicable only if you are using Oracle with the Parallel Server option in parallel mode. Specifies the number of free list groups which allow the table to have several segment headers. This enables multiple tasks to insert

into the index; thereby alleviating segment header contention. Free list groups should be set to the number of Oracle Parallel Server instances that access the index. For partitioned objects and cases of segment header contention, free list groups my be set for non-RAC systems.

PostgreSQL Table Storage Options

- *Table Space*: Specifies the name of the tablespace on which the table data is stored.

Sybase Table Storage Options

The options available depend on the version of the database platform.

- *DB Space*: Specifies the name of the database where the table data is stored.
- *Segment*: Specifies the name of the segment where the table is stored.
- *Locking Scheme*: Specifies the locking scheme to implement for DDL transactions.
- *ALLPAGES*: Locks the data pages and index pages affected by the query.
- *DATAPAGES*: Locks the data pages but not the index pages, and the lock is held until the end of the transaction.
- *DATAROWS*: Locks the individual rows on data pages; rows and pages are not locked. This scheme requires more overhead but is beneficial for small tables with much contention.
- *Max Rows per Page*: Specifies the maximum number of rows per page.
- *Reserve Page Gap*: Specify the reserve gap parameters.
- *Identity Gap*: Controls the allocation of ID numbers and potential gaps resulting from restarting the server. Creates ID number blocks of a specific size which improves performance by reducing contention for the table. Overrides the server-wide identity burning set factor. If the identity gap is set to 1000m the server allocates the first 1000 numbers and stores the highest number of the block to disk. When all the numbers are used, the server takes the next 1000 numbers. If after inserting row 2050 the power fails, then when the server restarts it will start with ID 3000, leaving an ID gap of 950 numbers. Each time the server must write the highest number of a block to disk, performance is affected so you must find the best setting to achieve the optimal performance with the lower gap value acceptable for your situation.
- *Replacement Strategy*: Select to use the fetch-and-discard (or MRU) replacement strategy that reads pages into cache at the most recently-used (MRU) end of the buffer, so it does not flush other pages. But these pages remain in cache a much shorter time, so subsequent queries are less likely to find the page in cache.
- *Prefetch Strategies*: Enables or disables large I/O for a session.

Define Table Overflow Options

The Table Editor Overflow tab specifies how to store the overflow records when a row is updated and no longer fits on the page where it was originally written. The Overflow tab of the Table Editor is available for index-organized tables on the Oracle 8.x, 9i, 10g, 11g and 12c platforms.

1. In the *Data Model Window*, double-click the table for which you want to specify storage options.
2. From the *Table Editor*, select the *Overflow* tab.
3. Complete the *Overflow* tab options as required and then click *OK* to exit the editor.

The *Overflow* tab creates an *ORGANIZATION INDEX* clause in the *CREATE TABLE* statement. Any options you specify on this tab will follow this clause.

The following describes options on the Overflow tab that require additional explanation.

- *Tablespace*: Specifies the name of the tablespace on which the overflow is stored.
- *No Logging*: Select if you do not want the DLL operations to be logged in the redo file. This can reduce index creation and updates by up to 30%.
- *Pct Threshold*: (*PCTTHRESHOLD*): Specifies the percentage of space reserved in the index block for the index-organized table. Any portion of a row that exceeds this threshold is stored in the overflow segment.
- *Including*: (*INCLUDING*) Select the column to include in the overflow.
- *Pct Free*: (*PCTFREE*) Specifies the maximum percentage of space in each data block to reserve for future updates. This reserved space helps to avoid row migration and chaining caused by an update operation that extends a data row's length. Tables that will not be updated should have this value set to 0.
- *Pct Used*: (*PCTUSED*) Specifies the maximum percentage of available space in each data block before re-adding it to the list of available blocks. When deletes take place and the room available in a block falls below this value, the block is made available for new inserts to take place. Tables that won't be updated should have this value set to 99. The default value is 40% means that blocks are available for insertion when they are less than 40% full.
- *Initial Trans*: (*INITTRANS*) Specifies the number of DML transactions for which space is initially reserved in the data block header. Oracle stores control information in the data block to indicate which rows in the block contain committed and uncommitted changes. The amount of history that is retained is controlled by this parameter. If too many transactions concurrently modify the same data block in a very short period, Oracle may not have sufficient history information to determine whether a row has been updated by a 'too recent' transaction. Specify a higher value for indexes that may experience many transactions updating the same blocks.

- *Max Trans*: (*MAXTRANS*) Once the space reserved by *INITRANS* is depleted, space for additional transaction entries is allocated out of any available free space. Once allocated, this space effectively becomes a permanent part of the block header. This parameter limits the number of transaction entries that can concurrently use data in a data block and therefore limits the amount of free space that can be allocated for transaction entries in a data block.
- *Initial Extent*: (*INITEXTENT*) Specifies the initial number of data blocks that Oracle should reserve. Oracle will reserve the number of data blocks that correspond to the initial extent for that table's rows.
- *Next Extent*: (*NEXT*) Specifies the size in kilobytes of the next extent. You should monitor this figure against the largest available chunk of free space in the tablespace. If a table cannot allocate its next extent, it will no longer be able to extend and, therefore, cannot accept additional data.
- *Pct Increase*: (*PCTINCREASE*) Specifies the percentage by which the next extent should grow over the previous extent's size. Be careful when setting Percent Increase because it magnifies how an object grows and, therefore, can materially affect available free space in a tablespace.
- *Min Extents*: (*MINEXTENTS*) Specifies the number of extents to allocate when the segment is created. Controls free space fragmentation by making sure that every used or free extent is at least as large as the value you specify.
- *Max Extents*: (*MAXEXTENTS*) Specifies the maximum number of extents that Oracle can allocate to the index. Once this limit is reached, Oracle prevents further growth of the cluster and cannot accept additional data. Carefully monitor the number extents already allocated to the index with this limit.
- *Free Lists*: (*FREELISTS*) Specifies the number of free lists to apply to the index. The default and minimum value is 1. Free lists can help manage the allocation of data blocks when concurrent processes are issued against the index. Oracle uses the free list to determine which data block to use when an *INSERT* operation occurs. Oracle allows table and indexes to be defined with multiple free lists. All tables and index free lists should be set to the high-water mark of concurrent *INSERT* or *UPDATE* activity. For example, if the index has up to 20 end users performing *INSERT* s at any time, then the index should have *FREELISTS=20*. Too low a value for free lists will cause poor Oracle performance. An increase in *FREELISTS* or *FREELIST_GROUPS* can alleviate segment header contention.
- *Free List Groups*: (*FREELIST GROUPS*) Applicable only if you are using Oracle with the Parallel Server option in parallel mode. Specifies the number of free list groups which allow the index to have several segment headers. This enables multiple tasks to insert into the index; thereby alleviating segment header contention. Free list groups should be set to the number of Oracle Parallel Server instances that access the index. For partitioned objects and cases of segment header contention, freelist_groups can be set for non-RAC systems.
- *Buffer Pool*: (*BUFFER_POOL*) Specifies the memory structure that is used for caching data blocks in memory, providing faster data access.

- *DEFAULT*: Caches data blocks in the default buffer pool.
- *KEEP*: Retains the object in memory to avoid I/O conflicts.
- *RECYCLE*: Removes data blocks from memory as soon as they are no longer in use, thereby saving cache space.

See Also

- Physical Storage Considerations
- Creating and Editing Entities and Tables

Distributing Columns for GreenPlum

The Distribution tab of the Table Editor for the GreenPlum platform allows you to create the DISTRIBUTED BY or DISTRIBUTED RANDOMLY statements that specify the distribution key. If you do not specify the columns of the distribution key and do not activate the random distribution option, the GreenPlum server will assign to the distribution key by default by the first column created in the table.

- *Available Columns*: Displays all the columns available to add to the distribution key. Select the column you want to add to the distribution key and move it to the Selected Columns box. Use the left and right arrows to move columns to and from the Selected Columns box. When no unique indexes or primary key columns are specified, this list includes all the columns of the table.
- *Selected Columns*: Displays the columns that make up the distribution key.
- *Up / Down* buttons: Let you reorder the columns in the distribution key. The column order can affect the access speed. The most frequently accessed columns should be at the top of the Selected Columns list. The Up/Down buttons are disabled when a single unique or primary index exists. In that case, the columns in the distribution key will be ordered automatically to must match the order of the index columns.
- *Random*: This option is available on if there are no unique indexes or primary keys specified for the table. If selected, the table is created using a random round-robin distribution key.

Completing this tab with the Random option selected will produce SQL code similar to the following:

```
CREATE TABLE "Entity1"(
"PK1" CHAR(10) NOT NULL,
"A1" CHAR(10),
"A2" CHAR(10),
)
DISTRIBUTED RANDOMLY
```

Selecting PK1 to be the only column in the distribution key will produce SQL code similar to the following:

```
CREATE TABLE "Entity1"(
"PK1" CHAR(10) NOT NULL,
"A1" CHAR(10),
"A2" CHAR(10),
CONSTRAINT "PK1" PRIMARY KEY ("PK1")
```

```
)
DISTRIBUTED BY ("PK1")
```

See Also

- Creating and Editing Columns
- Creating and Editing Attributes and Columns

Distributing Columns for IBM DB2 for LUW 9.x (and later versions)

For IBM DB2 for LUW 9.x (and later versions). The Distribute Columns tab of the Table Editor allows you to create the DISTIBUTE BY... statement for IBM DB2 that specifies how the table should be distributed across databases partitions within a single tablespace.

- *Available Columns*: Displays all the columns available to add to the partition. Select the column you want to add to the partition and move it to the Selected Columns box.
- *Selected Columns*: Displays the columns that make up the partition key. This guides how data is divided along the partition boundaries. Typically, a partition key is date-based or based on a numerically ordered data grouping. Use the left and right arrows to move columns to and from this grid. Columns that are in this grid make up the partition.
- *Up / Down* buttons: Let you reorder the columns on the partition. The partition order can affect the access speed. The most frequently accessed columns should be at the top of the partition list.
- *By Hash*: For IBM DB2. If selected, ER Studio Data Architect partitions the table according to a hash function, which can speed up table lookup or data comparison tasks.

Completing this tab will produce SQL code similar to the following:

```
CREATE TABLE Entity1(

PK1 CHAR(10) NOT NULL,

A1 CHAR(10),

A2 CHAR(10),

CONSTRAINT PK1 PRIMARY KEY (PK1)

)

DISTRIBUTE BY HASH (PK1,A1)
```

A column cannot be used as part of a distribution key if the data type of the column is LONG VARCHAR, LONG VAR GRAPHIC, BLOB, CLOB, DATALINK, XML distinct type on any of these types, or structured type. Available columns are limited to columns that are common to all existing PKs or unique constraints.

See Also

- Creating and Editing Columns
- Creating and Editing Attributes and Columns

Distributing Columns for Netezza

The Distribute tab of the Table Editor for the Netezza platform allows you to create the DISTRIBUTE ON statement that specifies the distribution key. The distribution key is used by the SPUs (Snippet Processing Units or intelligent storage nodes) to store data on the server. If you do not specify the columns of the distribution key and do not activate the random distribution option, the Netezza server will assign to the distribution key by default the first column created in the table.

- *Available Columns*: Displays all the columns available to add to the distribution key. Select the column you want to add to the distribution key and move it to the Selected Columns box. Use the left and right arrows to move columns to and from the Selected Columns box. You can select a maximum of four columns to use in the distribution key.
- *Selected Columns*: Displays the columns that make up the distribution key.
- *Up / Down* buttons: Let you reorder the columns in the distribution key. The column order can affect the access speed. The most frequently accessed columns should be at the top of the Selected Columns list.
- *Random*: If selected, the table is created using a random or round-robin distribution key.

Completing this tab with the Random option selected will produce SQL code similar to the following:

```
CREATE TABLE Entity1(

PK1 CHAR(10) NOT NULL,

A1 CHAR(10),

A2 CHAR(10),

CONSTRAINT PK1 PRIMARY KEY (PK1)

)

DISTRIBUTE ON RANDOM
```

Selecting PK1 to be the only column in the distribution key and not selecting Random will produce SQL code similar to the following:

```
CREATE TABLE Entity1(

PK1 CHAR(10) NOT NULL,

A1 CHAR(10),

A2 CHAR(10),

CONSTRAINT PK1 PRIMARY KEY (PK1)

)
```

DISTRIBUTE ON (PK1)

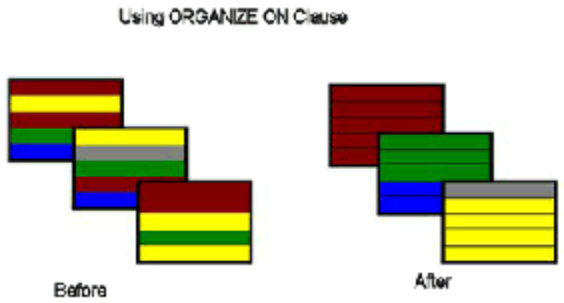
See Also

- Creating and Editing Columns
- Creating and Editing Attributes and Columns

Organizing Columns for Netezza

On the Organization tab of the Table Editor for the Netezza 6.0 (and later versions) platform, you can select up to four columns of a table on which to base the organization of the table. Tables with columns specified in the ORGANIZE ON clause are called Clustered Base Tables (CBT). Organizing the table records in this way causes Netezza to save the selected table records in the same or nearby extents. This can enhance query performance and disk I/O.

CBTs are most often used for large fact or event tables with many millions or billions of rows. Scanning records of such large tables requires full disk scans to gather the relevant records. Organizing tables to correspond with your queries can take advantage of zone maps to improve performance.



- *Available Columns*: Displays all the columns available to add to the organization key. Select the column you want to add to the organizing keys and move it to the Selected Columns box. Use the left and right arrows to move columns to and from the Selected Columns box. You can select a maximum of four columns to use in the organizing key.
- *Selected Columns*: Displays the columns that make up the organizing key.
- *Up / Down* buttons: Let you reorder the columns in the organizing key. The column order can affect the access speed. The most frequently accessed columns should be at the top of the Selected Columns list.

See Also

- Creating and Editing Columns
- Creating and Editing Attributes and Columns

Partitioning a Table

The performance of almost all database applications is I/O bound. To improve I/O throughput, you should physically separate tables that are frequently joined together. You should also separate tables from their indexes. The objective is to have the database read or write data in parallel as much as possible.

1. In the *Data Model Window*, double-click the table that you want to partition.
2. From the *Table Editor*, click the *Partitions* tab and then select the index for which you want to specify storage options.
3. Complete the *Partitions* options as required, click *Add* to launch the *Table Partition Editor*.
4. Complete the *Table Partition Editor* options as required, click *OK* to exit the editor and then click *OK* again to exit the *Table Editor*.

- The index storage options available on the Partitions tab and the Table Partition Editor depend on the database platform and version.

The following describe options that require additional explanation. For information on the storage options available for your database platform, click a link below:

Partition Options for Sybase	Table Partition Options for Oracle 8.x, 9i, 10g, 11g and 12c
Table Partition Editor for Oracle 8.x, 9i, 10g, 11g and 12c	Table Partition Options for IBM DB2 for OS/390 8.x, 9.x, and 10.x
Table Partition Editor for IBM DB2 for OS/390 8.x, 9.x, and 10.x	Table Partition Options for IBM DB2 for LUW 9.x and 10.x
Table Partition Editor for IBM DB2 for LUW 9.x (and later versions)	Table Partition Columns for IBM DB2 for LUW 5.x, 6.x, 7.x, 8.x

Table Partition Options for Sybase

- *Partition Table*: (*PARTITION num*) Select if you want to partition the table and then enter the number of partitions desired.

Table Partition Options for Oracle 8.x, 9i, 10g, 11g and 12c

- *Type*: Select the partition type.
    - *Composite*: Select to hash partition range partitions on a different key. When selected, specify the subpartitions by clicking *Subpartition Type*. More partitions increase the possibilities for parallelism and reduce contention. For example the following range partitions a table on *invoiceDate* and subpartitions these partitions by *invoiceNo*, creating 16 subpartitions.

CREATE TABLE invoices

(invoice_no NUMBER NOT NULL,

invoice_date DATE NOT NULL)

PARTITION BY RANGE (invoiceDate)

SUBPARTITION BY HASH (invoiceNo)

SUBPARTITIONS 4

(PARTITION invoices_Q1 VALUES LESS THAN (TO_DATE('01/04/2007', 'DD/MM/YYYY')),

PARTITION invoices_Q2 VALUES LESS THAN (TO_DATE('01/07/2007', 'DD/MM/YYYY')),

PARTITION invoices_Q3 VALUES LESS THAN (TO_DATE('01/09/2007', 'DD/MM/YYYY')),

PARTITION invoices_Q4 VALUES LESS THAN (TO_DATE('01/01/2008', 'DD/MM/YYYY'));

    -
        - *Range*: (*PARTITION BY RANGE (column_name)*) Select to store distinct ranges of data together, such as dates. Searches can then be limited to partitions with data that is the correct age. Also, once historical data is no longer useful, the whole partition can be removed.
        - *Hash*: (*PARTITION BY HASH (column_name) PARTITIONS count STORE IN tablespace)*) Select if there is no obvious range key or if range partitioning might cause uneven data distribution.
        - *List*: (*PARTITION BY LIST*) Select to control how rows map to partitions. Specify a list of discrete values for the partitoning key in the description of each partition.
- *Subpartition Type* button: Launches the *Subpartition Type Editor* where you can specify the columns to use in the subpartition. For a *HASH* subpartition, you can specify the number of subpartitions and where they are stored. For a *LIST* subpartition, you can specify the subpartitions to include.
- *Available Columns*: Displays the columns available to add to a partition.
- *Selected Columns*: Displays the columns that make up the partition key. This guides how data is divided along the partition boundaries. Typically, a partition key is date-based or based on a numerically ordered data grouping. Use the left and right arrows to move columns to and from this grid. Columns that are in this grid make up the partition.
- *Up / Down* buttons: Let you reorder the columns on the partition. The partition order can affect the access speed. The most frequently accessed columns should be at the top of the partition list.
- *Add*: Click to add table partitions. When clicked, launches the Table Partition Editor for Oracle 8.x, 9i, 10g, 11g and 12c.
- *Edit*: Select a partition and click to modify it. When clicked, launches the Table Partition Editor for Oracle 8.x, 9i, 10g, 11g and 12c.

Table Partition Editor for Oracle 8.x, 9i, 10g, 11g and 12c

- *Max Value*: (*VALUES LESS THAN (MAXVALUE)*) Select to create a catch-all partition for values that exceed all specified ranges. Otherwise, specify a High Value for the partition range.
- *High Value*: (*VALUES LESS THAN (high value)*) Specifies the ending value for the partition range.
- *Tablespace*: Specifies the name of the tablespace on which the table is stored.
    - *No Logging*: Select if you do not want the DLL operations to be logged in the redo file. This can reduce index creation and updates by up to 30%.
- *Pct Free*: (*PCTFREE*) Specifies the maximum percentage of space in each data block to reserve for future updates. This reserved space helps to avoid row migration and chaining caused by an update operation that extends a data row's length. Tables that will not be updated should have this value set to 0.
- *Pct Used*: (*PCTUSED*) Specifies the maximum percentage of available space in each data block before re-adding it to the list of available blocks. When deletes take place and the room available in a block falls below this value, the block is made available for new inserts to take place. Tables that will not be updated should have this value set to 99. The default value is 40% means that blocks are available for insertion when they are less than 40% full.
- *Initial Trans*: (*INITTRANS*) Specifies the number of DML transactions for which space is initially reserved in the data block header. Oracle stores control information in the data block to indicate which rows in the block contain committed and uncommitted changes. The amount of history that is retained is controlled by this parameter. If too many transactions concurrently modify the same data block in a very short period, Oracle may not have sufficient history information to determine whether a row has been updated by a "too recent" transaction. Specify a higher value for indexes that may experience many transactions updating the same blocks.
- *Max Trans*: (*MAXTRANS*) Once the space reserved by *INITRANS* is depleted, space for additional transaction entries is allocated out of any available free space. Once allocated, this space effectively becomes a permanent part of the block header. This parameter limits the number of transaction entries that can concurrently use data in a data block and therefore limits the amount of free space that can be allocated for transaction entries in a data block.
- *Edit Subpartition* button: Launches the Subpartition Editor where you can specify how to subpartition the partition and where the subpartitions should be stored.
- *Initial Extent*: (*INITEXTENT*) Specifies the initial number of data blocks that Oracle should reserve. Oracle will reserve the number of data blocks that correspond to the initial extent for that table's rows.
- *Next Extent*: (*NEXT*) Specifies the size in kilobytes of the next extent. You should monitor this figure against the largest available chunk of free space in the tablespace. If

a table cannot allocate its next extent, it will no longer be able to extend and, therefore, cannot accept additional data.

- *Pct Increase*: (*PCTINCREASE*) Specifies the percentage by which the next extent should grow over the previous extent's size. Be careful when setting Percent Increase because it magnifies how an object grows and, therefore, can materially affect available free space in a tablespace.
- *Min Extents*: (*MINEXTENTS*) Specifies the number of extents to allocate when the segment is created. Controls free space fragmentation by making sure that every used or free extent is at least as large as the value you specify.
- *Max Extents*: (*MAXEXTENTS*) Specifies the maximum number of extents that Oracle can allocate to the index. Once this limit is reached, Oracle prevents further growth of the cluster and cannot accept additional data. Carefully monitor the number extents already allocated to the index with this limit.
- *Free Lists*: (*FREELISTS*) Specifies the number of free lists to apply to the index. The default and minimum value is 1. Free lists can help manage the allocation of data blocks when concurrent processes are issued against the index. Oracle uses the free list to determine which data block to use when an *INSERT* operation occurs. Oracle allows table and indexes to be defined with multiple free lists. All tables and index free lists should be set to the high-water mark of concurrent *INSERT* or *UPDATE* activity. For example, if the index has up to 20 end users performing INSERTs at any time, then the index should have *FREELISTS=20*.Too low a value for free lists will cause poor Oracle performance. An increase in *FREELISTS* or *FREELIST_GROUPS* can alleviate segment header contention.
- *Free List Groups*: (*FREELIST GROUPS*) Applicable only if you are using Oracle with the Parallel Server option in parallel mode. Specifies the number of free list groups which allow the index to have several segment headers. This enables multiple tasks to insert into the index; thereby alleviating segment header contention. Free list groups should be set to the number of Oracle Parallel Server instances that access the index. For partitioned objects and cases of segment header contention, freelist_groups my be set for non-RAC systems.
- *Buffer Pool*: (*BUFFER_POOL*) Specifies the memory structure that is used for caching data blocks in memory, providing faster data access.
  DEFAULT: Caches data blocks in the default buffer pool.
  KEEP: Retains the object in memory to avoid I/O conflicts.
  RECYCLE: Removes data blocks from memory as soon as they are no longer in use, thereby saving cache space.

Table Partition Options for IBM DB2 for OS/390 8.x, 9.x, and 10.x

- *Available Columns*: Displays the columns available to add to a partition.
- *Selected Columns*: Displays the columns that make up the partition key. This guides how data is divided along the partition boundaries. Typically, a partition key is date-based or based on a numerically ordered data grouping. Use the left and right arrows to move columns to and from this grid. Columns that are in this grid make up the partition.

- *Up / Down* buttons: Let you reorder the columns on the partition. The partition order can affect the access speed. The most frequently accessed columns should be at the top of the partition list.
- *Add*: Click to add table partitions. When clicked, launches the Table Partition Editor for IBM DB2 for OS/390 8.x, 9.x, and 10.x.
- *Edit*: Select a partition and click to modify *Edit* to modify it. When clicked, launches the Table Partition Editor for IBM DB2 for OS/390 8.x, 9.x, and 10.x.

Table Partition Editor for IBM DB2 for OS/390 8.x, 9.x, and 10.x

- *Partitions*: Displays existing partitions. If more than one column is used to create the partition, the ending values of the partition keys (columns) are listed in the Ending At column in the same order as the columns in the partition definition. You can change the partition definition by selecting the partition and then changing the High Values in the Partition Definition area.
- *Partition Definition*: Defines the high values of each partition key. You must enter a high value for each partition key listed.
- *Inclusive*: If selected, includes the high value in the partition; otherwise the partition will end just before the high value is reached.

Table Partition Options for IBM DB2 for LUW 9.x and 10.x

- *Available Columns*: Displays the columns available to add to a partition.
- *Selected Columns*: Displays the columns that make up the partition key. This guides how data is divided along the partition boundaries. Typically, a partition key is date-based or based on a numerically ordered data grouping. Use the left and right arrows to move columns to and from this grid. Columns that are in this grid make up the partition.
- *Up / Down* buttons: Let you reorder the columns on the partition. The partition order can affect the access speed. The most frequently accessed columns should be at the top of the partition list.
- *Add*: Click to add table partitions. When clicked, launches the Table Partition Editor for IBM DB2 for LUW 9.x (and later versions).
- *Edit*: Select a partition and click to modify *Edit* to modify it. When clicked, launches the Table Partition Editor for IBM DB2 for LUW 9.x (and later versions).

Table Partition Editor for IBM DB2 for LUW 9.x (and later versions)

Lets you create a *PARTITION BY RANGE* statement similar to the following:

CREATE TABLE orders (id INT, shipdate DATE,...)
PARTITION BY RANGE(shipdate)
(
STARTING '1/1/2006' ENDING '12/31/2006'

EVERY 3 MONTHS
)

- *Partitions*: Displays existing partitions. If more than one column is used to create the partition, the ending values of the partition keys (columns) are listed in the *End Value* column in the same order as the columns in the partition definition. You can change the partition definition by selecting the partition and then changing the Start Value in the Partition Definition area.
- *Name*: Lets you define the name of the partition. This field is not used when creating several partitions automatically. Use the *Range Width* and *Duration Label* fields to create a range partitioning statement such as *EVERY 3 MONTHS*.
- *Column Name*: Displays the column names selected to be part of the partition on the Partitions tab.
- *Start Value*: Lets you specify the earliest possible date or time, or the lowest integer to include in the partition. In the partitioning statement, the start value is the *STARTING FROM* value.
  A date string can be specified using one of the following formats:
  • *yyyy - mm - dd*
  • *mm / dd / yyyy*
  • *dd. mm. yyyy*
  A timestamp string can specified as:
  • *yyyy - mm - ddhh. mm. ss. nnnnnn*
  • *yyyy - mm - dd - hh. mm. ss. nnnnnn*
  You can also choose *MINVALUE* or *MAXVALUE* from the list, the values of which depend upon the datatype.After specifying the Start Value, specify the End Value.
- *Start Value Inclusive*: Select to include the specified start value in the partition. Keywords inserted are *INCLUSIVE IN*. If not selected, the partition will start at the value just greater than the specified start value, and the keywords inserted will be *EXCLUSIVE IN*.
- *End Value*: Lets you specify the latest possible date/time or the largest integer to include in the partition. When working with a date-partitioned table, this would be the *ENDING* value. See Start Value above for acceptable formats for date and time stamp strings.
- *End Value Inclusive*: Select to include the specified end value in the partition. Keywords inserted are *INCLUSIVE IN*. If not selected, the partition will end at the date/time or integer just greater than the specified end value, and the keywords inserted will be *EXCLUSIVE IN*.
- *Range Width*: Use Range Width and Duration Label to automatically create multiple partitions. The Range Width is the date, time or integer interval used to partition the table. For example, in the *EVERY 3 MONTHS* clause, the Range Width is 3.
- *Duration Label*: From the list, select the unit of measure to use to partition the table. In the *EVERY 3 MONTHS* clause, the Duration Label is *MONTHS*. For an integer-partitioned table, select (none) as the duration label.
- *Tablespace*: Specifies the tablespace where the partition is stored.

- *LONG IN*: Lets you specify a large tablespace where long data is to be stored. Large objects are by default stored in the same tablespace as the corresponding data objects, whether the partitioned table uses only one tablespace or multiple tablespaces. The *LONG IN* clause overrides this default behavior, causing long data to be stored in the tablespace specified, which can improve database performance.

Table Partition Columns for IBM DB2 for LUW 5.x, 6.x, 7.x, 8.x

- *Available Columns*: Displays all the columns available to add to the partition. Select the column you want to add to the partition and move it to the Selected Columns box.
- *Selected Columns*: Displays the columns that make up the partition key. This guides how data is divided along the partition boundaries. Typically, a partition key is date-based or based on a numerically ordered data grouping. Use the left and right arrows to move columns to and from this grid. Columns that are in this grid make up the partition.
- *Up / Down* buttons: Let you reorder the columns on the partition. The partition order can affect the access speed. The most frequently accessed columns should be at the top of the partition list.
- *Use Hashing*: If selected, ER Studio Data Architect partitions the table according to a hash function, which can speed up table lookup or data comparison tasks.

See Also

- Defining Table Storage
- Creating and Editing Tables
- Partitioning a Table Index

Defining Index Storage

Storage parameters affect both how long it takes to access data stored in the database and how efficiently space in the database is used. For heavily accessed tables, place these tables in one location and place the table's indexes in a different location on different physical disk arrays. This will improve performance, because separate threads will be created to access the tables and indexes.

You can define how your tables are stored using the options in the Index Editor, which is accessible through the Index tab of the Table Editor.

1. In the *Data Model Window*, double-click the table for which you want to specify index storage options.
2. From the *Table Editor*, click the *Indexes* tab, select the index for which you want to specify storage options, and then click *Edit*.
3. The Index Editor appears with options that are specific to the database platform and version.
4. Complete the *Index Editor* options as required, click *OK* to exit the *Index Editor*, and then click *OK* again to exit the *Table Editor*.

For information on the storage options available for your database platform, click a link below:

GreenPlum Index Storage Options	Hitachi HiRDB Index Storage Options
IBM DB2 for OS/390 Index Storage Options	Informix Index Storage Options
Interbase Index Storage Options	Microsoft SQL Server Index Storage Options
Oracle 7.x Index Storage Options	Oracle 8.x, 9i, 10g, 11g and 12c Index Storage Options, see Properties tab
Postgre SQL Index Storage Options	Sybase SQL Anywhere Index Storage Options
Sybase System 10, ASE 10, 11 Index Storage Options	

GreenPlum Index Storage Options

- *Tablespace*: (*TABLESPACE*) Specifies the name of the tablespace where the index is stored.

Hitachi HiRDB Index Storage Options

- *Index Storage*: (*ON*) Specifies where to store the index.
- *PCTFREE*: (*PCTFREE*) Specifies the maximum percentage of space in each data block to reserve for future updates. This reserved space helps to avoid row migration and chaining caused by an update operation that extends a data row's length. Tables that will not be updated should have this value set to 0.

- *Unbalanced Split*: (*UNBALANCED SPLIT*) Specifies that in order to maintain the balance of the index tree and improve performance the index pages can be split.

IBM DB2 for OS/390 Index Storage Options

The storage options for an IBM DB2 for OS/390 appear on the Options tab of the Index Editor for that platform. For more information, see IBM DB2 for OS/390 Index Options.

Informix Index Storage Options

- *Fill Factor*: (*FILLFACTOR*) Specifies the percent or degree of index-page fullness. A low value provides room for index growth. A high value compacts the index. If an index is full (100 percent), any new inserts result in splitting nodes.

Interbase Index Storage Options

- *External File*: Specifies where to store the index.

Microsoft SQL Server Index Storage Options

- The options available depend on the version of the database.
- *Segment*: (*ON*) Specifies where the index will be stored.
- *File Group*: (*ON*) Specifies where the index will be stored.
- *Fill Factor*: (*FILLFACTOR*) Specifies the percent or degree of index-page fullness. A low value provides room for index growth. A high value compacts the index. If an index is full (100 percent), any new inserts result in splitting nodes.
- *Pad Index*: (*PAD_INDEX*) Specifies how much space to leave available on each page in the intermediate levels of the index. If *PAD_INDEX* is not used, by default each intermediate index page is given only enough empty space to hold at least one row of the maximum size the index can have. For tables with heavy index updates, this can cause the intermediate index pages to split often, causing unnecessary overhead. To prevent intermediate index page splitting, assign both the *FILLFACTOR* and the *PAD_INDEX* options. The *PAD_INDEX* option uses the *FILLFACTOR*, and applies it to the intermediate index pages. For example, if a *FILLFACTOR* of 80 is specified, then the *PAD_INDEX* will use this value, and only fill 80% of the intermediate index page space, instead of leaving enough space for just one new row.
- *Sort In Tempdb*: (*SORT_IN_TEMPDB*) Specifies to use tempdb to store intermediate sort results that are used to build the index. Although this increases the amount of temporary disk space that is used to create an index, it could reduce the time required to create or rebuild an index when tempdb is on a disparate set of disks from that of the user database.
- *Partition Keys*: Select the key to use when partitioning the index. Partitioning the index can substantially reduce query time, improving the load time and maintainability of the index.

Oracle 7.x Index Storage Options

- *Tablespace*: (*TABLESPACE*) Specifies the name of the tablespace where the index is stored.
- *Initial Transactions*: (*INITTRANS*) Specifies the number of DML transactions for which space is initially reserved in the data block header. Oracle stores control information in the data block to indicate which rows in the block contain committed and uncommitted changes. The amount of history that is retained is controlled by this parameter. If too many transactions concurrently modify the same data block in a very short period, Oracle may not have sufficient history information to determine whether a row has been updated by a "too recent" transaction. Specify a higher value for indexes that may experience many transactions updating the same blocks.
- *Max Transactions*: (*MAXTRANS*) Once the space reserved by INITRANS is depleted, space for additional transaction entries is allocated out of any available free space. Once allocated, this space effectively becomes a permanent part of the block header. This parameter limits the number of transaction entries that can concurrently use data in a data block and therefore limits the amount of free space that can be allocated for transaction entries in a data block.
- *Percent Free*: (*PCTFREE*) Specifies the maximum percentage of space in each data block to reserve for future updates. This reserved space helps to avoid row migration and chaining caused by an update operation that extends a data row's length. Tables that will not be updated should have this value set to 0.
- *Initial Extent*: (*INITEXTENT*) Specifies the initial number of data blocks that Oracle should reserve. Oracle will reserve the number of data blocks that correspond to the initial extent for that table's rows.
- *Next Extent*: (*NEXT*) Specifies the size in kilobytes of the next extent. You should monitor this figure against the largest available chunk of free space in the tablespace. If a table cannot allocate its next extent, it will no longer be able to extend and, therefore, cannot accept additional data.
- *Percent Increase*: (*PCTINCREASE*) Specifies the percentage by which the next extent should grow over the previous extent's size. Be careful when setting Percent Increase because it magnifies how an object grows and, therefore, can materially affect available free space in a tablespace.
- *Min Extents*: (*MINEXTENTS*) Specifies the number of extents to allocate when the segment is created. Controls free space fragmentation by making sure that every used or free extent is at least as large as the value you specify.
- *Max Extents*: (*STORAGE (MAXEXTENTS)*) Species the maximum number of extents that Oracle can allocate to the index. Once this limit is reached, Oracle prevents further growth of the cluster and cannot accept additional data. Carefully monitor the number extents already allocated to the index with this limit.

Postgre SQL Index Storage Options

- *Tablespace*: (*TABLESPACE*) Specifies the name of the tablespace where the index is stored.

Sybase SQL Anywhere Index Storage Options

- *DB Space*: (*IN*) Specifies where the index is stored.

Sybase System 10, ASE 10, 11 Index Storage Options

- *Segment*: (*ON*) Specifies where the index will be stored.
- *Max Rows per Page*: (*MAX_ROWS_PER_PAGE*) Specifies the maximum number of rows per page. This restricts the number of rows allowed on a page, reducing lock contention and improving concurrency for frequently accessed tables.The default value is 0, which creates clustered indexes with full data pages, non clustered indexes with full leaf pages, and leaves a comfortable amount of space within the index B-tree in both the clustered and nonclustered indexes.
  For heap tables and clustered indexes, the range for *MAX_ROWS_PER_PAGE* is 0-256. For non clustered indexes, the maximum value for *MAX_ROWS_PER_PAGE* is the number of index rows that fit on the leaf page, without exceeding 256. To determine the maximum value, subtract 32 (the size of the page header) from the page size and divide the difference by the index key size.
- *Fill Factor*: (*FILLFACTOR*) Specifies the percent or degree of index-page fullness. A low value provides room for index growth. A high value compacts the index. If an index is full (100 percent), any new inserts result in splitting nodes.

See Also

- Creating and Editing Indexes
- Defining Index Storage
- Partitioning a Table Index

Partitioning a Table Index

For database platforms that support this feature, partitioning the index can substantially reduce query time, limiting access to relevant partitions only. A partitioned index can be efficiently maintained. Partition independence enables a partition to be maintained without affecting the availability of other partitions.

You can access the *Index Partition Editor* through the *Table Editor > Index* tab, where you can choose to partition the index and add new partitions as required. The options available in the Index Partition Editor depend upon the selected database platform. For information on the options for your database, click one of the following links:

- IBM DB2 for OS/390 Index Partitions Options
- Oracle 8.x, 9i, 10g, 11g and 12c Index Partitions Options
- Index Partition Editor for Oracle 8.x, 9i, 10g, 11g and 12c Options

IBM DB2 for OS/390 Index Partitions Options

- For IBM DB2 for OS/390, the Cluster option on the Options tab must be selected in order to enable the partitioning options on the Partitions tab. Click Add on the Partitions tab to launch the Index Partition Editor.
- *High Value*: Lets you specify the maximum value of the column for this partition, entries that exceed this amount will not be included in the partition.
- *Vcat*: (*VCAT*) If selected, allocates space for the index on a data set managed using the virtual catalog (*VCAT*) whose name is specified next to this option, and the user manages the data sets required for the index. The *VCAT* method is usually used for defining system catalogs. It requires that the VSAM dataset be pre-defined before the index space is created.
- *StoGRoup*: (*STOGROUP*) If selected, stores the index on a data set managed by DB2 in the named storage group. Stogroup defined index spaces let DB2 do all the VSAM allocation work for you. If you have created a Stogroup for the data model, using the *Stogroup Editor* (see [ERStudio.htm#50614277_67903 See Creating and Editing StoGroups]), you can select that Stogroup from the list. If you select *Stogroup*, you can also define values for *PRIQTY* and *SECQTY*. To prevent wasted space for non-partitioned indexes, you should not define *PRIQTY* and *SECQTY* ; instead let DB2 manage the amount of primary and secondary space allocated for the index.
- *PriQty*: (*PRIQTY*) Specifies the minimum number of disk space in kilobytes to initially allocate for the index. The primary allocation should be large enough to handle the storage needs that you anticipate.
- *SecQty*: (*SECQTY*) Specifies the amount of disk space in kilobytes to allocate to the index when it reaches the size of the primary allocation. If the secondary allocation space is too small, the data set might have to be extended more times to satisfy those activities that need a large space. The default value of -1 indicates that DB2 should

determine the size of the secondary allocation, which is usually 10% of the primary allocation.

- *Percent Free*: (*PCTFREE*) Specifies the maximum percentage of space in each data block to reserve for future updates. This reserved space helps to avoid row migration and chaining caused by an update operation that extends a data row's length. Tables that will not be updated should have this value set to 0.
- *Free Page*: (*FREEPAGE*) Specifies how often to leave a page of free space when index entries are created. One free page is left for every number of pages specified here in the range of 0 to 255.
- *GBP Cache*: (*GPBCACHE*) In a data sharing environment, specifies what index pages are written to the group buffer pool. This option is ignored in a non-data-sharing environment unless the index is on a declared temporary table.
- *Changed*: Writes updated pages to the group buffer pool.
- *All*: Caches all pages to the group buffer pool as they are read in from DASD.
- *None*: Uses the buffer pool only for cross-invalidation.

Oracle 8.x, 9i, 10g, 11g and 12c Index Partitions Options

Options available on the Partitions tab of the Index Editor depend on the version of the database platform.

- *Type*: Select the partition type.
- *Range*: (*PARTITION BY RANGE (column_name)*) Select to store distinct ranges of data together, such as dates. Searches can then be limited to partitions with data that is the correct age. Also, once historical data is no longer useful, the whole partition can be removed.
- *Hash*: (*PARTITION BY HASH (column_name) PARTITIONS count STORE IN tablespace)*) Select if there is no obvious range key or if range partitioning might cause uneven data distribution. If selected, the *Partition Count* option is available.
- *Partition Count*: Click to launch the *Hash Partition Editor* where you can specify the number of subpartitions and where to store them.
- *Available Columns*: Displays the columns available to add to a partition.
- *Selected Columns*: Displays the columns that make up the partition key. Guides how data is divided along the partition boundaries. Typically, a partition key is date-based or based on a numerically ordered data grouping. Use the left and right arrows to move columns to and from this grid.
- *Up / Down* buttons: Let you reorder the columns on the partition. The partition order can affect the access speed. The most frequently accessed columns should be at the top of the partition list.

- *Index Scope*: Lets you specify the type of index partition to create.
- *Local*: (*LOCAL*) All index entries in the partition correspond to a one table partition (equi-partitioned). Supports partition independence, allowing for more efficient queries.
- *Global*: (*GLOBAL*) The index in the partition can correspond to many table partitions. Does not support partition independence and must be partitioned by range. If specified, you must specify the partition range values.
- *Add*: Click to launch the Index Partition Editor where you can add table partitions. For more information, see Index Partition Editor for Oracle 8.x, 9i, 10g, 11g and 12c Options.
- *Edit*: Click to launch the Index Partition Editor where you can add table partitions. For more information, see Index Partition Editor for Oracle 8.x, 9i, 10g, 11g and 12c Options.

Index Partition Editor for Oracle 8.x, 9i, 10g, 11g and 12c Options

- *Max Value*: (*VALUES LESS THAN (MAXVALUE)*) Select to create a catch-all partition for values that exceed all specified ranges. Otherwise, specify a High Value for the partition range.
- *High Value*: (*VALUES LESS THAN (high value)*) Specifies the ending value for the partition range.
- *Pct Free*: (*PCTFREE*) Specifies the maximum percentage of space in each data block to reserve for future updates. This reserved space helps to avoid row migration and chaining caused by an update operation that extends a data row's length. Tables that will not be updated should have this value set to 0.
- *Initial*: (*INITTRANS*) Specifies the number of DML transactions for which space is initially reserved in the data block header. Oracle stores control information in the data block to indicate which rows in the block contain committed and uncommitted changes. The amount of history that is retained is controlled by this parameter. If too many transactions concurrently modify the same data block in a very short period, Oracle may not have sufficient history information to determine whether a row has been updated by a "too recent" transaction. Specify a higher value for indexes that may experience many transactions updating the same blocks.
- *Max Trans*: (*MAXTRANS*) Once the space reserved by INITRANS is depleted, space for additional transaction entries is allocated out of any available free space. Once allocated, this space effectively becomes a permanent part of the block header. This parameter limits the number of transaction entries that can concurrently use data in a data block and therefore limits the amount of free space that can be allocated for transaction entries in a data block.
- *Initial Extent*: (*INITEXTENT*) Specifies the initial number of data blocks that Oracle should reserve. Oracle will reserve the number of data blocks that correspond to the initial extent for that table's rows.
- *Next Extent*: (*NEXT*) Specifies the size in kilobytes of the next extent. You should monitor this figure against the largest available chunk of free space in the tablespace. If

a table cannot allocate its next extent, it will no longer be able to extend and, therefore, cannot accept additional data.

- *Pct Increase*: (*PCTINCREASE*) Specifies the percentage by which the next extent should grow over the previous extent's size. Be careful when setting Percent Increase because it magnifies how an object grows and, therefore, can materially affect available free space in a tablespace.
- *Min Extents*: (*MINEXTENTS*) Specifies the number of extents to allocate when the segment is created. Controls free space fragmentation by making sure that every used or free extent is at least as large as the value you specify.
- *Max Extents*: (*MAXEXTENTS*) Specifies the maximum number of extents that Oracle can allocate to the index. Once this limit is reached, Oracle prevents further growth of the cluster and cannot accept additional data. Carefully monitor the number extents already allocated to the index with this limit.
- *Free Lists*: (*FREELISTS*) Specifies the number of free lists to apply to the index. The default and minimum value is 1. Free lists can help manage the allocation of data blocks when concurrent processes are issued against the index. Oracle uses the free list to determine which data block to use when an INSERT operation occurs. Oracle allows table and indexes to be defined with multiple free lists. All tables and index free lists should be set to the high-water mark of concurrent INSERT or UPDATE activity. For example, if the index has up to 20 end users performing INSERTs at any time, then the index should have FREELISTS=20.Too low a value for free lists will cause poor Oracle performance. An increase in FREELISTS or FREELIST_GROUPS can alleviate segment header contention.
- *Free List Groups*: (*FREELIST GROUPS*) Applicable only if you are using Oracle with the Parallel Server option in parallel mode. Specifies the number of free list groups which allow the index to have several segment headers. This enables multiple tasks to insert into the index; thereby alleviating segment header contention. Free list groups should be set to the number of Oracle Parallel Server instances that access the index. For partitioned objects and cases of segment header contention, freelist_groups my be set for non-RAC systems.
- *Buffer Pool*: (*BUFFER_POOL*) Specifies the memory structure that is used for caching data blocks in memory, providing faster data access.
- *DEFAULT* caches data blocks in the default buffer pool.
- *KEEP* retains the object in memory to avoid I/O conflicts.
- *RECYCLE* removes data blocks from memory as soon as they are no longer in use, thereby saving cache space.

See Also

- Partitioning a Table
- Creating and Editing Tables

Customizing Table DDL

The options selected in the Table Editor determine the clauses used in the *CREATE TABLE* and *CREATE INDEX* statements. Options on the DDL tab can give you more control over the DDL created.

1. Double-click a table to open the *Table Editor*.
2. Click the *DDL* tab and then click *Customize*.


Note: If the DDL is not displaying the options you expect to see, it may be because the relevant options on the DDL Tab Options dialog have not been selected.

The following describe options that require additional explanation:

- **Enclose names in quotes/brackets**: To avoid object name conflicts, you can enclose in brackets or quotations all names or enclose only names that conflict with SQL keywords. If the database platform supports object names enclosed in brackets or quotes, these options are available in the General Options section of the DDL tab Options and the *DDL Generation Wizard*. If you select *Enclose Names That Conflict with Keywords*, then any column names that contain SQL keywords will be automatically enclosed in brackets.
- **Use Complete Syntax for Virtual Columns**: This option is available for Oracle 11g and later. When this option is not selected, virtual columns will be added to the DDL with the following syntax:

```
column [<datatype>] AS (<column_expression>)
```

where `<datatype>` is set to the data type of the column if the Show Datatype option is on for the column and is omitted otherwise, and `<column_expression>` is the expression stored as the virtual column expression.

When this is selected, virtual columns will be added to the DDL with the following syntax:

```
column [<datatype>] GENERATED ALWAYS AS (<column_expression>) VIRTUAL
```

- **NUMBER datatype**: NUMBER or NUMBER(*) you must set the width and scale to *. A width of 0 will be converted to * when generating DDL.

See Also

- Partitioning a Table Index
- Partitioning a Table

Creating and Editing Indexes

An index is an ordered set of pointers to rows in a base table. Each index is based on the values of data in one or more table columns. An index is an object that is separate from the data in the table. When an index is created, the database builds and maintains it automatically. Indexes are used to improve performance; in most cases, access to data is faster with an index. Although an index cannot be created for a view, an index created for the table on which a view is based can improve the performance of operations on that view. Indexes are also used to ensure uniqueness; a table with a unique index cannot have rows with identical keys. If *Unique* is checked or *UNIQUE* is selected, each row must have a unique index value. If *Unique* is unchecked, or *NONUNIQUE* is selected, rows can have duplicate index values. Oracle offers a third option, *BITMAP*, type of index commonly used in data warehousing and other environments with large amounts of data and ad hoc queries, but a low level of concurrent DML transactions. DB2: Allow Reverse

Indexes improve performance by providing an efficient mechanism for locating data. Indexes work like a card catalog in a library: instead of searching every shelf for a book, you can find a reference to the book in the card catalog, which directs you to the book's specific location. Logical indexes store pointers to data so that a search of all of the underlying data is not necessary.

Indexes are one of the most important mechanisms for improving query performance. However, injudiciously using indexes can negatively affect performance. You must determine the optimal number of indexes to place on a table, the index type and their placement in order to maximize query efficiency.

Consider the following rules to optimize your indexes:

- *Index Number* - While indexes can improve read (query) performance, they degrade write (insert, update, and delete) performance. This is because the indexes themselves are modified whenever the data is modified. As a result, you must be judicious in the use of indexes. If you know that a table is subject to a high level of insert, update, and delete activity, you should limit the number of indexes placed on the table. Conversely, if a table is basically static, like most lookup tables, then a high number of indexes should not impair overall performance.
- *Index Type* - Generally, there are two types of queries: point queries, which return a narrow data set, and range queries, which return a larger data set. For those databases that support them, clustered indexes are better suited to satisfying range queries, or a set of index columns that have a relatively low cardinality. Non-clustered indexes are well suited to satisfying point queries.
- *Index Placement* - Almost every SQL database provides some physical storage mechanism for placing tables and indexes. At a minimum, you should take advantage of this database feature by separating the placement of tables from their indexes. In this manner, the database can read both the index and the table data in parallel, leading to faster performance.

In addition to indexing tables, you can also define indexes for auxiliary tables. For more information, see Creating and Editing Auxiliary Tables.

Create an Index

1. In the *Data Model Explorer*, select a physical data model.
2. In the *Data Model Window*, double-click the table you want to add an index to.
3. On the *Columns* tab of the *Table Editor*, click a column and then click *Add to Primary Key*.
   An index is automatically created and appears under the Indexes node in the Data Model Explorer.
4. In the *Table Editor*, select the *Indexes* tab.
5. An index is also created when you create an identifying relationship that propagates the primary key of the parent table to the child table as a foreign key.

Edit an Index

1. In the *Data Model Explorer*, expand the Index node of a physical data model, and then double-click the index you want to change.
2. You can also edit an index on the Index tab of the Table Editor for the table to which the index belongs.
3. Complete the *Index Editor* as required and then click *OK*.
4. If you change the index options and can see no reflection of the change in the DDL, click *Customize* on the *DDL* tab and ensure the options required to display your changes are selected. For example, If you create a non-unique index for a MySQL table and turn on full text, you must customize the DDL by selecting Generate Defined Nonunique Indexes for the index to show up.

Index Editor Options

The following help you complete the *Index Editor* options:

- The options and tabs available depend on the database platform.
- *Set as PK Index*: Read-only. If selected, indicates that the index selected is the primary key.
- *Constraint*: Enter a constraint to specify that every value in the named column must be unique. Unique constraints are similar to Primary keys. They require unique values throughout the named column or columns in the table.
- *Physical Only*: Select to specify that the index be ignored when comparing the physical model to a logical model.
- *Disable*: Adds CONSTRAINT... DISABLE to the syntax of the table that contains the PK. Disabling a PK constraint requires that you also disable the FK index of the child table. When you click DISABLE, ER Studio Data Architect automatically presents you with an option to disable all the FK constraints. Disabling a constraint can improve performance

when performing large data loads, making mass changes to tables, or importing or tables one at a time.

- You cannot disable the primary key of an index-organised table, hence the Disable option described above will not be available for that index type.
- *NoValidate*: This option is available for the Oracle 9i, 10g, and 11g platforms. Specify NoValidate if you do not want the constraint to be applied to the data in the table.

If generation of the FK or PK constraint is enabled, NOVALIDATE will be generated if the option is set for the constraint. For example, if the PK is not disabled, but NoValidate is set, the DDL would be

CREATE TABLE Entity1

(

PKCOL CHAR(10) NOT NULL,

CONSTRAINT PK1 PRIMARY KEY (PKCOL) NOVALIDATE

);

The same syntax will be used for FKs when created inside the CREATE TABLE or by ALTER TABLE.

ALTER TABLE ENTITY2 ADD CONSTRAINT BLIVOT

FOREIGN KEY (PK1)

REFERENCES ENTITY1(PK1) NOVALIDATE

If Disable is set but NoValidate is not set, the DDL for an FK created by ALTER TABLE would be the following, since NOVALIDATE is the default option. The same syntax is used for PKs.

ALTER TABLE ENTITY2 ADD CONSTRAINT BLIVOT

FOREIGN KEY (PK1)

REFERENCES ENTITY1(PK1) DISABLE VALIDATE

For information on the various tabs available, click a link below:

- Columns tab
- Properties tab
- Options tab
- Storage tab
- Partitions tab
- Compare Options tab
- Attachment Bindings tab

Columns tab

Select the columns or keys, the index should be based on. The efficiency of the index depends upon the keys chosen and their search order.

Properties tab

This tab is available for Oracle 8.x, 9i, 10g, 11g and 12c.

- *Index Type*: Select the appropriate index type; it can have a big impact on performance.
- *Bitmap*: Select for an index that will not be frequently updated by concurrent applications, although be aware that when applied to the index of a unique column (primary key candidate) you need to consider the space required, which depends on the cardinality of the column and the data distribution. A bitmap for each key value is used instead of a list of row IDs.
- *B-tree*: Select when indexing a primary key, because Oracle does not support primary key indexes, or select B-tree when space is a consideration. B-tree indexes are balanced to equalize access times to any row.
- *Unique*: (*UNIQUE*) If selected, specifies that the index key contains no duplicate values and therefore every row in the table is in some way unique. If not selected, a non-unique index is created that lets table rows have duplicate values in the columns that define the index. For a unique index, there is one rowid per data value. If selected, Oracle will generate a separate unique index before creating the PK constraint.
- *No Sort*: (*NOSORT*) If selected, specifies to not sort the data when the index is rebuilt.
- *Reverse Byte Order*: If selected, reverses the bytes of each column indexed (except the rowid) while keeping the column order. By reversing the keys of the index, the insertions will be distributed across all leaf keys in the index.
- *Enable Compression*: (*COMPRESS*) Select to enable index compression. Compression can repress duplication of keys in non-unique indexes. For concatenated indexes (indexes with multiple columns), the compress option can reduce the size of the index by more than half. For concatenated indexes (indexes with multiple columns), compression can reduce the size of the index by more than half. The compress option allows you to specify the prefix length for multiple column indexes.
- *Prefix Length*: (*COMPRESSprefix*) This is the prefix length for multiple column indexes and is available when Enable Compression is selected.
- *Tablespace*: Displays the tablespace on which the table index is stored.
- *No Logging*: Select, if you do not want the DLL operations to be logged in the redo file. This can reduce index creation and updates by up to 30%.
- *Pct Free*: (*PCTFREE*) Specifies the maximum percentage of space in each data block to reserve for future updates. This reserved space helps to avoid row migration and chaining caused by an update operation that extends a data row's length. Tables that will not be updated should have this value set to 0.

- *Initial Trans*: (*INITTRANS*) Specifies the number of DML transactions for which space is initially reserved in the data block header. Oracle stores control information in the data block to indicate which rows in the block contain committed and uncommitted changes. The amount of history that is retained is controlled by this parameter. If too many transactions concurrently modify the same data block in a very short period, Oracle may not have sufficient history information to determine whether a row has been updated by a *too recent* transaction. Initial Transactions limit the minimum number of concurrent transactions that can update a data block to avoid the overhead of allocating a transaction entry dynamically. Specify a higher value for indexes that may experience many transactions updating the same blocks.
- *Max Trans*: (*MAXTRANS*) Specifies the maximum number of concurrent transactions that can update a data block to avoid performance problems. Once the space reserved by INITRANS is depleted, space for additional transaction entries is allocated out of any available free space. Once allocated, this space effectively becomes a permanent part of the block header. This parameter limits the number of transaction entries that can concurrently use data in a data block and therefore limits the amount of free space that can be allocated for transaction entries in a data block.
- *Parallel*: Selects Oracle's parallel query option. You can achieve substantial performance gains by using Oracle's parallel query option.
- *Degrees*: Specifies the number of query server processes that should be used in the operation.
- *Instances*: Specifies how you want the parallel query partitioned between the parallel servers.
- *Initial Extent*: (*INITEXTENT*) Specifies the initial number of data blocks that Oracle should reserve. Oracle will reserve the number of data blocks that correspond to the initial extent for that table's rows.
- *Next Extent*: (*NEXT*) Specifies the size in kilobytes of the next extent. You should monitor this figure against the largest available chunk of free space in the tablespace. If a table cannot allocate its next extent, it will no longer be able to extend and, therefore, cannot accept additional data.
- *Pct Increase*: (*PCTINCREASE*) Specifies the percentage by which the next extent should grow over the previous extent's size. Be careful when setting Percent Increase because it magnifies how an object grows and, therefore, can materially affect available free space in a tablespace.
- *Min Extents*: (*MINEXTENTS*) Specifies the number of extents to allocate when the segment is created. Controls free space fragmentation by making sure that every used or free extent is at least as large as the value you specify.
- *Max Extents*: (*MAXEXTENTS*) Specifies the maximum number of extents that Oracle can allocate to the index. Once this limit is reached, Oracle prevents further growth of the cluster and cannot accept additional data. Carefully monitor the number extents already allocated to the index with this limit.

- *Free Lists*: (*FREELISTS*) Specifies the number of free lists to apply to the table. The default and minimum value is 1. Free lists can help manage the allocation of data blocks when concurrent processes are issued against the index. Oracle uses the free list to determine which data block to use when an *INSERT* operation occurs. Oracle allows table and indexes to be defined with multiple free lists. All tables and index free lists should be set to the high-water mark of concurrent *INSERT* or *UPDATE* activity. For example, if the index has up to 20 end users performing *INSERT* s at any time, then the index should have *FREELISTS=20*.Too low a value for free lists will cause poor Oracle performance. An increase in *FREELISTS* or *FREELIST GROUPS* can alleviate segment header contention.
- *Free List Groups*: (*FREELIST GROUPS*) Applicable only if you are using Oracle with the Parallel Server option in parallel mode. Specifies the number of free list groups which allow the table to have several segment headers. This enables multiple tasks to insert into the index; thereby alleviating segment header contention. Free list groups should be set to the number of Oracle Parallel Server instances that access the index. For partitioned objects and cases of segment header contention, free list groups my be set for non-RAC systems.
- *Buffer Pools*: (*BUFFER_POOL*) Specifies the memory structure or buffer pool that is used for caching data blocks in memory, providing faster data access.
- *DEFAULT:* Caches data blocks in the default buffer pool.
- *KEEP*: Retains the object in memory to avoid I/O conflicts.
- *RECYCLE*: Moves data blocks from memory as soon as they are no longer in use, thereby saving cache space.

Options tab

Options available depend on the database platform selected. For information on the index options available for your database platform, click a link below:

- Hitachi HiRDB Index Options
- IBM DB2 for AS/400, Common Server, and UDB Versions 5 through 9 Index Options
- IBM DB2 for OS/390 Index Options
- Informix Index Options
- Interbase Index Options
- Microsoft Access and SQL Server Index Options
- MySQL Index Options
- Teradata Index Options
- Oracle 7 Index Options
- Postgre SQL Index Options

- Sybase Index Options

Hitachi HiRDB Index Options

- *Unique*: (*UNIQUE*) If selected, specifies that the index key contains no duplicate values and therefore every row in the table is in some way unique. If not selected, a non-unique index is created that lets table rows have duplicate values in the columns that define the index.

IBM DB2 for AS/400, Common Server, and UDB Versions 5 through 9 Index Options

- *Unique*: (*UNIQUE*) If selected, specifies that the index key contains no duplicate values and therefore every row in the table is in some way unique. If not selected, a non-unique index is created that lets table rows have duplicate values in the columns that define the index.
- *Where Not Null*: (*WHERE_NOT_NULL*) If selected, specifies that multiple nulls can exist in a unique index. Useful when an index contains at least one nillable column, but all non-null entries must be unique.
- *Cluster*: If selected, specifies to create a clustered index. A clustered index specifies that the rows of the table should be inserted sequentially on data pages, which generally requires less disk I/O to retrieve the data. Generally, the clustered index should be on a column that monotonically increases, such as an identity column, or some other column where the value is increasing, and is unique. This is especially true if the table is subject to many *INSERTS*, *UPDATES*, and *DELETES*. Clustered indexes can greatly increase access speed, but usually only where the data is accessed sequentially in the same or reverse order of the clustered index, or when a range of items are selected.
- *Percent Free*: (*PCTFREE*) Specifies the maximum percentage of space in each data block to reserve for future updates. This reserved space helps to avoid row migration and chaining caused by an update operation that extends a data row's length. Tables that will not be updated should have this value set to 0.
- *Min Percent Used*: (*MINPCTUSED*) Specifies the minimum amount of used space on an index leaf page. If used, online index defragmentation is enabled for this index.
- *Allow Reverse Scans*: Specifies that the index be created in a way so that scans can also be performed in the direction opposite to that with which they were defined.
- *Include*: Specifies additional columns to be appended to the set of index key columns. The included columns may improve the performance of some queries through index-only access.

IBM DB2 for OS/390 Index Options

- *Unique*: (*UNIQUE*) If selected, specifies that the index key contains no duplicate values and therefore every row in the table is in some way unique. If not selected, a non-unique index is created that lets table rows have duplicate values in the columns that define the index.

- *Cluster*: (*CLUSTER*) Select to create a clustered index. A clustered index specifies that the rows of the table should be inserted sequentially on data pages, which generally requires less disk I/O to retrieve the data. Generally, the clustered index should be on a column that monotonically increases--such as an identity column, or some other column where the value is increasing--and is unique. This is especially true if the table is subject to many *INSERTS*, *UPDATES*, and *DELETES*. Clustered indexes can greatly increase access speed, but usually only where the data is accessed sequentially in the same or reverse order of the clustered index, or when a range of items are selected.
- For IBM DB2 for OS/390 5.x, the Cluster option must be selected to enable the index partitioning options on the Partitions tab of the Index Editor. For more information, see Partitioning a Table Index.
- *Where Not Null*: (*WHERE NOT NULL*) Specifies that multiple nulls can exist in a unique index. Useful when an index contains at least one nillable column, but all non-null entries must be unique.
- *Concurrent Copy*: (*COPY*) If selected, concurrent copies of the index are allowed, which can increase index availability by supporting index recovery.
- *Do Not Close Dataset*: (*CLOSE*) If selected, the index will not be closed when not in use. If not selected, the dataset can be closed if it is not in use and the maximum number of open datasets is reached or exceeded.
- *Defer Index Creation*: (*DEFER*) Specifies that index modifications are to be deferred during INSERT, UPDATE, and *DELETE* operations. This allows the data updates to complete quicker, but queries may not return the most up-to-date information until the index is synchronized. This option is not supported for an index on a declared temporary or auxiliary table.
- *Padded*: (*PADDED*) If selected, the index entry is padded to the full length of its datatype which makes scanning the index efficient if most of the index entries are close to or at the maximum length. If not selected, the index supports true variable length key which allows for shorter indexes and more index entries on each page. It also allows index-only access. If the index is not padded, data can be retrieved directly from the index without having to look up the data length in the data page. Not padding the index can be more efficient if you will save a lot of disk space by not padding them.
- *Partitioned*: Select if you want to partition the index. Then the options in the Index Partitions Editor accessible from the Index Editor > Partitions tab become available. For more information, see Partitioning a Table Index.
- *Using Claus*: (*USING*) Specifies what kind of index space is created for the index and who manages the index. If you do not specify a *USING* claus, DB2 manages the index space on volumes listed in the default storage group of the table's database, which must exist. DB2 uses the defaults values *PRIQTY*, *SECQTY*, and *Erase Data*.
- *VCAT*: (*VCAT*) If selected, allocates space for the index on a data set managed using the virtual catalog (VCAT) whose name is specified next to this option, and the user manages the data sets required for the index. The VCAT method is usually used for defining system catalogs. It requires that the VSAM dataset be pre-defined before the index space is created.

- *STOGROUP*: (*STOGROUP*) If selected, stores the index on a data set managed by DB2 in the named storage group. Stogroup defined index spaces let DB2 do all the VSAM allocation work for you. If you have created a Stogroup for the data model, using the Stogroup Editor (see [ERStudio.htm#50614277_67903 See Creating and Editing StoGroups]), you can select that Stogroup from the list. If you select Stogroup, you can also define values for *PRIQTY* and *SECQTY*. To prevent wasted space for non-partitioned indexes, you should not define *PRIQTY* and *SECQTY* ; instead let DB2 manage the amount of primary and secondary space allocated for the index.
- *PRIQTY*:(*PRIQTY*) Specifies the minimum number of disk space in kilobytes to initially allocate for the index. The primary allocation should be large enough to handle the storage needs that you anticipate.
- *SECQTY*: (*SECQTY*) Specifies the amount of disk space in kilobytes to allocate to the index when it reaches the size of the primary allocation. If the secondary allocation space is too small, the data set might have to be extended more times to satisfy those activities that need a large space. The default value of -1 indicates that DB2 should determine the size of the secondary allocation, which is usually 10% of the primary allocation.
- *Erase Data*: (*ERASE*) If selected, specifies to erase the index data set when the index is deleted.
- *None*: If selected, no USING claus is included in the *CREATE INDEX* statement.
- *Buffer Pool*: (*BUFFERPOOL*) Identifies the buffer pool to be used for the index. The buffer pool used determines how the data blocks are cached in memory, in order to facilitate faster data access. *BP0* is the default buffer pool for sorting.
- *Percent Free*: (*PCTFREE*) Specifies the maximum percentage of space in each data block to reserve for future updates. This reserved space helps to avoid row migration and chaining caused by an update operation that extends a data row's length. Tables that will not be updated should have this value set to 0.
- *Free Page*: (*FREEPAGE*) Specifies how often to leave a page of free space when index entries are created. One free page is left for every number of pages specified here in the range of 0 to 255.
- *Piece Size*: (*PIECESIZE*) Specifies the maximum addressable space of each data set for a secondary index. Specify the size followed by K (kilobyte), M (Megabyte), of G (Gigabyte). If you do not specify the unit of measure, kilobytes is assumed. Sizes specified in gigabytes are converted to megabytes in the DDL.
- *GBP Cache*: (*GPBCACHE*) In a data sharing environment, specifies what index pages are written to the group buffer pool. This option is ignored in a non-data-sharing environment unless the index is on a declared temporary table. Specify Changed to write updated pages to the group buffer pool, All to cache all pages to the group buffer pool as they are read in from DASD or None to indicate that the buffer pool is only to be used for cross-invalidation.

Informix Index Options

- *Unique*: (*UNIQUE*) If selected, specifies that the index key contains no duplicate values and therefore every row in the table is in some way unique. If not selected, a non-unique index is created that lets table rows have duplicate values in the columns that define the index.
- *Cluster*: (*CLUSTER*) Select to create a clustered index. A clustered index specifies that the rows of the table should be inserted sequentially on data pages, which generally requires less disk I/O to retrieve the data. Generally, the clustered index should be on a column that monotonically increases--such as an identity column, or some other column where the value is increasing--and is unique. This is especially true if the table is subject to many *INSERTS*, *UPDATES*, and *DELETES*. Clustered indexes can greatly increase access speed, but usually only where the data is accessed sequentially in the same or reverse order of the clustered index, or when a range of items are selected.

Interbase Index Options

- *Unique*: (*UNIQUE*) If selected, specifies that the index key contains no duplicate values and therefore every row in the table is in some way unique. If not selected, a non-unique index is created that lets table rows have duplicate values in the columns that define the index.
- *Sort Order*: Specify the sort order of the index, Ascending (*ASC*) or Descending (*DESC*).

Microsoft Access and SQL Server Index Options

The options available depend on the database platform and version.

- *Unique*: (*UNIQUE*) If selected, specifies that the index key contains no duplicate values and therefore every row in the table is in some way unique. If not selected, a non-unique index is created that lets table rows have duplicate values in the columns that define the index.
- *Clustered*: (*CLUSTERED* or *NONCLUSTERED*) Select to create a clustered index. A clustered index specifies that the rows of the table should be inserted sequentially on data pages, which generally requires less disk I/O to retrieve the data. Generally, the clustered index should be on a column that monotonically increases, such as an identity column, or some other column where the value is increasing, and is unique. This is especially true if the table is subject to many *INSERTS*, *UPDATES*, and *DELETES*. Clustered indexes can greatly increase access speed, but usually only where the data is accessed sequentially in the same or reverse order of the clustered index, or when a range of items are selected. If the index is not unique and the *Clustered* option is selected, the *Non-Unique Clustered Options* become available.
- *Ignore Dup Keys*: (*IGNORE_DUP_KEY*) Specifies that if you attempt to create duplicate keys by adding or updating data that affects multiple rows (with the *INSERT* or *UPDATE* statement), the row that causes the duplicates is not added or, in the case of an update, is discarded.

- *Sorted Data*: (*SORTED_DATA*) Specifies to sort the data when the index is rebuilt.
- *Non-Unique Clustered Options*: Specifies the action to take when a duplicate row is added.

MySQL Index Options

- *Unique*: (*UNIQUE*) If selected, specifies that the index key contains no duplicate values and therefore every row in the table is in some way unique. If not selected, a non-unique index is created that lets table rows have duplicate values in the columns that define the index.
- *Fulltext*: Lets you set the index as a full-text index. Full-text indexes in MySQL are an index of type *FULLTEXT*. *FULLTEXT* indexes are used with MyISAM tables only and can be created from *CHAR*, *VARCHAR*, or *TEXT* columns at *CREATE TABLE* time or added later with *ALTER TABLE* or *CREATE INDEX*.

Teradata Index Options

- *Unique*: (*UNIQUE*) If selected, specifies that the index key contains no duplicate values and therefore every row in the table is in some way unique. If not selected, a non-unique index is created that lets table rows have duplicate values in the columns that define the index.
- *Primary*: (*PRIMARY*) If selected, specifies that this index is the primary index. If you do not define a primary index, by default the first column in the table becomes the primary index in the table.
- *Partitioning Expression*: (*PARTITION BYexpression*) Available if you selected Primary. Enter an expression to partition the data by a particular attribute, such as date by week, for some time. Partitioning the data enables you to more efficiently retrieve summarized data for a particular week because you only have to read the data for that week. Good candidates for a Partitioned Primary Index are tables where there is a high volume of daily inserts, so there is a bias toward partitioning on the data date. Partitioning an index by product_code or agent_id can also improve some queries. If you need more space than available in the text box to enter your expression, click More to launch the Partition Expression Editor where you can input a multi-line expression. The following is an example of a partitioning expression:
  PARTITION BY RANGE_N(CAPTURE_DATE BETWEEN DATE '2007-11-30' AND DATE '2008-2-30' EACH INTERVAL '1' MONTH).
- *Collect Statistics*: (*COLLECT STATISTICS)* Collects demographic data for one or more columns of a table, computes a statistical profile of the collected data, and stores the profile in the data dictionary. Collecting full statistics involves scanning the base table, and sorting to determine the number of occurrences for each unique value. The Teradata Optimizer uses this profiling data when it generates its table access and join plans. Full statistics is generally recommended for relevant columns and indexes on tables with less than 100 rows per Access Module Processor (AMP).

This is an example of the SQL generated when *Collect Statistics* has been selected in the *Index Editor* for an index in the table.

COLLECT STATISTICS ON table_1 INDEX unique_1;

- *Using Sample* Collecting sampled full-table statistics is generally faster than collecting full table statistics, but may not produce the best table access and join plans. When you specify this option, Teradata will determine the appropriate percentage to the data to scan. The best choices for USING SAMPLE are columns or indexes that are unique or 95% unique.

This is an example of the SQL generated when *Collect Statistics* and *Using Sample* have been selected in the *Index Editor* for an index in the table.

COLLECT STATISTICS USING SAMPLE ON table_1 INDEX unique_1;

Oracle 7 Index Options

- *Unique*: (*UNIQUE*) If selected, specifies that the index key contains no duplicate values and therefore every row in the table is in some way unique. If not selected, a non-unique index is created that lets table rows have duplicate values in the columns that define the index.
- *No Sort*: (*NOSORT*) If selected, specifies to not sort the data when the index is rebuilt.

Postgre SQL Index Options

- *Unique*: (*UNIQUE*) If selected, specifies that the index key contains no duplicate values and therefore every row in the table is in some way unique. If not selected, a non-unique index is created that lets table rows have duplicate values in the columns that define the index.
- *Clustered*: (*CLUSTER*) Select to create a clustered index. A clustered index specifies that the rows of the table should be inserted sequentially on data pages, which generally requires less disk I/O to retrieve the data. Generally, the clustered index should be on a column that monotonically increases, such as an identity column, or some other column where the value is increasing, and is unique. This is especially true if the table is subject to many *INSERTS*, *UPDATES*, and *DELETES*. Clustered indexes can greatly increase access speed, but usually only where the data is accessed sequentially in the same or reverse order of the clustered index, or when a range of items are selected.

Sybase Index Options

The options available depend upon the database platform version.

- *Unique*: (*UNIQUE*) If selected, specifies that the index key contains no duplicate values and therefore every row in the table is in some way unique. If not selected, a non-unique index is created where table rows can have duplicate values in the columns that define the index.
- *Type*: Sybase IQ uses *HG* or *LF* indexes to generate better or faster query plans to execute the query. The *HG* or *LF* index may also be appropriate for a *DATE*, *TIME*, *DATETIME*, or *TIMESTAMP* column, if you are evaluating equality predicates against the column. An LF index is also recommended, if you frequently use the column in the

GROUP BY clause and there are less than 1000 distinct values, for example, less than three years of dates.

- *HG*: Use an *HG* (High Group) index when the column will be used in a join predicate or the column has more than 1,000 unique values. The High_Group index is commonly used for join columns with integer data types because it handles *GROUP BY* efficiently. Each foreign key columns requires its own HG index; however, if a join index exists, the same column cannot have both an explicitly created HG index and a foreign key constraint.
- *LF*: *LF* is the fastest index in Sybase IQ. Use an *LF* (Low Fast) index when a column has less than 1,000 unique values or a column has less than 1,000 unique values and is used in a join predicate. This index is ideal for columns that have a few unique values (under 1,000) such as sex, Yes/No, True/False, number of dependents, and wage class.Never use an LF index for a column with 10,000 or more unique values. If the table has less than 25,000 rows, use an HG index, which requires fewer disk I/O operations for the same operation.
- *Clustered*: (*CLUSTER*) Select to create a clustered index. A clustered index specifies that the rows of the table should be inserted sequentially on data pages, which generally requires less disk I/O to retrieve the data. Generally, the clustered index should be on a column that monotonically increases, such as an identity column, or some other column where the value is increasing, and is unique. This is especially true if the table is subject to many *INSERTS*, *UPDATES*, and *DELETES*. Clustered indexes can greatly increase access speed, but usually only where the data is accessed sequentially in the same or reverse order of the clustered index, or when a range of items are selected.
- *Ignore Dup Keys*: (*IGNORE_DUP_KEY*) Specifies that if you attempt to create duplicate keys by adding or updating data that affects multiple rows (with the *INSERT* or *UPDATE* statement), the row that causes the duplicates is not added or, in the case of an update, is discarded.
- *Sorted* or *Sorted Data*: (*SORTED_DATA*) Specifies to sort the data when the index is rebuilt.
- *Non-Unique Clustered Options*: Choose how you want to handle duplicate rows.
- *Ignore Dup Rows*: (*IGNORE_DUP_ROW*) Eliminates duplicate rows from a batch of data and cancels any insert or update that would create a duplicate row, but does not roll back the entire transaction.
- *Allow Dup Rows*: (*ALLOW_DUP_ROW*) Allows you to create a new, non-unique clustered index on a table that includes duplicate rows. If a table has a non-unique clustered index that was created without the allow_dup_row option, you cannot create new duplicate rows using the insert or update command.
- *None*: If neither Ignore Dup Rows or Allow Dup Rows has been selected, when you attempt to create an index on a table that has duplicate rows the creation fails, and the insertion fails If you try to insert a duplicate row to an indexed table.

Storage tab

Lets you select various storage options depending upon the selected database platform. For more information, see Defining Index Storage.

Partitions tab

Lets you select various storage options depending upon the selected database platform. For more information, see Partitioning a Table Index.

Compare Options tab

Select the differences you want the Compare and Merge Utility to ignore.

Attachment Bindings tab

Bind an external piece of information, or attachment to the index. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary. For more information, see Attaching External Documents to the Data Model.

See Also

- Creating and Editing Database Dependent Objects
- Developing the Physical Model
- Creating and Editing Relationships

Creating and Editing Database Dependent Objects

The availability of many physical data model objects depends on the database platform selected. For more information, see the following topics:

Creating and Editing Aliases	Creating and Editing Object Types	Creating and Editing Auxiliary Tables
Creating and Editing Packages	Creating and Editing Buffer Pools	Creating and Editing Rollback Segments
Creating and Editing DataBases	Creating and Editing Sequences	Creating and Editing Database Schemas
Creating and Editing Generators	Creating and Editing SQL Procedures	Creating and Editing Functions
Creating and Editing StoGroups	Creating and Editing Materialized Query Tables	Creating and Editing Synonyms
Creating and Editing Materialized Views	Creating and Editing Tablespaces	Creating and Editing Node Groups
Creating and Editing Triggers		

See Also

- Creating and Editing Relationships
- Creating and Editing Columns

Creating and Editing Aliases

An alias is an alternate name for an alias, synonym, table, view or for another undetermined object type. (You can only create an Alias for a synonym on the IBM DB2 for OS/390 or z/OS platforms.) An alias can provide a more meaningful object name for the target audience. An alias can also act as a low-overhead view. Table aliases can refer to each other in a process called chaining. Aliases let you assign the permissions of a database user to an object without creating a separate user entity or masking a user's identity.

The following database platforms support Aliases:

- IBM DB2 for LUW 5.x, 6.x, 7.x, 8.x, 9.x, 10.x
- IBM DB2 for OS/390 5.x and 6.x
- IBM DB2 for z/OS 7.x, 8.x, 9.x, and 10.x
- The DB2 Alias Wizard and DB2 Alias Editor share the same options, except for Attachment Bindings options which are present only in the editor.

Create an Alias

1. In the *Data Model Explorer*, expand the Physical *Main Model*, right-click the *Aliases* node, and then select *New Alias*.
2. *Complete theDB2 Alias Wizardand then clickFinishto create the alias.*
3. Once you have created the alias, you can edit it by right-clicking the alias you want to change, and then selecting Edit Alias.

The following describe *DB2 Alias Wizard* options that require additional explanation:

Name page/tab Reference page/tab

Definition page/tab DDL page/tab

Attachment Bindings tab

Name page/tab

Enter the name of alias and its owner.

- The alias name is the fully-qualified name of the alias and must not be the same as any table, view, alias, or synonym on the current server.
- The owner name is the user ID for the owner of the alias. This name is the authorization ID that qualifies the alias.

Reference page/tab

Select the type of object you want to reference. If there are any objects in the model that match your selection, you can click the identifier list to choose the appropriate identifier.

Definition page/tab

Enter or edit a definition for the alias. If the target database supports it, ER Studio Data Architect adds this definition as a comment when generating SQL code. The maximum length of comments in the DDL is 254 characters.

DDL page/tab

Displays the *CREATE ALIAS* statement needed to build the alias. ER Studio Data Architect uses the platform-specific parser of the model's selected database platform to generate the alias.

Attachment Bindings tab

Bind an external piece of information, or attachment to the alias. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary and must be applied to the alias before they will display on this tab. For more information, see Attaching External Documents to the Data Model.

See Also

- Creating and Editing Relationships
- Creating and Editing Columns
- Attaching External Documents to the Data Model

Creating and Editing Auxiliary Tables

Auxiliary Tables are created to store the data of large object (LOB) Columns. LOB Columns are columns in a regular table, called the Base Table, whose data type is BLOB or CLOB and whose data is large objects. Since the data is very large, it is not stored with the other data in the table. Even though the base table holds the LOB Column logically, as a VARCHAR(4), the LOB Column's data is physically stored in the auxiliary table. Each LOB Column has at least one auxiliary table associated with it. Either one auxiliary table for each LOB Column of a non-partitioned table, or one auxiliary table for each LOB Column for each partition, if the LOB Column is defined in a partitioned base table. For example, if you have two LOB Columns in a Base Table with four partitions, you must also have a total of eight auxiliary tables.


Note: You must store Auxiliary tables each in their own auxiliary tablespaces, and you must create a unique index on them.

The following database platforms support auxiliary tables:

- IBM DB2 for OS/390 5.x and 6.x
- IBM DB2 for z/OS 7.x, 8.x, 9.x, and 10.x


Note: The Auxiliary Table Wizard and Auxiliary Table Editor share the same options, except for Attachment Bindings options which are present only in the editor.

Add an Auxiliary Table

1. In the *Data Model Explorer*, expand the Physical *Main Model*, right-click the *Auxiliary Tables* node, and then select *New Auxiliary Table*.
2. *Complete theAuxiliary Tables Wizardand then clickFinishto create the* auxiliary table.

TIP: Once you have created the auxiliary table, you can edit it by right-clicking the auxiliary table you want to change, and then selecting Edit Auxiliary Table.

The following helps you complete the *Auxiliary Tables Wizard* options:

Name page/tab	Base Table page/tab
Storage page/tab	Index Options page/tab
Definition page/tab	DDL page/tab
Attachment Bindings tab	DDL page/tab

Name page/tab

Enter the name of auxiliary table and its owner.

- The table name is the fully-qualified name of the auxiliary table and must not be the same as any table, view, alias, or synonym on the current server.

- The owner name is the name of the owner of the base table whose column you want to store in the auxiliary table. This name is the authorization ID that qualifies the table name. If not specified, the DBS subsystem user identifier for the connection used to create the database is used.

Base Table page/tab

- The base table identifier is the name of the base table whose column you want to store in the auxiliary table. The auxiliary table is created to hold the data of a *LOB* column. The *LOB* column logically belongs to a regular table, called the base table. However, because its data is too large to physically stored in the base table, the *LOB* column data is physically stored in the auxiliary table. The base table identifier is also made up of owner name, dot, table name. The drop-down list displays the name of all tables in the model that have tables defined with *BLOB* or *CLOB* columns. The base table identifier is optional ER Studio Data Architect, but required for a valid *CREATE AUX TABLE SQL* statement.
- You must have one auxiliary table per *LOB* column for a non-partitioned base table. However, if a *LOB* Column is defined in a partitioned base table, then each *LOB* column must have a separate auxiliary table for each partition in the base table. The drop-down list displays the name of all columns in the selected base table that are defined as *BLOB* or *CLOB* datatypes. The *LOB* column name is optional ER Studio Data Architect, but required for a valid *CREATE AUX TABLE SQL* statement.
- When creating an auxiliary table for a partitioned table, you can specify the number of the partition whose data you want to store. This is only valid for LOB Columns defined in base tables stored in partitioned tablespaces.

Storage page/tab

- If you have any defined databases in the model, you can click the list and select the database where the auxiliary table will be created. For more information, see Creating and Editing DataBases.
- Each auxiliary table must be defined in its own tablespace; i.e. you must have exactly one tablespace per auxiliary table. The tablespace name can be qualified with the database name in which the tablespace resides. Together the database.tablespace make up the tablespace identifier. The tablespace name is optional ER Studio Data Architect, but required for a valid *CREATE AUX TABLE SQL* statement. For more information, see Creating and Editing Tablespaces.
- Each auxiliary table requires a Unique Index.

Index Options page/tab

- *Concurrent Copy*: (*COPY*) If selected, concurrent copies of the index are allowed, which can increase index availability by supporting index recovery.
- *Do Not Close Dataset*: (*CLOSE*) If selected, the index will not be closed when not in use. If not selected, the dataset can be closed if it is not in use and the maximum number of open datasets is reached or exceeded.

- *None*: If selected, no *USING* clause is included in the *CREATE INDEX* statement.
- *VCAT*: (*VCAT*) If selected, allocates space for the index on a data set managed using the virtual catalog (*VCAT*) whose name is specified next to this option, and the user manages the data sets required for the index. The *VCAT* method is usually used for defining system catalogs. It requires that the VSAM dataset be pre-defined before the index space is created.
- *STOGROUP*: (*STOGROUP*) If selected, stores the index on a data set managed by DB2 in the named storage group. Stogroup defined index spaces let DB2 do all the VSAM allocation work for you. If you have created a Stogroup for the data model, using the *Stogroup Editor* (see Creating and Editing StoGroups), you can select that Stogroup from the list. If you select *STOGROUP*, you can also define values for *PRIQTY* and SECQTY. To prevent wasted space for non-partitioned indexes, you should not define *PRIQTY* and *SECQTY* ; instead let DB2 manage the amount of primary and secondary space allocated for the index.
- *PRIQTY*:(PRIQTY) Specifies the minimum number of disk space in kilobytes to initially allocate for the index. The primary allocation should be large enough to handle the storage needs that you anticipate.
- *SECQTY*: (SECQTY) Specifies the amount of disk space in kilobytes to allocate to the index when it reaches the size of the primary allocation. If the secondary allocation space is too small, the data set might have to be extended more times to satisfy those activities that need a large space. The default value of -1 indicates that DB2 should determine the size of the secondary allocation, which is usually 10% of the primary allocation.
- *Erase Data*: (*ERASE*) If selected, specifies to erase the index data set when the index is deleted.
- *Buffer Pool*: (*BUFFERPOOL*) Identifies the buffer pool to be used for the index. The buffer pool used determines how the data blocks are cached in memory, in order to facilitate faster data access. BP0 is the default buffer pool for sorting.
- *Percent Free*: (*PCTFREE*) Specifies the maximum percentage of space in each data block to reserve for future updates. This reserved space helps to avoid row migration and chaining caused by an update operation that extends a data row's length. Tables that will not be updated should have this value set to 0.
- *Free Page*: (*FREEPAGE*) Specifies how often to leave a page of free space when index entries are created. One free page is left for every number of pages specified here in the range of 0 to 255.
- *Piece Size*: (*PIECESIZE*) Specifies the maximum addressable space of each data set for a secondary index. Specify the size followed by K (kilobyte), M (Megabyte), of G (Gigabyte). If you do not specify the unit of measure, kilobytes is assumed. Sizes specified in gigabytes are converted to megabytes in the DDL.
- *GBP Cache*: (*GPBCACHE*) In a data sharing environment, specifies what index pages are written to the group buffer pool. This option is ignored in a non-data-sharing environment unless the index is on a declared temporary table.

- *Changed*: Writes updated pages to the group buffer pool.
- *All*: Caches all pages to the group buffer pool as they are read in from DASD.
- *None* Uses the buffer pool only for cross-invalidation.

Definition page/tab

Enter or edit a definition for the auxiliary table. If the target database supports it, ER Studio Data Architect adds this definition as a comment when generating SQL code.

DDL page/tab

Displays the *CREATE AUXILIARY TABLE*.statement needed to build the auxiliary table. ER Studio Data Architect uses the platform-specific parser of the model's selected database platform to generate the auxiliary table.

Attachment Bindings tab

Bind an external piece of information, or attachment to the auxiliary table. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary. For more information, see Attaching External Documents to the Data Model.

See Also

- Creating and Editing Tablespaces
- Attaching External Documents to the Data Model
- Creating and Editing DataBases

Creating and Editing Buffer Pools

The buffer pool is a memory structure that can cache data blocks in memory, providing faster data access by retaining the object in memory to avoid I/O conflicts or remove data blocks from memory as soon as they are no longer in use, thereby saving cache space. You can choose to use the buffer pool to store indexes and tables for many different database platforms.

The default page size of a buffer pool is the size specified when the database was created unless you explicitly specify another page size. Pages can be ready into a buffer pool only if the table space page size is the same as the buffer pool page size, so ensure the buffer pool page size matches that of the table space.

The following database platforms support buffer pool creation.

- IBM DB2 for LUW 5.x, 6.x, 7.x, 8.x, 9.x, 10.x

**Note:** The Bufferpool Wizard and Bufferpool Editor share the same options, except for Attachment Bindings options which are present only in the editor.

Create a Buffer Pool

1. In the *Data Model Explorer*, expand the Logical *Main Model*, right-click the *Buffer Pools* node, and then click *New Bufferpool*.
2. *Complete theBufferpool Wizardand then clickFinishto create the* buffer pool.

TIP: Once you have created the auxiliary table, you can edit it by right-clicking the auxiliary table you want to change, and then selecting Edit Auxiliary Table.

The following helps you complete the *Bufferpool Wizard*:

Name page/tab Nodes page/tab

Definition page/tab DDL page/tab

Attachment Bindings tab

Name page/tab

- Specify the size of the buffer pool in the number of pages. If you do not specify the size, the default size is -1.
- Click the list to specify the size of the pages for the buffer pool in Kilobytes.
- *Extended Storage*: Select to use extended storage on unix systems, then if the pages requested are not in the buffer pool, they can be accessed from extended storage, which can be faster than reading the pages from disk. This option is ignored in IBM DB2 for LUW version 9 and later.


Note: Ensure your system has enough real memory for all the buffer pools, in addition to the memory required for the database manager and application.

Nodes page/tab

In version 9, IBM started using *DBPARITITOINNUM* instead of *NODE*.

Definition page/tab

Enter or edit a definition for the buffer pool. If the target database supports it, ER Studio Data Architect adds this definition as a comment when generating SQL code.

DDL page/tab

Displays the *CREATE BUFFERPOOL* statement needed to build the buffer pool. ER Studio Data Architect uses the platform-specific parser of the model's selected database platform to generate the buffer pool.

Attachment Bindings tab

Bind an external piece of information, or attachment to the buffer pool. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary. For more information, see Attaching External Documents to the Data Model.

See Also

- Developing the Physical Model
- Best Practices

Creating and Editing DataBases

Databases are DB2 structures such as collections of tables with associated indexes, as well as the tablespaces and index spaces that contain them. Using databases for administration, you can limit access to the data in the database in one operation, or give authorization to access the data as a single unit.

When you create a database the following tasks are performed:

- System catalog tables required by the database are setup.
- The database recovery log is allocated.
- The database configuration file is created and the default values are set.
- The database utilities are bound to the database.

Before creating the database, ensure you have considered the contents, layout, potential growth and usage.

The following database platforms support database creation:

- IBM DB2 for OS/390 5.x, 6.x, 7.x, 8.x, 9.x and10.x


Note: The Database Wizard and Database Editor share the same options, except for Attachment Bindings options which are present only in the editor.

Add a Database

1. In the *Data Model Explorer*, expand a physical model, right-click the *Databases* node, and then click *New Database*.
2. *Complete theDatabase Wizardand then clickFinishto create the* database.
3. Once you have created the auxiliary table, you can edit it by right-clicking the auxiliary table you want to change, and then clicking Edit Auxiliary Table.

The following helps you complete the *Database Wizard* options:

Name & Type page/tab Option page/tab

Definition page/tab DDL page/tab

Attachment Bindings tab

Name & Type page/tab

Name the database and specify the type. The database cannot already exist on the current server. The name must not start with DSNDB, unless it is a workfile database and must not start with DSN followed by five digits. Select WorkFile Database (*AS WORKFILE*) if the database operates in a data sharing environment and you are creating a database that will be used for purposes such as managing the data of temporary tables, storing intermediate SQL results,

created global and temporary tables. If you choose to create a WorkFile Database, you must also specify the member or DB2 subsystem that can use the workfile. Only one member of the data sharing group can use this workfile.

Option page/tab

To make the database easier to manager, keep work files in a separate buffer pool.

- Tablespace and index buffer pools: (*BUFFERPOOL and INDEXBP*) Identifies the buffer pools to be used. The buffer pool used determines how the datablocks are cached in memory, in order to facilitate faster data access. BP0 is the default buffer pool for sorting.
- *Storage Group*: (*STOGROUP*) If selected, stores the index on a data set managed by DB2 in the named storage group. Stogroup defined index spaces let DB2 do all the VSAM allocation work for you. If you have created a Stogroup for the data model, using the Stogroup Editor (see Creating and Editing StoGroups), you can select that Stogroup from the list. If you select Stogroup, you can also define values for *PRIQTY* and *SECQTY*. To prevent wasted space for non-partitioned indexes, you should not define *PRIQTY* and *SECQTY* ; instead let DB2 manage the amount of primary and secondary space allocated for the index.
- *Encoding Scheme*: (*CCSID*) Specifies the encoding scheme: *ASCII*, *EBCDIC*, *UNICODE* or the system installed default. This is not valid if this database is to be used as a work file in a data sharing environment. If you do not want to use the default encoding scheme that was selected when you installed DB2 for OS/390 and z/OS, you can override your installation selection here. You cannot specify an encoding scheme for a workfile database.

Definition page/tab

Enter or edit a definition for the database. If the target database supports it, ER Studio Data Architect adds this definition as a comment when generating SQL code.

DDL page/tab

Displays the *CREATE DATABASE* statement needed to build the database. ER Studio Data Architect uses the platform-specific parser of the model's selected database platform to generate the database.

Attachment Bindings tab

Bind an external piece of information, or attachment to the database. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the *Attachments* folder of the *Data Dictionary*. For more information, see Attaching External Documents to the Data Model.

See Also

- Creating and Editing Database Dependent Objects
- Creating and Editing Sequences
- Attaching External Documents to the Data Model

Creating and Editing Generators

A generator is a sequential number that can be automatically inserted in a column. Generators are typically used for purposes such as ensuring a unique value in a primary key that uniquely identifies an associated row. The Generator wizard lets you create a generator and sets its starting value to zero.

The following database platforms support generator creation.

- Interbase 2007, Interbase 2009, Interbase XE, Interbase XE3

Note: The Bufferpool Wizard and Bufferpool Editor share the same options, except for *PreSQL & PostSQL* and *Attachment Bindings* options, which are present only in the editor.

To Create a Generator

1. In the *Data Model Explorer*, expand an InterBase physical model, and then click *New Generator*.
2. *Complete theGenerator Wizardand then clickFinishto create the* buffer pool.

Tip: Once you have created the generator, you can edit it by right-clicking the generator you want to change, and then selecting *Edit Generator*.

The following helps you complete the *Bufferpool Wizard*:

Name page/tab Definition page/tab DDL page/tab PreSQL & PostSQL tab Attachment Bindings tab

Name page/tab

Provide a name for a new generator or edit the name of an existing generator.

Definition page/tab

Provide a description for the generator. It will be visible in the model.

DDL page/tab

Displays the *CREATE GENERATOR* statement needed to build the generator. ER Studio Data Architect uses the platform-specific parser of the model's selected database platform to generate the generator DDL.

PreSQL & PostSQL tab

Lets you enter SQL to be applied before or after the *CREATE GENERATOR* statement. The PreSQL and PostSQL scripts entered here are included in the script when you generate the physical database.

Attachment Bindings tab

Bind an external piece of information, or attachment to the generator. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary. For more information, see Attaching External Documents to the Data Model.

See Also

- Attaching External Documents to the Data Model
- Creating and Editing SQL Procedures

Creating and Editing SQL Procedures

SQL procedures are reusable blocks of PL/SQL that you create, which are stored in the database. Applications can call procedures to streamline code development, debugging, and maintenance. Procedures can also enhance database security by letting you write procedures granting users execution privileges to tables rather than letting them access tables directly.

Stored procedures can contain program flow, logic, and database queries. They can accept and generate parameters, return single or multiple result sets, and return values.

In addition to doing anything that can be accomplished with SQL statements, stored procedures can also:

- Execute a series of SQL statements in a single stored procedure.
- Reference other stored procedures, which can simplify a series of complex statements.
- Execute faster than individual SQL statements because the stored procedure is compiled on the server when it is created.
- Be cached in memory, for faster execution.

There are several ways that you can implement SQL procedures:

- SQL that runs before or after creating an object. For more information, see Creating and Editing PreSQL and PostSQL for the CREATE TABLE Statement
- Scripted and templated procedures that can store application code into the database for all applications to used via the CALL statement. For more information, see Creating and Editing Procedures.
- Scripted and templated triggers that enforce referential integrity rules; enforce business rules on table data; synchronize information in another table when a table is updated; initiate an action external to the DBMS when an update occurs; and prevent some types of operations, such as UPDATEs. For more information, see Creating and Editing Triggers.
- Functions that can execute multiple SQL statements from within an SQL statement. For more information, see Creating and Editing Functions.
- Packages that can store together all the code necessary to process SQL statements from a single source file. You can use packages to process and call batches of SQL. For more information, see Creating and Editing Packages.

This section also covers the following:

- Creating and Editing PreSQL and PostSQL for the CREATE TABLE Statement
- Creating and Editing Procedures
- Create a Scripted Procedure
- Apply a Scripted Procedure to a Submodel or Model

- Create Reusable Procedure
- Create a Scripted User-Defined Procedure
- Create a Templated User-Defined Procedure
- Bind a Reusable Procedure to a Table
- Create an SQL Procedure
- Import an SQL Procedure
- Export an SQL Procedure

See Also

- Creating and Editing Database Dependent Objects
- Developing the Physical Model

Create a Scripted Procedure

1. In the *Data Model Explorer*, expand the physical *Main Model*, right-click the *Procedures* node, and then click *New Procedure*.
2. Complete the *Procedure SQL Editor* and then click *Validate* to ensure the SQL is valid.
3. Click *OK* to create the procedure and exit the editor.
4. Once you have created the auxiliary table, you can edit it by right-clicking the auxiliary table you want to change, and then selecting Procedure.

In order for code to be generated for procedural logic such as triggers and procedures, the results of the logic must be stored in a variable called *resultstring*. For example, in the *Northwind.dm1* sample model, the Data Dictionary lists a reusable trigger, SYSUPDATE, for the Oracle platform. The code for this trigger includes the following statement, which is required for code to be generated for the trigger:
resultstring = trigBody

The following help you complete the options of the *Procedure SQL Editor*:

SQL page/tab Description page/tab

Permissions page/tab Attachment Bindings tab

SQL page/tab

Enter the SQL for the procedure.

Description page/tab

Enter or edit a definition for the procedure. If the target database supports it, ER Studio Data Architect adds this definition as a comment when generating SQL code.

Permissions page/tab

Sets access roles and user permissions for the procedure. Keep in mind that the more specific permissions are, the more time you may have to spend maintaining and updating them. The roles and users must be previously assigned to appear in the Procedure Editor. For more information, see Granting and Revoking Permissions to Modify Database Objects, Creating and Editing Database Roles, and Creating and Editing Database Users.

- Against InterBase platforms, no *Permissions* page/tab is available.

Attachment Bindings tab

Bind an external piece of information, or attachment to the procedure. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype.

Attachments are created in the Attachments folder of the Data Dictionary. For more information, see Attaching External Documents to the Data Model.

See Also

- Apply a Scripted Procedure to a Submodel or Model
- Creating and Editing Database Dependent Objects
- Developing the Physical Model

Apply a Scripted Procedure to a Submodel or Model

You can copy a scripted procedure to from a submodel to the physical model or to one or more submodels using the *Apply to Submodel(s)* command. For more information, see Copying a Database Object to a Submodel (Apply to Submodel).

See Also

- Create a Scripted Procedure
- Creating and Editing Database Dependent Objects
- Developing the Physical Model

Creating and Editing PreSQL and PostSQL for the CREATE TABLE Statement

In the PreSQL & PostSQL tab of the Table Editor you can enter SQL procedures to be applied before and after the CREATE TABLE statement.

- The SQL in the procedure will be validated in context of the table selected and the database platform.

1. In the *Data Model Window* or *Data Model Explorer*, double-click a table to launch the *Table Editor*.
2. Click the *PreSQL & PostSQL* tab.
3. Enter or edit an SQL procedure to run before the *CREATE TABLE* statement.
4. Click the *PostSQL* tab and then enter or edit an SQL procedures to run before the *CREATE TABLE* statement.
5. Click *OK* to implement the changes and exit the editor.

See Also

- Developing the Physical Model
- Create a Scripted Procedure

Creating and Editing Procedures

Procedures are reusable blocks of PL/SQL, stored in the database, that applications can call. Procedures streamline code development, debugging and maintenance by being reusable. Procedures can enhance database security by letting you write procedures granting users execution privileges to tables rather than letting them access tables directly. Also, procedures can increase the efficiency of your database. They can consolidate and centralize logic that might otherwise have existed in multiple applications that access the same database, making the code easier to maintain. Since only the procedure executable is stored on the server, decreasing its storage requirements, and the procedure is executed on the server, reducing network traffic.

Procedures are similar to functions except that functions can be used within SQL statements, whereas procedures must be invoked using the *CALL* statement.

ER Studio Data Architect supports the following procedure types:

- *Regular or scripted procedures:* Specific to the table on which they are created. You can write scripted procedures in SQL You can create a scripted procedure through the Table Editor - Dependencies Tab or though the Procedure SQL Editor, which you can access by double-clicking the procedure name in the Procedures node of the Data Model Explorer.
- *Templated Procedures*: Specific to the table on which they are created or modified. You can write templated procedures in BASIC. Once created, they do not appear in the list of procedures on the *Procedures* node of the *Data Model Explorer*.
- *Reusable procedures*: Created in the data dictionary, reusable procedures can become templated if they are modified through the Table Editor. You can write templated procedures in BASIC. Reusable procedures are templated procedures written in BASIC code. These procedures can be applied to any table, since they use the table's context in the code. The Reusable Procedure node in the Data Dictionary includes DBMS platform nodes. ER Studio Data Architect organizes reusable procedures by platform. When you want to create, edit, or delete a reusable procedure, right-click the target DBMS node and then click the appropriate short cut menu.

The following database platforms support Procedures:

- IBM DB2 for LUW 5.x, 6.x, 7.x, 8.x, 9.x, and 10.x
- InterBase 2007, XE, XE3
- Microsoft Access 6.x and 7.x
- Microsoft SQL Server 6.x, 7.x, 2000 - 2014
- Oracle 7.x, 8.x, 9i, 10g, 11g and 12c
- Sybase ASA 6.0, 7.0, 8.0, 9.0, Adaptive Server IQ 12.
- Sybase Adaptive Server IQ 12.
- Sybase ASE 11.0, 11.5, 11.9, 12.0, 12.5, and 15

- The *Northwind.dm1* sample model illustrates how procedures can be used to gather information and perform computations on them to provide summary information.

Notes

- You can view the *templated* procedures associated with a table on the Dependencies tab of the Table Editor.
- You can view the scripted procedures you create under the Procedures tab of the node.
- In order for code to be generated for procedural logic such as triggers and procedures, the results of the code must be stored in a variable called *resultstring*. For example, in the *Northwind.dm1* sample model, the Data Dictionary lists a reusable trigger, SYSUPDATE, for the Oracle platform. The code for this trigger includes the following statement, which is required for code to be generated for the trigger:
resultstring = trigBody

See Also

- Creating and Editing Database Dependent Objects
- Creating and Editing SQL Procedures

Create Reusable Procedure

Reusable procedures are maintained in the database and can save you time because they do not have to be recreated for every database application in which you want to use the procedure.

Reusable user-defined procedures are created in the Data Dictionary. For more information, see Reusing Procedural Logic

See Also

- Create a Scripted Procedure
- Creating and Editing Procedures

Create a Scripted User-Defined Procedure


Note: The SQL in the procedure will be validated in context of the table selected.

1. In the *Data Model Window* or *Data Model Explorer*, double-click a table to launch the *Table Editor*.
2. Click the *PreSQL & PostSQL* tab.
3. Enter or edit an SQL procedure to run before the *CREATE TABLE* statement.
4. Click the *PostSQL* tab and then enter or edit an SQL procedures to run before the *CREATE TABLE* statement.
5. Click *OK* to implement the changes and exit the editor.

In order for code to be generated for procedural logic such as triggers and procedures, the results of the logic must be stored in a variable called *resultstring*. For example, in the *Northwind.dm1* sample model, the Data Dictionary lists a reusable trigger, SYSUPDATE, for the Oracle platform. The code for this trigger includes the following statement, which is required for code to be generated for the trigger: `resultstring = trigBody`

See Also

- Creating and Editing Procedures
- Create a Scripted Procedure

Create a Templated User-Defined Procedure

The Templated Procedure Editor lets you create or edit a reusable procedure. Templated procedures are specific to the specific table on which they are created or modified. Reusable procedures created in the data dictionary can become templated if they are modified through the Table Editor. You can write templated procedures in BASIC.

Notes

- In order for code to be generated for procedural logic such as triggers and procedures, the results of the logic must be stored in a variable called *resultstring*. For example, in the *Northwind.dm1* sample model, the Data Dictionary lists a reusable trigger, SYSUPDATE, for the Oracle platform. The code for this trigger includes the following statement, which is required for code to be generated for the trigger:
  resultstring = trigBody
- The Templated Procedure Editor includes find and replace functionality for working with SQL text strings.
- Templated procedures are different from scripted procedures in that they are reusable.
- You can only access the Templated Procedure Editor from the Dependencies tab of the Table Editor.
- Procedures are considered to be data model schema, along with packages, functions, materialized views, auxiliary tables, synonyms, and triggers.

The schema objects tab of the options editor allows you to set default trigger actions for Parent and Child actions. It also enables you to apply default triggers to tables when relationships are created.

See Also

- Creating and Editing Procedures
- Create a Scripted Procedure
- Create a Scripted User-Defined Procedure

Bind a Reusable Procedure to a Table

Once you have defined the reusable procedure in the Data Dictionary, you can drag the procedure from the Reusable Procedural Logic node in the Data Dictionary onto a table in the Data Model Window. This binds the procedure to the table. You can see any bound procedures on the Dependencies tab of the Table Editor.

See Also

- Creating and Editing Procedures
- Create a Scripted Procedure

Create an SQL Procedure

1. In the *Data Model Explorer*, expand the Physical *Main Model*, right-click the *Procedures* node, and then click *New Procedure*.
2. Complete the *Procedure SQL Editor* and then click *OK* to create the procedure.
3. Once you have created the SQL procedure, you can edit it by right-clicking the procedure you want to change, and then selecting Edit SQL Procedure.

The following helps you complete the options of the *Procedure SQL Editor*:

Description tab Attachment Bindings tab

Permissions tab Notes

Description tab

Enter or edit a definition for the procedure. If the target database supports it, ER Studio Data Architect adds this definition as a table comment when generating SQL code.

Attachment Bindings tab

Bind an external piece of information, or attachment to the procedure. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary. For more information, see Attaching External Documents to the Data Model.

Permissions tab

Sets access roles and user permissions for the procedure. Keep in mind that the more specific permissions are, the more time you may have to spend maintaining and updating them. The roles and users must be previously assigned to appear in the Procedure SQL Editor. For more information, see Creating and Editing Database Roles and Creating and Editing Database Users.

- Against InterBase platforms, no *Permissions* page/tab is available.

Notes

- You can choose to launch your preferred SQL editor when editing functions and procedures. Using the Tools tab of the Options Editor, you can enter the name and location of the SQL editor that will launch and manage the SQL ER Studio Data Architect outputs. ChooseTools > Options > Tools > ISQL Path.
- For all new models you can choose:

- To display the schema object description or its associated symbol; choose Tools > Options > Display and then select the desired display mode.
- To hide invalid schema objects; choose Tools > Options > Schema Objects and then select *Hide Invalid Schema Object Bitmap*.
- For the current data model, you can choose:
- To display the schema objects or objects relationships, and object definition or object symbol; choose View > Diagram And Object Display Options > Schema Object and then select the desired display options.
- What to display when you mouse over a schema object; choose Tools > Options > Diagram > Schema Object Display and choose to display the object description or its associated DDL.
- For physical models, you can enter SQL to be applied before and after the CREATE OBJECT clause in the CREATE TABLE statement. Choose Model > Model Properties > PreSQL & Post SQL.
- The sample physical model for SQL Server 2008 in *Northwind.dm1* provides some good examples of how SQL procedures can be used.

See Also

- Creating and Editing Database Dependent Objects
- Developing the Physical Model
- Creating and Editing Procedures

Export an SQL Procedure

1. In the *Data Model Explorer*, expand the Physical *Main Model*, right-click the *Procedures* node, and then click *New Procedure*.
2. On the *Description* tab, click *Export* and then type, or browse to and then select the name of the SQL procedure file name where you want to save the SQL procedure.
3. Complete the *Procedure SQL Editor* and then click *OK* to create the procedure.

See Also

- Creating and Editing Database Dependent Objects
- Developing the Physical Model
- Import an SQL Procedure

Import an SQL Procedure

1. In the *Data Model Explorer*, expand the Physical *Main Model*, right-click the *Procedures* node, and then click *New Procedure*.
2. On the *Description* tab, click *Import* and then type, or browse to and then select the name of SQL procedure file you want to import.
3. Complete the *Procedure SQL Editor* and then click *OK* to create the procedure.

See Also

- Creating and Editing Procedures
- Developing the Physical Model
- Export an SQL Procedure

Creating and Editing Functions

Functions are subroutines that you define. Functions can be written in a programming language such as C, COBOL, or Java that returns a scalar value or a complete table, or they can call another function, or they can be written in SQL and return a scalar value. Functions are useful for reusable application logic. You can use functions to determine the best methods for controlling access and manipulation of the underlying data contained in an object. Functions accept a number of parameters and pass a single value back to the calling program. Functions can be used in the database to check the validity of the data being entered. For example, functions can be used to validate zip codes. By invoking a routine with the zip code, the function can return a true or false value based on if the zip code is valid.

You can create a reusable function Function SQL Editor where you can create a function and bind attachments to it. You can also add a Function through the Table Editor - Dependencies tab. They are not associated with a particular table; they are stored in the database.

The following database platforms support SQL functions:

- IBM DB2 for LUW 5.x, 6.x, 7.x, 8.x, 9.x, 10.x
- Microsoft SQL Server 2000 - 2014
- Oracle 7.x, 8.x, 9i, 10g, 11g, 12c
- Sybase ASA 6.0, 7.0, 8.0, 9.0, Adaptive Server IQ 12

Create a Stored SQL Function

1. In the *Data Model Explorer*, expand the Physical model, right-click the *Functions* node, and then click *New Function*.
2. *Complete theFunction SQL Editorand then clickOKto create the* function.
3. Once you have created the function, you can edit it by right-clicking the function you want to change, and then selecting *Edit Function*.

The following helps you complete the *Function SQL Editor* options:

Description tab Permissions tab

Attachment Bindings tab Notes

Description tab

Enter or edit a definition for the function. If the target database supports it, ER Studio Data Architect adds this definition as a table comment when generating SQL code.

Permissions tab

Sets access roles and user permissions for the function. Keep in mind that the more specific permissions are, the more time you may have to spend maintaining and updating them. The

Roles and Users must be previously assigned to appear in the Function Editor. For more information, see Creating and Editing Database Roles and Creating and Editing Database Users.

Attachment Bindings tab

Bind an external piece of information or attachment to the function. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary. For more information, see Attaching External Documents to the Data Model.

Notes

- You can choose to launch your preferred SQL editor when editing functions and procedures. Using the Tools tab of the Options Editor, you can enter the name and location of the SQL editor that will launch and manage the SQL ER Studio Data Architect outputs. Choose Tools > Options > Tools > ISQL Path.
- For all new models you can choose:
- To display the schema object description or its associated symbol, choose Tools > Options > Display and then select the desired display mode.
- To hide invalid schema objects, choose Tools > Options > Schema Objects and then select *Hide Invalid Schema Object Bitmap*.
- For the current data model, you can choose:
- To display the schema objects or objects relationships, and object definition or object symbol; choose View > Diagram And Object Display Options > Schema Object and then select the desired display options.
- What to display when you mouse over a schema object; choose Tools > Options > Diagram > Schema Object Display and choose to display the object description or its associated DDL.

See Also

- Developing the Physical Model
- Creating and Editing Packages

Creating and Editing Packages

A package is a group of procedures, functions, variables, and SQL statements, used to store together related objects. Packages contain all the information needed to process SQL statements from a single source file. You can use packages to process and call batches of SQL. They are not associated with a particular table; they are stored in the database.

The following database platforms support packages:

- Oracle 7.x, 8.x, 9i, 10g, 11g and 12c

Create a Package

1. In the *Data Model Explorer*, expand the Logical *Main Model*, right-click the *Packages* node, and then click *New Package*.
2. Complete the *Package Editor* and then click *OK* to create the *package.*

TIPS:

- Once you have created the auxiliary table, you can edit it by right-clicking the package you want to change, and then clicking Edit Package.
- The Package Editor includes find and replace functionality for working with SQL text strings.

The following helps you complete the options of the *Package Editor*:

Header tab Body tab

Definition tab Attachment Bindings tab

Permissions tab

Header tab

The header contains the package specification. Here you can declare types, variables, constants, exceptions, cursors and subprograms. The header does not contain any code. You can specify a package that contains only global variables that will be used by subprograms or cursors. This type of package declares only types, constants, variables, exceptions, and call specifications and does not require a package body.

Body tab

The package body provides the implementation for the subprograms and the queries for the cursors declared in the header.

Definition tab

Enter or edit a definition for the package. If the target database supports it, ER Studio Data Architect adds this definition as a comment when generating SQL code.

Attachment Bindings tab

Bind an external piece of information, or attachment to the package. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary. For more information, see Attaching External Documents to the Data Model.

Permissions tab

Sets access roles and user permissions for the package. Keep in mind that the more specific permissions are, the more time you may have to spend maintaining and updating them. The Roles and Users must be previously assigned to appear in the Package Editor. For more information, see Creating and Editing Database Roles and Creating and Editing Database Users.

See Also

- Creating and Editing Database Roles
- Creating and Editing Database Users
- Attaching External Documents to the Data Model

Creating and Editing Triggers

Triggers are code objects associated with a table that automatically execute when the table is modified by an SQL INSERT, UPDATE, or DELETE statement. Triggers can contain complex SQL statements, such as those required to query other tables, but cannot accept parameters or arguments, and cannot perform commit or rollback operations. They are also subject to coding errors like every other script or application.

Because triggers are customizable and execute automatically, they are often used to:

- *Enforce referential integrity rules*. For example, when a new customer in the eastern territory is added to the customer table, the new customer information is also added to the table for the sales representative handling that region.
- *Enforce business rules on table data*. For example, restricting an order insertion based on the customer's account status. If the customer's payment is overdue or the order exceeds the customer's credit limit, the order is rejected.
- *Synchronize information in another table when a table is updated*. For example, you can create a delete trigger on the *product* table that the trigger uses to locate and delete matching rows in the *sales* table and *pending orders* table.
- *Initiate an action external to the SQL Server when an update occurs*. For example, when a new employee is hired and added into the employees table, an e-mail is sent to the human resources department that prompts them to perform certain actions like arranging desk space and computer equipment, and ensuring all that all required paperwork is sent to the new employee and is returned completed to the human resources department.
- *Prevent some types of operations, such as UPDATEs.* For example, a trigger can roll back updates that attempt to apply a *bonus* (stored in the *renumeration* table) to *employees* (stored in the *persons* table) who work less than 16 hours a week.

You can view the triggers associated with your models from the Triggers node in the Data Model Explorer, but to create or edit triggers, you must use the Triggers tab of the table with which you want to associate the trigger.

The generated trigger code is customized for each target database platform. ER Studio Data Architect uses templates to implement trigger code generation. SQL Procedures can be triggers that are a special type of procedure.

ER Studio Data Architect supports the following trigger types:

- *Referential Integrity Triggers* ensure implied relationships in the database are enforced, by preventing users from creating discrepancies in the database. They inform the database how to manage or process the procedural SQL commands, which enforce the business rules of an organization. ER Studio Data Architect defines Referential Integrity Triggers by establishing insert, update and delete options between parent and child tables.

On the Trigger tab of the Relationship Editor, you can define the level of referential integrity and SQL insert, update, and delete behaviors. For more information, see Create a Referential Integrity Trigger.

- *User-Defined Triggers* are usually customized for special situations to override the default capabilities of database generated code. User-defined triggers are closely tied to DBMS-specific code, and therefore, are implemented only in physical models. There are three types of user-defined triggers:
- *Scripted Triggers* that are written in the raw procedural code of the database such as Oracle's PL/SQL or Microsoft's Transact-SQL. You can reverse-engineer these objects if they exist on the database. You can also display them in the Data Model Window. Reverse-engineered triggers have table names and other object references hard-coded into their underlying code. For more information, see Create a Scripted User-Defined Trigger.
- *Templated Triggers* that do not have names of specific objects the trigger references, such as table names and specific columns hard-coded into their underlying code. This enables these triggers to be reused across various objects of an application, and the body of the trigger code to be reused across models. For more information, see Create a Templated User-Defined Trigger.
- *Reusable Triggers* that can perform actions against any number of tables. For more information, see Create a Reusable User-Defined Trigger.

Notes

- To view triggers on the Data Model Window, you must enable them. To display triggers, choose *Tools* > *Options* > *Schema Objects*.
- Specify default trigger actions for new triggers on the Schema Objects tab of the Options editor. On this tab you can also choose to apply the default triggers. For more information, see
- You can only access the Trigger Editors from the Dependencies tab of the Table Editor.
- You can choose to display only specific attributes of a relationship, including trigger actions on the Relationship tab of the Diagram And Object Display Options. For more information, see Customizing the Display of Diagrams and Objects.
- In order for code to be generated for procedural logic such as triggers and procedures, the results of the code must be stored in a variable called *resultstring*. For example, in the *Northwind.dm1* sample model, the Data Dictionary lists a reusable trigger, SYSUPDATE, for the Oracle platform. The code for this trigger includes the following statement, which is required for code to be generated for the trigger:
  resultstring = trigBody

Trigger Example

```
Sub Main
```

```
Dim trigName As String

Dim trigBody As String

Dim attrib As AttributeObj

Dim crlf As String

crlf = Chr(13) + Chr(10)

trigName = Left(CurrEntity.TableName, 24) + "UpdTrg"

trigBody = "CREATE OR REPLACE TRIGGER " + trigName + " AFTER UPDATE ON " +
CurrEntity.TableName + crlf

trigBody = trigBody + "REFERENCING OLD AS OLD NEW AS NEW" + crlf

trigBody = trigBody + "FOR EACH ROW" + crlf

trigBody = trigBody + "BEGIN" + crlf

trigBody = trigBody + "UPDATE " + CurrEntity.TableName + " SET UPDATEDATE =
SYSDATE WHERE "

For Each attrib In CurrEntity.Attributes

If attrib.PrimaryKey = True Then

If Right(trigBody,6) <> "WHERE " Then

trigBody = trigBody + " AND "

End If

trigBody = trigBody + attrib.ColumnName + " =:OLD." + attrib.ColumnName

End If

Next attrib

trigBody = trigBody + ";" + crlf + "END;"

'Resultstring outputs the trigger to the DDL script when the Generate
Database

'wizard is used. The string variable used to generate the trigger DDL needs
to be set to it.

resultstring = trigBody

'This message box is used to view the SQL when debugging the VB code. A table
has to be selected.

MsgBox(trigBody)

End Sub
```

The following database platforms support Triggers:

- IBM DB2 for LUW 5.x, 6.x, 7.x, 8.x, 9.x, and 10.x
- IBM DB2 for OS/390 6.x and for z/OS 7.x, 8.x, 9.x, and 10.x
- InterBase 2009, XE, XE3

- Microsoft SQL Server 4.x, 6.x, 7.x, 2000 - 2014
- Oracle 7.x, 8.x, 9i, 10g, 11g and 12c
- Sybase ASA 6.0, 7.0, 8.0, 9.0
- Sybase Adaptive Server IQ 12.
- Sybase SQL Server System ASE 11.0, 11.5, 11.9, 12.0, 12.5, 15

Notes

- When generating a database using the DDL Generation Wizard, you can choose which triggers to include and whether they should be generated using either CREATE or DROP statements. On the General Options tab, you can choose to create system triggers such as referential integrity triggers.
- Triggers will display in physical model of the Data Model Explorer in this format: `triggerName. assocatedTableName.`
- You can display the trigger actions by selecting Trigger Actions on the Relationship tab of the Diagram and Display Options dialog. For example, when trigger actions are displayed U:R displays next to an Employee table with an Update Restrict trigger.
- There are certain commands, statements, and operations that cannot be used in trigger programs. To avoid generating exceptions with your triggers, consult your DBMS documentation.

Create a Referential Integrity Trigger

Referential integrity triggers, or system triggers, ensure implied relationships in the database are enforced, by preventing users from creating discrepancies in the database. They inform the database how to manage or process the templated procedures, which enforce the business rules of an organization. On the Trigger tab of the Relationship Editor, you can define referential integrity triggers by establishing insert, update and delete behaviors for the parent and child tables.


Note: You cannot create a referential integrity trigger for a non-specific relationship.

1. On the Data Model Window, create a relationship between two tables you want to ensure get updated when an SQL INSERT, UPDATE or DELETE operation is performed on one of the tables.
2. On the *Data Model Window,* double-click the relationship.
3. On the *Relationship Editor*, click the *Trigger* tab.

The following describes the options that require additional explanation.

- In order for code to be generated for procedural logic such as triggers and procedures, the results of the trigger must be stored in a variable called *resultstring.* For example, in the *Northwind.dm1* sample model, the Data Dictionary lists a reusable trigger, SYSUPDATE, for the Oracle platform. The code for this trigger includes the following

statement, which is required for code to be generated for the trigger:
resultstring = trigBody

- ER Studio Data Architect uses templates to implement referential integrity trigger code generation. The generated trigger code is customized for each target database platform. For each INSERT, UPDATE or DELETE action, you can apply the following trigger templates based on type, selections, and level of data modification required:
- *Parent Actions*: Defines what the child does when the parent is updated.
- *Insert*: *None*: This cannot be changed.
- *None* (No Action): The trigger verifies the existence of the foreign key values in the parent table's primary key. If the values cannot be validated, the update or delete fails the referential integrity checks.
- *Set Null*: The trigger verifies the existence of the foreign key values in the parent table's primary key. If the values cannot be validated, then the trigger sets the foreign key value to null in the child table and then updates or deletes the parent. For example, you have a parent table *persons* and a child table *tasks*, which through a mandatory relationship share an *id*. By using a *Set Null* trigger you can allow the user to insert data into the tasks table that contains an id for which their is no corresponding value in the persons table. The database will not throw a referential integrity error in this case.
- *Set Default*: Sets the foreign key to its DEFAULT value, then updates or deletes the parent.
- *Restrict*: The trigger verifies the existence of foreign key values in the parent table's primary key. If the values cannot be validated, then the trigger prevents the insertion, update or deletion of data. For example, you have a parent table *persons* and a child table *tasks*, which through a mandatory relationship share an id. By using a *Restrict* trigger you can prevent the user from inserting data into the tasks table that contains an id for which their is no corresponding value in the persons table. The database will throw a referential integrity error in this case.
- *Cascade*: If a primary key is updated or deleted, the trigger cascades that modification to corresponding foreign key values in dependent tables.

Create a Scripted User-Defined Trigger

1. On the *Data Model Window* or the *Data Model Explorer*, select the table on which you want to create a Trigger.
2. Choose Edit > Edit Database Table.
   Tip: Alternatively, on the *Data Model Window*, open the *Table Editor* by double-clicking the table or right-clicking the object and then selecting *Edit Table*.
3. On the *Table Editor*, click the *Dependencies* tab.
4. Click the *Add* button and then select *Triggers* > *Scripted*.

5. Complete the *Trigger Editor* and then click *Finish* to create the trigger.

The following helps you complete the *Trigger Editor* options:

Notes SQL tab

Description tab Attachment Bindings tab

Notes

- In order for code to be generated for procedural logic such as triggers and procedures, the results of the trigger must be stored in a variable called *resultstring*. For example, in the *Northwind.dm1* sample model, the Data Dictionary lists a reusable trigger, SYSUPDATE, for the Oracle platform. The code for this trigger includes the following statement, which is required for code to be generated for the trigger:
  resultstring = trigBody
- The Trigger Editor includes find and replace functionality for working with SQL text strings.
- **Name**: Lets you enter a trigger name or edit the existing one.

SQL tab

Lets you enter a *CREATE TRIGGER* statement for the trigger, or edit the existing script. If you do not want to enter an SQL script, you can use the Import button to import it.

- *Export*: Lets you export the SQL script to a *.sql file. This is useful for when you want to create additional triggers based on the one you are currently creating. You can export the code, and later import it into another procedure. When you click Export, the Save As dialog box opens. Enter the file name. ER Studio Data Architect saves the files in the Model folder, if you want to save your files in a different folder, browse and locate it.
- *Import*: Lets you import **.sql* files. When you click Import, the Open dialog box opens. You can enter the name of the **.sql* file, or browse and locate it.
- *Validate*: Lets you validate the SQL script. If ER Studio Data Architect detects any errors, it returns them in a message box. Errors include the error type, line, and column information.

Description tab

Includes a text box where you can enter a Trigger description.

Attachment Bindings tab

Bind an external piece of information, or attachment to the trigger. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype.

Attachments are created in the Attachments folder of the Data Dictionary. For more information, see Attaching External Documents to the Data Model.

Create a Templated User-Defined Trigger

1. In the *Data Model Explorer*, expand the Physical *Main Model*, right-click the Triggers node, and then click *New* Trigger.
2. *Complete the* Trigger Wizard *and then click* Finish *to create the trigger.*

Tip: Once you have created the auxiliary table, you can edit it by right-clicking the auxiliary table you want to change, and then selecting Edit Auxiliary Table.

The following helps you complete the options of the *Trigger Wizard*:

Notes Description page/tab

Attachment Bindings tab

Notes

- The Trigger Editor includes find and replace functionality for working with SQL text strings.
- In order for code to be generated for procedural logic such as triggers and procedures, the results of the trigger must be stored in a variable called *resultstring*. For example, in the *Northwind.dm1* sample model, the Data Dictionary lists a reusable trigger, SYSUPDATE, for the Oracle platform. The code for this trigger includes the following statement, which is required for code to be generated for the trigger:
  resultstring = trigBody

Description page/tab

Enter or edit a definition for the description. If the target database supports it, ER Studio Data Architect adds this definition as a comment when generating SQL code.

Attachment Bindings tab

Bind an external piece of information, or attachment to the trigger. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary. For more information, see Attaching External Documents to the Data Model.

Create a Reusable User-Defined Trigger

Reusable user-defined triggers are created in the Data Dictionary. For more information, see Create and Edit Reusable Triggers

See Also

- Attaching External Documents to the Data Model
- Reusing Procedural Logic

Creating and Editing Materialized Query Tables

A materialized query table is a table that, like a materialized view, is defined based upon the result of a query addressing one or more tables. While the query on which a view is based is run whenever the view is referenced, a materialized query table stores the query results as data that you can work with instead of the data that is in the underlying tables.

Materialized query tables can significantly improve query performance, especially for complex queries.

The following database platforms support Materialized Query Tables:

- IBM DB2 for LUW 5.x, 6.x, 7.x, 8.x, 9.x, and 10.x


Note: The DB2 Materialized Query Table Wizard and DB2 Materialized Query Table Editor share the same options, except for Attachment Bindings options which are present only in the editor.

Add a Materialized Query Table

1. In the *Data Model Explorer*, expand the Physical *Main Model*, right-click the *Materialized Query Table* node, and then click *New Materialized Query Table*.
2. *Complete theDB2 Materialized Query Table Wizardand then clickFinishto create the* materialized query table.

Tip: Once you have created the materialized query table, you can edit it by right-clicking the materialized query table you want to change, and then clicking Edit Materialized Query Table.

The following helps you complete the options of the *DB2 Materialized Query Table Wizard*:

Name page/tab Location page/tab

SQL page/tab Definition page/tab

DDL page/tab Attachment Bindings tab

Notes

Name page/tab

The name of the materialized query table is derived from the information entered here, in the format of *owner*. *name* where:

- *owner* is the owner of the materialized query table.
- *name* is the name of the materialized query table.

Location page/tab

- *What is the table tablespace*: To avoid the performance overhead created by logging changes to the data, create materialized query tables in a table space that is defined as NOT LOGGED.
- *Refresh Preferences*: Determines when the materialized query table is update when its base tables are updated.
- *Query Optimization*: If enabled, DB2 can consider the materialized query table in automatic query rewrite. When query optimization is enabled, DB2 is more restrictive of what can be selected in the full select for a materialized query table.

SQL page/tab

Enter the entire select statement for the materialized view, starting with the keyword *SELECT*.

Definition page/tab

Enter or edit a definition or description for the materialized view. If the target database supports it, ER Studio Data Architect adds this definition as a comment when generating SQL code.

DDL page/tab

Displays the *CREATE SUMMARY TABLE AS... (SELECT...)* statement needed to build the node group. ER Studio Data Architect uses the platform-specific parser of the model's selected database platform to generate the materialized query table.

Attachment Bindings tab

Bind an external piece of information, or attachment to the materialized query table. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary. For more information, see Attaching External Documents to the Data Model.

Notes

- The materialized query table is not populated when first created. You must explicitly populate the materialized query table using the REFRESH TABLE statement for system-maintained materialized query tables or by using the LOAD utility, INSERT statement or the REFRESH TABLE statement for user-maintained materialized query tables.

See Also

- Attaching External Documents to the Data Model
- Creating and Editing Materialized Views

Creating and Editing Materialized Views

The performance of data warehousing and decision support system tools can be significantly increased when materialized views are used. Materialized views increase the speed of queries that access many records, allowing you to query terabytes of data in seconds. A materialized view is a pre-computed table comprising aggregated or joined data from fact and possibly dimension tables which is also known as a summary or aggregate table. An example of such a pre-defined query would be the sum of sales by region for a specific period of time. The storage requirements of the pre-computed summations and data joins of a materialized view are small compared to the original source data. A materialized view creates a real table that can be indexed and analyzed.

The following database platforms support materialized views:

- Oracle 8.x, 9i, 10g, 11g and 12c

**Note:** The Oracle Materialized View Wizard and Oracle Materialized View Editor share the same options, except for PreSQL & PostSQL, Permissions, and Attachment Bindings options which are available only in the editor.

Create a Materialized View

1. In the *Data Model Explorer*, expand the Physical *Main Model*, right-click the *Materialized Views* node, and then click *New Materialized View*.
2. *Complete theOracle Materialized View Wizardand then clickFinishto create the* materialized view.

Tip: Once you have created the materialized view, you can edit it by right-clicking the materialized view you want to change, and then clicking Edit Materialized View.

The following helps you complete the options of the *Oracle Materialized View Wizard*:

Name page/tab	Refresh page/tab
Query page/tab	Transaction page/tab
Extent page/tab	Options page/tab
Definition page/tab	DDL page/tab
PreSQL & PostSQL tab	Permissions tab
Attachment Bindings tab	Requirements for the Fast Refresh Method

Name page/tab

Users already defined in the logical or physical model can be selected from the list or owners for the materialized view.

Refresh page/tab

- *How should the materialized view be refreshed?* Choose how the materialized view is synchronized with respect to the base tables it was created from. The query defining the materialized view determines which refresh method can be used; the materialized view can always be rebuilt from scratch but it may not always be possible to perform a fast refresh on it. For more information, see Requirements for the Fast Refresh Method.
    - *Complete*: Build from scratch.
    - *Fast*: Apply only the changes.
    - *Force*: Perform a fast refresh if possible, if not possible perform a complete refresh.
    - *Never*: Do not refresh the view automatically.
- *Choose a refresh mechanism*: Choose when the materialized view is refreshed.
    - *ON DEMAND*: If selected, you can run a build-in Oracle procedure to refresh the materialized view such as *DBMS_MVIEW.REFRESH* (refreshes specific materialized view), *DBMS_MVIEW.REFRESH_DEPENDENT* (refreshes materialized views associated with specific base tables) or *DBMS_MVIEW.REFRESH_ALL_MVIEWS* (refreshes all materialized views).
    - *ON COMMIT*: If selected, the materialized view is refreshed when a transaction is committed that updates tables referenced in the materialized view.
    - *Automatically*: If selected, Oracle refreshes the materialized view at the specified date and time or the specified frequency.
- *Select a refresh method*: If this materialized view is based on the Primary Key, it allows the master tables of the view to be reorganized without affecting the eligibility of the view for fast refresh. In this case the master table must contain a primary key constraint. Select Rowid if the materialized view is organized based on the RowID, which is useful if the view does not include all primary key columns of the master tables.
- *Primary key*: If selected, the materialized view master tables can be reorganized without affecting its ability to be refreshed using the Fast refresh method. Select this method only if the master table contains an enabled primary key constraint

Query page/tab

- Select a pre-defined tablespace from the list or specify the name of another tablespace where you want the materialized view placed.
- Specify *Default* for the rollback segment if you want Oracle to select the rollback segment to use. Default is most useful when modifying a materialized view.
- One *Master Rollback Segment* is stored for each materialized view and is validated when the materialized view is created and refreshed. The master rollback segment is ignored if the materialized view is complex.

- *What is the materialized view query?* Enter the SQL query to be used to populate and to refresh the materialized view. The SQL must be valid to create the materialized view.

Transaction page/tab

- *Initial* transactions: (*INITTRANS*) Specifies the number of DML transactions for which space is initially reserved in the data block header. Oracle stores control information in the data block to indicate which rows in the block contain committed and uncommitted changes. The amount of history that is retained is controlled by this parameter. If too many transactions concurrently modify the same data block in a very short period, Oracle may not have sufficient history information to determine whether a row has been updated by a 'too recent' transaction. Specify a higher value for materialized views based on tables that may experience many transactions updating the same blocks.
- *Max* transactions: (*MAXTRANS*) Once the space reserved by *INITRANS* is depleted, space for additional transaction entries is allocated out of any available free space. Once allocated, this space effectively becomes a permanent part of the block header. This parameter limits the number of transaction entries that can concurrently use data in a data block and therefore limits the amount of free space that can be allocated for transaction entries in a data block.
- *Percent Free*: (*PCTFREE*) Specifies the maximum percentage of space in each data block to reserve for future updates. This reserved space helps to avoid row migration and chaining caused by an update operation that extends a data row's length. Tables that will not be updated should have this value set to 0.
- *Percent Used*: (*PCTUSED*) Specifies the maximum percentage of available space in each data block before re-adding it to the list of available blocks. When deletes take place and the room available in a block falls below this value, the block is made available for new inserts to take place. Materialized views that will not be updated should have this value set to 99. The default value is 40% means that blocks are available for insertion when they are less than 40% full.

Extent page/tab

- *Initial Extent*: (*INITEXTENT*) Specifies the initial number of data blocks that Oracle should reserve. Oracle will reserve the number of data blocks that correspond to the initial extent for that table's rows.
- *Next Extent*: (*NEXT*) Specifies the size in kilobytes of the next extent. Monitor this figure against the largest available chunk of free space in the tablespace. If a table cannot allocate its next extent, it will no longer be able to extend and, therefore, cannot accept additional data.
- *Percent Increase*: (*PCTINCREASE*) Specifies the percentage by which the next extent should grow over the previous extent's size. Be careful when setting Percent Increase because it magnifies how an object grows and, therefore, can materially affect available free space in a tablespace.

- *Min Extents*: (*MINEXTENTS*) Specifies the number of extents to allocate when the segment is created. Controls free space fragmentation by making sure that every used or free extent is at least as large as the value you specify.
- *Max Extents*: (*MAXEXTENTS*) Specifies the maximum number of extents that Oracle can allocate to the materialized view. Once this limit is reached, Oracle prevents further growth of the cluster and cannot accept additional data. As a result, you should carefully monitor the number extents already allocated to the table with this limit.

Options page/tab

- *Do you want to register a prebuilt table to the view?* If selected, registers and existing table as a pre-initialized materialized view, which is particularly useful when registering large materialized views in a data warehousing environment.
- *Should the materialized view be immediately filled?* If not selected, the materialized view is populated during the next refresh operation.
- *Should data for the materialized view be cached?* If selected, Oracle puts data accessed frequently at the most recently used end of the list in the buffer cache when a full table scan is performed. This option is useful for small lookup tables.
- *Is the materialized view eligible for query rewrite?* Only enable query rewrite if expressions in the statement are repeatable.
- *How many threads should be in a parallel operation?* Enter an integer to set the default degree of parallelism for queries and DML on the materialized view after creation.
- *Is the materialized view compressed?* If selected, compresses data segments to reduce disk and memory use.

Definition page/tab

Enter or edit a definition for the materialized view. If the target database supports it, ER Studio Data Architect adds this definition as a materialized view comment when generating SQL code.

DDL page/tab

Displays the *CREATE MATERIALIZED VIEW* statement needed to build the materialized view. ER Studio Data Architect uses the platform-specific parser of the model's selected database platform to generate the materialized view.

PreSQL & PostSQL tab

Lets you enter SQL to be applied before or after the *CREATE MATERIALIZED VIEW* statement. The PreSQL and PostSQL scripts entered here are included in the script when you generate the physical database.

Permissions tab

Sets access roles and user permissions for the materialized view. Keep in mind that the more specific permissions are, the more time you may have to spend maintaining and updating them.

The Roles and Users must be previously assigned to appear in the Materialized View Editor. For more information, see Creating and Editing Database Roles and Creating and Editing Database Users.

Attachment Bindings tab

Bind an external piece of information, or attachment to the materialized view. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary. For more information, see Attaching External Documents to the Data Model.

Requirements for the Fast Refresh Method

Materialized View Characteristics	Fast Refresh Method is Supported When the Materialized View has:		
	Only Joins	Joins and Aggregates	Am Aggregate on a Single Table

Detail tables only	X	X	X
Single table only			X
Table appears only once in the *FROM* list	X	X	X
No non-repeating expressions like *SYSDATE* and *ROWNUM*	X	X	X
No references to *RAW* or *LONG RAW*	X	X	X
No *GROUP BY*	X		
Rowids of all detail tables appear in the *SELECT* list of the query	X		
Expressions are allowed in the *GROUP BY* and *SELECT* clauses provided they are the same		X	X
Aggregates allowed but cannot be nested		X	X
AVG with *COUNT*		X	X
SUM with *COUNT*			X
VARIANCE with *COUNT* and *SUM*		X	X
STDDEV with *COUNT* and *SUM*		X	X
WHERE clause contains join predicates which can be ANDed bit not ORed.	X	X	
No *WHERE* clause			X
No *HAVING* or *CONNECT* BY	X	X	X
No subqueries, inline views, or set functions like *UNION* or *MINUS*	X	X	X
COUNT(*) must be present			X
No *MIN* and *MAX* allowed			X
If outer joins, then unique constraints must exist on the join columns of the inner join table	X		
Materialized view logs must exist and contain all columns referenced in the materialized view and have been created with the *LOG NEW VALUES* clause			X

Materialized view logs must exist with rowids of all the detail tables	X		
Non-aggregate expression in *SELECT* and *GROUP BY* must be straight columns			X
DML to detail table	X		X
Direct path data load	X	X	X
ON COMMIT	X		X
ON DEMAND	X	X	X

See Also

- Creating and Editing Materialized Query Tables
- Creating and Editing Database Users
- Attaching External Documents to the Data Model

Creating and Editing Node Groups

A node group is a named set of one or more database partitions. A node group can have 2 to 32 nodes defined for it. The number of nodes that is defined for the node group determines the number of systems across which the database file is created.

You may need to create a node group when you want the database files to be visible across a set of System i models in a load sharing and balancing environment. Nodegroups enable workload to be processed in parallel across a partitioned database for large tables, while smaller tables can be stored on one partition or on a small number of partitions if desired.

The following database platforms support node groups:

- IBM DB2 for LUW 5.x, 6.x, 7.x, 8.x, 9.x, 10.x

Note: The DB2 Nodegroup Wizard and DB2 Nodegroup Editor share the same options, except for Attachment Bindings options which are defined in the editor only.

Create a Node Group

1. In the *Data Model Explorer*, expand the physical *Main Model*, right-click the *Node Groups* node, and then click *New Nodegroup*.
2. *Complete theNode Groups Wizardand then clickFinishto create the* node group.

Tip: Once you have created the auxiliary table, you can edit it by right-clicking the node group you want to change, and then clicking Edit Node Group.

The following helps you complete the options of the *Node Groups Wizard*:

Name page/tab Definition page/tab

DDL page/tab Attachment Bindings tab

Name page/tab

- *What is the name of the NodeGroup?* The name must not begin with SYS or IBM.
- *Enter the nodes (partitions) to include*: Specify the partition number(s). This must be a valid partition number between 0 - 999

Definition page/tab

Enter or edit a definition for the node group. If the target database supports it, ER Studio Data Architect adds this definition as a comment when generating SQL code.

DDL page/tab

Displays the *CREATE NODEGROUP* statement needed to build the node group. ER Studio Data Architect uses the platform-specific parser of the model's selected database platform to generate the node group.

Attachment Bindings tab

Bind an external piece of information, or attachment to the node group. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary. For more information, see Attaching External Documents to the Data Model.

See Also

- Attaching External Documents to the Data Model
- Creating and Editing Object Types

Creating and Editing Object Types

Types define an abstract data type or object composed of a collection of similar types of data. For example, you can create an object type that defines a full address rather than the pieces of an address, such as city, state and postal code in this case the object type stores the pieces of an address as a single type, in the same location and the full address can be accessed and manipulated as single unit rather than multiple units. You could also use an object type to create a purchase order object that includes a method to sum the cost of all purchased items.

Object types are useful for ensuring uniformity and consistency as they are defined as single encapsulated entity that can be reused in other object types and objects. They also offer flexibility by allowing for the creation of objects that represent real-world situations which is limited in relational objects.

Once created, you can use an object type as a datatype in a table.

You can create the following object types:

- *Incomplete*: Specifies no attributes or methods and can be used for circular references such as person - female. It lets the type be referenced before it is complete. Incomplete types can also be used to create types that contain REF attributes to a subtype that has not yet been created. To create such a supertype, first create an incomplete type of the subtype to be referenced and then create the complete subtype after you create the supertype.

A subtype is a specialized version of its direct supertype with an explicit dependency on it. To ensure that subtypes are abandoned after a supertype is dropped, drop all subtypes first; a supertype cannot be dropped until all its subtypes are dropped.

- *Complete*: An incomplete type can be completed by executing a *CREATE TYPE* statement that specifies the attributes and methods.
- *VARRAY*: Used to store small sets of related data. For example, if you have ten offices (each one with a different description) at a particular division in your company, you could create a VARRAY of 10 to hold the details of these offices. The values for a VARRAY type must be fixed and known and be small values because they are stored in RAW format.
- *Nested Table*: Used when data is repeated for the same entity an unknown number of times and storage is a concern.
- A combination of complete, varray, and nested table types.

The following database platforms support object types:

- Oracle 8.x, 9i, 10g, 11g and 12c

Create an Object Type

1. In the *Data Model Explorer*, expand the physical *Main Model*, right-click the *Object Types* node, and then click *New Object Type*.

2. *Complete the tabs of theObject Type Editorand then clickOKto create the object type.*

TIP:

- Once you have created the *object type*, you can edit it by right-clicking the object type you want to change, and then clicking Edit Object Type.
- The Object Type Editor includes find and replace functionality for working with SQL text strings.

The following describe options that require additional explanation:

Header and Body tab

Enter the SQL to create the object type. The SQL generated from each these tabs will appear in their own SQL blocks.

Description tab

Enter or edit a description for the object type.

Attachment Bindings tab

Bind an external piece of information, or attachment to the object type. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary. For more information, see Attaching External Documents to the Data Model.

See Also

- Creating and Editing Relationships
- Creating and Editing Columns
- Attaching External Documents to the Data Model

Creating and Editing Rollback Segments

Rollback Segments allow you to undo, or rollback, DML transactions such as SQL deletes, inserts, and updates that have been executed on a database. A rollback segment entry contains information about the database records before they were modified by a transaction.

The following database platforms support rollback segments:

- Oracle 7.x, 8.x, 9i, 10g, 11g and 12c

Create or Edit a Rollback Segment

1. In the *Data Model Explorer*, expand the Physical *Main Model*, right-click the *Rollback Segments* node, and then click *New Rollback Segment*.
2. *Complete theRollback Segments Wizardand then clickFinishto create the* Rollback Segment.

TIP:: Once you have created the auxiliary table, you can edit it by right-clicking the rollback segment you want to change, and then clicking Edit Rollback Segments.

The following helps you complete the options of the *Rollback Segments Wizard*:

Name tab | Storage tab

Definition tab | DDL tab

Attachment Bindings tab

Name tab

- *Should this rollback segment be made public?* This option only applies to a database operating in a parallel server environment. A public rollback segment can be brought online by any instance of the database in a parallel server environment. A private rollback segment can only be brought online by the database instance that has the rollback segment specified in its initialization file.

Storage tab

- *What extent sizes do you want to assign to this rollback segment?*

The before-image details for multiple transactions can be recorded in one rollback segment but each transaction must fit entirely into one segment and the blocks of a rollback segment are not overwritten until the transaction that wrote to those blocks has been committed. You must therefore, ensure the rollback segment is big enough to hold all the information for the longest running query without wrapping and is big enough to hold the undo information for the *largest transaction* x *number of concurrent transactions* ÷ *number of rollback segments.*

- The default extent size is the size of 5 data blocks. The default minimum extent size is the size of 2 data blocks. The maximum extent size varies by operating system.

- *Optimal Size*: The optimal extent size cannot be less that the space initially allocated for the rolback segment.
- *What are the minimum and maximum number of extents...?* The default minimum number of extents is 2.The default maximum number of extents varies by operating system.

Definition tab

Enter or edit a definition for the rollback segment. If the target database supports it, ER Studio Data Architect adds this definition as a comment when generating SQL code.

DDL tab

Displays the *CREATE ROLLBACK SEGMENT* statement needed to build the rollback segment. ER Studio Data Architect uses the platform-specific parser of the model's selected database platform to generate the rollback segment.

Attachment Bindings tab

Bind an external piece of information, or attachment to the rollback segment. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary. For more information, see Attaching External Documents to the Data Model.

See Also

- Attaching External Documents to the Data Model
- Developing the Physical Model

Creating and Editing Sequences

Sequences are used to generate a series of integer values according to rules defined in the sequence. Database sequences are generally are used to create primary keys; they can also be used to generate random numbers.

The following database platforms support sequences:

- IBM DB2 for LUW 9.x, 10.x
- Oracle 8.x, 9i, 9i, 10g, 11g and 12c
- SQL Server 2012, 2014

**Note:** The *Sequence Wizard* and *Sequence Editor* share the same options, except for Attachment Bindings options which are present only in the editor.

Create or edit a Sequence

1. In the *Data Model Explorer*, expand the Physical *Main Model*, right-click the *Sequences* node, and then select *New Sequence*.
2. *Complete theSequences Wizardand then clickFinishto create the sequence.*

Tip: Once you have created the sequence, you can edit it by right-clicking the sequence you want to change, and then clicking Edit Sequence.

The following helps you complete the options in the *Sequences Wizard*: **Note:** The options available depend on the database platform selected.

Name page/tab Numbers Page/Tab Options page/tab Definition page/tab

DDL page/tab PreSQL & PostSQL page/tab Attachment Bindings page/ tab Permissions page/tab

Name page/tab

- If you have already defined users for the selected database, you can click the list and choose a defined user for the sequence owner. Depending on the platform, you can also select a datatype for the sequence.

Numbers Page/Tab

Lets you specify

- The starting number for the sequence.
- The increment interval value for the sequence.
- The maximum value that can be generated for the sequence and the minimum value for the sequence.

Options page/tab

- *Cycling*: If you have specified a *Maximum Value* on the previous page/tab, you can choose to cycle the values.
- *Ordering* (DB2 and Oracle): Specify whether or not the sequence numbers must be generated in order of request. SQL Server 2012 does not support ordering
- *Cache*: Consider the following performance and application requirements trade-offs when choosing to enable the cache and the optimal cache size:
- Caching sequence numbers enables a range of sequence numbers to be kept in memory for fast access. If the next sequence number can be allocated from cache, number allocation can happen quickly; however, if the next sequence number cannot be allocated from cache, the application may have to wait for I/O operations to complete.
- Not caching sequence values ensures that there is no loss of values in the event of a system failure, shutdown or database deactivation. The values of the sequence are not stored in the cache and every sequence request results in synchronous I/O to the log.

Definition page/tab

Enter or edit a definition for the sequence. If the target database supports it, ER Studio Data Architect adds this definition as a comment when generating SQL code.

DDL page/tab

Displays the CREATE SEQUENCE statement needed to build the sequence. ER Studio Data Architect uses the platform-specific parser of the model's selected database platform to generate the sequence.

PreSQL & PostSQL page/tab

Lets you enter SQL to be applied before or after the *CREATE SEQUENCE* statement. If you select Generate, the PreSQL and PostSQL scripts entered here are included in the script when you generate the physical database.

Attachment Bindings page/ tab

Bind an external piece of information, or attachment to the sequence. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary. For more information, see Attaching External Documents to the Data Model.

Permissions page/tab

Sets access roles and user permissions for the sequence. Keep in mind that the more specific permissions are, the more time you may have to spend maintaining and updating them. The

Roles and Users must be previously assigned to appear in the Sequence Editor. For more information, see Granting and Revoking Permissions to Modify Database Objects.

See Also

- Attaching External Documents to the Data Model
- Creating and Editing Tables
- Granting and Revoking Permissions to Modify Database Objects

Creating and Editing StoGroups

A StoGroup is a set of volumes on direct access storage devices (DASD) that hold the data sets in which tables and indexes are stored. The *CREATE STOGROUP* statement creates a storage group on the current server. You can then allocate storage from the identified volumes for table and index spaces. Different parts of a single database can be stored in different StoGroups.

If a dataset reaches its maximum number of extents, any transaction that writes data to tables or indexes associated with that stogroup fails and the database can become inaccessible until more space is allocated to the stogroup. It is important to allocate sufficient storage for the database and to monitor its growth carefully. For information on capacity planning, see Planning for and Predicting Database Growth.

You can use stogroup you create to define storage for the index of an auxiliary table or a table, or storage for a tablespace. For more information, see Creating and Editing Auxiliary Tables, Defining Index Storage, and Creating and Editing Tablespaces.

The following database platforms support StoGroups:

- IBM DB2 for OS/390 5.x, 6.x, 7.x, 8.x, 9.x, and 10.x
- The StoGroups Wizard and StoGroups Editor share the same options, except for Attachment Bindings options which are present only in the editor.

Create a StoGroup

1. In the *Data Model Explorer*, expand the Physical *Main Model*, right-click the *Entities* node, and then click *New Stogroup*.
2. *Complete theStoGroup Wizardand then clickFinishto create the* stogroup.
3. Once you have created the stogroup, you can edit it by right-clicking the stogroup you want to change, and then clicking Edit StoGroup.

The following help you complete the *StoGroup Wizard* options:

Options page/tab Definition page/tab

DDL page/tab Attachment Bindings tab

Options page/tab

What is the name of the Stogroup? The stogroup name must identify a storage group that does not yet exist at the current server.

- *VCAT*: Specify the integrated catalog facility catalog for the storage group. Specify an alias if the name of the VCAT is longer than 8 characters.
- *Enter the volumes to include in the Stogroup, separated by commas*: Includes the specified volumes in the StoGroup. To include all volumes in the stogroup, enter an asterisk (*).

Definition page/tab

Enter or edit a definition for the stogroup. If the target database supports it, ER Studio Data Architect adds this definition as a comment when generating SQL code.

DDL page/tab

Displays the *CREATE STOGROUP* statement needed to build the stogroup.ER Studio Data Architect uses the platform-specific parser of the model's selected database platform to generate the stogroup.

Attachment Bindings tab

Bind an external piece of information, or attachment to the stogroup. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary. For more information, see Attaching External Documents to the Data Model.

See Also

- Planning for and Predicting Database Growth
- Developing the Physical Model
- Creating and Editing Auxiliary Tables

Creating and Editing Synonyms

A synonym is an alternate name for a database object that lets you:

- Reference an object without specifying its owner (The synonym definition references the owner, but to use the synonym thereafter you do not have to know the owner name.)
- Reference a remote object without specifying its database.
- Alias an object so its purpose becomes more obvious.

The following database platforms support Synonyms:

- IBM DB2 for OS/390 5.x and 6.x and for z/OS 7.x, 8.x, 9.x, and 10x
- Microsoft SQL Server 2005, 2008, 2012, 2014
- Netezza 4.6 and 5.0
- Oracle 8.x, 9i, 10g, 11g and 12c


Note: The Synonym Wizard and Synonym Editor share the same options, except for Attachment Bindings options which are present only in the editor.

Create a Synonym

1. In the *Data Model Explorer*, expand the Physical *Main Model*, right-click the *Synonyms* node, and then select *New Synonym*.
2. *Complete theSynonyms Wizardand then clickFinishto create the synonym.*

Tip: Once you have created the *synonym*, you can edit it by right-clicking the *synonym* you want to change, and then clicking Edit Synonym.

The following helps you complete the options of the *Synonyms Wizard*:

Name page/tab Reference page/tab

Definition page/tab DDL page/tab

Attachment Bindings tab

- The options available depend on the database platform selected.

Name page/tab

- *Make this synonym accessible to all users?* If selected, creates a *public synonym* that can be used by everyone in the database, otherwise a *private synonym* is created that belongs to the synonym owner.

Reference page/tab

- *What type of object does this synonym reference?* The list of objects available depends on the selected database platform.
- IBM DB2 for OS/390 5.x and 6.x and for z/OS 7.x, 8.x, 9.x, and 10.x: Supports synonyms for aliases, synonyms, tables, unknown, and views.
- Microsoft SQL Server 2005 and 2008: Supports synonyms for functions, procedures, and views.
- Microsoft SQL Server 2012, 2014: Supports synonyms for sequences.
- Netezza 4.6 and 5.0: Supports synonyms for tables and views.
- Oracle 8.x, 9i, 10g, 11g and 12c: Supports synonyms for functions, materialized views, object types, packages, procedures, sequences, synonyms, tables, and views.
- If you specify Unknown, the object type will change to correspond to the type of object named below.
- *What is the schema of the referenced object? What is the name of the referenced object?* The list is populated with the names of all the objects in the model matching the model type specified as the type of object. But you can also enter your own value in this field.
- *Who is the owner of the referenced object? What is the name of the referenced object?* The list is populated with the names of all the objects in the model matching the model type specified as the type of object. But you can also enter your own value in this field.
- *If the object is on a remote database, please enter the DB link.* Type the path of the remote database, or click the list and select it.

Definition page/tab

Enter or edit a definition for the synonym. If the target database supports it, ER Studio Data Architect adds this definition as a comment when generating SQL code.

DDL page/tab

Displays the *CREATE SYNONYM* statement needed to build the synonym. In physical models, ER Studio Data Architect uses the platform-specific parser of the model's selected database platform to generate the view. If you're adding a view to a logical data model, the default parser AINSI SQL is used unless you change the View Parser in the Logical Model Options to another parser. For more information, see Defining Model Options for the Selected Model.

Attachment Bindings tab

The Attachment Bindings tab is available in the Synonym Editor. Bind an external piece of information, or attachment to the synonym. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value

Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary. For more information, see Attaching External Documents to the Data Model.

See Also

- Defining Model Options for the Selected Model
- Attaching External Documents to the Data Model

Creating and Editing Tablespaces

A tablespace is the physical location where the data in a database is stored. Using tablespaces, the administrator can control disk layout to optimize database performance, by for example storing a frequently used index on a fast disk. The manner in which space is allocated to database objects directly impacts their ability to grow, which can ultimately affect database performance.

The following database platforms support tablespaces:

- IBM DB2 for LUW 5.x, 6.x, 7.x, 8.x, 9.x, 10.x
- IBM DB2 for OS/390 5.x and 6.x and for z/OS 7.x, 8.x, 9.x, and 10.x
- Oracle 7.x, 8.x, 9i, 10g, 11g and 12c
- The Tablespace Wizard and Tablespace Editor share the same options, except for Attachment Bindings options which are present only in the editor.

Create a TableSpace

1. In the *Data Model Explorer*, expand the Physical *Main Model*, right-click the *Tablespaces* node, and then click *New Tablespace*.
2. Complete the *Tablespace Wizard* and then click *Finish* to create the alias.
3. Once you have created the tablespace, you can edit it by right-clicking the tablespace you want to change, and then selecting Edit Tablespace.

The following helps you complete the options of the *Tablespace Wizard*:

Name & Type page/tab
Name page/tab
Datafile page/tab
Options page/tab
Options 1 page/tab
Options 2 page/tab
Definition page/tab
DDL page/tab
Attachment Bindings page/ tab

The options available depend on the database platform selected.

- Many of the table space parameters are also used when defining table storage. For information on defining storage parameters not described here, see Defining Table Storage.

Name & Type page/tab

- *Choose the type of data to store in the tablespace*:
- *Regular*: Choose for all data types except for temporary tables.

- *Long*: Choose for long or LOB tables columns, or for structured type columns or index data. The table space must be a DMS table space.
- *Temporary*: Choose if you want to store temporary data. A database must have at least one temporary table space to store temporary tables and another that allows the declaration of temporary tables.
- *Universal (Range-partitioned)*: A range-partitioned tablespace is segmented and based on partitioning ranges. It contains a single table and you can create any type of index on tables in the range-partitioned tablespace. If you specify this option, you must also specify the Segment Size and the number of partitions. You can either accept the default number of rows per page or specify a different number.

Name page/tab

- *Do you want space management to be performed through the dictionary or locally in the tablespace?* Specify how extents should be managed.
- *Dictionary Managed*: Relies on SQL dictionary tables to track space utilization. Adding an extent to a table requires the database manager to find free space, mark it as non-free and potentially causes rollback segments to extend. These operations may jeopardize database performance because they cannot be performed simultaneously. However, dictionary managed tablespaces are very flexible, different objects sharing the same tablespace can have different extent sizes.
- *Locally Managed*: Eliminates the problem of tablespace fragmentation, and potential bottlenecks associated with the data dictionary. The database manager maintains a bitmap of free and used blocks, or sets of blocks. Locally managed tablespaces can be used for temporary tablespaces and can often sustain high extent-allocation and deallocation activities. If a table is created with the wrong extent size, causing tablespace fragmentation, this type of tablespace management can enforce extent uniformity at the tablespace level.
- *Disable logging when schema/data is modified*: If selected, disables logging when schema or data is modified.
- *Place tablespace offline after creation*: If selected, places tablespace offline after creation.
- *AutoAllocate Extents*: If selected, Oracle automatically assigns and allocates appropriately sized extents depending on the objects in the tablespace.
- *Uniform Extent Size*: If selected, extent sizes are managed at the tablespace level, not at the object level. The extent size is specified when the tablespace is created and all the extents for objects created in the tablespace will be the same.The uniform extent size cannot be overridden when a schema object, such as a table or an index, is created. This option may eliminate fragmentation.

Datafile page/tab

- *Defined Datafiles*: Double-click the Size or Unit fields to edit them. Click Add only when you need to define another datafile.

- *Reuse Existing File*: If *Use Oracle Managed Files to automatically generate Defaults* is not enabled, this option is available. If selected, lets you reuse an existing datafile, which the database manager overwrites the first time the file is accessed.

Storage page/tab

- If the table space of an Oracle table space is dictionary managed, as specified on the Name tab of the editor, then define the parameters of the extents on this tab.
- *Specify the containers for the tablespace below*: Specify an absolute or relative directory name. The relative path is relative to the database directory. If the directory name does not exist, it will be created.
- Click Add only if you want to specify another container.
- *Select a NodeGroup to assign to the tablespace*: Specify either a nodegroup or a database partition number.
- *Select a page size*: The page size must be the same as the page size of the buffer pool that is associated with the database.

Options page/tab

- *Extent Size*: Specify the number of pages that will be written to a container before writing to the next container. The database manager cycles through the containers as it stores data. The default extent size is specified by the database configuration parameter *DFT_EXTENT_SZ*.
- *Prefetch Size*: Specify the number of pages that will be read when prefetching tablespace data. The default prefetch size is specified by the database configuration parameter *DFT_PREFETCH_SZ*.
- *Overhead*: Specifies the overhead, disk seek, and latency time of the I/O controller. This should be an average for all containers belonging to the tablespace. This value is used during query optimization to determine the I/O cost.
- *Transfer Rate*: Specifies the time required to read one page into memory. This should be an average for all containers belonging to the tablespace. This value is used during query optimization to determine the I/O cost.
- *Select the recovery status*: If *ON* is selected, dropped tables in this tablespace may be recovered using the *ROLLFORWARD* command with the *RECOVER TABLE ON* option.

Options 1 page/tab

- *GBP Cache*: Specifies a group buffer pool cache.
- *Sizing*: Specifies the settings for the different Sizing parameters:
- If you are creating a non-partitioned tablespace, enter a value between 1 and 255 in the *Max rows per page* field.

- If you are creating a partitioned tablespace, enter the number of partitions you want to create for the tablespace (between 1 and 254) in the *Number of Partitions* field, then select a size for the partitions from the Partition Size (*DSSIZE*) list. Enter a value between 1 and 255 in the *Max rows per page* field.
- If you are creating a segmented tablespace, select a value from the Segment Size list, and enter a value between 1 and 255 in the Max rows per page field.
- If you are creating an LOB tablespace, select a value from the *Partition Size* (*DSSIZE*) list.
- *TRACKMOD*: When set to Yes, the database manager tracks database modifications to enable the backup utility to detect which subsets of the database pages should be examined by an incremental backup and included in the backup image if necessary.

Options 2 page/tab

- *Locking*: To control concurrent access and prevent uncontrolled data access, you can create locks which associates a database manager resource with an application, called the lock owner, to control how other applications access the same resource. The Lock Size determines the granularity of the locks.
- *Maximum Locks*: Set this to a low number to avoid contention. Setting *LOCKMAX* to zero increases tablespace availability by preventing lock escalations which can lock the tablespace. Setting *LOCKMAX* to *SYSTEM* allows the database manager to control the maximum number of locks by a setting in the database configuration file.
- *Close Rule*: Determines whether the data sets can be closed when the table space is not in use, and the limit on the number of open data sets (*DSMAX*) is reached. If you select No, no data sets can be closed. However, if *DSMAX* is reached and no *CLOSE YES* page sets exist, then even *CLOSE NO* page sets will be closed.
- *Partitions*: For information on partition parameters, see Partitioning a Table.

Definition page/tab

Enter or edit a definition for the tablespace. If the target database supports it, ER Studio Data Architect adds this definition as a comment when generating SQL code.

DDL page/tab

Displays the *CREATE TABLESPACE* statement needed to build the tablespace. ER Studio Data Architect uses the platform-specific parser of the model's selected database platform to generate the tablespace.

Attachment Bindings page/ tab

Bind an external piece of information, or attachment to the tablespace. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype.

Attachments are created in the Attachments folder of the Data Dictionary. For more information, see Attaching External Documents to the Data Model.

See Also

- Creating and Editing Database Dependent Objects
- Developing the Physical Model

Creating and Editing Database Schemas

A database schema is a collection of unique objects in a namespace. Schemas allow you to group tables and other related objects together, such as a table and an index. Once the schema has been defined you can

The following database platforms support Schemas:

- IBM DB2 for LUW 5.x, 6.x, 7.x, 8.x, 9.x, and 10.x
- Microsoft SQL Server 2005 - 2014
- The Database Schema Wizard and Database Schema Editor share the same options, except for Attachment Bindings options which are present only in the editor.

Create a Database Schema

1. In the *Data Model Explorer*, expand the Physical *Main Model*, right-click the *Schema* node and then click *New Database Schema*.
2. Complete the *Database Schema Wizard* and then click *Finish* to create the database schema.
3. Once you have created the schema, you can edit it by right-clicking the schema you want to change, and then selecting Edit Schema.

The following helps you complete the options of the *Database Schema Wizard*:

Object Privilege page/tab Definition page/tab

DDL page/tab Attachment Bindings tab

Object Privilege page/tab

Sets access roles and user permissions for the schema. Keep in mind that the more specific permissions are, the more time you may have to spend maintaining and updating them. The Roles and Users must be previously assigned to appear in the Sequence Editor. For more information, see Granting and Revoking Permissions to Modify Database Objects.

Definition page/tab

Enter or edit a definition for the schema. If the target database supports it, ER Studio Data Architect adds this definition as a comment when generating SQL code.

DDL page/tab

Displays the *CREATE SCHEMA* statement needed to build the schema. ER Studio Data Architect uses the platform-specific parser of the model's selected database platform to generate the tablespace.

Attachment Bindings tab

Bind an external piece of information, or attachment to the schema. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the *Selected Attachments* grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary. For more information, see Attaching External Documents to the Data Model.

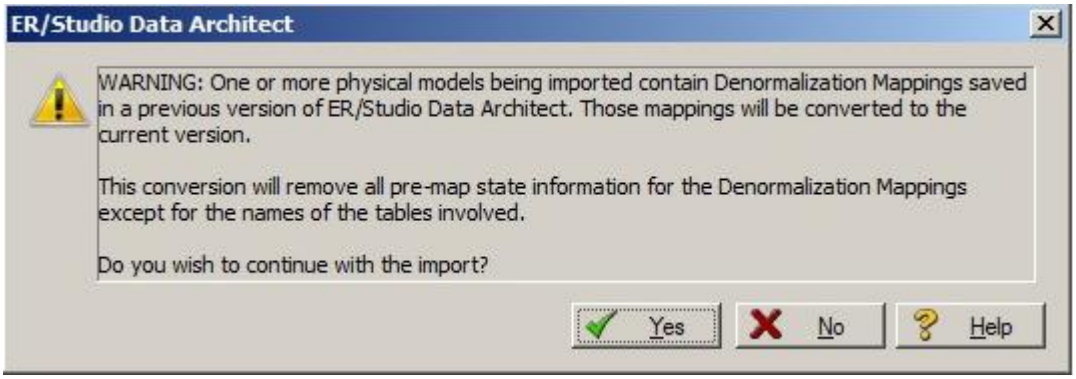
See Also

- Granting and Revoking Permissions to Modify Database Objects
- Attaching External Documents to the Data Model

Upgrading Diagrams with Denormalization Mappings

Instead of showing Denormalizations in the Data Model Tree, denormalizations such as Horizontal and Vertical Splits, and Rollups and Rolldowns, are maintained with the table data and appear on the Where Used tab of the Table Editor. From the Where Used tab you can see the denormalization history of the table.

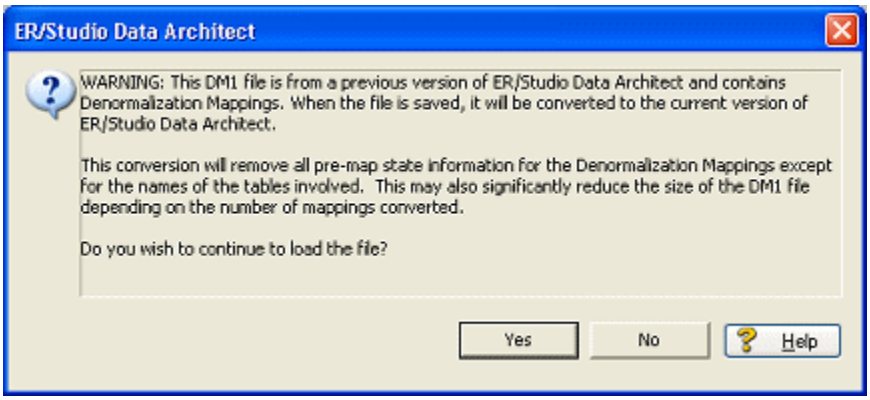
Since the denormalization feature has changed in 9.0, any documents containing denormalization mappings will be upgraded to use the new features of ER Studio Data Architect 9.0. For example, when you import a physical model from a file that uses the denormalization mappings from earlier versions of ER/Studio, you will receive the following warning message:



Denormalization mappings are now retained with the table data and can be viewed and edited on the Where Used tab of the Table Editor. Denormalization history, including Columns, Indexes, and Relationships of pre-map tables, is no longer retained, so the undo and make permanent functions specifically for denormalizations are no longer available. Denormalization mappings have been worked into the normal undo function of the product allowing them to be undone like other operations. However, the names of the pre-map tables are retained. Retaining only essential pre-map information means that the DM1 file with denormalizations is smaller and therefore opens faster.

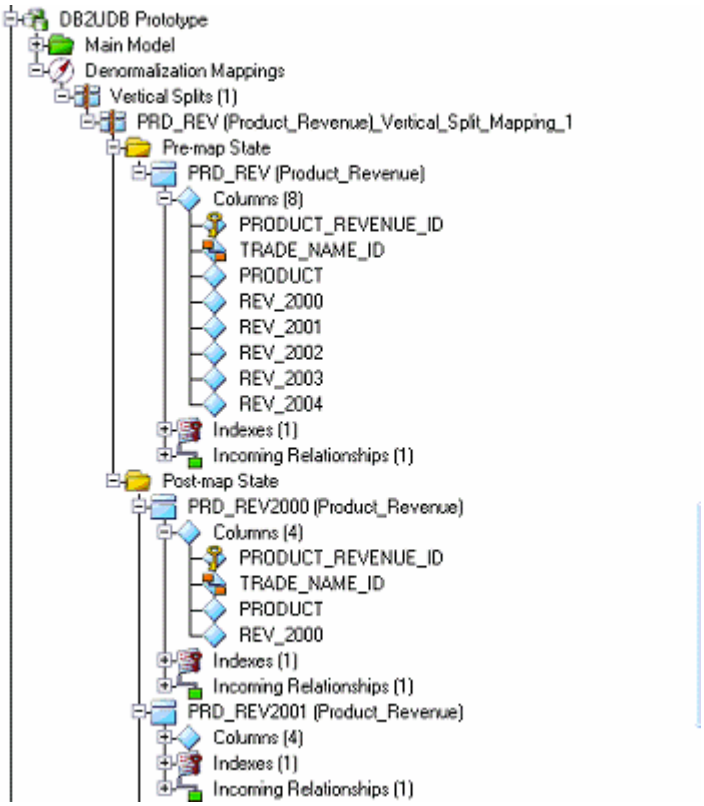
For more information, see Optimizing Query Performance on the Physical Model (Denormalizing the Physical Model).

When upgrading version 8.5 and earlier dm1 files to version 9.0 and higher, you will see the following message when you attempt to open a diagram that contains denormalization mappings.

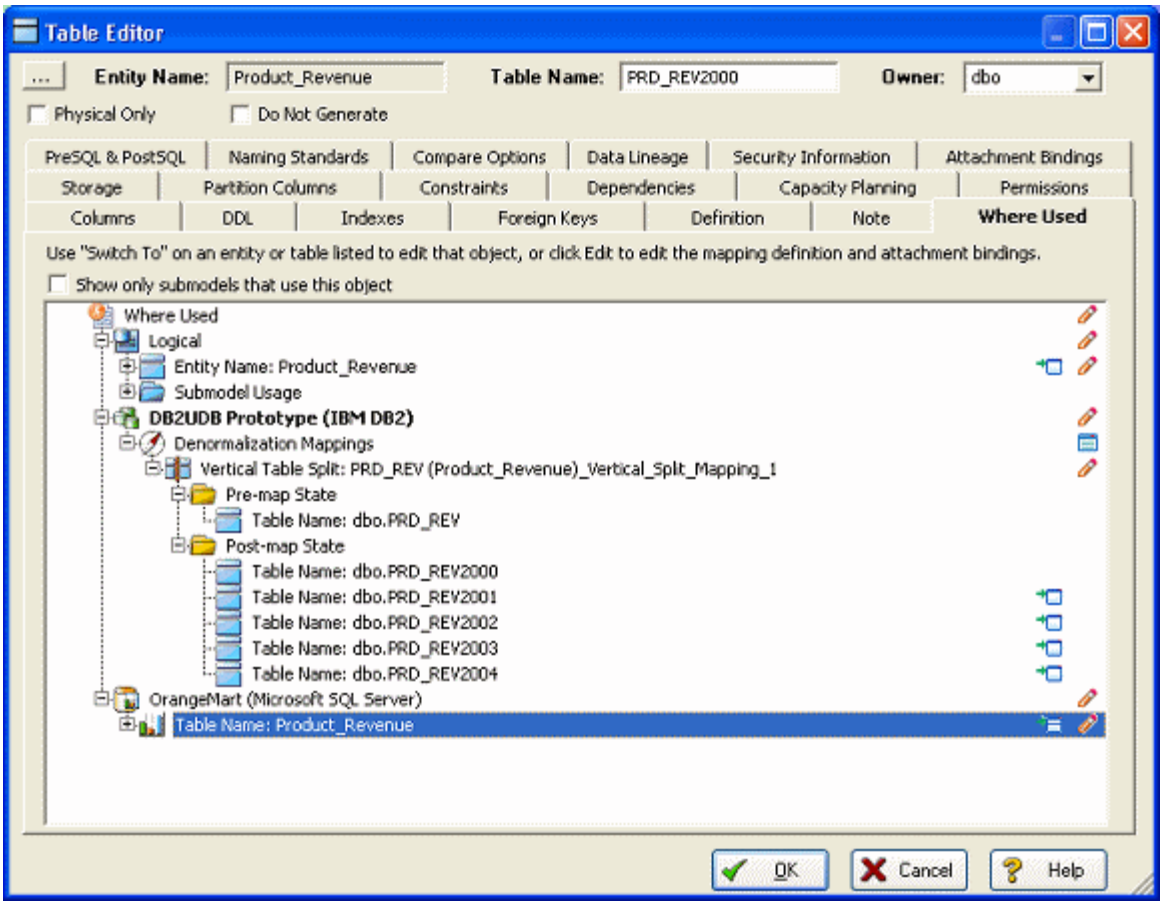


A similar message displays when you run the Diagram Data Version Upgrade utility on the Repository.

Converting the file to 9.0 changes the denormalization information from something similar to the following in version 8.5.



to the following in 9.0 and higher.



See Also

- Optimizing Query Performance on the Physical Model (Denormalizing the Physical Model)
- Associate Similar Objects Using User-Defined Mappings

Optimizing Query Performance on the Physical Model (Denormalizing the Physical Model)

No matter how elegant a logical design is on paper, it often breaks down in practice because of the complex or expensive queries required to make it work. Sometimes, the best remedy for the performance problems is to depart from the logical design. Denormalization is perhaps the most important reason for separating logical and physical designs, because then you do not need to compromise your logical design to address real-world performance issues.

Here are some common scenarios you should consider when denormalizing a physical design:

- *Duplicated Non-Key Columns*: You may decide to duplicate non-key columns in other tables to eliminate the need to access, or join to that table in a query, such as when you duplicate the description or name column of a lookup table. When duplicating non-key columns in tables, ensure that the procedural logic synchronizes the column data so that it does not differ from the values in the reference table.
- *Horizontal Table Splits*: When tables have many rows of data, you may want to partition the data by splitting the table into multiple tables where each table uses the same schema but stores a different range of primary key values. Splitting the table reduces the size and density of the indexes used to retrieve data, thereby improving their efficiency. On the other hand, once the table is split you must access data in multiple tables instead of in a single table which requires more complicated procedural logic to manage the tables.
- *Vertical Table Splits*: When tables with many columns have high read/write activity, you may want to split the table into multiple tables with different column sets, where each column set uses the same primary key. This lets you spread update activity across multiple tables, thus decreasing the impact of maintaining multiple indexes. As with horizontal table splits, vertical table splits require more complex procedural logic to manage multiple tables.

After generating a physical model from a logical model, you can use denormalization mappings to improve performance in the physical model implementation.

Unlike a manual edit of the physical model, a denormalization mapping provides a record of what has been changed and retains the link back to the logical model (visible in the Table Editor Where Used tab). This makes the denormalization self-documenting, and reduces the manual labor required when merging with an updated logical model. Also, a mapping can be undone at any time (see Synchronizing Physical and Logical Models).

The following denormalization mappings are available:

- About Roll-Up Denormalization: Remove one or more unnecessary child tables, consolidating their columns into the parent table.

- About Roll Down Denormalization: Remove an unnecessary parent table, copying all of its non-primary-key columns into one or more of its child tables. The child tables can be further consolidated via a Table Merge Denormalization.
- About Horizontal Split Denormalization: Partition a table into two or more tables with identical columns.
- About Vertical Split Denormalization: Remove a table and distribute its columns among two or more new tables. The new tables have the same primary keys as the original; each of the other columns can appear in one, several, or all of the new tables, or can be dropped.
- About Column Map Denormalization: Copy a column from one table to another to improve query performance by eliminating the need for a join.
- About Table Merge Denormalization: Remove unnecessary tables by combining two or more unrelated tables with the same primary key into a single table containing both tables' columns.

Removing and Consolidating Child Columns

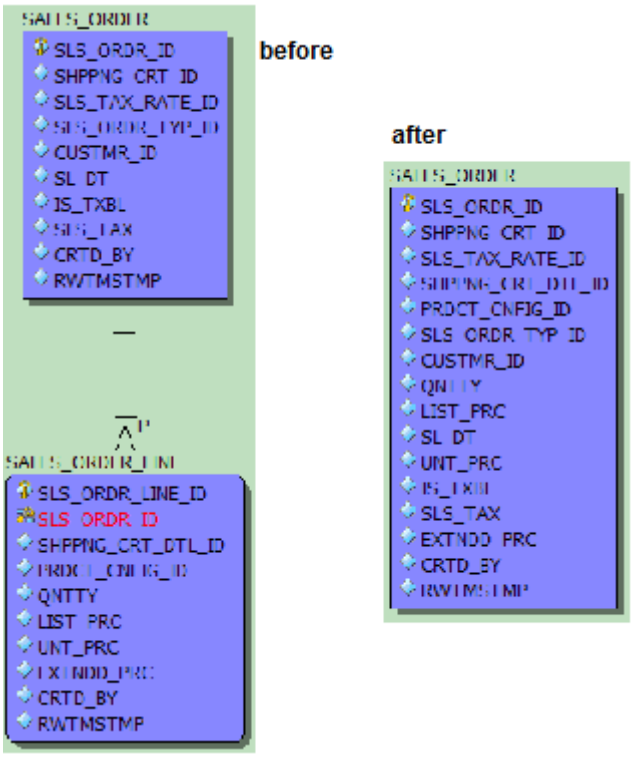
This sections contains the following topics:

- About Roll-Up Denormalization
- Create a New Roll-up Mapping

About Roll-Up Denormalization

A roll-up denormalization mapping removes one or more unnecessary child tables from a physical model, consolidating their columns into the parent table. For a roll-up operation, the PARENT primary key is preserved. Any native PK columns in the child are discarded.

In the following example, Sales_Order_Line has been rolled up into Sales_Order, all of the columns of Sales_Order_Line have been consumed by Sales_Order, and only the parent's key remains.



Create a New Roll-up Mapping

1. Display the physical model.
2. Click and drag or *CTRL-click* to select the parent and child tables.
3. Right-click one of the selected tables and then choose Denormalization Mapping > Rollups.
4. Complete the *Roll Up Denormalization Wizard* as required.

Note:

- The Convert Subtype Relationships to Recursive Relationships check box allows you to preserve relationships between subtypes when rolling up subtypes into a supertype. The relationships will become recursive. The rolenames in the subtypes will be used in the supertype. Otherwise this page is merely informative. If the correct tables are listed, click Next; otherwise, click Cancel and redo your selection.
- If the Rollups menu choice is grayed, check to make sure you have not accidentally selected another table or some other object.
- When appropriate, child columns can be repeated in the unified table. For example, if a parent-child relationship normalizes month-specific data using a cardinality of 1:12, the wizard would by default repeat the child table's columns 12 times in the unified table. The repeating columns are named by appending numeric suffixes to the original column name; you should rename them appropriately after creating the mapping.

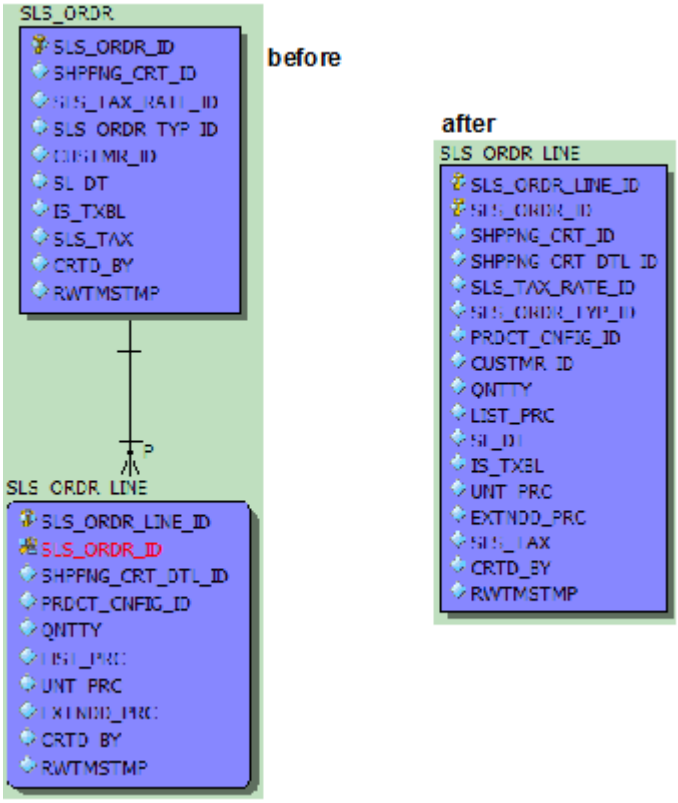
Removing Unnecessary Parent Tables

This section contains the following topics:

- About Roll Down Denormalization
- Create a New Roll-Down Mapping

About Roll Down Denormalization

A roll-down denormalization mapping removes an unnecessary parent table, copying all of its non-primary-key columns into one or more of its child tables. In the following example, the SLS_ORDR table has been rolled down into the SLS_ORDR_LINE table. The child will keep the compound key from the propagating SLSORDRID identifying relationship. For a Rolldown operation, the CHILD primary key is preserved. Any native PK columns in the child are kept.



Create a New Roll-Down Mapping

1. Display the physical model.
2. Click and drag or *CTRL-click* to select the parent and child tables.
3. Right-click one of the selected tables, then from the pop-up menu choose *Denormalization Mapping > Rolldowns*.

4. Complete the *Roll Down Denormalization Wizard* as required.

Note:

- If the Rolldowns menu choice is dimmed, check to make sure you have not accidentally selected another table or some other object.
- The first panel of the wizard is merely informative. If the correct tables are listed, click Next; otherwise, click Cancel and redo your selection.
- If appropriate, the denormalized child tables can be further consolidated via a About Table Merge Denormalization.

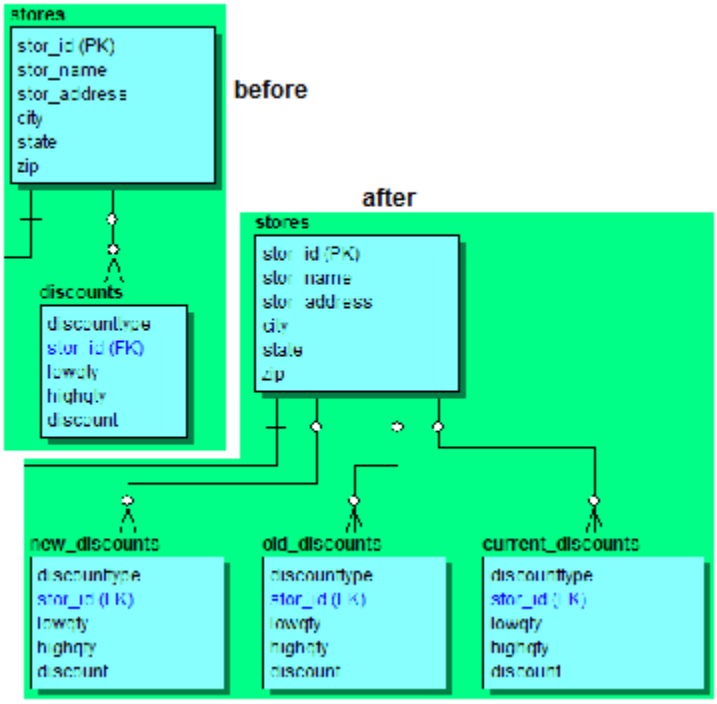
Partitioning a Table and Duplicating Columns

This section contains the following topics:

- About Horizontal Split Denormalization
- Create a New Horizontal Split Mapping

About Horizontal Split Denormalization

A horizontal split denormalization mapping partitions a table into two or more tables with identical columns. In the following example, a single discounts table is split into three separate tables



Create a New Horizontal Split Mapping

1. Display the physical model.
2. Right-click the table to be partitioned and then choose *Denormalization Mapping* > *Horizontal Splits*.
3. Complete the *Horizontal Table Split Wizard* as required.

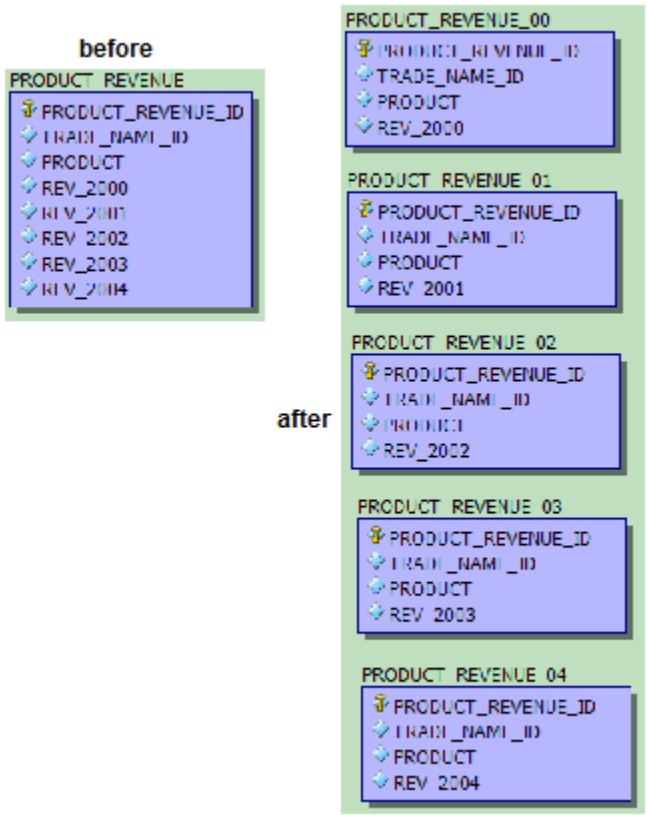
Replacing a Table with New Tables That Share the Primary Key

This section contains the following topics:

- About Vertical Split Denormalization
- Create a Vertical Split Mapping

About Vertical Split Denormalization

A vertical split denormalization mapping removes a table and distributes its columns among two or more new tables. The new tables have the same primary keys as the original; each of the other columns can appear in one, several, or all of the new tables, or can be dropped. In the following example, a table with separate columns for each of five years' revenues, is split into five tables, one for each year.



Create a Vertical Split Mapping

1. Display the physical model.
2. Right-click the table to be split and then choose *Denormalization Mapping* > *Vertical Splits*.
3. Complete the *Vertical Table Split Wizard* as required.
4. When apportioning columns, you can select multiple columns using Shift-Click or CTRL-Click, then drag them all at once to one of the new tables. The columns remain selected, so if appropriate you can drag the same set to another new table.

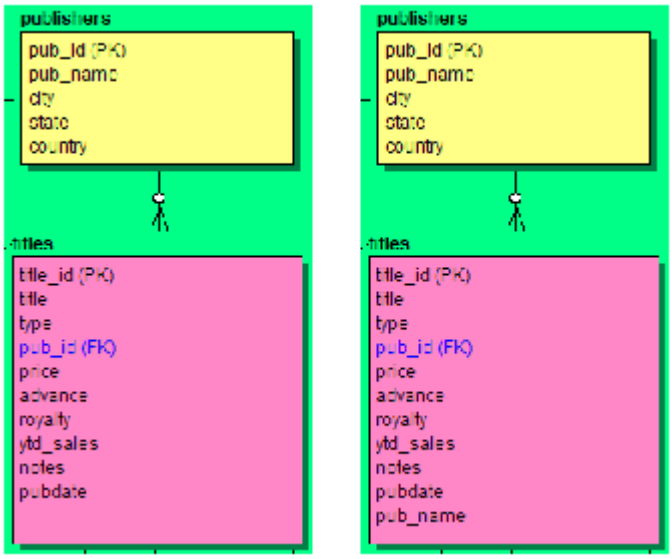
Eliminating Joins by Copying Columns

This section contains the following topics:

- About Column Map Denormalization
- Create a New Column Map Mapping

About Column Map Denormalization

A column map denormalization mapping copies a column from one table to another to improve query performance by eliminating the need for a join. In the following example, the pubname column is copied into the titles table, eliminating the need for a join when querying for the name of a book's publisher.



Create a New Column Map Mapping

1. In the *Data Model* tab of the *Data Model Explorer*, right-click the column you want to map and then choose *Denormalization Mapping* > *Column Mappings*.
2. Complete the *Column Mapping Wizard* as required.

3. The first panel of the wizard is merely informative. If the correct column is listed, click *Next*; otherwise, click *Cancel* and redo your selection.

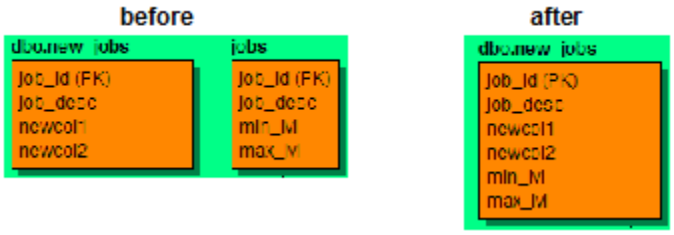
Removing Unnecessary Tables

This section contains the following topics:

- About Table Merge Denormalization
- Create a New Table Merge Mapping

About Table Merge Denormalization

A table merge denormalization removes unnecessary tables by combining two or more unrelated tables with the same primary key into a single table containing all tables' columns.In the following example, all the fields from the jobs table are merged into the dbo.newjobs table.



Create a New Table Merge Mapping

1. Display the physical model.
2. Click and drag or *CTRL-click* to select the tables to be merged.
3. Right-click one of the selected tables and then choose Denormalization Mapping > Table Merges.
4. Complete the *Table Merge Denormalization Wizard* as required.

Note:

- If the Table Merges menu choice is dimmed, check to make sure you have not accidentally selected another table or some other object.
- The first panel of the wizard is merely informative. If the correct tables are listed, click *Next*; otherwise, click *Cancel* and redo your selection.

See Also

- Physical Storage Considerations
- Planning for and Predicting Database Growth

Synchronizing Physical and Logical Models

After normalizing or denormalizing the model you may want to synchronize it with the real-world database. Using the Submodel Synchronization Utility, you can see the differences found and decide which changes you want to change. For more information, see Synchronizing Submodels.

See Also

- Synchronizing Submodels
- Synchronizing Comments

Planning for and Predicting Database Growth

In order to ensure optimal database performance, it is important that you assess your storage needs and plan accordingly. This section will help you to do this and is comprised of the following topics:

- Planning for Table and Index Storage
- Predicting Table Growth
- Predicting Database Growth
- Reporting Database Growth Analysis

Planning for Table and Index Storage

Planning the storage of tables and indexes is best done during the physical design stage in order to improve performance and to streamline data administration tasks. When planning physical storage, carefully consider both the placement and size of tables and indexes. The performance of almost all database applications is I/O bound. To improve I/O throughput, physically separate tables that are frequently joined together. Also, separate tables from their indexes. The objective is to have the database read or write data in parallel as much as possible.

Two key concerns of every database administrator are free space management and data fragmentation. If you do not properly plan for the volume and growth of your tables and indexes, these two administrative issues could severely impact system availability and performance. Therefore, when designing the physical model, consider the initial extent size and logical partition size.It

As a starting point, estimate the size of each table and its indexes based on a projected row count. The Capacity Planning tool helps project the size of the databases given certain growth parameters. For databases that let you specify initial extent sizes, such as Oracle, set the initial extent size to the estimated size in order to avoid data fragmentation as the table grows. By keeping tables within a single extent, data access times decreases and table reorganization can be determined.

Once you have determined the placement and sizes of the tables and indexes in your database, estimate the total size requirements of the database in order to avoid running out of free space for the database.

Predicting Table Growth

Capacity planning metrics can be recorded for each table so that proper sizing can be done before the table is implemented in a database.

The Capacity Planning tab of the Table Editor helps predict change using metrics such as table row count, table growth rate, growth type, growth period and table maximum size.

1. In the *Data Model Explorer*, double-click a table in the physical model.

2. Click and then complete the *Capacity Planning* tab.

Tip: Change the column Avg. Width and Percent Null values to more accurately predict the table growth.

The growth predictions you make in the Table Editor display in the Capacity Planning utility and can be edited there.

Predicting Database Growth

Using the Capacity Planning utility, you can forecast storage requirements of newly implemented or existing database systems so budgeting or engineering can be accounted for well in advance.

1. In the *Data Model Explorer,* select and then right-click a physical model.
2. Click and then complete the Capacity Planning utility.

The following describe options that require additional explanation:

- *Sizing Options* tab: *Index Options*: Select any of the following options: Primary Key (PK), Foreign Key (FK), Alternate Key (AK), and Inversion Entry (IE).
- *Growth Parameters* tab: Specifies overhead elements that contribute to database growth. The amount of overhead required depends on the DBMS platform and the storage options you have selected for the database, such as the minimum percentage of a data block to reserve as free space, and options for logging, auditing, and data recovery. The values provided here are reflected in the Projected Size column of the Growth Analysis tab.
    - *Row Overhead*: Tailors the formulae to account for any overhead used by the chosen DBMS platforms of the physical model to store each record. This number represents the number of bytes of overhead the DBMS may use to store records. This should be set to 0 initially, but you can specify any number of bytes to add on to the row size.
    - *Table Overhead*: Accounts for overhead to store data for a table. This is a multiplier applied to table size. A value of 1.2 corresponds to 20% overhead. The default value of 1 indicates no overhead.
    - *Index Overhead*: Accounts for overhead to store indexes. The number of bytes is added to the indexed column sizes. For example, in Oracle the index overhead is 6 bytes, so 6 is added to the sum of the indexed column sizes.
    - *Blob Overhead*: Accounts for overhead to store BLOB columns. Similar to Table Overhead, this is a multiplier applied to BLOB storage. A value of 1.2 corresponds to 20% overhead. 1 indicates no overhead.

Blob BlockSize: Enter an estimate of the average size of BLOB columns. This will be used as the average size of each BLOB column.

Note:

- Changes you make for a table on the Sizing Options tab are automatically reflected in the Sizing Estimates, which is also displayed this tab.
- Changes made to all Sizing Options and Growth Parameters are reflected on the Growth Analysis tab.
- The capacity predictions you make in the Capacity Planning utility display in the Table Editor and can also be edited there.

Reporting Database Growth Analysis

From the Capacity Planning Utility you can export the metrics produced in formats such as RTF or CSV, so, for example, you can draft your own reports for the growth patterns of a physical model.

1. In the Data Model Explorer, select and then right-click a physical model.
2. Click *Capacity Planning* and then complete the *Sizing Options* and *Growth Parameters* of the Capacity Planning utility.
3. Click the *Growth Analysis* tab and select the *Report Type* and *Report Options* desired.

A preview of the report contents appears in the Output area.

See Also

- Physical Storage Considerations
- Defining Index Storage

Connecting to Database Sources and Targets

In ER Studio Data Architect, you can use the Reverse Engineer Wizard to connect to a database and then create a new data model based on an existing database. You can also connect to a database using the DDL Generation Wizard and automatically generate the SQL source code to build the database. In ER Studio Data Architect you also connect to databases using the Model Compare and Merge facility.

ER Studio Data Architect recognizes the following database connectivity methods:

- Open Database Connectivity (ODBC), using ODBC drivers. (See Configuring ODBC Data Source and Target Connections.)
- Native connectivity, using database client software. (See Using Native or Direct Database Connections.)
- For a list of supported databases, see the *Installation Guide* or choose Tools > Datatype Mapping Editor to view the default datatype mapping for each supported database.

See Also

- Reverse Engineering an Existing Database
- Generating a Script File or Database
- Using the Compare and Merge Utility
- Troubleshooting Database Connections

Configuring ODBC Data Source and Target Connections

ER Studio Data Architect can use 32-bit ODBC drivers to connect to database servers to reverse engineer a database or to create SQL from database objects. Before using the Reverse-Engineering Wizard or DDL Generate Wizard to connect to a database using the ODBC option, you must first install the appropriate ODBC driver for the target database platform and configure the ODBC data source. Using the ODBC Data Source Administrator you can configure data connections and then access them from the Reverse Engineer, Compare and Merge, and DDL Generation wizards.  **Note:** Most database vendors publish ODBC drivers for their respective platforms. In addition, several third-party developers also market ODBC drivers for many database platforms. You must use 32-bit ODBC drivers. ER Studio Data Architect is not compatible with 16-bit ODBC

1. Choose Tools > ODBC Setup.
2. Follow the on screen prompts to configure the data source.

Notes

- Configure the ODBC driver by following the instructions provided by the ODBC vendor.
- Data sources that are configured on the User DSN tab of the ODBC Data Source Administrator are available for the logged in user only, while data sources configured on the System DSN tab are available for all users of that system.
- You can also access the ODBC Data Source Administrator through Windows by choosing Start > Settings > Control areal > Administrative Tools > Data Sources (ODBC).
- You can share your ODBC data source configuration with other users by distributing the..\Windows\ODBC.ini file that contains the data source definitions.
- ODBC drivers are used for all databases not listed Using Native or Direct Database Connections, such as GreenPlum and PostgreSQL.

See Also

- Using Native or Direct Database Connections
- Troubleshooting

Using Native or Direct Database Connections

Using a native or direct database connection you can reverse engineer all the database objects supported by MS SQL Server, IBM DB2 for z/OS, IBM DB2 for LUW, Oracle, and Sybase ASE. To connect to these databases you must first install the corresponding DBMS client utility software available from your database vendor. (For information on the client utility versions supported, see ER/Studio Repository Database Server Requirements in the *ER/Studio Repository Installation and Administration Guide*.)

- On the machine running ER Studio Data Architect, install the RDBMS client utility software that is available from your database vendor.
    - The following table lists the databases you can connect to using native or direct connections and the RDMBS client utility software required.

Database	Required RDBMS Client Utility Software
IBM DB2 for z/OS 5.x, 6.x, 7.x, 8.x, 9.x, and 10.x	IBM DB2 Connect
IBM DB2 for LUW 5.x, 6.x, 7.x, 8.x, 9.x, 10.x Server	Corresponding version of IBM DB2 Client Utilities
IBM DB2 for AS/400 V4R5 and V5R2	Corresponding version of IBM DB2 Client Utilities
Microsoft SQL Server 4, 6, 7, 2000, 2005, 2008, 2012, 2014	Corresponding version of SQL Server Client Access
Oracle 7.x, 8.x, 9i, 10g, 11g and 12c	Corresponding version of Oracle Client Utilities
Sybase ASE 11, 11.5, 11.9, 12..0, 12.5, and 15.x	Sybase Client Utilities 11.9 or later

- Provide to your ER Studio Data Architect users the data source connection strings required for your databases so they can enter them in the Data Source fields of the Reverse Engineer, DDL Generation or Compare and Merge wizards.

Database	Required Data Source Connection String
IBM DB2 for z/OS	
IBM DB2 for LUW	Use the ODBC data source you have specified in the ODBC Data Source Administrator.
MS SQL Server	Use the server name.
Oracle	Use the SQL*Net connection string or an alias if you have defined one in: • Oracle SQL*Net Easy Config • Oracle Net8 Easy Config • Oracle Net8 Configuration Assistant
Sybase ASE	

See Also

- Reverse Engineering an Existing Database
- Generating a Script File or Database
- Using the Compare and Merge Utility

Troubleshooting Database Connections

Use the following checklists to resolve any connectivity problems you can encounter. Once you have confirmed your setup, if you are unable to connect please contact Embarcadero's Technical Support.

This section contains the following topics:

- IBM DB2 Connectivity Checklist
- Informix Connectivity Checklist
- Interbase Connectivity Checklist
- Microsoft Access Connectivity Checklist
- Microsoft SQL Server Connectivity Checklist
- Oracle Connectivity Checklist
- Sybase Connectivity Checklist

IBM DB2 Connectivity Checklist

- You have configured an ODBC data source entry for each IBM DB2 database that you want to access from ER Studio Data Architect.
- ER Studio Data Architect *uses ODBC with IBM DB2 for connectivity purposes only. It does not use ODBC catalog calls;* ER Studio Data Architect *has complete knowledge of the DB2 system catalog.*
- You have installed the IBM DB2 Client Connection Software on your local machine
- You have SYSADMIN, SYSCTRL, SYSMAINT or DBADM database privileges (not OS privileges) on your local machine.
- Unlike other database platforms, local IBM DB2 data source registration requires IBM DB2 server approval.
- *Only DBADM privileges can be granted on databases using the GRANT statement. The other privileges can be assigned only through the database manager configuration.*
- Your servers have been configured in the IBM DB2 Client Configuration Assistant.
- You have tested the connection in the IBM DB2 Client Configuration Assistant and it was a valid connection.
- The *SQLLIB\BIN* subdirectory is in the Windows Path variable.
- If you are using DBArtisan, you can see a list of servers in the server list in the Datasource Registration Wizard or in the Discover Datasource dialog box.
- You can connect to your server through your specific client utility.

- IBM DB2 version 5.x client utilities are installed locally. They are included on the IBM DB2 CD.
- You have read access to SYSIBM.SYSDATABASE and SYSIBM.SYSTABLES for DB2 OS/390 SYSCAT.SCHEMATA for DB2 UDB QSYS2.SYSTABLES for AS400. You need to be able to query these tables.

Informix Connectivity Checklist

You can connect to your server through your specific client utility.

1. You have a 32-bit Informix ODBC driver installed locally that is the correct version for the Informix server version to which you want to connect (Informix 5, Informix 7, Informix CLI for 7.2).
2. You have 32-bit Informix SetNet installed locally and configured correctly.
3. You have configured a data source in the ODBC Administrator using the Informix ODBC driver and pointed it to the correct SetNet information.

Interbase Connectivity Checklist

You can connect to your server through your specific client utility.

1. You have a 32-bit InterBase 4.2 ODBC driver installed locally.
2. You have configured a data source in the ODBC Administrator using the InterBase ODBC driver and pointed it to a valid InterBase database.

Microsoft Access Connectivity Checklist

The 32-bit Microsoft Access ODBC driver is installed locally. Search your local drives. This driver is included on the Microsoft Access media or from Microsoft website.

1. You have configured a data source in the ODBC Data Source Administrator using the Microsoft Access ODBC driver and pointed it to your database.
2. You are logging into your database using the Admin login ID.
3. If your database is in a compacted mode, you have repaired the database in the ODBC setup.
4. The database file is in standard archive mode and not locked or in read-only mode.

Microsoft SQL Server Connectivity Checklist

You can connect to your server through your specific client utility.

1. The Microsoft SQL Server version 6.5 or 7.0 client utilities are installed locally. They are included on the Microsoft SQL Server CD.
2. Your servers have been configured in one of the Microsoft SQL Server configuration tools, either SQL Server Client Configuration Utility or Client Network Utility.

3. If you are using DBArtisan, you can see a list of servers in the server list in the Datasource Registration Wizard or in the Discover Datasource dialog box.
4. For SQL Server 6.5, the following subdirectories are in the Windows Path variable: MSSQL\BIN or MSSQL\BINN and MSSQL\DLL.
5. For SQL Server 6.5, you can ping the server from ISQLW.exe.
6. For SQL Server 6.5, the Automatic ANSI to OEM and Use International Settings options are selected in the DBLibrary tab of the Microsoft SQL Client Configuration Utility.
7. For SQL Server 7.0, the MSSQL7 subdirectory is in the Windows Path variable.
8. For SQL Server 7.0, the Automatic ANSI to OEM Conversion and Use International Settings options have been selected in the DB Library Options tab of the Client Network Utility.

Oracle Connectivity Checklist

You can connect to your server through your specific client utility.

1. Your servers have been configured in one of the Oracle configuration tools: SQL*Net Easy Config, Net8 Easy Config or Net8 Configuration Assistant.
2. If using Net8 Easy Config or Ne8 Configuration Assistant, you can ping the server.
3. One of the following subdirectories is in the Windows Path variable: ORANT\BIN, ORAWIN95\BIN, ORAWIN\BIN, ORA81\BIN.
4. A least one of the following libraries in the Oracle home BIN directory: Ora73.dll, Ora803.dll, Ora805.dll or Oran8.dll.
5. The Oracle client configuration tool points to the correct TNSNAMES file: TNSNAMES.ora. If you are using DBArtisan, you can see a list of servers from your TNSNAMES file in the Server drop-down list in the Datasource Registration Wizard or in the Discover Datasource dialog box.

Sybase Connectivity Checklist

Sybase client utilities version 11.x are installed locally. Search your local drives as it is included on the Sybase media.

1. The Sybase network library LIBSYBDB.dll is installed locally.
2. Your servers have been configured in one of the Sybase configuration tools: SQLEDIT or DSEDIT.
3. You are can ping the server from either SQLEDIT.exe and SYBPING.exe or DSEDIT.exe.
4. The following subdirectories are in the Windows Path variable: SYBASE\BIN and SYBASE\DLL.
5. If you are running Windows 95 or 98, the Autoexec.bat file is calling the SYBSET.bat file.

6. If you are running Windows NT, the Sybase System Environment Variable (Control Page\System\Environment) points to only one Sybase directory.
7. If you are using the 16-bit Open Client Software, it is installed in the same directory as the 32-bit Open Client.
8. The Sybase client configuration tool points to the correct SQL.ini file. If you are using DBArtisan, you can see a list of servers from your SQL.ini file in the Server drop-down list in the Datasource Registration Wizard or in the Discover Datasource dialog box.

See Also

- Troubleshooting
- Configuring ODBC Data Source and Target Connections

Working with the Data Dictionary

Using the Data Dictionary you can enforce standards, promote reuse, and build a common framework across all models.

A well-defined Data Dictionary prevents the constant reinventing of the wheel and gives you everything that is needed right at your fingertips. Domains used in conjunction with Reference Values, Defaults, and Rules let you build a common list of data elements that you can use in any logical or physical model. You can use the attachment system to add additional metadata to any object in a logical or physical model. Every object editor includes an Attachment Bindings tab where you can add attachments.

Once you define objects in the Data Dictionary, you can use them in either the logical or physical models. You can define domains to create "template" or reusable attributes and columns. For example a domain, *Timestamp*, can be created for attributes or columns tracking a modify date. Now any *LastModifyDate* attribute or column can be bound to the *Timestamp* domain so that they can all be managed through that single domain. Reference values, defaults and rules can be implemented directly on attributes and columns, or used to further define the definition of a domain. Using the previous example, a common default for *LastModifyDate* is the current system date. So a default can be created in the Data Dictionary called *CurrentSystemDate* with a different definition depending on the target DBMS. If this default is then bound to the *Timestamp* domain under the default tab, every attribute or column that is bound to the domain will now get a default of the current system date.

You can use Data Dictionaries across many **.dm1* files. If the Repository is in use, the Enterprise Data Dictionary system can be used to implement one dictionary across disparate models. Changes to an object in an Enterprise Data Dictionary will be reflected in any model where that object is used. If the Repository is not in use, use the Import Data Dictionary command (see Importing a Data Dictionary) to import dictionary objects from another **.dm1* file. An important note to remember when using the latter implementation is that changes to one dictionary will not affect the source dictionary.

Macros can also be used to import and export domain definitions and reference values from Excel if they need to be imported from other sources such as an external metadata repository. For more information on Macros, go to the Macro tab of the Data Model Explorer to view documentation and sample macros.

- You can find a good example of a well structured Data Dictionary in the Northwind data model that ships with ER Studio Data Architect.
- If you are using the ER/Studio Repository, you can create an Enterprise Data Dictionary, which you can use in multiple diagrams. For more information see, Working with Data Dictionaries in the Repository.

The following reusable dictionary objects are stored in object-specific nodes on the Data Model Explorer Data Dictionary tab:

- *Attachments*: Associate database objects with datatypes such as a text note or a pointer to an external file. For more information, see Attaching External Documents to the Data Model.

- *Data Security Information: Enforce Enterprise Data Security standards and custom levels. For more information, see Enforcing Security Using Data Security Types and Properties.*
- *Defaults*: Make database population more efficient by using default values. For more information, see Enhancing Data Integrity Using Defaults.
- *Rules*: Enforce data integrity by disallowing transactions that violate the rules. For more information, see Promoting Data Integrity Through Rules.
- *Reference Values*: Lets you create and manage reference values. For more information, see Defining Valid Attribute Data Using Reference Values.
- *Naming Standards Templates*: Enforces naming standards. For more information, see Enforcing Naming Standards Using Naming Standards Templates.
- *User Datatypes*: Customized datatypes allow for greater flexibility. For more information, see Ensuring Consistent Domain Definitions Using User Datatypes.
- *Domains*: Reusable attributes and columns that make model creation more efficient. Quickly create a model by populating entities and tables with attributes and columns defined in domains. For more information, see Reusing Attribute Definitions Using Domains.
- *Reusable Procedural Logic*: Reusable triggers, reusable procedures, and libraries. For more information, see Reusing Procedural Logic.

See Also

- Importing a Data Dictionary
- Copying and Pasting Data Dictionary Objects

Importing a Data Dictionary

You can import the Data Dictionary of another data model diagram (DM1 file), making it building a new model easier and more efficient. You can also use macros to perform task such as importing items like domains and object definitions, setting storage parameters on physical tables, and add naming conventions to constraints.

Using a common data dictionary in your models facilitates the re-use and enforcement of established data definitions across multiple data models. When you important a Data Dictionary, the new objects are added to the existing Data Dictionary. ER Studio Data Architect maintains all existing dictionary objects as well as organization structures.

Import a Data Dictionary

1. Choose File > Import Data Dictionary.
2. In the *File Name* box, type the name of the data model where the target data Data Dictionary is stored, or browse and locate it.
3. Choose how you want to handle name duplications by selecting one of the option listed in the *Resolve Imported Objects with Duplicate Names* pane.
4. Click *Open*.

See Also

- Copying and Pasting Data Dictionary Objects
- Changing Data Dictionary Object Bindings and Values

Copying and Pasting Data Dictionary Objects

You can select any standard Data Dictionary object and copy and paste it either within the current diagram on in another diagram. When you copy or paste a Data Dictionary object, ER Studio Data Architect copies and pastes all of its dependent objects too. You can also select multiple standard Data Dictionary objects, such as Defaults and Rules, or multiple Procedural Logic Data Dictionary objects, such as Reusable Triggers or Reusable Procedures.

1. In the *Data Model Explorer*, click the *Data Dictionary* tab.
2. In the *Data Dictionary* tab, right-click one or more target objects and then select *Copy Dictionary Objects*.
3. Locate the target Data Dictionary (either the current diagram or to another diagram), right-click any node and then select *Paste Data Dictionary Objects*.

Notes

- You can also copy and paste objects from a local dictionary to an Enterprise Data Dictionary.
- When pasting domains, you can use the Switch Domain Bindings macro to map local properties to enterprise properties. You can find the macro in the sample macros on the Macro tab of the Data Model Explorer (Modeling Productivity Macros) This macro scans all the columns and attributes in the active model or all models and switches the domain bindings from the source domain to the target domain. The information for each bound column will be updated with the target domain. Any domain overrides will be preserved.
- When making multiple selections, Procedural Logic Data Dictionary objects, they must be the same type (i.e. reusable triggers or reusable procedures, but not both), all other objects can be different types.
- To enable to the Copy Data Dictionary menu item, you must click within a Data Dictionary object node.
- The Paste Data Dictionary menu item is available both at the root Data Dictionary object node as well as with in the node.

See Also

- Importing a Data Dictionary
- Changing Data Dictionary Object Bindings and Values
- Checking in the Data Dictionary

Attaching External Documents to the Data Model

Attachments and attachment types offer a structured method for you to associate an external piece of information to your data model. You can extend your metadata this way and include supporting documentation, such as meeting notes, risk analysis and spreadsheets. You can bind attachments to virtually any file or application on your system.

Attachments are organized into two key components:

- *Attachment Types*: An organizational system where you can group together similar attachments and specify which objects can have this type of attachment bound to them. Attachment Types define the scope of the attachments created underneath them. Use attachment types to organize attachments by either the object types to which they are applied or the scope of the business data they are supposed to capture for a model. Once you create an attachment, you can bind it to any data model objects associated with the attachment type.
- *Attachment*: Information that you can associate with the diagram objects types selected for the Attachment Type. Attachments can take many different forms such as an external file, a date, or a test list.

TIP: You can find a good example of a well-structured attachments folder in the Northwind sample model.

This section is comprised of the following topics:

Create and Edit Attachment Types Create and Edit Attachments

Bind an Attachment to an Object Bind an Attachment to a Specific Object

Override an Attachment Value

Create and Edit Attachment Types

1. In the *Data Dictionary*, right-click the *Attachments* folder and then click *New Attachment Type*.
2. Define the attachment type as required and then click *OK* to complete the editor.

The following describe options that require additional explanation:

Name page/tab

Define the name of the attachment type and provide a description of the attachment type, such as when and why attachments of this type are used.

Attachment Type Usage page/tab

Lists all supported object classes to which you can bind attachments of this type. Any object classes you select here will be bound to this attachment type and will be displayed subsequently on the Binding Information tab of the Attachment Editor.

Notes

- When you want to only allow specific attachments to be bound to specific objects, you can create an Attachment Type for this purpose and on this tab, indicate the specific objects by selecting the corresponding boxes.
- If you are editing an existing attachment type, and you remove object types from the object list on the Attachment Type Usage tab, bound attachments associated with this attachment type will be affected.
- To only allow specific attachments to be bound to specific objects, create an Attachment Type for this purpose and on the Attachment Type Usage tab, indicate the specific objects by selecting the corresponding boxes.

Create and Edit Attachments

1. In the *Data Dictionary*, right-click the appropriate attachment type folder and then click *New Attachment*.
2. Define the attachment as required and then click *OK* to complete the editor.
3. Once you have created the attachment, you can edit it by right-clicking the attachment you want to change, and then selecting Edit Attachment.

The following helps you complete the describes the options that require additional explanation:

Name page/tab

Define the name of the attachment and provide a description of the attachment, such as when and why this attachment is used.

Value page/tab

Select what kind of data this attachment contains. For Text List datatypes, you can list possible data values and then set the default value.

Binding Information tab

Select the object classes and/or specific objects to which you want to bind this attachment. You can override this setting using the Attachment Bindings tab of data model object editors.

Bind an Attachment to an Object

Ensure the attachment can be used with the object you want to bind it to. In the Data Dictionary, double-click the attachment you want to bind to ensure it can be bound to the type of object selected.

Bind an Attachment to a Specific Object

Once you have bound an attachment to an object type in the Data Dictionary, you can choose to bind that attachment to a specific object of that type from within the object editor. For example, you have bound an attachment to an entity type object. From within the Entity Editor, click the Attachment Bindings tab and then from the list of Available Attachments, select the attachment you want to bind to that specific entity and then click the arrow to move the attachment into the list of attachments already associated with that entity.

- Attachments bound to a primary key is not transferred to the foreign key of the child in the relationship.

Override an Attachment Value

The Value Override Editor lets you override an Attachment's default value with a new value. This lets you add additional information to a bound attachments. To override an attachment value, click the value field. The Value field varies depending on your datatype selection. For example, if the attachment is a text datatype, the Value Override Editor lets you edit the text. The following describes the different available datatypes and their associated values and how you would change their values:

- *Boolean*: Lets you select None, True, or False.
- *Date*: Displays the current date. To change the date, click the list, and use the calendar to select the target date.
- *External File Path*: Lets you change an external file path. Enter a file path or browse and locate it. If you want to view the attached file, click the View File button.
- *Numeric*: Lets you change the numeric value. Enter a numeric value in the box.
- *Text*: Lets you change the character value. Enter text in the box.
- *Text List*: Double-click the blank line in the text box, enter a value and then use the TAB key to go to the next line. Repeat these steps until you finished adding values. Use the Up or Down buttons to reorder the values. Use the Delete button to remove a value. The first entry is the default value. To change this, click the target value and then click the Set as Default button.
- *Time*: Lets you change the time value. Select one of the time parameters (hours, minutes, seconds, AM/PM) in the box, and enter a value or scroll to the appropriate value. Continue selecting time parameters and changing the value until the appropriate time is entered.

See Also

- Associating a Data Dictionary with a Diagram
- Working with the Data Dictionary

Enforcing Security Using Data Security Types and Properties

The Data Security management function of the Data Dictionary extension helps enforce security standards within the data model. It allows the data architect to classify objects in the model such as an entity, attribute, submodel or model and associate the database object with various security impact levels or compliance mappings. You can document the security of your data as you would document other properties such as the data steward would for functional business areas or external documentation. Use reporting features to communicate security information to other stakeholders.

- *Data Security Type*: A top-level description of a security aspect to be observed. Default descriptions are Compliance and Classification. You can add or edit your own types and then add properties to the types. For example, you may want to describe how security is enforced, so you might add an "Enforcement Mechanism" type and then give it properties such as "Project and Model passwords required," "Read-Only Granted By Default," and so on.
- *Data Security Property*: A specific security attribute to be observed. A default example is "Compliance Mapping" which describes which of several industry compliance and reporting standards should be maintained. You can add or edit your own types and then add properties to the types. For example, you may want to describe how security is enforced, so you might add an "Enforcement Mechanism" type and then give it properties such as "Project and Model passwords required," "Read-Only Granted By Default," and so on.

The topics in this section include:

- Generate Default Security Information
- Create and Edit a Data Security Type
- Create and Edit a Data Security Property

Generate Default Security Information

1. On the *Data Dictionary* tab, right-click the *Data Security Information* folder and then click *Generate Default Security Information*.
2. if you have already generated the default security information, choose how to resolve duplicate properties.

This generates Compliance Mapping, Privacy Level, and Security Information security properties that you can customize for your environment.

Notes

- View and edit the default security properties, right-click a security property object and then select Edit Security Property.

- If, after editing the default security properties, you want to reset the properties to their default values, generate the properties again and, when prompted, overwrite the existing security objects.

Create and Edit a Data Security Type

1. On the *Data Dictionary* tab, right-click the *Data Security Information* folder and then click *New Security Type*.
2. Enter a name for the security property type, provide a description of it, such as when and why this security property is used, and then click *OK*.
3. Once you have created the Data Security Type, you can edit it by right-clicking the type you want to change, and then selecting Edit Data Security Type.

Create and Edit a Data Security Property

1. On the *Data Dictionary* tab, right-click the appropriate security type folder and then select *New Security Property*.
2. Define the security property as required and then click *OK* to complete the editor.
3. Once you have created the Data Security Property, you can edit it by right-clicking the property object you want to change, and then selecting Edit Data Security Property.

The following describe options that require additional explanation:

Name tab

Define the name of the security object and provide a description of it, such as when and why this data security property is used.

Value page/tab

Select what kind of data this security object contains. For Text List datatypes, you can list possible data values and then set the default value.

Binding Information tab

Select the object classes and/or specific objects to which you want to bind this security property. You can override this setting using the Security Information tab of object class editors.

See Also

- Importing a Data Dictionary
- Enhancing Data Integrity Using Defaults

Enhancing Data Integrity Using Defaults

Defaults enhance data integrity in your model by supplying a value to an attribute or column when you do not specify one. Defaults ensure that attributes and columns always adopt a value, even when the user or transaction does not. provide one.

You can bind defaults to columns, attributes, and domains, along with user-defined datatype created in the Data Dictionary. Defaults are created in the Data Dictionary, which lets you manage and reuse commonly used defaults.

Create and Edit Defaults

1. In the *Data Dictionary*, right-click the *Defaults* folder and then click *New Default*.
2. Define the default as required and then click *OK* to complete the editor.
3. Once you have created the Default, you can edit it by right-clicking the default you want to change, and then selecting Edit Default.

The following describe options that require additional explanation:

Default Value tab

You must add the value in the same syntax as the database platform in which you intend to use the default.

Attachment Bindings tab

Bind an external piece of information, or attachment to the domain. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary and must be applied to the default before they will display on this tab. For more information, see Attaching External Documents to the Data Model.

Binding Information tab

Select the object classes and/or specific objects to which you want to bind this default. You can override this setting using the Default tab of the data model object editor.

See Also

- Determining Which Objects are Bound by Data Dictionary Objects
- Changing Data Dictionary Object Bindings and Values
- Enforcing Data Integrity Using Check Constraints

Promoting Data Integrity Through Rules

Rules promote data integrity by validating the data supplied to an attribute or table column. For example, you could define a rule, NotLessThanZero, which specifies that data values for an attribute or column be zero or greater. Once bound to a table column, the rule would reject any transactions that supply a negative value to the table column. Rules are independent database objects that behave like check constraints, but can be re-used throughout your data model.

Rules, like Defaults in the Data Dictionary let you manage and reuse commonly used rules. Rules can be bound to other dictionary objects such as datatype or UDTs as well as attributes/columns. Rules can be created directly in the domain or under the Rules node.

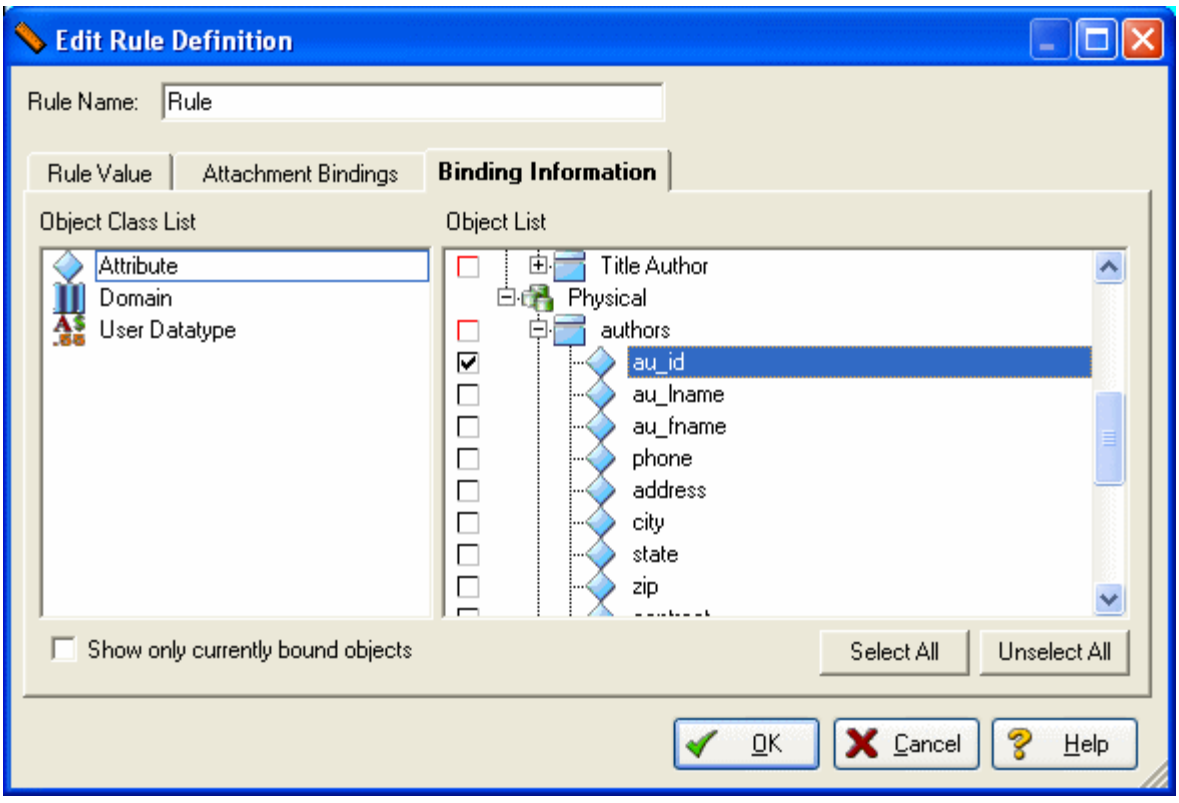
Example

If you have a generic constraint that will apply to many fields, you can create a domain that will allow you to specify the constraint in that domain. You can use a place-holder for the column in the table. Use @var to substitute the column name. The column name replaces @var when a rule in the data dictionary is bound to a column.

For example, a rule was created in the *Northwind.dm1* sample model, with the following syntax:

UPPER(@var) in ('Y', 'N')

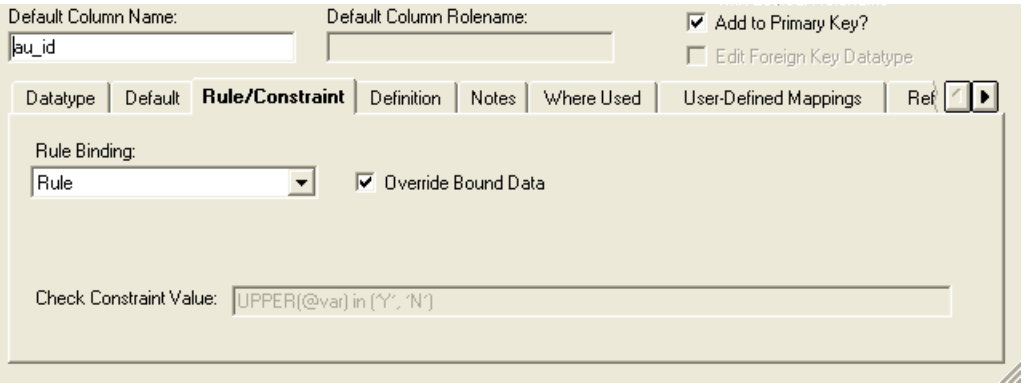
When that rule is bound to column au_id as follows:



the rules translates to:

UPPER(CURRENT_CUSTOMER) in ('Y', 'N')

as shown in the Check Constraint Value area in the following screenshot:



Create and Edit Data Dictionary Rules

1. On the *Data Model Explorer*, click the *Data Dictionary* tab.
2. On the *Data Dictionary* tab, right-click the *Rules* node and then select *New Rule*.
3. Complete the *Add/Edit Dictionary Rule* Editor as required and then click *OK* to implement the changes.
4. Once created, you can edit the rule by double-clicking it to launch the editor.

The following describe options that require additional explanation:

Rule Value tab

Specifies the rule. You must specify the rule value in the same syntax as the database platform in which you intend to use the rule.

Attachment Bindings tab

Bind an external piece of information, or attachment to the rule. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary and must be applied to the default before they will display on this tab. For more information, see Attaching External Documents to the Data Model.

Binding Information tab

Select the object classes and/or specific objects to which you want to bind this rule. You can override this setting using the Rule/Constraint tab of the object class editor.

See Also

- Determining Which Objects are Bound by Data Dictionary Objects

- Changing Data Dictionary Object Bindings and Values

Defining Valid Attribute Data Using Reference Values

Reference Values are attributes that define allowed data. They represent look-up table columns, code-value ranges, or a rule or constraint applied to a column. Reference values can be defined as a range of values or as an itemized list.

- You can import or export multiple Reference Values from Excel using a macro. On the Macro Tab of the Data Model Explorer, choose SampleMacros > Import Reference Values from Excel for information on different macros.

1. In the *Data Dictionary*, right-click the *Reference Values* node and then click *New Reference Value*.
2. Define the reference value as required and then click *OK* to complete the editor.
3. Once created, you can edit the reference value by double-clicking it to launch the editor.

The following helps you complete the Reference Value Editor:

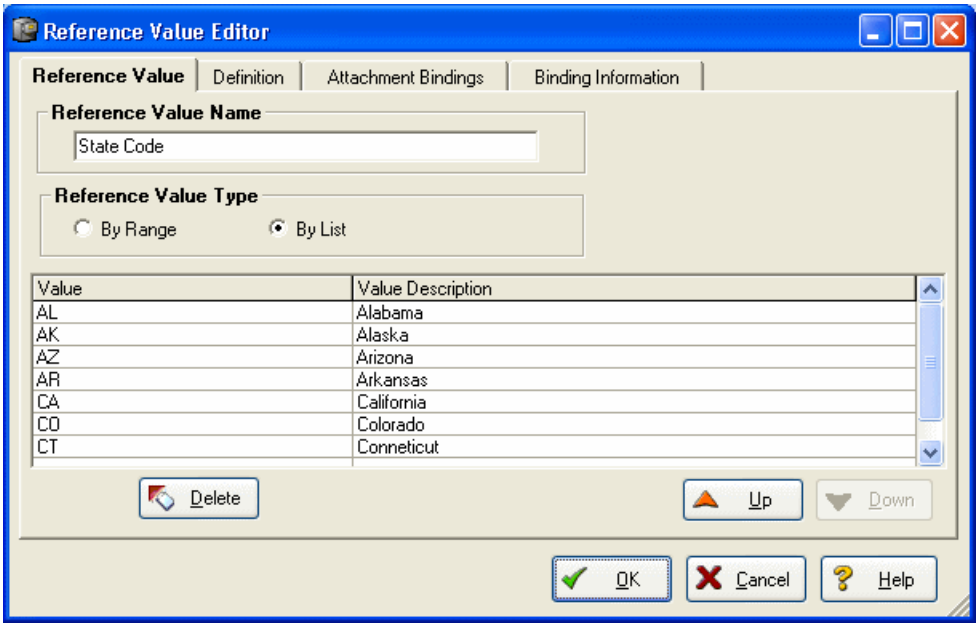
Reference Value tab Definition tab

Attachment Bindings tab Binding Information tab

See Also

Reference Value tab

- *Values 'NOT' Between*: If selected, lets you define a range of values for the bound domain, attribute, or column that are not considered valid.
- *Reference Value Type*:
- *By Range*: If selected, lets you define a range of values for the bound attribute, domain, or column. enter values in the Minimum and Maximum fields in the Reference Value Range section.
- *By List*: If selected, lets you define a list of values for the bound attribute, domain, or column. If you select By List, the Editor changes to display a Value and Value Description box. Double-click in the Value field in the first line of the table and enter a value. To enter a description, double-click the Value Description field for the first line to enter a corresponding description. Continue entering values in this manner until you have entered all desired values. The blank line at the end of the list is disregarded when saving the Reference Value.



Definition tab

Enter or edit a definition for the reference value. This description helps you to document the usage of the reference value.

Attachment Bindings tab

Bind an external piece of information, or attachment to the reference value. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary and must be applied to the reference value before they will display on this tab. For more information, see Attaching External Documents to the Data Model.

Binding Information tab

Select which object classes and/or specific objects can use this reference value. You can override this setting using the Reference Values tab of the data model object editor.

See Also

- Determining Which Objects are Bound by Data Dictionary Objects
- Changing Data Dictionary Object Bindings and Values

Enforcing Naming Standards Using Naming Standards Templates

In a Naming Standards Template you can define standards for character case, or add prefixes/suffixes to make the object types more easily recognizable. The Naming Standards Templates is comprised of an abbreviation list used to automatically convert logical to physical names and vice versa, and options for a given model that define case standards by object type, name length standards, and prefix/suffix standards. The Naming Standards Utility is used to apply the naming standards templates to a model or objects within a model.

A naming standards template can be created and stored either as either as part of a Naming Standards Template object in the Data Dictionary of a dm1 file or as XML data in an external file. Naming standard templates are supported in Enterprise Data Dictionaries as well as in local data dictionaries.

Once created and saved, the naming standards can be used when using other wizards within ER Studio Data Architect, such as XML Schema Generation, Generate Physical Model, Compare and Merge, Model Validation, and Add New Physical Model. The naming standards templates can also be applied to domains, attributes/columns, and entities/tables through a Naming Standards tab in the object editors. When selected, the *Freeze Names* option in these editors prevents any naming standards applied from changing the name of the object selected.

Reference the following topics to create a new naming standards template, import an existing template, define, view or customize the naming standard template, export the naming standards template, and apply a naming standards template to a model using the Naming Standards Utility.

- Creating a Naming Standards Template
- Track Versions of Naming Standards Templates

Creating a Naming Standards Template

There are two implementations of the Naming Standards Template Editor; one for creating an external template that can be imported for use with any diagram, and another for creating a template that is bound to the active diagram. The Naming Standards Editor, used to bind objects in the diagram to a naming standards template includes two tabs not present in the external template editor, Attachment Bindings and Binding Information.

1. To create an external naming standards template and that is saved as an XML file, choose Tools > Naming Standards Editor.
   To create a naming standards template that is saved as an object in the Data Dictionary of the active diagram, *from the* Data Model Explorer, Data Dictionary *tab, right-click* Naming Standards Templates *node and then click* New Naming Standards Template.
   Using either of the methods above launches the Naming Standard Template Editor.
2. To save the new file and exit the utility, click *OK*.
   If this template is a Data Dictionary object, the new naming standards template appears within the Data Model Explorer, Data Dictionary, Naming Standards Templates node.

If this template is an external file, a new naming standards template file is created in the Model directory. Naming standards templates have an extension of.*nst*.

Tip:: Once created, you can edit the naming standards template by double-clicking its object on the Data Dictionary tab to launch the editor.

The following help you complete the options of the *Naming Standards Template Editor*:

Name tab
Logical and Physical tabs
Mapping tab
Order tab
General tab
Attachment Bindings tab
Binding Information tab
Track Versions of Naming Standards Templates
See Also

Name tab

- *Import from External Naming Standards* Template: Enables you to important an XML-formatted naming standards template file into the diagram. This could be an external naming standards template that was created using the Naming Standards Template Editor accessible from the Tools menu.
- *Export to External Naming Standards* Template: The naming standards template is saved with a file name extension of.nst. Other ER Studio Data Architect users can then use the import function to open the naming standards template in their diagram.

Logical and Physical tabs

- *Case* column: Select a conversion option to apply to each logical data model object type:

The results achieved by applying the case conversion options are described in the table that follows:

Case conversion option	Data model name	Converted name (spaces removed)
[none]	CUSTOMER ADDRESS CODE	CUSTOMERADDRESSCODE
UPPER	Customer	CUSTOMER
lower	CUSTOMER	customer
UpperCamel	CUSTOMER ADDRESS CODE	CustomerAddressCode
lowerCamel	CUSTOMER ADDRESS CODE	customerAddressCode

- *Prefix* and *Suffix* columns: To enhance readability of models and code, you can add prefixes/suffixes so people reading the code from the model will more easily recognize object types. In the physical model you could add a T_ or V_ in front of each table or

view name (prefix) so when users see the reference in a view, procedure or function, others will know what type of object it is. Adding a suffix can be used in the same way.

Mapping tab

- *Import/Export*: Click to import or export an abbreviation list in a comma-delimited formatted file (.csv)
- *Abbreviations* tab: Map logical names to physical names and then further qualify the name type. On this tab, you can also define illegal names, import an abbreviation list from a comma delimited file (.csv), or export the abbreviation list that displays to a.csv file.
- To delete a row, simply select it and press Delete on the keyboard. The easiest way to select a row is by right-clicking on it. You can also select multiple rows using Shift or Ctrl while right-clicking
- *Prime*: Indicates that the word entered in the Logical or Physical column makes the name unique such as, Customer, Employee, and Account.
- *Qualifier*: Qualifies the class word to further define the meaning of the attribute. A name could have multiple modifiers. For example, a logical column name is Customer First Name ID, Customer is the Prime and ID is the Class. If there are abbreviations in the Naming Standards Template for the logical texts First and Name, and you check the qualifier box, ER Studio Data Architect sorts First and Name as qualifiers when it reorders the text during translation.
- *Class*: The type of data in an attribute. Class data is typically used in attribute and column names, not entity or table names. Most new naming standards have the class word at the end of the attribute name such as, name, date, code, id, and document.
- *Illegal*: Words that are not allowed in names. These will mainly be used during the validation process.
- *Priority*: Used to resolve duplicate physical abbreviations as names are transformed from physical to logical. The default value of the Priority field is Primary. The priority changes to Secondary when you define a mapping for a physical name that is already mapped to a logical name. You can change the priority by selecting the priority field, clicking the down arrow and selecting another priority. When you change the priority of an abbreviation from Secondary to Primary, the abbreviation that was previously Primary is changed to Secondary. A physical name can only have one primary abbreviation, but can have many secondary abbreviations.

Order tab

Specify the order of each part of the name for both logical and physical names. If the order is not specified, the order of the parts in the original name is maintained.

For each object type, select the corresponding cell in the Part 1 column, click the arrow, and then select which part of the name should come first when translating between logical and physical names. Then define the second and third part of the name if required.

- To specify that an object part is unordered, select the appropriate cell in the part column, click the arrow and then select the ellipsis (...).

General tab

- *Delimiter Options* area: Defines how to break up words in both the logical and physical model. For example, should Customer Number be translated to CustNum or Cust_Num. On this tab,
  To separate a word when a specific character is encountered, in the *Delimiter Options* section, click *By Character* and then type the character.
  To separate a word when a cases change is encountered, in the *Delimiter Options* section. select *By Case.*
- *Special Characters* area: Specifies which characters are to be treated as special characters and how they are processed.

Attachment Bindings tab

Bind an external piece of information, or attachment to the naming standards template. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary and must be applied to the reference value before they will display on this tab. For more information, see Attaching External Documents to the Data Model.

Binding Information tab

Select which object classes and/or specific objects can use this naming standards template. You can override this setting using the Naming Standards tab of the data model object editor.

Track Versions of Naming Standards Templates

You can track versions of a naming standards template (NST) by importing the NST into an NST object of an Enterprise Data Dictionary; a data dictionary that has been or will be added to the Repository. Whenever you check in the Data Dictionary, you can add notes describing the changes applied. You can view the version history of any object. in the Repository.

See Also

- Working with Data Dictionaries in the Repository
- Viewing the Checkin History of a Repository Object

Ensuring Consistent Domain Definitions Using User Datatypes

User-defined datatypes are a powerful mechanism for ensuring the consistent definition of domain properties throughout a data model. You can build a user-defined datatype from base datatypes, specifying width, precision and scale, as applicable. In addition, you can bind rules and defaults to the user-defined datatype to enforce domain integrity. After creating a user-defined datatype, you can use it in attributes, table columns, and domains without needing to define its underlying definition each time. User-defined datatypes are particularly useful with commonly-referenced table columns in a database, such as *PhoneNo*, *ZipCode* or *PartNo*.

The central maintenance of user datatypes eliminates the tedium and potential for errors users can encounter when manually editing each referenced table column. For example, you can define a surrogate key in the Part table, PartNo, as an integer for efficiency. You have defined a user datatype, *partnumber*, to represent every use of PartNo throughout the database design. Subsequently, you learn that the data to be converted from a legacy system contains some part numbers in character format. To accommodate the change, you only need to edit the definition of *partnumber* from an integer to a character-based user datatype. ER Studio Data Architect automatically converts all table columns using *partnumber* to the new definition.

Although only a few database platforms currently support user-defined datatypes, ER Studio Data Architect extends their utility to all database platforms in the physical model. ER Studio Data Architect automatically converts user-defined datatypes to their base definitions when generating SQL code for any database platforms that do not provide native support.

- For SQL Server, Sybase, and IBM DB2 only, the *CREATE* statements of the user datatype can be included in the DDL when generating the database.

Create and Edit User Datatypes

1. On the *Data Dictionary* tab, right-click the *User Datatypes* node and then select *New UserDatatype*.
2. Define the user-defined datatype as required and then click *OK* to complete the editor.
3. Once created, you can edit the naming standards template by double-clicking its object on the Data Dictionary tab to launch the editor.

The following describe options that require additional explanation:

UDT tab

- *Apply nullability to all bound columns*: If you indicate Yes Allow Nulls, this check box is available. When you are editing the user datatype and you change this selection from what it was previously, selecting this check box broadcasts the change to all objects bound to this user datatype.
- *Default Binding*: *Optional* If you want to bind the user datatype to a Default, select it from the list.

- *Rule Binding*: *Optional* If you want to bind the user datatype to a Rule, select it from the list.

Attachment Bindings tab

Bind an external piece of information, or attachment to the user datatype. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary and must be applied to the user datatype before they will display on this tab. For more information, see Attaching External Documents to the Data Model.

Binding Information tab

Select which object classes and/or specific objects can use this user-defined datatype. You can override this setting using the Attributes or Columns tab of the entity or table editor, respectively, or on the Datatype tab of the domain editor.

- a user datatype and change its definition, ER Studio Data Architect automatically updates your entire model so that all affected objects use the new definition.

See Also

- Enforcing Naming Standards Using Naming Standards Templates
- Enforcing Security Using Data Security Types and Properties
- Reusing Attribute Definitions Using Domains

Reusing Attribute Definitions Using Domains

Domains are very useful for creating entity attributes or table columns commonly used in your models. Creating a domain allows you to define the object once and then use it repeatedly by applying the domain to the entity attributes and table columns.

Domains are reusable attribute templates that promote consistent domain definitions. You construct domains as you would attributes, specifying a name, datatype properties, null status, default values and validation rules. After creating domains, you can re-use them in your data model by applying them to attributes and table columns. By defining a domain, you also gain the power of propagating changes to the underlying domain definition to all attributes and table columns that reference it.

One important method for enforcing business rules is to define and to apply domain restrictions. Domain integrity refers to the rules governing what values an attribute can take. By restricting and validating an attribute's values, you can implement important business rules such as ensuring that a checking account maintains a positive balance, or preventing the entry of invalid phone numbers.

ER Studio Data Architect lets you organize your domains using domain folders. Domain folders let you create and manage your Data Dictionary Domains. You can classify domains in unique groups by creating different domain folders. You can create nodes within nodes. You can move existing domains into a domain folder, or create new domains within the folder.

Domains can be dragged and dropped into a table or entity to create a column or attribute. Domains can also be dragged onto another domain to create a subdomain, by moving the selected domain beneath the target domain. Domains can also be copied and pasted onto another domain to create a copy of the selected domain as a subdomain or the target domain. Note, changes to the copied domain will not be inherited by the pasted subdomain.

Domains can have nested subdomains or child domains that initially duplicate most of the attributes of the parent domain, but can have divergent behaviors once created. A Domain Override option in the Add Domain or Edit Domain functions creates inheritance and overrides the domain system without losing the valuable relationship with the attributes/columns (binding information).

- When you delete a domain folder, all domains and folders within that folder are also deleted. All bindings to any columns and attributes are unbound.

Create a Domain Folder

1. On the *Data Dictionary* tab, right-click the *Domains* node and then click *New Domain Folder*.
2. Enter a name for the domain and then click *OK*.

Create and Edit a Domain or Subdomain

To create a domain at the root of the Domains node, on the **Data Dictionary** tab, right-click the *Domains* node and then click *New Domain*.

*To create a domain within a domain folde*r, on the *Data Dictionary* tab, expand the *Domains* node, right-click the node of a domain folder, and then click *New Domain*.

To create a subdomain (a domain within a domain folder), on the Data Dictionary *tab, navigate to the domain folder where you want to create the subdomain, right-click the domain folder, and then click* New Domain.

1. Define the domain as required and then click *OK* to complete the editor.
2. Once created, you can edit the domain by double-clicking its object on the Data Dictionary tab to launch the editor.

The following helps you complete the *Domain Editor* options:

Datatype tab	Default tab
Rule/Constraint tab	Reference Values tab
Naming Standards tab	Definition tab
Note tab	Override Controls tab
Attachment Bindings tab	Binding Information tab

Datatype tab

- *Receive Parent Modifications*: *For subdomains*, lets you synchronize changes made to the attributes of a parent domain with a those of the child domain. Selecting this option automatically synchronizes parent and child attributes, except those specified on the Reference Values, Naming Standards, Attachment Binding, and Binding Information tabs. Not selecting this option creates a child domain that is initially identical to the parent domain, except that the child domain is created without any attachment bindings. Subsequent changes to the parent domain will not be inherited by the subdomain. Any changes you make to the subdomain will override the values inherited from the parent domain. Henceforth, the child can only inherit attributes from the parent if these attributes were not previously specified for the child.
- *Synchronize Domain and Attribute/Column Names*: Select this option to synchronize the Domain, Attribute, and Column Names. If you entered names in the Attribute and/or Column Name fields that are different than the Domain name, selecting this option overrides your entries.
- *Apply nullability to all bound columns?* This option becomes available when you select a datatype that can have nulls but you choose to disallow nulls by clicking No in the Allow Nulls area. Selecting this check box does not impact if there are no bound columns.

- *Identity Property* area: This area becomes available if the Identity Property is supported by the database platform of a model in the diagram, and the datatype selected, such as TINYINIT, supports this property.

Default tab

- *Default Binding*: If a default value has not already been bound to the domain (Data Dictionary > Defaults > Edit Default Definition > Binding Information), you can enter a Declarative Default value or choose another Default from the Default Binding list. The declarative default will appear in the SQL produced when you select Generate Database from the physical model.
- *Declarative Default*: Specifies the default domain value when a value is not explicitly specified during an insert. DEFAULT definitions can be applied to any columns except those defined as timestamp, or those with the IDENTITY property. Only a constant value, such as a character string; a system function, such as SYSTEM_USER(); a NULL; or a constraint name can be assigned to a DEFAULT.

Rule/Constraint tab

Add, modify, or delete a constraint. Check constraints are used to ensure the validity of data in a database and to provide data integrity. For more information, see Enforcing Data Integrity Using Check Constraints.

Reference Values tab

Reference Values are attributes that define allowed data. They represent look-up table columns or code-value ranges or a rule or constraint applied to columns. For more information, see Defining Valid Attribute Data Using Reference Values.

Naming Standards tab

Controls user and object permissions to modify the affected item. When a Naming Standards Template is bound to an object, then that template is used instead of the Naming Standards Template selected in the Naming Standards Utility or the Generate Physical Model utility. This allows you to apply different naming standards to different objects when portions of a model require different versions of a standard. For example, some objects already exist in the database and you do not want to apply a newer naming standard to them. For more information, see Enforcing Naming Standards Using Naming Standards Templates.

Definition tab

Enter or edit a definition for the domain. If the target database supports it, ER Studio Data Architect adds this definition as a domain comment when generating SQL code.

Note tab

The notes added here are included as part of the HTML report when you generate a report for the model. You can format the notes using standard HTML tags.

Override Controls tab

Controls what can or cannot be overridden on columns bound to this domain.

- *General Overrides* area: The allow/disallow options disable the "override" indicator on the bound attributes/columns if set to Disallow. When you modify the domain override controls and set one or more of the general override options to Disallow, ER Studio Data Architect (and the RepoServer for Enterprise Data Dictionaries) removes all existing overrides for that particular property.
- *Name Synchronization*:
    - *Never Sync:* Completely detaches the domain name from the bound attributes/columns. This avoids inadvertent changes when the attribute or column names are bound to the same domain, and attribute/column name in the domain editor is changed. Changes to the names in the domain do not affect the names of attributes/columns that are bound to the domain.
    - *Partial Sync*: Changes to the attribute/column name in the domain editor will update the name of any bound attributes or columns that have the old name, i.e. where the name has not been overridden. For example, if you create two entities with an attribute from the domain, one with the default name and one with an overridden attribute name, then modify the attribute name in the Domain editor and have selected Partial Sync, then the domain change will modify the attribute name in the entity where the name has not been overridden, but will not change the attribute name in the entity where the name has been overridden. If the names do not match then the changes are not propagated.

**Note:** The *Name Synchronization* controls do not apply when switching bound domains. Those controls only apply when editing a bound domain.

- *Attachment Synchronization*: Keeps attachments in the domain in sync with the bound column. Any attachments added to a domain will propagate to the attributes/columns. Always Migrate always migrates new attachments on the domain to the columns. You can still add other attachments directly to the columns.

Attachment Bindings tab

Bind an external piece of information, or attachment to the domain. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary and must be applied to the domain before they will display on this tab. For more information, see Attaching External Documents to the Data Model.

Binding Information tab

Select the object classes and/or specific objects to which you want to bind this attachment. You can override this setting using the Attribute or Column tab of the entity or table editor, respectively.

See Also

- Reusing Procedural Logic
- Importing a Data Dictionary

Reusing Procedural Logic

The Data Dictionary supports the following types of procedural logic and enables you to organize them by database platform:

- *Reusable Triggers*: A special type of stored procedure that automatically executes when data modification operations such as INSERT, UPDATE, or DELETE occur. Because triggers are customizable and fire automatically, they are often used to maintain referential integrity in a database.
- *Reusable Procedures*: Templated procedures written in BASIC code. These procedures can be applied to any table, since they use the table's context in the code.
- *Libraries*: Lets you compartmentalize blocks of code used to generate SQL for Reusable Triggers and Reusable Procedures. This lets you reuse blocks of code. Reusable triggers or procedure code can call library functions.

The editor that launches when creating new procedural logic is the Sax Basic editor, which is compatible with Microsoft Visual Basic.

Topics covered in this section are:

- Create and Edit Reusable Triggers
- Create and Edit Reusable Procedures
- Create and Edit Libraries

Create and Edit Reusable Triggers

1. Select a table.
    - The SQL in the procedure will be validated in context of the table selected and the database platform. If no table is selected, the SQL cannot be properly validated.
2. In the *Data Dictionary*, expand the *Reusable Procedural Logic* node and then expand the *Reusable Triggers* node.
3. Right-click the node for the appropriate database platform and then click *New Trigger*.
4. Define the trigger as required and then click *OK* to complete the editor.
    - **Tip:** Once created, you can edit the trigger by double-clicking its object on the Data Dictionary tab to launch the editor.

The following describe options that require additional explanation:

- In order for code to be generated for procedural logic such as triggers and procedures, the results of the logic must be stored in a variable called *resultstring*. For example, in the *Northwind.dm1* sample model, the Data Dictionary lists a reusable trigger, SYSUPDATE, for the Oracle platform. The code for this trigger includes the following

statement, which is required for code to be generated for the trigger:
resultstring = trigBody

BASIC tab Includes a text editor as well as all the editing and formatting tools that are available on the ER Studio Data Architect Macro Editor. You can enter trigger code or you can use the Import button to import it. You can use Visual BASIC and/or AI to create the code for the reusable trigger.

- *DBMS*: If you change the DBMS from the one you clicked when you opened the editor, you can use the Copy Dictionary Object functionality to move the Trigger to the correct DBMS node.
- *Select Libraries*: Click to select a previously defined library to add to the trigger code. When using library functions, the library file needs to be selected by selecting this option.

Create and Edit Reusable Procedures

1. Select a table.
    - The SQL in the procedure will be validated in context of the table selected and the database platform. If no table is selected, the SQL cannot be properly validated.
2. In the *Data Dictionary*, expand the *Reusable Procedural Logic* folder and then expand the *Reusable Procedures* folder.
3. Right-click the folder for the appropriate database platform and then click *New Procedure*.
4. Define the reusable procedure as required and then click *OK* to complete the editor.
Tip:* Once created, you can edit the procedure by double-clicking its object on the Data Dictionary tab to launch the editor.

The following describe options that require additional explanation:

- In order for code to be generated for procedural logic such as triggers and procedures, the results of the logic must be stored in a variable called *resultstring*. For example, in the *Northwind.dm1* sample model, the Data Dictionary lists a reusable trigger, SYSUPDATE, for the Oracle platform. The code for this trigger includes the following statement, which is required for code to be generated for the trigger:
resultstring = trigBody

BASIC tab Includes a text editor as well as all the editing and formatting tools that are available on the ER Studio Data Architect Macro Editor. You can enter procedure code or you can use the Import button to import it. You can use Visual BASIC and/or AI to create the code for the reusable procedure.

- *DBMS*: If you change the DBMS from the one you clicked when you opened the editor, you can use the Copy Dictionary Object functionality to move the procedure to the correct DBMS node.

- *Select Libraries*: Click to select a previously defined library to add to the procedure code.

Create and Edit Libraries

1. In the *Data Dictionary*, right-click the *Libraries* folder and then click *New Library*.
2. Define the library as required and then click *OK* to complete the editor.

The following describe options that require additional explanation:

BASIC tab Includes a text editor as well as all the editing and formatting tools that are available on the ER Studio Data Architect Macro Editor. You can enter library code or you can use the Import button to import it. You can use Visual BASIC and/or AI to create the code for the library.

Library Example

'This function generates the header for a templated insert
'procedure. Input is entity name that is used to generate
'procedure name. Platform is Oracle.

Function insertprocheader (ent As Entity) As String

Dim result As String

Dim attr As AttributeObj

'add create statement for the procedure with naming convention

result = "CREATE OR REPLACE PROCEDURE P" & ent.TableName & "INSERT" & vbCrLf

result = result & "(" & vbCrLf

'add parameter list for insert statement
'loop in actual sequence order

For i = 1 To ent.Attributes.Count

For Each attr In ent.Attributes

If attr.SequenceNumber = i Then

'make parameter line for column

result = result & "V" & attr.ColumnName & vbTab & vbTab & vbTab

result = result & "IN "

result = result & attr.Datatype & "," & vbCrLf

End If

Next attr

```
Next i

'trim last comma of the parameter list

result = Left(result, Len(result) - 3)

'add last closed parantheses

result = result & ")" & vbCrLf

result = result & "AS" & vbCrLf & "BEGIN" & vbCrLf

'return header

insertprocheader = result

End Function
```

Call to Library Example

This sub main routine demonstrates how to call a library function.


Note: The library needs to be added to the reusable trigger or stored procedure.

```
Sub Main

Dim procstring As String

procstring = insertprocheader(CurrEntity)

'call other library functions for body of procedure
'call other library functions for end of procedure
'check procedure text with a message box
'disable when generating DDL

MsgBox(procstring)

'output procedure string variable to DDL wizard

resultstring = procstring

End Sub
```

See Also

- Reusing Attribute Definitions Using Domains
- Importing a Data Dictionary

Determining Which Objects are Bound by Data Dictionary Objects

The View Bindings dialog box displays the names of the attributes or columns to which the selected Data Dictionary object is bound. Selected objects show which domains are bound to which attributes and columns.

1. On the *Data Dictionary* tab, navigate to the dictionary object you want to investigate, such as a specific rule.
2. Right-click the data dictionary object and select *View.... Bindings*, where... is the name of the data dictionary object.

Notes

- The View Bindings dialog is read-only.
- To unbind an object, use the Binding Information tab of the data dictionary object editor or the relevant tab of the editor of the database class or object you want to unbind.

See Also

- Changing Data Dictionary Object Bindings and Values
- Importing a Data Dictionary

Changing Data Dictionary Object Bindings and Values

The bindings of data dictionary objects can be overridden in the data model object editor. For example, you have created a security property and applied it to an entity. Unbinding the security property from an attribute of that entity or changing the value of the security property can be accomplished through the Security Information tab of the Attribute Editor.

- In order to bind an data dictionary object to a database object, that specific object or class to which the object belongs, must have been bound to the data dictionary object in the data dictionary object editor.

1. In the *Data Model Explorer*, right-click the data model object, and then click *Edit*.
2. In the editor, click the relevant tab, as described in the table below.

To change bindings of this type	Use this tab in the database object editor
Attachments	Attachment Bindings
Data Security Information	Security Information
Defaults	Default
Rules	Rule/Constraint or Constraints
Reference Values	Reference Values
Naming Standards Templates	Naming standards
User Datatypes	Datatype
Domains	Attributes

- You can also bind Attachments to an entire diagram. You can change the diagram bindings on the Diagram Properties dialog. Choose *File > Diagram Properties*.
- Make your changes and then click *OK* to exit the editor.

Notes

- You can override the value of an attachment by double-clicking the value of the attachment in the grid, which brings up the Value Override Editor.
- When you create a report, you can choose to include details of bound attachments for the objects selected to described in the report. For more information, see Generating RTF and HTML Model Reports.

See Also

- Generating RTF and HTML Model Reports
- Working with the Enterprise Data Dictionary

Working with the Enterprise Data Dictionary

If your installation provides access to the ER/Studio Repository, you can promote a local data dictionary to the Repository, thereby creating an Enterprise Data Dictionary which can be shared amongst Repository users. For more information, see Working with Data Dictionaries in the Repository.

See Also

- Working with Data Dictionaries in the Repository
- Importing a Data Dictionary

Documenting Data Extraction, Transformation, and Load

This section is comprised of the following topics:

- Introduction to Data Lineage
- Data Lineage Workflow
- Creating and Editing Data Lineage Data Flows
- Defining Source Systems
- Relating Source and Target Tables and Columns
- Assigning Model Data Movement Properties
- Creating and Editing Data Lineage Data Flows
- Documenting Table/Entity Data ETL
- Documenting Column/Attribute ETL

See Also

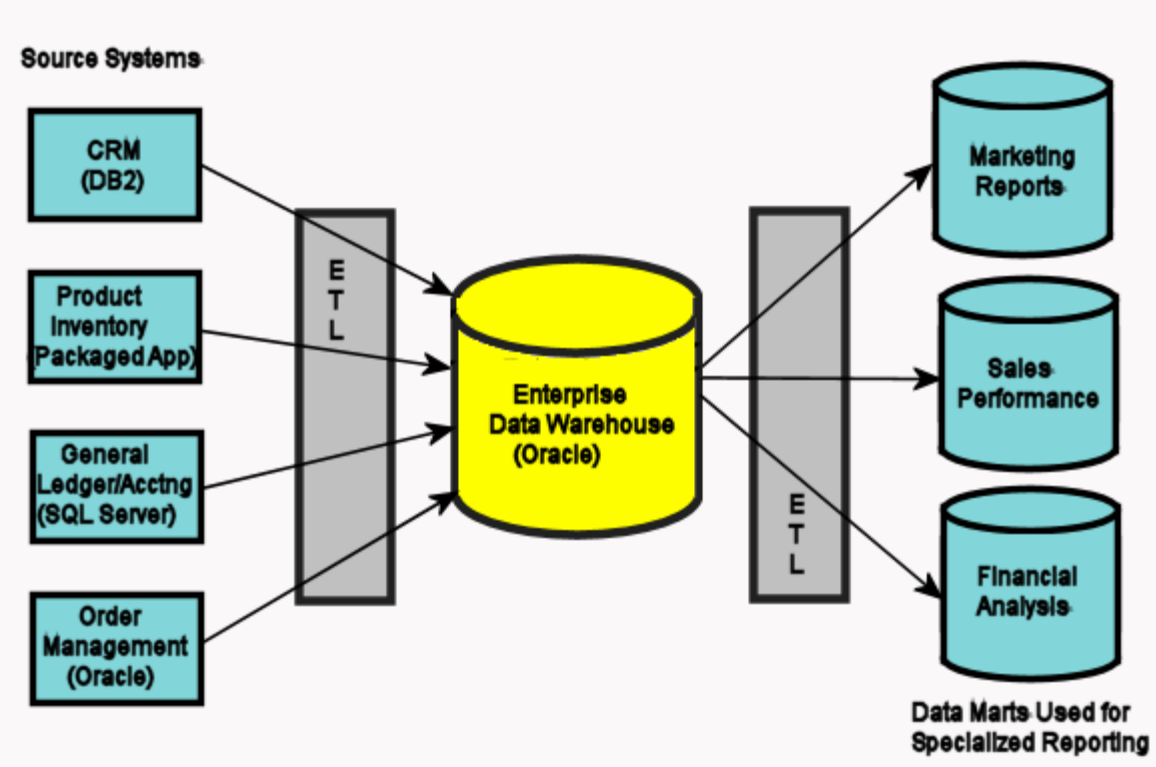
- Tutorials:Documenting an Existing Database
- Exporting a Data Model to XML

Introduction to Data Lineage

Data Lineage enables you to document the movement of data from point A to point B, and any intermediate steps in between. This movement is sometimes referred to as Extraction, Transformation and Load (ETL). Points A and B can be anything from flat files, high-end databases like Oracles and DB2, XML files, Access databases, and Excel worksheets. This is sometimes referred to as source and target mapping. A model produced in ERStudio can represent any point along the way. Data Architects need the ability to specify the "source" or "target" of data, down to the column/attribute level. Along with the metadata that defines the source and target mapping are rules for how the data is manipulated along the way.

The task of moving the data is often assigned to specialized developers who have little data modeling experience. They need good documentation to guide them in deciding what data goes where. Documenting the movement or mappings for the developers is usually the task of a data architect. Data Architects need to track data between very complex systems. For example, an organization may have a data warehouse that consists of data sourced from a number of different systems, such as CRM (Customer Relationship Management), payroll, general ledger, accounting, product, and inventory systems. The data warehouse may in turn have on-line data marts that receive data from it to produce reports for Sales Executives, HR Executives and marketing teams that are generating reports about the performance of various aspects of the business.

The following illustrates a generalized data movement process:



In this illustration, the data is sourced from various systems on the left and fed to a data warehouse that stores the data in a format that is more conducive to reporting. This reduces the amount of overhead on the source systems so resources are not used for reporting directly on

them. The data also must be cleansed to ensure the quality of the data used for reports. Reports can be generated directly from the Enterprise Data Warehouse (EDW), but many times are fed to other specialized data marts targeted for specific audiences. Further cleansing or manipulation of the data is performed in the process of moving it to the Online Data Manager (ODM). An organization using ERStudio would most likely have a model of each part of this process, the Customer Relation Management (CRM), General Ledger/Accounting, Orders, Inventory, EDW, and ODMs. It is the job of the data architect to "map" these systems together. They manage the models and need to define the mappings which produce reports that assist the data warehouse engineers. An ETL developer feeding data into one of the data marts needs to know what columns and tables to get the data from in the EDW. For consistency sake, they may also need the mappings from the legacy or Online Transaction Processing (OLTP) systems.

ER Studio Data Architect provides several ways for you to document how your data moves from system to system.

In Reports and generated SQL you can view the ETL information you have specified as follows:

- In the Table editor, you can document how often the data is extracted along with the rules that regulate data movement
- In the Table Column editor, you can document how the data is transformed as it moves from source to target, known as source-to-target mapping.

Using the Data Lineage tab you can create a visualization of the data movement and transformation, so you can see the relationships between the source and target, how the data flows from one table to another, and how the data is transformed.

See Also

- Data Lineage Workflow
- Data Lineage

Data Lineage Workflow

Data Lineage workflow generally goes as follows:

1. Defining Source Systems
2. Relating Source and Target Models
3. Relating Source and Target Tables and Columns
4. Creating and Editing Data Lineage Data Flows
5. Documenting Table/Entity Data ETL
6. Documenting Column/Attribute ETL
7. Share the data lineage with your colleagues through reports, SQL, XML, exported Metadata or through the ER/Studio Team Server and ER/Studio Viewer.

See Also

- Data Modeling Fundamentals
- Data Lineage

Defining Source Systems

Data sources can originate from models in the active diagram (local models), from external sources imported into the active diagram or from a data sources created on the Data Lineage tab. The source can be imported from **.dm1* files, **.dt1* files, database or an SQL files, flat files, and other common application files. Potential sources from the open diagram are found on the Data Lineage tab, which can be seen by expanding Data Sources > Local Models.

- Source data imported through the Data Lineage tab only includes information such as table and column name, datatype, nillability, primary key, and column definitions. To obtain more details, reverse engineer the database or import it into ER Studio Data Architect using the Metadata Wizard.

This section is comprised of the following topics:

- Creating a New Source
- Importing External Source into Other Sources

Creating a New Source

1. From the *Data Lineage* tab, expand the *Data Sources* node.
2. Right-click *Other Sources* and choose *New Source*.
3. Complete the *Data Source Properties* dialog as required and then click *OK* to exit the editor.

The new source will appear under the Other Sources node with one empty object node, which may be Tables or Entities depending on the data source type defined. You can edit the new source model by expanding the Other Sources node, and then the new source node. Then, right-click the source object and select Add Table/Entity to launch the Table or Entity Editor.

The following describe options that require additional explanation:

General tab

- *General Properties*
- *Name*: The name you give it here will be displayed as a data source in the Other Sources node within the Data Lineage tab.
- *Type*: Select the source type. This setting affects available options in the Connectivity group. For example, select Relational for a DBMS such as MySQL.
- *Connectivity Properties*
- *Host, Server/Service, Port*: For connecting to an external DBMS. For an existing ER Studio Data Architect physical model, leave blank.
- *DBMS Type*, *Version*, *Location/Path*, *File Type*, *Encoding*: These settings depend on the Type selected above.

Model Usage tab

This is a read-only display of the source defined on the General tab. Ensure that it matches your intentions.

Importing External Source into Other Sources

1. From the *Data Lineage* tab, expand the *Data Sources* node.
2. Right-click *Other Sources* and choose *Import New Source*.
3. Complete the *Data Source Properties* dialog as required and then click *OK* to exit the *Import Source* wizard.

The new source will appear under the Other Sources node.

The following describe options that require additional explanation:

Page 1 - Please select where you would like to import the source metadata from

- *From a Repository based DM1 file*: Lets you obtain source from data models and Named Releases managed within the ER/Studio Repository. When you select this option, ER Studio Data Architect opens the Repository Operation Status dialog box and the Get From Repository dialog box. This process connects to the current Repository Server defined in the Repository settings. The Import Source wizard automatically gets the diagram.
- *From an SQL file* ER Studio Data Architect imports the SQL file.
- *Compare against a live database:* If you select this option, a page appears where you can select the database and connection type. The connection type can be either ODBC or Native/Direct Connection. For information about connecting to databases, including troubleshooting information, see Connecting to Database Sources and Targets.
- *Comparison Quick Launch*: The Compare Quick Launch data is saved as an *.rvo file. For information on using the Quick Launch option in the wizard, see *Saving and Using Quick Launch Settings*.

Page 5 - Results

- *Current and Target Model Display Grid*: Between the Source and Target models is a Resolution column. The default merge decision is Merge the data into the new source file. You can click on any item in the Resolution column to enable the decision list. If you want to change the decision, click the list and then click the new resolution. When you change the default resolution of an object, the decisions of their dependent properties and objects are automatically updated. You can also click the category folders, like the Tables Resolution column to change all the decisions for all the underlying objects in that object category. And, you can use the CTRL key to select multiple items, and then right click to enable the decision list.
- *SQL DIfference*: To enable the SQL Difference utility, select any difference that is a long text field, such as a Definition, Note, or DDL, and then click SQL Difference to view the differences between the SQL of the models. This utility only allows you to view the

differences; difference resolutions are performed on the Results page of the Compare and Merge Utility.

- *Filter Report on Results:* Create a report of the source content and your chosen resolutions. You can choose to create an HTML or an RTF report.
- You can modify the default display using the options at the bottom of the page.

See Also

- Relating Source and Target Models
- Creating and Editing Data Lineage Data Flows

Relating Source and Target Models

Data movement properties describe how source and target models are related outside of the Data Lineage tab. You can relate a source model to one or more models in the same diagram or to models imported from external systems. The rules defined here are used at the table/entity level on the Data Lineage tab of the entity and table editors.

1. On the Data Model Explorer, right-click the top-level node for the particular model and select *Data Movement Properties*.
2. Select the *Source* or *Targets* tab.
3. Select the level of mapping you intend to add or edit.
4. Click *Add* and select from among the models listed.
5. Select the *Synch registered data...* option if you want changes to the current model that extend to other models in the.dm1 file to be automatically updated.

Once complete, you can double-click any table to edit table-level and column-level movement rules.

See Also

- Data Lineage Workflow
- Data Lineage

Relating Source and Target Tables and Columns

Data movement rules describe how source and target tables and entities are related. You can relate source data to one or more tables and entities in the same model, the active diagram, or to tables imported from legacy systems. The rules defined here are used at the table level on the Data Lineage tab of the entity and table editors.

Create and edit data movement rules

1. On the *Data Lineage* tab, right-click *Data Movement Rules* and choose *New Data Movement Rule*.
2. Complete the *Data Movement Rule* editor as required and then click *OK* to exit the editor.

Once created, you can edit the Data Movement rule by double-clicking it to launch the Data Movement Rule editor.

The following describe options that require additional explanation:

Rule Information tab

- *Rule Name*: Enter a name that indicates the operation and objects acted on, depending on the specifics of your binding definition.
- *Rule Type*: Select a generic movement rule type that best describes the data movement.
- *Rule Text*: Document your data movement plan here, perhaps adding instructions or contingency plans.

Binding Information tab

Select the object classes and/or specific objects to which you want to bind this rule. You can override this setting using the Data Lineage tab of the entity or table editor. For more information, see Creating and Editing Entities.

See Also

- Data Lineage Workflow
- Data Lineage

Assigning Model Data Movement Properties

Assigns data sources or targets and the level of mapping you intend to use, direct or secondary. You must first have added a source/target in the Data Lineage tab using the *Data Source/Targets Properties*. To add a data source/target to a physical model:

1. Right-click the top-level node for your particular physical model name and select *Data Movement Properties*.
2. Select the *Source* or *Targets* tab.
3. Select the level of mapping you intend to add or edit.
4. Click *Add* and select from among the models listed.
5. Select the *Synch registered data...* option if you want changes to the current model that extend to other models in the.dm1 file to be automatically updated.

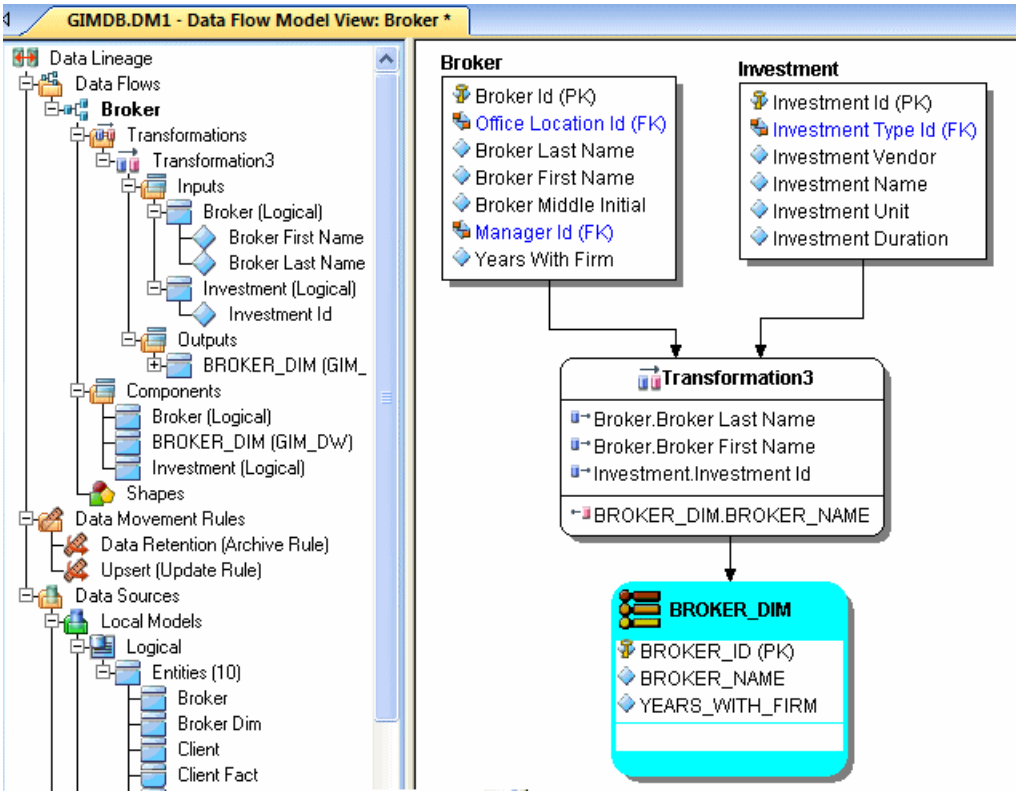
Once complete, you can double-click any table to edit table-level and column-level movement rules.

See Also

- Defining Source Systems
- Introduction to Data Lineage

Creating and Editing Data Lineage Data Flows

The Data Lineage Data Flow organizes and encapsulates data transformations and the source tables and columns used in the transformations to produce the target data. Multi-tiered mappings are possible and there can be multiple transformations involving different attributes/columns between two entities/tables as illustrated below.



Tip: Displaying the transformation definition, inputs and outputs, and code is controlled by the Transformation options in the Diagram and Object Display Options editor. For more information, see Configuring and Customizing.

Create a Data Lineage Data Flow

1. From the data model explorer, click the *Data Lineage* tab at the bottom of the application window.
2. *If this is the first time you click the Data Lineage tab after opening a diagram*, you are prompted to create a *Data Lineage Data Flow*. Click *Yes*.
   If this is not the first time you click the Data Lineage tab after opening a diagram, from the Data Lineage explorer, right-click the *Data Flow* node in the Data Lineage explorer and then click *Create Data Flow*.
3. Enter a data flow name and then click *OK*.

Note The name that appears in the diagram title tab at the top of the application window is appended with: *data flow name*, when you click a data flow in the Data Lineage explorer, such as *GIMDB.DM1 - Data Flow Model View: Data Flow Name**.

4. If you have not already created your data movement rules and data sources and targets, proceed to Relating Source and Target Tables and Columns or Defining Source Systems, and then return to Step 5.

5. Navigate to and then drag and drop the source and target tables or entities onto the Data Lineage window.

   Tip: You can have multiple source and target tables or entities. Drag and drop as many as you need.

   Tip: You can choose to display the inputs and outputs of the transformation, by right-clicking an empty space on the Data Lineage window and selecting Diagram and Object Display Options, then clicking the Transformation tab and selecting Input and Output Columns.

   Note: The Data Lineage Window will continue to display the data flow visualization if you click the Repository, Data Dictionary or Macros explorer tabs, but if you click the Data Model tab, the Data Lineage Window is replaced with the Data Model window. Clicking the Data Lineage tab brings back the Data Lineage window.

6. Right-click an empty space of the Data Lineage window and then click *Insert Transformation*.

7. Double-click the new transformation to open the *Transformation Editor*.

8. Complete the *Transformation Editor* as required and then click *OK* to exit the editor.

   Tip: Once the data flow is created, you can double-click it to change its name, or double-click a transformation or component to change its properties.

9. Right-click an empty space of the Data Lineage window and then click *Insert Data Stream*.

10. Click an input source and then click the transformation object. Repeat as many times as necessary to link all the inputs to the transformation object.

11. Click the transformation object and then click an output. Repeat as many times as necessary to link all the outputs to the transformation object.

The following helps you complete the options of the *Transformation Editor*.

All tabs Columns tab

Definition tab Data Movement Rules tab

Attachments tab See Also

All tabs

Name: The name of the transformation can be edited in this field.

Type: Click the type list to choose a transformation; or in the text field, type the name of an existing transformation type or enter a name for a new transformation type.

Columns tab

This tab (also used for Attributes) displays the Parent Model, Parent Object, Attribute/Column name, Data Type and Definition of the transformation inputs and outputs.

- *Inputs*: Click the ellipsis (...) button to choose the source data to be transformed in this transformation. Here you can select all the columns of the table or only selected columns to be the input of the transformation.
- *Outputs*: Click the ellipsis (...) button to choose the output data to be created by this transformation

Definition tab

- *Business*: Describe the transformation for your audience.
- *Code*: Enter the code that will perform the transformation, such as a SELECT statement, of a VB Basic or JavaScript function or procedure.

Data Movement Rules tab

These are the rules from the Data Movement Rules node of the Data Lineage explorer.

**Note:** You can delete or edit an input or output column by double-clicking the transformation in the Data Lineage window, clicking the ellipsis in the Transformation Editor and then deselecting the column you want to remove.

Attachments tab

Bind an external piece of information, or attachment to the transformation. You can also remove an attachment from an object, override an attachment binding's default value, or change the position of a bound attachment. To override the value of the attachment you have moved to the Selected Attachments grid, double-click the Value field of the target attachment. ER Studio Data Architect opens the Value Override Editor or a list depending on the attachment datatype. Attachments are created in the Attachments folder of the Data Dictionary and must be applied to the default before they will display on this tab.

- Like any other database object, you can right-click a transformation, choose Comments and add a comment to the transformation.

See Also

- Data Lineage Workflow
- Tutorials:Creating a Data Flow

Documenting Column/Attribute ETL

At the column/attribute level, you can document how data is transformed as it moves from one system to another, by adding transformation logic for source/target attribute mapping. You can map multiple source/targets to one column/attribute in the physical/logical model.

You can register the same source/target or physical model more than once for a given physical model. For example, in the Adventure Works.dm1 model, you can register the Adventure Works physical model to the Adventure Works DW physical model as a direct source AND direct target if the data warehouse is used for analytical reporting and to perform some data cleansing. Additionally, you can also register the Adventure Works physical model to the Adventure Works DW physical model as both a secondary source and direct source.

Document column/attribute data sources

1. From the *Data Model* tab, navigate to and then double-click the table/entity whose data lineage you want to document.
2. In the table/entity editor, select the column or attribute to document and then click the *Data Lineage* tab.
3. Click the *Direct Sources* tab.
4. Click *Edit*.
5. In the *Edit Source/Target Mappings* dialog, select from the list of sources to map to your target and edit the transformation logic accordingly.
6. A description is optional, but a very good engineering practice vital to maintainability.
7. Continue in the same manner as above, to map secondary sources, and direct and secondary targets.
8. Click *OK* to exit the editor.

See Also

- Documenting Table/Entity Data ETL
- Documenting Data Extraction, Transformation, and Load

Documenting Table/Entity Data ETL

Use the Data Lineage tab of the entity or table editor to document how often the table/entity data is sourced and when it was last sourced. Once you have created data movement rules in the Data Lineage Explorer, you can apply these rules to the selected table/entity. You can associate as many data movement rules as you want to the table/entity. You can report data movement rules when creating reports, generating SQL or XML Schema, or Metadata is exported.

Document table data movement

1. From the *Data Model* tab, navigate to and then double-click the table/entity whose data lineage you want to document.
2. In the Table Editor, click the *Data Lineage* tab.
3. Complete the *Data Lineage* dialog as required and then click *OK* to exit the editor.

See Also

- Data Lineage Workflow
- Documenting Data Extraction, Transformation, and Load
- Documenting Column/Attribute ETL

Saving and Using Quick Launch Settings

To improve efficiency when performing repetitive tasks, Quick Launch lets you save settings and selections for select ER Studio Data Architect wizards and utilities for later reuse. You can save settings in separate files for each of the following wizards and utilities.

- Compare and Merge
- DDL Generation
- Model Validation
- Naming Standards
- Reports, Reverse Engineering
- Reverse Engineering
- Submodel Synchronization
- XML Schema Generation

This section is comprised of the following topics:

- Save Quick Launch Settings to a File
- Use Quick Launch Settings from a File
- Rename Quick Launch Settings File
- Delete Quick Launch Settings

Save Quick Launch Settings to a File

1. On the last page of the wizard or utility, deselect *Use file-based Quick Launch settings*.
2. Click *Save As*.
3. Enter a name for the quick launch settings file.
4. Click *OK*.

Use Quick Launch Settings from a File

1. Select *Use file-based Quick Launch settings*.
2. Click *Load File*.
3. Navigate to and then double-click the desired Quick Launch settings file.

The Quick Launch file extensions are:

- CMO for Compare and Merge
- DDO for DDL Generation

- MVO for Model Validation
- NSO for Naming Standards
- RPO for ER Studio Data Architect Report
- RVO for Reverse Engineering
- SSO for Submodel Synchronization
- XQO for XML Schema Generation

By default, these files are stored in the XML subdirectory of the ER Studio Data Architect application data directory; the default directory can be changed in *Tools* > *Options* > *Directories* > *Quick Launch*.

- **Windows XP**: C:\Documents and Settings\<user>\Application Data\Embarcadero\ERStudio\XML
- **Windows 8, Windows 7, and Windows Vista**: C:\Users\<user>\AppData\Roaming\Embarcadero\ERStudio\XML
- **AppWave ION**: My Documents\ERStudio Data Architect XE\XML
- Select *Settings and Objects* if you want to use the saved selection or *Settings Only* to use the default selections.
- Click *Go!*

Rename Quick Launch Settings File

1. On the last page of the wizard or utility, deselect *Use file-based Quick Launch settings*.
2. Select a quick launch setting file from the list.
3. Click *Rename*.
4. Enter a new name for the quick launch settings file.
5. Click *OK*.

Delete Quick Launch Settings

1. On the last page of the wizard or utility, deselect *Use file-based Quick Launch settings*.
2. Select a quick launch setting file from the list.
3. Click *Delete*.

See Also

- Automating ERStudio DA
- Administrator Guide

Generating RTF and HTML Model Reports

One of ER Studio Data Architect's most powerful applications is that of a documentation generator to communicate complex databases and associated metadata to the Enterprise. ER Studio Data Architect is equipped with extensive report generation capabilities:

- HTML Report Generation: Instantaneous generation of an HTML-based Web site designed to provide simple navigability through data models and model metadata using standard browsers such as Microsoft's Internet Explorer or Netscape Navigator.
- RTF Report Generation: Instantaneous documentation generation compatible with applications like Microsoft Word.
- In addition to providing model reviewers with reports, you can also make the model available so reviewers using ER/Studio Viewer or ER/Studio Team Server can see the actual model, but not make any changes to it.

The ER Studio Data Architect Report Wizard guides you through the process of creating rich text format (RTF) and HTML reports based on your logical or physical models. The ER Studio Data Architect Report Wizard lets you generate RTF and HTML reports. Due to the formatting differences between RTF reports and HTML reports, the information is presented in slightly different ways, even though the data, with few exceptions, is the same for both formats. Differences in the type of information reported is noted for each format type in the pages that have specific information about each object type's reports.

Why Use RTF Reports?

RTF reports are suitable for hard copy distribution. You can view the RTF-formatted reports using almost any popular word processing application.

You can select the object types to include in the report (such as entities, relationships, etc.) You can also select the amount of information about each object type to include in the report such as the summary of all instances of each object type, and/or detailed information about each object instance.

RTF reports include a title page, followed by a table of contents, followed by the reports. In general, and subject to the selections made for information that should be included in the report, each object type such as entity, relationship, etc. is introduced in a summary report that gives general information about each instance of each object type, followed by detailed reports for each instance of an object type. Much of the information in the report is presented in easy-to-read tables.

Why Use HTML Reports?

HTML reports are suitable for sharing on the internet or local Intranet. The reports require a browser that supports frames (and has them enabled). You can include a JPG, GIF, or BMP image of the submodels in your report and you can set the zoom level for each submodel individually.

Reports generated in HTML format have a similar feel to web pages. The title page is opened when the report is open. At the top of the title page and at the top of all pages in the report is a

list that lets you select an object type's report page. The list includes only those object types that have a report page. If there is no information reported about a particular diagram object type, it is not included in the list. The report page for each object type has a list in a left-hand pane that provides links to specific instances of an object type. Information is presented in easy-to-read tables.

See Also

- Generate a Report
- docwiki.embarcadero.com/Connect/en/Using_Reports

Generate a Report

ER Studio Data Architect generates reports on all data model objects each with their own specific properties.

1. On the *Data Model Explorer*, locate the Logical or Physical model for which you want to create a report.
2. Right-click the model source for the report and then click *Generate Reports*.
3. Complete the pages of the *ER Studio Data Architect Report Wizard* as required.

ER Studio Data Architect generates reports on all data model objects each with their own specific properties.

The following helps you complete the options of the *ER Studio Data Architect Report Wizard*:

Page 1 Page 2

Page 3 for RTF Reports Page 3 for HTML Reports

See Also

- The options differ depending on whether the model selected is a Logical or a Physical model and whether you selected to create an RTF or an HTML report.

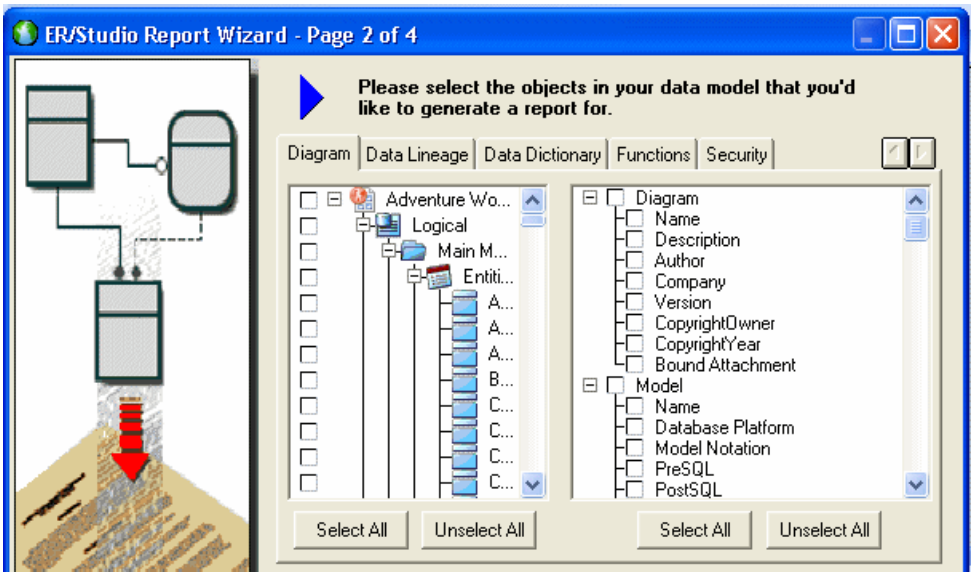
Page 1

- *Do you want to invoke an Editor to view the report after it is generated?*: If Yes is selected, launches the default application for viewing and editing.rtf or.html files, for example, Microsoft Word for.rtf files or Firefox for.html files.
- *Use enhanced version.* Use this option if your model contains an attribute with more than 200 objects. If selected, ER Studio Data Architect splits the HTML report into three areas or frames, ensuring quick navigation of all objects in an attribute. In the report, viewable data is divided into Object Type, Object List, and Object detail.

Page 2

Select the database and data lineage objects to report on.

- To view the full name of the database object in the tree on the left side of this page, pause the cursor over the representative icon and the full name will appear.



- The tabs available depend on the database platform you have selected.
- If you right-click anywhere on the object tree, you can select all or deselect all objects using the short-cut menu. You can also display or hide object owner names for those objects with owners.
- If you select diagram objects on the data model before opening the wizard, only those selected object are selected in the object tree saving you from reselecting objects.
- You can expand or collapse portions of the tree by clicking the + and - nodes.
- *Select All* button: Selects all objects. If you do not click Select All and if you are using Quick Launch settings saved from versions earlier than 6.6, object selections will be overwritten due to a more recent feature that organizes objects by submodel.

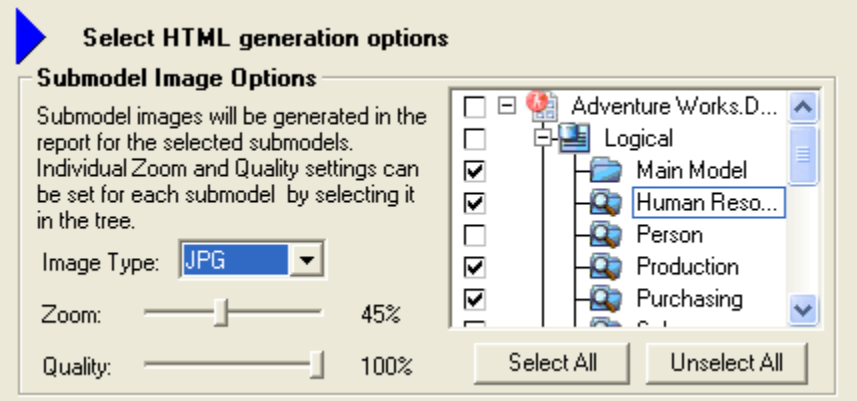
Page 3 for RTF Reports

- *Create Title Page*: Lets you create a title page, which consists of the report title, Diagram Report, followed by diagram details as specified on the Information tab of Diagram Properties.
- *Page Break Options*: Lets you specify where to insert page breaks in the report. By default, a page break occurs after each section in the report. *Note:* Inserting page breaks after each section can significantly increase the length of the report.

Page 3 for HTML Reports

- *Submodel Image Options*: Select either a.gif,.jpg or.png formatted image of the model or submodel, similar to a screenshot, to include in the report.
- Generating a JPG is resource intensive and may not be possible for large data models. If you need to capture an entire large data model, you can try reducing quality or zoom size, changing the image type format, or select individual options and ER Studio Data Architect will capture individual objects for each selected object. For example, do not

select the Logical Model, which selects each model and submodel in that model; rather, select the models and submodels individually.



- *Logo and Link Options*
- *Image File*: Select an image, such as your company logo, to include on the to left-hand side of the page.
- *New Link*: Enter the URL of the content you want to display when the report reader clicks the image specified above.
- *Report Formatting*: Choose separately to retain the format, which can include.html tags, of the object definition and note, or if you are not formatting the text within the object editor, choose to wrap the text.

See Also

- Printing the Data Model
- Exporting an Image of the Data Model

Exporting the Data Model

You can export the logical or physical data model to a variety of different formats and platforms, allowing you to collaborate with other database users, communicate your database design, create SQL from your data model to create a physical database, and create an XSD Schema you can share with application developers who can use this schema.

Click a link below for information on exporting data models:

- Exporting Metadata for a Particular Model
- Exporting Metadata for all Models and Submodels
- Exporting a Data Model to XML
- Generating a Script File or Database

See Also

- Exporting an Image of the Data Model
- Printing the Data Model

Exporting Metadata for a Particular Model

- You need a separate license to access the MetaData Export bridges. See the About ERStudio dialog for a list of the modules you are licensed to use.
- This feature is not supported by the Developer Edition of ER Studio Data Architect.

Using the MetaData Export Wizard, you can export your diagrams in formats compatible with a wide-variety of industry-standard tools. For a current list of supported platforms and bridges available, see:

- **Windows XP**: C:\Program Files\Embarcadero\ERStudio Data Architect X.X\MIRModelBridges.xml
- **Windows 8, Windows 7, and Windows Vista**: C:\ProgramData\Embarcadero\ERStudio Data Architect X.X\MIRModelBridges.xml

or see the pdf version at docs.embarcadero.com/products/er_studio].

1. On the Data Model Explorer, right-click the model or submodel you want to export.
2. Click Export Model Metadata.
3. Complete the *Export Model Metadata* dialog as required.
4. Click the field beside the option name to either enter a value or select a value from the list. Help text for each option appears in the area at the bottom of the page.
5. If you want to save or print the log, click *Save to File* or *Print*.

The following describe options that require additional explanation.

Page 1

Type: From the list, select the MIR Bridge you want to use to generates a file that can be imported into the third-party application supported by the selected bridge type.

Page 3

This page displays informational and warning messages describing successful and unsuccessful exports respectively.

See Also

- Exporting an Image of the Data Model
- Printing the Data Model

Exporting Metadata for all Models and Submodels

- This feature is not supported by the Developer Edition of ER Studio Data Architect.
- You need a separate license to access the MetaData Export bridges. See the *About ERStudio* dialog for a list of the modules you are licensed to use.

Using the MetaData Export Wizard, you can export your diagrams in formats compatible with a wide-variety of industry-standard tools. For a current list of supported platforms and bridges available, see:

- **Windows XP**: C:\Program Files\Embarcadero\ERStudio Data Architect X.X\MIRModelBridges.xml
- **Windows 8, Windows 7, and Windows Vista**: C:\ProgramData\Embarcadero\ERStudio Data Architect X.X\MIRModelBridges.xml

or see the pdf version at [docs.embarcadero.com/products/er_studio].

1. Click **File > Export File > Export Model Metadata**.
2. Complete the *Export Model Metadata* dialog as required.
3. Click the field beside the option name to either enter a value or select a value from the list. Help text for each option appears in the area at the bottom of the page.
4. If you want to save or print the log, choose *Save to File* or *Print*.

The following describe options that require additional explanation:

Page 1

Type: From the list, select the MIR Bridge you want to use to generates a file that can be imported into the third-party application supported by the selected bridge type.

Page 2

Model selection: Choose to export the logical model, physical model, or all models. If you choose to export the physical model, because there can be numerous physical models in an ER Studio Data Architect file you must enter the name of the model in the Physical model name field.

Page 3

This page displays informational and warning messages describing successful and unsuccessful exports respectively.

See Also

- Exporting an Image of the Data Model
- Printing the Data Model

Exporting a Data Model to XML

The eXtensible Markup Language (XML) is a document format that has recently emerged as a standard for data representation and exchange on the Internet. XML is also now supported by many applications and platforms, allowing the data modelers and application developers to collaborate on service-oriented architecture (SOA) initiatives. For example, the data modeler models the data and the relationships that describe how applications exchange data, and exports the model to XML, which the application developer can then use to ensure that the applications interface correctly.

ER Studio Data Architect allows you to generate the following XML-type files:

- Schema Definition (XSD): The XSD formally describes the elements in an XML document. It is an abstract representation of an object's characteristics and how the object relates to other objects. The XSD is used to verify that each element in an XML document conforms to the element rules described in the XSD. Schemas are most often used in e-commerce, data control, and database-style applications where character data content must be validated, strict data control is needed, or where strong data typing is required. For example, many applications use an XSD to validate parameters in configuration files.

With one click, you can export the entire logical model to an xsd file. This function makes assumptions about how the model elements map to the xsd element and gives you no control over the mappings. For more information, see Generating a Schema Definition (XSD).

- Document Type Definition (DTD): The DTD is another XML schema language used in traditional text document publishing applications.The DTD is a description in XML declaration syntax of a particular type or class of documents. It defines what names are used for certain element types, where they may occur, how the elements fit together, and ensures that all documents conforming to the DTD are constructed and named in a consistent manner. Validators in applications such as editors, search engines, browsers, and databases can read the DTD before reading the XML file in order to prepare to display or otherwise work with the XML document.
- XML Metadata Interchange (XMI): The XMI file format was designed primarily to allow the exchange of data between data modeling applications. Based on the XML format, XMI allows data modelers using Unified Modeling Language (UML) and programmers using different languages and development tools to exchange information about the model metadata, what a set of data consists of and how it is organized.

Using the MetaData Export Wizard, you can export the entire.dm1 file or an individual model to an.xsd or an.xmi file. This function makes assumptions about how the model elements map to the xsd elements but gives you some control over the export parameters. For more information, see Exporting Metadata for a Particular Model and Exporting Metadata for all Models and Submodels.

- XML Schema Generation (XML): The XML Schema Generation Wizard creates schemas that can be based on a logical or physical model, a submodel, or even specific components only of a model or submodel. The XML Schema Generation Wizard allows you to customize the XML schema structure by clicking the desired options and dragging the available objects from the database design into a tree representing the

XML schema. This wizard provides advanced options to transform relational entities and attributes into complex types, elements and attributes, define naming standards and datatype mapping, and incorporates the Quicklaunch system used in other wizards to save previous settings and streamline repetitive operations. For more information, see Generating a Customized XML Schema.

You can use XML editing applications such as Microsoft XML Notepad, Eclipse, Visual Studio, and Altova XML Spy to view and edit the schema.

This section contains the following topics:

- Generating a Schema Definition (XSD)
- Generating a Document Type Definition (DTD)
- Generating a Customized XML Schema

See Also

- Exporting an Image of the Data Model
- Printing the Data Model

Generating a Document Type Definition (DTD)

Use the Schema Definition (XSD) command to create a schema definition of the logical model.

1. Open the model you want to base the Schema Definition on.
2. Choose File > Export File > XML File > Document Type Definition (DTD).
3. Specify the output file and then click *Save*.

See Also

- Exporting an Image of the Data Model
- Printing the Data Model
- Exporting the Data Model

Generating a Schema Definition (XSD)

1. Open the model you want to base the XML Schema on.
2. Choose File > Export File > XML File > Schema Definition (XSD).
3. Specify the output file and then click *Save*.

See Also

- Exporting an Image of the Data Model
- Printing the Data Model
- Exporting the Data Model

Generating a Customized XML Schema

1. Open the model you want to base the XML Schema on.
2. Choose Models > XML Schema Generation.
3. On the *XML Schema Generation Wizard*, accept the default values and then select the database entities to include in the schema, or change the schema options to customize the schema contents.
4. Complete the *XML Schema Generation Wizard* and then click *Finish* to exit the editor.


Note: The XML Schema Generation Wizard contains many options, using standard XML terminology, to define the contents of the.*xsd* file created. The procedures that follow include brief descriptions of these options. For detailed information on XML terminology and coding standards, see www.w3.org/XML.

The following helps you complete the options of the *XML Schema Generation Wizard*:

Page 1 Page 2

Page 3 Datatype Conversion tab

Name Conversion tab Page 4

Page 1

The options on this dialog define the XML processing parameters that appear at the beginning of the XML schema and set up Quick Launch to enable you to reuse your choices to rerun the XML generator later. On this page a Quick Launch settings file can be loaded to regenerate the schema using the saved settings and objects, or you can modify the settings and objects before generating the schema.

- **Encoding**: Specify the name of the character encoding used for externally parsed entities. UTF-8 is the most common encoding used. Other possible encoding names include UTF-16, ISO-10646-UCS-2, ISO-10646-UCS-4, ISO-8859-1, ISO-8859-02, ISO-2022-JP, Shirft-JIS, and EUC-JP.
- **Standalone**: Select the appropriate option to indicate the presence of attribute defaults and entity declarations that are external to the document entity.
    - **Yes**: Indicates that there are no markup declarations external to the document entity that affects the information passed from the XML processor to the application.
    - **No**: Indicates that there are markup declarations external to the document entity that affects the information passed from the XML processor to the application.
    - **Unspecified**: Indicates that he XML processor should determine whether the document is standalone or not depending on the presence of external markup

declarations. No standalone declaration is included in the document type declaration.

- Schema Level Comment: Select the type and then enter the content of comments to include in the schema.
    - XML Comment: If selected, indicates that comments use standard XML comments syntax. XML parsers ignore XML comments. Information in the Schema Level Comment box describes the intended purpose of the schema. For example,
    - xs:documentation: If selected, indicates that comments are enclosed within the *xs:documentation* element, which is a child of the annotation element. Information in the Schema Level Comment box describes the intended purpose of the schema and is available to any processing application. For example,
      <xs:annotation>
      <xs:documentation>this is an XSD comment
      </xs:documentation>
      </xs:annotation>
    - xs:appinfo: If selected, indicates that comments are enclosed within the *xs:appinfo* element, which is a child of the annotation element. Information in the Schema Level Comment box provides additional information to a processing application, stylesheet or other tool. For example,
      <xs:annotation>
      <xs:appinfo>this is an XSD comment
      </xs:appinfo>
      </xs:annotation>

Page 2

Set the schema attributes and namespace parameters, which are not required parameters but various XML schema development tools use the schema attributes and namespace parameters to validate the XSD.

- Namespace: Defines the XML vocabulary used in the schema, which is comprised of unique element and attribute names. The namespace defines any elements and attributes in the schema that use the prefix of that namespace. Each vocabulary used should have its own namespaces to enable the XML parser to resolve ambiguities that can exist between identically named elements or attributes. In the example that follows, the element lib:Title is defined in the XML vocabulary located at http://www.library.com.
  <?xml version="1.0"?>
  <Book xmlns:lib="http://www.library.com">
  <lib:Title>Sherlock Holmes</lib:Title>
  <lib:Author>Arthur Conan Doyle</lib:Author>
  </Book>
- Location: Defines the Uniform Resource Identifier (URI) of the namespace, usually using Web address conventions. You do not need to define the vocabulary of a namespace but usually the namespace URI location contains either a Document Type Definition (DTD) or an XML schema that defines the precise data structure of the XML. To add

another Namespace declaration, click the space below the first Namespace and Location entries and type the Namespace declaration.

- Target Namespace: Specifies the URI of a DTD or XML schema that defines any new elements and attributes created in the XML instance. The XML parser uses the target namespace to validate the XML, ensuring that any elements and attributes used in the XML exist in the declared namespace. The XML parser also checks the target namespace for other constraints on the structure and datatype of the elements and attributes used. If the target and default namespaces are different, you can choose the namespace in from the list of the type property. The type list is populated with the namespaces that are defined on this page.

Default Namespace: Specifies the URI of a DTD or DML schema that implicitly defines any element within the XML schema that is not explicitly defined, with the prefix of another namespace.

Element Form Default/Attribute Form Default: The elementFormDefault and attributeFormDefault declarations are interdependent. The following describes the results of specifying the various combinations of these declarations

Element form default value	Attribute form default value	Results
Not specified	Not specified	No rules define how globally and locally declared elements and attributes must be defined.
Not specified	qualified	All locally and globally declared attributes must be namespace qualified.
Not specified	unqualified	All locally and globally declared attributes must not be namespace qualified.
qualified	Not specified	All locally and globally declared elements must be namespace qualified.
qualified	qualified	All locally and globally declared elements and attributes must be namespace qualified.
qualified	unqualified	All elements and globally declared attributes must be namespace qualified and locally declared attributes must not be namespace qualified.
unqualified	Not specified	All locally and globally declared elements must not be namespace qualified.
unqualified	qualified	Attributes and globally declared elements must be namespace qualified and locally declared elements must not be namespace qualified.

unqualified	unqualified	All globally declared elements and attributes must not be namespace qualified and locally declared elements and attributes must be namespace qualified.
unqualified	qualified	Attributes and globally declared elements must be namespace qualified and locally declared elements must not be namespace qualified.

Page 3


Note: Some of the options on the tabs of Page 3 are very similar to each other, such as *Add element refs to parent entity elements* and *Include view columns for complex types, elements, and groups*. Where possible, similar options are described together.

Model Options and Dictionary Options: The Model Options control the default behavior for the entity relationship diagram (ERD) to XML mappings as the model objects are dropped into the Schema Body on page 4 of the utility. The Dictionary Options control the default behavior for the entity relationship diagram (ERD) to XML mappings as the domain objects are dropped into the Schema Body on page 4 of the utility.

- **Include definitions as annotations**: If selected, specifies to include the definition as an annotation to the entity, attribute, or view declaration. For example, in the following entity declaration, the entity definition *Someone we employ*, is included within the *xs:annotation* element.

```
<xs:element name="Employee" nillable="true">
<xs:annotation>
<xs:documentation>Someone we employ.
</xs:documentation>
</xs:annotation>
</xs:element>
```

- **Include Nillable property in XSD**: If selected, specifies to generate the *nillable="true"* option for elements that were created from objects where null values are permitted. In the example above, the nillable option was generated for the element.

- **Include attributes for complex types, elements and groups**: If selected, specifies to include entity attribute or view columns when dragging and dropping entities from the model to the schema. Attributes and columns are added to the complexType declaration. The default for this option includes the attributes within the *xs:sequence* element. In the following example, the Employee entity was dragged onto the complexType element, which created the sequence element containing the entity attributes.

```
<xs:element name="Employee nillable="true">
<xs:complexType>
<xs:sequence>
<xs:element name="EmployeeID" type="empid"/>
<xs:element name="FirstName" type="xs:string"/>
<xs:element name="LastName" type="xs:string"/>
```

```
</xs:sequence>
</xs:complexType>
</xs:element>
```

- Include PK attributes: If selected, specifies to include the Primary Key (PK) indices as attributes. For example, the following entity declaration incudes EmployeeID, which is the primary key for the Employee entity.

```
<xs:element name="Employee nillable="true">
<xs:complexType>
<xs:sequence>
<xs:element name="EmployeeID" type="empid"/>
<xs:element name="FirstName" type="xs:string"/>
<xs:element name="LastName" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
```

- Include FK attributes: If selected, specifies to include the Foreign Key attributes as an entity is added to the complex type or element node. For example, the following entity declaration incudes the foreign keys for the Employee entity, which are JobID and PublisherId.

```
<xs:element name="Employee" nillable="true">
<xs:complexType>
<xs:sequence>
<xs:element name="FirstName" type="xs:string"/>
<xs:element name="LastName" type="xs:string"/>
<xs:element name="JobID" type="xs:short"/>
<xs:element name="PublisherID" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
```

- Add element refs to parent entity refs (Include view columns for complex types, elements, and groups): If selected, specifies to create an element reference to show relationships. For example, if this option is selected and a parent entity such as CUSTOMER is implemented as an element under the element node and you drag a child entity ORDER under the complex type node, then an *<xs:element ref ="CUSTOMER"/>* is created.
  This option automatically populates the cardinality of the element ref. For identifying relationships minOccurs=1 and maxOccurs="unbounded", and for non-identifying relationships minOccurs= 0 and maxOccurs="unbounded".
    - Default to xs:sequence: If selected, specifies to include the model details in the *xs:sequence* element. The sequence element specifies that child elements must appear in a sequence. Each child element can occur from 0 to any number of times. In the employee entity declaration that follows, the entity attributes are

included within the *xs:sequence* element. The employee attributes appear in the same sequence in which they are defined in the model.

```
<xs:element name="Employee" nillable="true">
<xs:complexType>
<xs:sequence>
<xs:element name="FirstName" type="xs:string"/>
<xs:element name="LastName" type="xs:string"/>
<xs:element name="JobID" type="xs:short"/>
<xs:element name="PublisherID" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
```

-
  - **Default to xs:choice**: If selected, specifies to include the model details in the *xs:choice* element. *xs:choice* specifies that only one of the elements contained in the *<choice>* declaration can be present within the containing element. In the employee entity declaration that follows, the entity attributes are included within the *xs:choice* element.

```
<xs:element name="Employee" nillable="true">
<xs:complexType>
<xs:choice >
<xs:element name="FirstName" type="xs:string"/>
<xs:element name="LastName" type="xs:string"/>
<xs:element name="JobID" type="xs:short"/>
<xs:element name="PublisherID" type="xs:string"/>
</xs:choice >
</xs:complexType>
</xs:element>
```

-
  - **Default to xs:all**: If selected, specifies to include the model details in the *xs:all* element. *xs:all* specifies that the child elements can appear in any order. In the employee entity declaration that follows, the entity attributes are included within the *xs:all* element.

```
<xs:element name="Employee" nillable="true">
<xs:complexType>
<xs:all >
<xs:element name="FirstName" type="xs:string"/>
<xs:element name="LastName" type="xs:string"/>
<xs:element name="JobID" type="xs:short"/>
<xs:element name="PublisherID" type="xs:string"/>
</xs:all >
```

</xs:complexType>
</xs:element>

- Include attribute defaults: If selected, specifies to include the attribute defaults as specified in Default tab of the Attribute Editor. For example, in the following attribute declaration, the default value is included.

<xs:element default="Toronto" name="Address" nillable="true" type="xs:string">

- Set the minOccurs based on the attribute (or view column) NULL option: If selected, sets the *minOccurs* value to 0 when the Nulls property of the entity attribute (or view column) is set to NULL.
  For example, in the OrangeMart.dm1 sample model, the Approved_Product entity in the logical model has *Therapeutic_equiv_code_ID* which allows NULLs. When the XSD for the attribute in this entity is generated with this option enabled, the *minOccurs* value is automatically set to zero similar to the following:

<xs:element maxOccurs="unbounded" minOccurs="0" name="Therapeutic_Equiv_Code_Id" nillable="true" type="xs:integer"/>
When the Nulls property of the entity attribute is set to *NOT NULL*, *minOccurs* is set to 1.

- Include Reference Values as Enumerations: Selecting this option suppresses enumerations that are created from list reference values. This way you can choose if the list reference values bound to attributes and domains should be in the generated XML file or not. When the option is selected, and a domain or attribute with a bound list reference value Is dragged over the schema body, only for simple types the enumerations will be created. In the same situation, if the same kind of object is dragged over the Elements category in the Schema body, only after creating a new local simple type of that element the enumeration of the reference value will be created. However, you can manually drag and drop the reference value over to an simple type if you want the enumeration."
- General Options tab: The General Options control the default mapping for attribute and domain datatypes as they are dropped into the Schema Body window on page 4 of the utility.
- Create local simple types by default: If selected, declares the local simple type under the *xs:element* where the element is defined. In the example that follows, the simple type, *xs:string* is declared for the *Description* element.

<xs:element name="PurchaseOrder">
<xs:annotation>
<xs:documentation>root element description</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:sequence>
<xs:element name="OrderDate" type="xs:date" minOccurs="1"/>
<xs:element name="Description" nillable="true">

```
<xs:simpleType>
<xs:restriction base="xs:string">
<xs:maxLength value="200"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
```

- Include Datatype Length: If selected, specifies to include the datatype length of attributes when elements and simple types are created from attributes.
- Default to global simple types: If selected, converts the datatype from the model to global
- Default to base datatypes: If selected, converts the datatype from the model to standard *xs:datatypes* using a datatype mapping dictionary, which is selected on page 5 of the wizard, on the Datatype Conversion tab.

```
<xs:element name="PurchaseOrder">
<xs:annotation>
<xs:documentation>root element description</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:sequence>
<xs:element name="OrderDate" type="xs:date" minOccurs="1"/>
<xs:element name="Description" type="xs:string" minOccurs="0"/>

</xs:sequence>
<xs:attribute name="OrderID" type="xs:date"/>
</xs:complexType>
</xs:element>
```

- Generation Order: Specifies the order of the list of nodes in the Schema Body in Page. Includes and Import will always be at the top. The order of the other nodes depends on the order option selected.
    - Most to least granular: Lists the nodes in the following sequence: Complex Types, Groups, Elements, and Simple Types.
    - Least to most granular: Lists the nodes in the following sequence: Simple Types, Elements, Groups, and Complex Types.

Datatype Conversion tab

The datatype dictionary converts datatypes using the dictionary created using the Datatype Mapping Editor. To select a datatype dictionary, type the dictionary name into the text box, or click the arrow to select a previously used dictionary.


Note: The naming dictionary is used to translate the names in the data model to the schema names. If a dictionary name is not provided, the other options on this dialog are used to transform the names.

Name Conversion tab

The name conversion options available in this dialog convert the model names to names that conform to XML standards. Although the utility can display all the data model names, some of the model names may not conform to XML standard naming conventions. For example, XML schemas cannot have element names that start with anything except a character, so "123Customer" would be invalid.


Note: The XML Schema Generation utility will error on the side of producing valid XSD. If there are any invalid characters in the names after the NST options are applied, the utility will automatically remove them.

- **Select Naming Standards Template**: To select a Naming Standards Template, type the dictionary name into the text box, or click Select External and browse to the location of the dictionary. The Naming Standards Template is used to translate the names in the data model to the schema names. If a dictionary name is not provided, the other options on this dialog are used to transform the names. If a template name is provided, the other options on this dialog become unavailable. Click Next to select the object to include in the schema. For information on the Naming Standards Template, see Copying and Pasting Data Dictionary Objects.
- **Case Conversion Options**: The following describes the case conversion options available and the results achieved by applying them:

Case conversion options	*Data model name*	*Converted name (spaces removed)*
Upper Camel	CUSTOMER ADDRESS CODE	CustomerAddressCode
Lower Camel	CUSTOMER ADDRESS CODE	customerAddressCode
Upper	Customer	CUSTOMER
Lower	CUSTOMER	customer
Preserve	CUSTOMER ADDRESS CODE	CUSTOMERADDRESSCODE

Page 4

On this page you create the schema, select the model objects, their grouping in the schema, and the order in which they appear in the schema.

The model objects from the active model populate the Available Objects list. All domains and reference values are included from the data dictionary and are organized by dictionary when multiple dictionaries exist in the data model.

If the schema includes entity definitions and attributes, changes to the property values for a selected object appear in the schema only. Changes made to the property values on this dialog do not affect the properties of the model.


Note: The settings of the minOccurs and maxOccurs properties of Schema Body elements reflects the Allow Nulls setting on the Datatype tab of the Entity Editor.

To create the schema, reference the following procedures:

- Include External Schemas.
- Import Referenced Namespaces.
- Add Domains and Attributes to the Simple Types Section
- Add Reference Values to a Domain
- Add Domains, Attributes, and Entity Objects to the Elements Section
- Add Domains, Attributes and Entities to the Groups Section
- Add Complex Entity Objects to the Complex Types Section
- Creating xs:attributes from Attributes or Columns
- Change the Property Values of Schema Objects
- Change the Object Order in a Schema Body Section
- Delete an Element from a Schema Body Section.
- Add a Composite to an Element.
- Add a Group Reference to an Element.
- Change the Type of an Existing Composite Type.
- Preview the Schema.
- Creating SQL to Run Before and After Generating a Physical Database.
- When you have finished creating the schema, click *Finish*.

The XML schema is generated and saved to the target directory and file name specified on page 1 of the Wizard.

Include External Schemas

The include declaration includes an external schema in the target namespace of the schema containing the include declaration

1. In the **Schema Body** section on page 4 of the XML Schema Generation Wizard, right-click *Includes* and then select *New*.
2. Type a name for the include declaration, and click *OK*.

The new include declaration appears highlighted within the Includes folder of the Schema Body tree and its properties display in the properties pane below.

1. In the **Property Value** column of the properties pane, click the name of the include declaration and then type the URI of the schema.
2. To create more include declarations, repeat steps 1 to 3.

**Note:** You can later edit the name of the include declaration or delete it by right-clicking its name under the Includes node and then selecting Edit or Delete.

Import Referenced Namespaces

The import declaration identifies a namespace whose schema components are referenced by the schema containing the import declaration.

**Note:** Use the import declaration to declare any namespaces not previously identified in the Schema Attributes on page 2 of the XML Schema Generation Wizard.

1. In the **Schema Body** on page 4 of the XML Schema Generation Wizard, right-click **Import** and then select **New**.
2. Type a name for the import declaration and then click **OK**.
   The new import declaration appears highlighted within the Import folder of the Schema Body tree and its properties display in the properties pane below.
3. In the **Property Value** column of the properties pane, click the space next to the **schemaLocation** property and then type the URI of the schema document containing the imported namespace.
4. In the **Property Value** column of the properties pane, click the name of the **namespace** declaration and then type the URI of the schema containing the namespace to import.
5. To create more import declarations, repeat steps 1 to 3.

**Note:** You can edit the name of the import declaration or delete it by right-clicking its name under the Import node and then selecting Edit or Delete.

Add Domains and Attributes to the Simple Types Section

Domains can be included in the Simple Types or Complex Types sections of the Schema Body.

**Note:** Dragging a domain that contains a subdomain into the Simple Types node does not include any of its subdomains in the schema.

From the list of Available Objects, click a simple domain or attribute and then drag it to the Simple Types node in the Schema Body pane.

The Simple Types node expands to show the newly added object.

**Note:** If the target domain is for the simple type added, you can select the schema element, scroll through the properties, and change the default type and type prefix by clicking

the Property Value list. The available namespaces are dependent on the namespaces declared on page 2 of the wizard.

Add Reference Values to a Domain

Drag and drop the reference value from the list of **Available Objects** onto the object in the **Schema Body**.

The reference values for the object appear as a property of the object in the **Schema Body**.

Notes:

- For information on reordering objects in Schema Body nodes, see Change the Object Order in a Schema Body Section
- For information on changing the property values of objects in Schema Body nodes, see Change the Property Values of Schema Objects.
- To preview the schema, click *Preview* or for more information, see Preview the Schema.

Add Domains, Attributes, and Entity Objects to the Elements Section

From the list of *Available Objects*, click a domain, attribute or entity object and then drag it to the *Elements* node in the *Schema Body* pane.

The *Elements* node expands to show the newly added objects.

Notes

- You cannot drag reference values onto domains in the Elements node; instead, use the Simple Types node.
- You can drag a simple entity or a complex entity, with all their attributes, to the Elements node and then drag other attributes to its choice, sequence, or all attribute container.
- To convert an attribute of an element to an element, drag the attribute out of the complexType container.
- To convert an element to an attribute of another element, drag the element into the complexType node.
- An element defined in the elements node may have a complex type underneath it. You can insert the complex type under the element and insert composites under that.
- If you drag an attribute to the complex type node under an element, it is included within the *xs:attribute* container below the composite.
- If you drag an attribute to a composite node, it is included within the *xs:element* container below the composite.

Add Domains, Attributes and Entities to the Groups Section

From the list of **Available Objects**, click a domain, attribute or entity and then drag it to the *Groups* node in the Schema Body pane.

The Groups node expands to show the newly added object.

Notes:

- Use the Groups node to define groups that another part of the schema, such as an element can reference.
- A group may have a composite underneath it just like an element or complex type.

Add Complex Entity Objects to the Complex Types Section

From the list of **Available Objects**, select a complex object, such as a domain or entity, and then drag it to the Complex Types node in the Schema Body.

The Complex Types node expands to show the newly added object.

Notes

- If the target domain is for the simple type added, you can select the schema element, scroll through the properties, and change the default type and type prefix by clicking the Property Value list. The available namespaces are dependent on the namespaces declared on page 2 of the wizard.
- Use the Complex Types node to define standalone complex types for later reuse.
- Typically, complex types define complex structures that elements in the element node can reference.
- Complex types promote reuse in an XSD.

Creating xs:attributes from Attributes or Columns

You can add attributes or columns to the xs:attributes declaration using either of the following methods:

- Drag an attribute/column from the Data Model tree to a Complex type node (not to the Sequence node underneath.)
- Drag an element from under a sequence node (within a Complex type) to the Complex type node above.

Using either of these methods, you should see the icon next to the attribute/column change to an "a", indicating that it is an xs:attribute.

Change the Property Values of Schema Objects

1. In the **Schema Body**, select the object.
2. In the property pane, change the property values by clicking in the space in the **Property Value** column next to the property name and then typing the new value.

Change the Object Order in a Schema Body Section

The object order in the schema body determines the order of elements in the generated schema. To convey relationships and improve schema readability, you may want to reorder object after dropping them into schema body nodes.

- To place an object first in the list of objects in that container, drag the object directly on top of the *Schema Body* node.
- To place an object below another object, in the Schema Body node drag the object on top of an object or the composite container below that object.
- To select a contiguous list of objects and drag them into the same node in the Schema Body, select the first object in the list, then press Shift and click the last object in the list, and finally drag the group of objects to a node in the Schema Body.
- To drag multiple objects into the same node in the Schema Body, select an object, press and hold CTRL while selecting more objects, and then drag the group of objects to a node in the Schema Body.


Note: Objects cannot be moved between nodes by dragging. If required, delete the object and then drag it from the list of Available Objects to another node in the Schema Body.

Delete an Element from a Schema Body Section

In the **Schema Body**, right-click the element name and select **Delete Element**.

The element and all subelements are deleted from the **Schema Body** node.

Add a Composite to an Element

Use composites in a schema to group elements together.

The following composite types are supported:

- xs:sequence
- xs:choice
- xs:all

The composite type node appears below the *complexType* container.

1. In the *Schema Body*, right-click the element name and then select *New Local complexType*.
2. Right-click the *Complex Types* node just added, select *New* and then select the type of composite you want to add.

You can now drag other objects to the composite node.


Note: To create a composite type node within an existing composite node, right-click the composite node, select New and then select the type of composite you want to add.

Add a Group Reference to an Element

You can add a reference to group to an empty composite node.

1. In the **Schema Body**, right-click the element name and then select **New Local complexType**.
2. Right-click the **Complex Type** node just added and then select **New Group**.
3. In the property box, click the space in the **Property Value** column that is next to the **Property Name ref** and then type the name of the group or click the arrow and select the name of the group to reference.

Change the Type of an Existing Composite Type

Once you have added a composite type to the Schema Body node, if desired you can change its type.

Composite node names can be.

- xs:sequence
- xs:choice
- xs:all

Right-click the composite node, select Convert to and then select the new composite type.

Preview the Schema

1. To preview the schema, click *Preview*.

A preview of the generated schema appears.

1. When you are finished with the preview, click *Close* to return to the previous dialog where you can continue to change the format and contents of the schema.
2. If necessary, click *Back* to return to a previous dialog and change the options.

Export UML Types

The following table illustrates the various database types and the UML types to which they map:

Database Type	UML Type
BIGINT	long
BINARY	byte
BIT	bool
CHAR	string
COUNTER	int
DATE	string
DATETIME	string
DATETIMN	string
DECIMAL	float
DECIMALN	float
DOUBLE PRECISION	double
FLOAT	float
FLOATN	float
IMAGE/LONG BINARY	byte
INTEGER	int
INTN	int
LONG VARCHAR	string
MLSLABEL/VARCHAR	string
MONEY	float
MONEYN	float
NCHAR	string
NTEXT/LONG NVARCHAR	string
NUMERIC	float
NUMERICN	float
NVARCHAR	string

PICTURE	byte
REAL/SMALLFLOAT	float
ROWID/VARCHAR	string
SERIAL/INTEGER	int
SMALLDATETIME	string
SMALLINT	short
SMALLMONEY	Float
TEXT	string
TIME/DATETIME	string
TIMESTAMP/DATE	string
TINYINT	short
UNIQUEID	int
VARBINARY/BLOB	byte
VARCHAR	string
VARIANT	char

See Also

- Exporting an Image of the Data Model
- Printing the Data Model
- Exporting the Data Model

Generating a Script File or Database

The DDL Generation Wizard lets you generate the DDL to build a database. You can use this wizard anytime you want to create a database from a physical model. The wizard guides you through the process of building a database and generating the SQL source code for the database without requiring you to know any of the underlying commands.

ER Studio Data Architect lets you generate DDL to update an existing database or you can create a brand new database. The wizard lets you select database objects specific to your platform and the options associated with those objects.


Note: The options of the wizard vary slightly depending on the database platform you are using.

Generate a script file or database

1. On the *Data Model Explorer*, select a physical model and then choose Database > Generate Database.

   
   Tip: If you select diagram objects on the data model before opening the wizard, the selected object are selected in the object tree within the wizard, saving you from re-selecting them.

2. Follow the prompts of *DDL Generation Wizard* as it guides you through the rest of the process.

The following helps you complete the options of the **DDL Generation Wizard**:

Page 1 Page 2

Tables tab General Options tab

Tips Notes

Page 1

Generate a Single, Ordered Script File: If selected, writes a single SQL file with specification of all objects.

Generate Objects To Multiple Files: If selected, writes a separate SQL file for each object selected. Objects are ordered in subdirectories for each selected object class.

Database Creation Script File or **Base Directory**: If you selected **Generate a Single, Ordered Script File**, this control lets you provide the filename and path for the SQL file generated. If you selected **Generate Objects To Multiple Files**, this control lets you specify the directory that is to contain the generated script files.

Generate Objects with a Database Connection: If selected, lets you create the selected objects in a database you're connected to. Clicking **Connect** opens the **Database Connection** Dialog Box where you can input your database connection information. Three methods are available:

- Generate Database Objects Using ODBC and Generate Database Objects Using Direct Connect - Lets you select a generic ODBC or native driver connection and provide relevant connection information and credentials.
- Generate Database Objects Using ER/Studio Team Server - If logged in to ER/Studio TS, you can choose a database type, select a datasource, and provide credentials. If not currently logged in to ER/Studio TS, you are prompted to log in before proceeding.

Unless you choose to include DROP Statements in the Generation Options, you'll receive DDL errors when ER Studio Data Architect tries to create an object that already exists in the target database. The Generation Options are on the tabs where you select which objects in the data model for which you'd like to build a database or generate SQL scripts.

If you selected Generate Database Objects Using ODBC or Generate Database Objects Using Direct Connect, the Create ER/Studio Team Server Data Source from current settings controls let you have a new datasource definition created on ER/Studio TS using the setting you provide on this wizard and provide a data source Name. The Bind Data Source to Model control, available only for repository models, lets you bind the current model to the new data source.

Wizard Quick Launch: These controls let you preload wizard settings from a previously saved session.

Page 2

Select (use) an Existing SQL Server 2005 Database: This option is available when Select or Create a Database is selected. If you choose to update an existing database, make sure to choose to include DROP Statements in your Table Generation Options; otherwise, you will receive DDL errors when ER Studio Data Architect tries to create an object that already exists in the target database. The Generation Options are on the tabs where you select which objects in the data model for which you would like to build a database or generate SQL scripts.

Tables tab

Use Complete Syntax for Virtual Columns: This option is available for Oracle 11g and later. When this option is not selected, virtual columns will be added to the DDL with the following syntax:

column [<datatype>] AS (<column_expression>)

where *<datatype>* is set to the data type of the column if the Show Datatype option is on for the column and is omitted otherwise, and <column_expression> is the expression stored as the virtual column expression.

When this is selected, virtual columns will be added to the DDL with the following syntax:

column [<datatype>] GENERATED ALWAYS AS (<column_expression>) VIRTUAL

General Options tab

Generate Object Creation Verification Code: Available for SQL Server and Sybase database platforms. When selected, this option generates code similar to the following for tables, views, indexes, procedures, and functions.

IF OBJECT_ID('Entity1') IS NOT NULL

PRINT '<<< CREATED TABLE Entity1 >>>'

ELSE

PRINT '<<< FAILED CREATING TABLE Entity1 >>>'

go

Tips:

- If you right-click anywhere on the object tree within the wizard, you can select or deselect all objects using the short-cut menu. You can also display or hide object owner names for those objects with owners.
- To view the DDL generated by the currently selected settings, click the SQL Preview button on the wizard at anytime. You can print or save the SQL script directly from the SQL Preview dialog.
- You can change the object types selected by default in the Generate Other Object Types area of the General Options tab on page 3 of the wizard; click your preferences in Tools > Options > Object Types.
- **NUMBER datatype**: NUMBER or NUMBER(*) you must set the width and scale to *. A width of 0 will be converted to * when generating DDL.

See Also

- Database Support and Connectivity
- Connecting to Database Sources and Targets
- Saving and Using Quick Launch Settings
- Exporting an Image of the Data Model
- Printing the Data Model

Creating SQL Procedures to Run Before and After Generating a Physical Model View

You can create PreSQL and PostSQL that ER Studio Data Architect runs before generating a view in the physical model when you choose the Database > Generate Database command and then click **Finish** in the wizard. The PreSQL and PostSQL you enter in the View Editor can be included in the SQL generated before and after the CREATE VIEW statement that creates the view.

1. Double-click a view to which you want to associate an SQL procedure.
2. In the **View Editor**, click the **PreSQL & PostSQL** tab.
3. Enter, or copy and paste the PreSQL and PostSQL onto their respective tabs.
4. If you want the PreSQL and PostSQL scripts to be generated, select **Generate**.
5. Click **OK**.

See Also

- Creating SQL to Run Before and After Generating a Physical Database
- Creating and Editing SQL Procedures

Creating SQL Procedures to Run Before and After Generating a Table

You can create PreSQL and PostSQL that ER Studio Data Architect runs before generating a table in the physical model when you choose the Database > Generate Database command and then click *Finish* in the wizard. The PreSQL and PostSQL you enter in the Table Editor can be included in the SQL generated before and after the CREATE TABLE statement that creates the table.

1. Double-click the table you want to associate the SQL procedure with.
2. In the *Table Editor*, click the *PreSQL & PostSQL* tab.
3. Enter or copy and paste the PreSQL and PostSQL onto their respective tabs.
4. If you want the PreSQL and PostSQL scripts to be generated, select *Generate*.
5. Click *OK*.

See Also

- Creating SQL to Run Before and After Generating a Physical Database
- Creating and Editing SQL Procedures

Creating SQL to Run Before and After Generating a Physical Database

You can create PreSQL and PostSQL procedures that ER Studio Data Architect runs before generating a physical database from the physical model when you choose the Database > Generate Database command and then click *Finish* in the wizard. Using the PreSQL and PostSQ of the Emergency Admissions sample model as an example, you can create users and assign them specific permissions before generating the database and then create a materialized view, after generating the database.

1. In the *Data Model Explorer*, select a physical data model and then choose Model > Model Properties.
2. Select the *PreSQL & PostSQL* tab.
3. Enter, or copy and paste the PreSQL and PostSQL on their respective tabs.
4. If you want the PreSQL and PostSQL scripts to be generated, select *Generate*.
5. Click *OK*.

See Also

- Creating SQL Procedures to Run Before and After Generating a Table
- Creating and Editing SQL Procedures
- Creating SQL Procedures to Run Before and After Generating a Physical Model View

Exporting an Image of the Data Model

From ER Studio Data Architect, you can create a screenshot image of your data models to distribute to your reviewers. You can choose from several different image formats, including EMF, JPEG, GIF, BMP, and PNG.

1. In the *Data Model Explorer*, select the physical or logical model, or the submodel that you want to capture as an image.
2. Choose File > Export Image.
3. Choose your options in the *Save As Image* dialog and click *OK*.


Note: When choosing the image characteristics, be aware that the lower the resolution, the smaller the file size will be. The higher the resolution, the bigger the file size will be.

See Also

- Exporting the Data Model
- Printing the Data Model

Printing the Data Model

From ER Studio Data Architect, you can print your data models in monochrome or color to distribute to your reviewers. You can use the Print dialog box to set specific print parameters such as, printing specific sections of your diagram, diagram appearance, page range, zoom level, and border specifications. The Print dialog also has navigational tools that let you navigate your diagram to select the print area. You can save your print settings to reuse at a later time.

- Before printing your model, display the page boundaries in the Data Model Window so you can optimally place your objects for better viewing and printing. Choose Tools > Options > Display > Page Boundaries.

Print the data model

1. In the *Data Model Explorer*, select the physical or logical model, or the submodel that you want to print.
2. Choose File > Print.
3. Choose the options desired in the *Print* dialog and then click *OK*.
4. To enable reviewers who do not have access to ER Studio Data Architect to view your models, in addition to printing the model you can also create an XML Schema, save the model and distribute it with ER Studio Data Architect Viewer, create RTF and HTML reports about the model, export the model to another format to be opened in a different application, or create an image of the model.

See Also

- Generating RTF and HTML Model Reports
- Exporting the Data Model
- Exporting an Image of the Data Model

Working with ERStudio Repository

- The Developer Edition of ER Studio Data Architect does not support ER/Studio Team Server access. Users of the Developer Edition will not be able to log in to a Repository Server or access the server in any way.

The Repository provides access to diagrams for more than one person at a time. Modelers can update or delete model information simultaneously by working with ER Studio Data Architect in the normal fashion and by merging their changes into a single model.

To use Repository operations such as check in, check out, and get diagram, you must first specify the Repository connection details and then log in to the Repository. To explore the Repository, you need to specify the ER/Studio Team Server server connection details and then log in to ER/Studio TS.

- During the Repository Installation, a default user name and password *Admin* is set. The Admin user name and password has Super User privileges, which means Admin has all available Repository Privileges. The Admin user should secure the Repository by creating users and roles. For more information, see Establishing Security for the Repository in the *ER/Studio Repository Installation and Administration Guide*.

When you are finished working with the Repository, you should log out. If you close ER Studio Data Architect, you are automatically logged out of the Repository.

This section provides usage information for Repository Users.

- Repository architecture, setup, maintenance, control, and query options are covered in the *ER/Studio Repository Installation and Administration Guide*.

Items covered in this section include:

- Specifying Repository and ER/Studio TS Options
- Logging In and Out of the Repository and ER/Studio TS
- Synchronizing Comments
- Multi-threading in the Repository
- Establishing Security for the Repository
- Working with Objects in the Repository
- Working with Data Dictionaries in the Repository
- Working with Repository Projects
- Cancelling a Repository Operation
- Best Practices

See Also

- ER/Studio TS wiki
- Administrator Guide

Specifying Repository and ER/Studio TS Options

Using configurable Repository settings you can specify and modify server particulars, Repository display settings, check out policy, and communication settings. There are two dialogs that control Repository behavior, Setting Repository and ER/Studio TS Options and Repository Properties. The ER/Studio Team Server settings are also defined in the **Repository Options** dialog.

See Also

- Setting Repository and ER/Studio TS Options
- Repository Properties
- ER/Studio TS wiki

Setting Repository and ER/Studio TS Options

1. Log in to the Repository.
2. Choose Repository > Options.
3. Specify your changes on the *Repository Options* tabs and then click *OK* to exit the editor.

The following helps you complete the **Repository Options**:

- General tab
- Check Out Policy tab
- ER/Studio Team Server tab
- Notes

General tab

Specify all Repository server settings.

- **Server Machine**: Displays the host name of the ER/Studio Repository. Click the Select button to list Previously Used Repositories and click Refresh to update the Repository list with the host names of all Repositories found on the network.
- **Refresh**: Click to refresh the list of Repository servers available to you on your network.
- **Active File Directory**: The directory to where files checked out or retrieved from the Repository are saved. The **Active File Directory** should be a directory on the local machine as opposed to a shared directory on the network; a shared network directory can result in Repository operation errors.
- Keep the Active File Directory on a local machine. A shared network directory can result in Repository operation errors.
- **Hide Repository Status Icons**: If selected, does not show the Repository Status icons, such as the lock, in the Data Model and Data Lineage explorers, and Data Model and Data Lineage windows. This option is useful when you want to print the Diagram without the icons displayed.
- **Show Repository Status Dialog**: Indicate whether you want to display or hide the Repository Status dialog box, which shows timing information for each step taken during a Repository operation.
- **View SQL Alerts**: If selected, the Repository Merge operation will display an alert during check-in to warn the user when multiple users have modified data used by a view. When the SQL alert is displayed, there will be an option to not show such messages. Using the *View SQL Alerts* option, you can re-enable these messages.

- **Enable Repo Events Polling**: Repository Events Polling returns information about Repository operations so that the Diagram or Objects' status are updated with the most current information available. If you enable Repo Events Polling, you can then determine the time interval at which the polling occurs. The default is enabled.
- **Repo Events Polling Interval**: The number of seconds between polls to synchronize the status of local diagrams and objects with the Repository.
- **Hide General Information**: For performance reasons, you may want to select this option which prevents the Repository status information from displaying in the **General Information** area of the **Get From Repository** dialog.
- **Display Synchronize Comments Warning**: When selected, this option will prompt the user to save any changes to the file before synchronizing the comments with the Repository. When not selected, the system automatically saves the file before synchronization.
- **Diagram Name Display Format**: Determines how the diagram name displays in Repository dialogs such as Get from Repository. If you choose, **Diagram Name (Filename.dm1)** the diagram name displayed might be Demo1 (Model.dm1) where Demo1 is the name field of the **Diagram Properties** dialog, which is accessed by choosing File > Diagram Properties.

Check Out Policy tab

Lets you change the properties for the check-out policy. For example, you can specify that you want a prompt when the user tries to edit a Repository item that is not checked out.

- **ER Objects** area: Determines how to store the core data, such as names and definitions of the model objects.
- **Layout Objects** area: Determines how to store the entity object's display or graphic information, such as the size, color, and placement of Objects.

  Immediate: The immediate Check Out policy requires an established server connection and a user login. When you attempt to edit an object that is not checked out, you are prompted to check out the object or to open the object in an editor in view, or read-only, model.

  Delayed: If Delayed and Confirmation Required are selected, when you attempt to edit an object that is not checked out, you are prompted to check out the object. If you select Yes in the Confirmation dialog or if Confirmation Required has not been selected, ER Studio Data Architect marks the object with an open red lock, indicating that check out is local, instead of an open green lock, which indicates that the object has been checked out from the Repository and is marked as checked out for all Repository users to see. The Repository is not informed of the objects checked out status and the actual check out does not occur until you run the command Repository > Diagrams > Run Delayed Check Out, at which time, you may need to resolve any conflicts with changes checked in by other users.

ER/Studio Team Server tab

URL: Lets you specify connection details to the ER/Studio Team Server server which allows you to specify the name of the server on which ER/Studio TS is running, such as datojt090, or the URL for the ER/Studio TS login, such as http://datojt090:8080.

Notes

- When you add or check in files to the Repository, you should do so to the Active File Directory.
- Consider your installation configuration when configuring your Repository Server Settings.
- The Repository database does not need to reside on the same machine as the Repository server, and when you are configuring your Repository server, the Repository database is not affected.

See Also

- Repository Properties
- Logging In and Out of the Repository and ER/Studio TS

Repository Properties

1. To view only Repository Properties, log in to the Repository.
   To change Repository Server Ports, log out of the Repository.
2. Choose Repository > Repository Properties.
3. Click the *Server* tab to view the server settings and then the *User* tab to view the log in name of the current user.
4. If necessary, when not logged in to the Repository, you can change the ports your system uses to communicate with the Repository. On the *Server* tab, click *Edit Ports*, change the port settings, click *Apply Changes*, and then click *OK* to exit the editor.

SSL Port: Since version 8.5, ER Studio Data Architect supports Secure Sockets Layer (SSL) communication protocol which encrypts messages between the client and Repository. You must select **Use SSL connection to Repository** before SSL will be used to encrypt ER Studio Data Architect network messages.

See Also

- Setting Repository and ER/Studio TS Options
- Logging In and Out of the Repository and ER/Studio TS

Logging In and Out of the Repository and ER/Studio TS

Log in to the Repository or ER/Studio Team Server

- To log in to the Repository, choose Repository > Log In.
- To log in to ER/Studio Team Server, choose Repository > Team Server Log In.
- Complete the log in dialog and then click *OK*.

The following describes the options that require additional explanation:

User Name: Enter your user name. This is case-sensitive.

Password: Enter your password. This is case-sensitive. You can change your password by selecting Repository > Security > Change Password.

Log in using current Windows Account: Select this option to log in to the Repository or ER/Studio TS using the user name and password of the current Windows user. This option accesses the LDAP server on the network to verify the user's credentials.

Keep me signed in: Select this option to remain logged in across shutdowns and startups of ER Studio Data Architect.

Notes

- During the Repository Installation, a default user name and password *Admin* is set as the default. The Administrator should change the Admin password after the initial log in.
- The Admin user cannot be deleted. This prevents an accidental lockout. You can set up additional users with Roles and Privileges to maintain Repository security.
- If an administrator changes the repository security, users must log in again to update their permissions.

Log out of the Repository or ER/Studio Team Server

1. To log out of the Repository, choose Repository > Log Out.
2. To log out of the ER/Studio Team Server, click **Repository > Team Server Log Out**.
3. In the **Repository Operation Status** dialog, select the desired options and operations and then click **Close**.
4. If you select **Do Not Display Dialog Again** and later want to see the Repository Operation Status dialog, you can re-enable it in Repository Options. For more information, see Setting Repository and ER/Studio TS Options.

See Also

- ER/Studio TS wiki
- Setting Repository and ER/Studio TS Options

Synchronizing Comments

You can add comments to any object in ER Studio Data Architect that are saved with the document and uploaded to the Repository. When adding a comment you have the option to immediately update the Repository or if you like you can wait and update the Repository with all your new comments at once. Synchronizing comments also ensures your model is updated with any comments other users have made. You do not have to check out an object in order to append a comment to it.

Choose Repository > Synchronize External Data.

- You can choose to be notified when comments are added by selecting Show Comment Notifications on the Tools > Options > Comments tab.

See Also

- Adding Comments to Database Objects
- Adding and Editing Object Comments

Multi-threading in the Repository

As of ER Studio Data Architect version 9.0, the Repository takes advantages of new multi-threading functionality that improves Repository performance over the previous sequential method used to handle transactions between the Repository server and the ER Studio Data Architect client. Mutli-threading also improves Repository performance with ER/Studio Business Architect in that you can log in or do some security operations concurrently like you can with ER Studio Data Architect.

Multi-threading allows the server to process multiple transaction concurrently. As long as a given transaction does not conflict with any currently processing transaction, the server opens a new thread for the new transaction and processes it immediately. For example, transactions necessary for User A to log in can be processed at the same time as User B is adding a diagram, and User A will be logged in just as quickly as if User B was not in the middle of adding a diagram.

However, not all transactions can be run concurrently. This means that in some cases, an operation must wait for a transaction initiated by a different user to complete. In addition, some transactions listed on the following table as being able to run concurrently might be blocked by the Repository database's own concurrency mechanisms if the same table is being accessed at the same time by different transactions.

The following describes some important rules that affect multi-threading on the Repository:

- If a Check In transaction is being processed on a given Diagram or Data Dictionary, all other operations on that Diagram or Data Dictionary must wait until the Check In completes.
- If an Add or Check In transaction is being processed, all other Add or Check In transactions must wait until the initial transaction completes. This same rule applies to Set Named Release, Branch and Branch/Merge Check In operations.
- Most other common tasks, including Log Ins, Log Outs, Gets, Check Outs and Security Center operations are processed concurrently by the server, but may be affected by database level-locking mechanisms.

The following describes concurrency restrictions:

- Multiple, concurrent '*Get* access on the same object (such as user, role, diagram, and dictionary) are allowed.
- Only one **Write** access on a particular object allowed at one time (for example, a delete table and a modify table will not run concurrently on the same table).
- Writes and Gets on a particular object will also be serialized (for example, a Get Diagram and a Check In Diagram will not run concurrently on the same diagram).

The following table describes which transactions can run simultaneously.

- **ER Studio Data Architect Operation** is the operation that invokes the Server Transaction.

- **Server Transaction** is the request that is seen by the Repository Server.
- The **Concurrency** columns show which Server transactions can run simultaneously with the process in the respective Server Transaction column.

ER Studio Data ArchitectOperation	Repository ServerTransaction	Concurrency				
		Get Latest Diagram(Dictionary)	Add Diagram(Dictionary)	Check In Diagram(Dictionary)	Security,Project Operations	Undo/Redo/ Check OutDiagram (Dictionary)
Add Diagram	Add Diagram	√	√	√		
	Add Dictionary	√			√	√
	Bind Dictionary to Diagram	√	√	√	√	√
	Modify Project	√	√	√	√	√
Add Enterprise Data Dictionary	Add Dictionary	√			√	√
	Bind Dictionary to Diagram	√	√	√	√	√
Bind Enterprise Data Dictionary	Bind Dictionary to Diagram	√	√	√	√	√
Branch	Branch	√			√	√
Branch/Merge Dialog	Get Branch Tree Info	√	√	√	√	√
Branch/Merge operation	Merge Check in Diagram	√			√	
	Merge Check In Dictionary	√			√	
	Merge Get Latest Diagram	√	√	√	√	√

	Merge Get Latest Dictionary	√	√	√	√	√
Change Diagram File Name	Change Diagram File Name	√	√	√		
Change Own Password	Change Own Password	√	√	√	√	√
Check In Data Dictionary	Check In Dictionary Objects	√			√	
	Get Latest Diagram	√	√	√	√	√
	Get Latest Dictionary	√	√	√	√	√
Check In Diagram	Check In Diagram	√			√	
	Get Latest Diagram	√	√	√	√	√
	Get Latest Dictionary	√	√	√	√	√

Check In Diagram Objects	Check In Diagram Objects	√			√	
Check Out Diagram	Check Out Diagram	√	√		√	
Check Out Diagram Objects	Check Out Diagram Objects	√	√		√	
Check Out Dictionary	Check Out Dictionary	√	√		√	
Delete Diagram	Delete Diagram	√			√	√
Delete Dictionary	Delete Dictionary	√		√	√	√
Delete Named Release	Delete Named Release	√	√	√	√	√
Enterprise Dictionary Bindings Dialog	Get All Dictionaries	√	√	√	√	√
	Get Dictionary Binding Info	√	√	√	√	√
	Get All Dictionary/Diagram Bindings	√	√	√	√	√
Get from Repository (Diagram)	Get Latest Diagram	√	√	√	√	√
	Get Latest Dictionary	√	√	√	√	√
Get from Repository (Submodel)	Get Latest Dictionary	√	√	√	√	√
	Get Latest Submodel	√	√	√	√	√

Get from Repository Dialog	Get Structural Info	√	√	√	√	√
	Get Submodel Info	√	√	√	√	√
Get Latest Version	Get Latest Diagram	√	√	√	√	√
	Get Latest Dictionary	√	√	√	√	√
Get Named Release	Get Dictionary Named Release	√	√	√	√	√
	Get Name Release	√	√	√	√	√
Get Named Release Dialog	Get All Named Releases	√	√	√	√	√
Log In	Get My Security Info	√	√	√	√	√
	Get All Projects	√	√	√	√	√
	Log In	√	√	√	√	√
Log Out	Log Out	√	√	√	√	√
Project Center	Add Project	√	√	√	√	√
	Delete Project	√	√	√	√	√
	Get All Projects	√	√	√	√	√
	Get Project Info	√	√	√	√	√
	Modify Project	√	√	√	√	√

Redo Check Out Diagram	Redo Check Out Diagram	√	√		√	
Redo Check Out Dictionary Objects	Redo Check Out Dictionary Objects	√	√		√	
Redo Check out Objects	Redo Check out Objects	√	√		√	
Security Center	Add Role	√	√	√	√	√
	Add User	√	√	√	√	√
	Delete Role	√	√	√	√	√
	Delete User	√	√	√	√	√
	Force Check In	√	√	√	√	√
	Force Log Out	√	√	√	√	√
	Get All Diagrams for Role	√	√	√	√	√
	Get All Object Level Security	√	√	√	√	√
	Get All Object Privileges	√	√	√	√	√
	Get All Privileges	√	√	√	√	√
	Get All Roles	√	√	√	√	√
	Get All User Level Security	√	√	√	√	√
	Get All Users	√	√	√	√	√
	Get Repository Security Info	√	√	√	√	√
	Get Role Info	√	√	√	√	√

	Get User Checked Out Info	√	√	√	√	√
	Get User Info	√	√	√	√	√
	Get User On Machine Info	√	√	√	√	√
	Get User Security Info	√	√	√	√	√
	Get User Status	√	√	√	√	√
	Modify Role	√	√	√	√	√
	Modify Security Info	√	√	√	√	√
Security Center...	Modify User	√	√	√	√	√
	Modify User Security Rights	√	√	√	√	√
Set Named Release	Set Dictionary Named Release	√			√	√
	Set Named Release	√			√	√
Synchronize Comments	Synchronize Data	√	√		√	√(no concur-rent Syncs)
Undo Check Out Diagram	Undo Check Out Diagram	√	√		√	
Undo Check Out Dictionary	Undo Check Out Dictionary	√	√		√	
View Object History	Get History	√	√	√	√	√

See Also

- Synchronizing Comments
- Logging In and Out of the Repository and ER/Studio TS

Establishing Security for the Repository

Security of the Repository contents is managed by the Repository Administrator through the Security Center by creating and then assigning users particular Groups, Roles and Privileges.

**Note:** To prevent an accidental lockout, the Admin user cannot be deleted.

For example, we create a user JimB, give him a Role and call it DBA. The DBA role has many privileges, but none allow the DBA to modify a logical model. A Repository administrator can associate JimB with any model in the Repository and he will not be able to modify the diagram's logical model, but he can view it.

The Administrator can restrict user and group access to Repository items at the project, diagram, model, submodel, and data dictionary levels. To restrict access to the Repository, the Administrator creates users and then assigns the users a default role or assign, which grants specific permissions to the user, or creates new roles to assign to users. The Administrator can alternatively create a group of users and assign a role to that particular group. In these ways, the Administrator can grant users full access to some Repository items and restrict or deny access to others items.

Before creating users, groups,and roles, the Administrator should understand the following Repository concepts:

- **Cascading Security**: To make it easy for Administrators to assign a role globally to users when there are many diagrams in the Repository, such as when the user must have the same access permissions to many of diagrams, the Administrator can assign a user and role to various 'levels' of the ER/Studio Repository.

For example, a Project can be created in the ER/Studio Repository that contains many diagrams. If you assign a user a role to this Project, any diagrams contained within the Project will be granted the same permission set. The Repository itself can act as this highest-level point to assign permissions that are cascaded down.

Higher levels can be overwritten by assigning the same user different roles at a lower level, for example UserA may have the Viewer role at repository level but Modeler role for a specific diagram or submodel.

- **Client Side Security Caching**: To promote the concept of detached modeling collaboration (e.g. being logged out of ER/Studio Repository and possibly detached from the network), all security associated with the user by diagram is cached on the client when the user logs into ER/Studio Repository.

**Note:** When security changes are made, users must re-login to update their permissions.

- **Super User**: Super User is a default Role created upon installation of ER/Studio Repository. The default 'Admin' user is assigned to the Super User role. This user can do anything to diagrams, users and roles managed in the ER/Studio Repository as it is granted all privileges. It is not removable or editable, and is only found at the 'Repository' level of the Repository Security area of the Security Center.

- No Access: *No Access* is a role that can be applied only at the project and diagram levels of the Repository. It is a quick and global mechanism to prevent any access whatsoever to diagrams managed in specific projects, or to individual diagrams themselves. For example, you can create a group and assign it to the no access role for a project, then any users added to the group will not have access.
- Additive permissions: Permission granted to a user by group membership are additive. The user will have all the permission granted to him plus the permissions afforded him by group membership.

Notes

- When a user gets a file from the Repository, ER Studio Data Architect tracks the user and machine combination for that particular file. In addition, to enable users to model while they are "offline" or unable to connect to the Repository server, ER Studio Data Architect stores information about the security rights of the user and the diagram with the file, which has the following effects:
- When a user is logged into the Repository, and attempts to open a DM1 file that was retrieved from the Repository by a different user/machine combination than the currently logged in user, ER Studio Data Architect indicates that the file cannot be opened due to the conflict. This ensures that one user does not attempt to work on or check in changes to a diagram that was originally retrieved by a different user. Because the Repository keeps track of object check outs based on the user and machine, only the user on the machine that originally checked out an object can check it back in.
- Even if a user is not logged into the Repository, Repository DM1 files can be opened and worked on in ER Studio Data Architect. ER Studio Data Architect will load the security permission data for the user that originally retrieved the file from the Repository, and that security data will govern the rest of the working session until the user either logs in to the Repository or closes the file. If multiple files are open while the user is NOT logged in to the Repository they must all have been retrieved by the same user. Of course, this also applies when the user IS logged in.
- The cached security data stored with each diagram is updated whenever an open Repository file is saved while logged in to the Repository, but if the file is not saved before closing it, then any changes to the security permissions for that user and diagram will not be cached with the file. So, if you plan to work offline, and the security settings have changed since the file was last saved, then you should open each file while logged into the Repository and save it before taking the files to work offline.
- Permission settings can be applied at the project or repository level, which might cause a conflict when two files are opened while a user is offline. If two files are opened that both have cached security information for the Repository level or for the same Project, and if the cached data differs, then the most recently saved data will be used and stored in both files if and when they are saved.
- If an admin makes changes to the security UI, such as changing permissions on folders to grant or revoke access or moving diagrams between projects with different permissions, users must re-login to update their permissions.

This section also includes the following:

- Creating and Managing Roles
- Creating and Managing Users
- Creating and Managing Groups
- Granting and Prohibiting User Access to Repository Items

See Also

- Creating and Managing Roles
- Creating and Managing Users
- Creating and Managing Groups
- Granting and Prohibiting User Access to Repository Items

Creating and Managing Groups

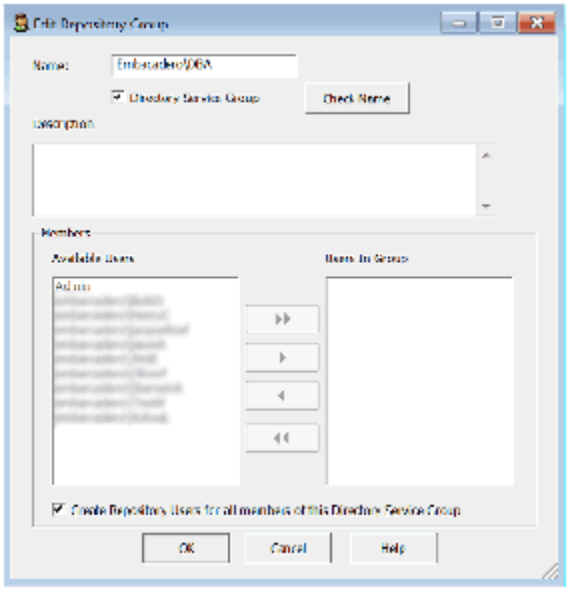
You can use groups in the Repository Security Center to make it easier to handle permissions. You can assign roles to a group at the Repository, Project, and Diagram levels and then add or remove users from the group to apply the settings to the user quickly and easily.

Groups can be created two ways:

- Create a group in the Repository Security Center and then add users to it manually
- Create a group with the same name as an LDAP/Active Directory group, and any LDAP/Active Directory users that are added to the Repository would automatically be members of that group based on the LDAP/Active Directory settings.

Create, Update, and Delete Group

1. Choose Repository > Security > Security Center.
2. Click the *Manage Groups* tab and select *New*.
3. Provide a name for the group and optionally, a description of the group.
4. *If you want to create a group based on an LDAP/Active Directory group*, click *Directory Service Group* and another option becomes available at the bottom of the dialog, *Create Repository Users for all members of this Directory Service Group*. If you select this option, then repository users in this group are automatically added to the list of Repository users. If you do not select this option, then you will need to create the users manually and as you add LDAP/Active Directory Service users, they will be added to the Repository groups that correspond to the LDAP/Active Directory Service groups that they are members of.



If you want to manually populate the group, do not select *Directory Service Group*, from the list of *Available Users*, select the group members by Ctrl-clicking each user, or click the double

right-facing arrow to select all the users. You can also remove group members by clicking the left-facing arrow.

Notes

- *Delete a group*: On the *Manage Groups* tab, you can delete a group by selecting the group and clicking *Delete*. Deleting a group does not delete the group members but the group members will no longer have the permissions afforded to them by the group membership.
- *Edit a group*: On the *Manage Groups* tab, you can edit the group name and list of group members by selecting the group and clicking *Edit* and making your changes on the *Edit Repository Group* dialog.

Assigning a Role to a Group

Assigning a role to a group grants the group all the permissions selected in the role.

1. Choose Repository > Security > Security Center.
2. Select a Repository item you want to assign permissions to.
3. *If the group is not currently assigned a role*, on the *Repository Security* tab, click the group name in the *Available Users* column and drag it onto a role name in the *Available Roles* column.

If the group was already assigned a role, on the Repository Security *tab, click the group name the* Available Roles *column and drag it onto another role name in the* Available Roles *column.*

1. Click *Apply* and then click *OK* to exit the *Security Center.*
2. Notify users in the group that they must log out and then log in again to receive the security updates.

Granting and Prohibiting User Access to Repository Items

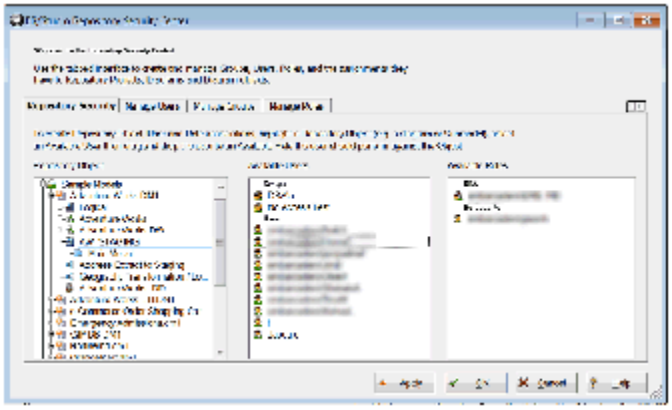
On the Repository Security tab of the Security Center, users with the appropriate privileges can assign other users the appropriate access privileges for different objects, granting them permission to perform specific operations on a Repository item and prohibiting them from performing other operations on the same object. You can assign the access privileges of a role to individual users or to an entire group.

Assigning Users/Groups Roles for Accessing Repository Items

1. Log in to the Repository.
2. Choose Repository > Security > Security Center.
3. Click the *Manage Roles* tab; explore the available roles to determine if an existing role provides the access you want to grant the user. If necessary, create a new role with the appropriate permissions.
4. Click the *Manage Groups* tab; explore the available groups to determine if an existing group encompassed the users you want to authorize users. You can double click the

name of a group to see the members of the group in the *Edit Repository Group* dialog. If necessary, create a new group. Assigning a role to a group is more efficient than assigning the role to the users one at a time.

5. Click the *Repository Security* tab.
6. In the *Repository Object* area, select a Repository item (diagram, object, submodel, data flow, transformation, or data dictionary) to which you want to grant or prohibit access.
7. Beneath *Available Users*, select a group or user to which you want to assign the role to the selected object and drag it onto the appropriate role in the *Available Roles* area.
8. To remove the role assignment of a user/group to a Repository item, click the Repository item, drag the user/group name from under role onto another role in the Available Roles area or back onto the Available Users list.



Notes

- All permissions available to a user through direct assignment to roles are added to the permissions available to any groups the user is a member of. Permissions are additive, so the user gets all the permissions assigned to the user whether they were assigned through a group or directly to the user.
- A user or group can only have one role for each object. But a user or group can have different roles for different objects.
- If a user is listed in the Available Users area, but cannot be selected, the user has been deactivated and must be reactivated before you can modify the user.

Prevent Users from Accessing Specific Repository Items

The system-defined No Access role can prevent a user from accessing selected projects or diagrams

1. Log in to the Repository.
2. Choose Repository > Security > Security Center.
3. In the *Repository Object* area, navigate to and then click to select a project or diagram you want to secure from specific users.

4. From the list of users in the *Available Users* area, click a user name or *CTRL-click* several names and then drag the users onto the *No Access* role in the *Available Roles* area.
5. Repeat Step3 and Step4 as required to secure other diagrams and projects.
6. Click *Apply*, continue changing security settings if required, and then click *OK* to exit the security center.

Notes

- If a user has been assigned the No Access role for an object, but that user is a member of a group with permissions for the object, then the user will have all permissions assigned to the group. All permissions available to a user through direct assignment to roles are added to the permissions available to any groups the user is a member of. Permissions are additive, so the user gets all the permissions assigned to the user whether they were assigned through a group or directly to the user.

Change User/Group Access to Repository Items

To change user/group access to Repository items, change the permissions of the role the user/group has for the Repository item or assign another role to the user/group for that particular Repository item. If required, you can force the user log out on the Manage Users tab of the Security Center, and then change the user's access privileges.

See Also

- Creating and Managing Roles
- Creating and Managing Users

Creating and Managing Roles

ER Studio Data Architect lets you create customized roles with different sets of permissions. Repository Object Type Permissions are pre-defined privileges to operate on Repository items. You can assign them to a role, which gives users assigned to that role, permission to perform certain Repository operations. You can use the many available privileges to create specific roles to suit your environment.

For example, you can create and assign some roles like the following:

- **Repo Level Basic**, which has all of the Repository level permissions, assigned to the target user at the Repository level.
- **Logical Only Modeling**, which has all of the Diagram, Model, and Submodel level privileges that do not apply to the physical model, assigned to the target user at the diagram level for each diagram in the Repository or at least the ones that the target user is allowed to work on.

With the above roles and assignment, when the target user adds the diagram, the user can check it out, but cannot modify it. To allow the user modify the logical model immediately after adding it, the Admin would have to apply the "Logical Only Modeling" role to the target user at the Repository level.

- To access, create, update, and delete user information, a Repository administrator must have Repository Object Type Permission, Access Security Info and Update Security Info privileges.

Create, Update, and Delete Roles

1. Choose Repository > Security > Security Center.
2. Select the *Manage Roles* tab.
3. Click *New*.
4. Click an item in the *Repository Object Type* list and then in the *Repository Object Type Permission* area, assign the permissions you want to assign to the role for the selected object type.
5. Repeat step 4 for each *Repository Object Type*.
6. Click *Apply* and continue changing security settings and then when finished, click *OK* to exit the security center.
7. To apply the new role to existing users, see Assigning a Role to a User/Group.
8. Inform affected users that they should log out of the Repository and then log in again to receive the security updates.

Notes

- Once created, the Administrator can select the role and update, delete or rename it at any time. Renaming the role does not affect the privileges users have, but if you delete a

role users who had access to Repository items through that role, will no longer have access to those items.

- Users cannot modify objects without the necessary permissions, regardless of whether the user has the objects checked out.
- To facilitate immediate access to a newly added diagram (not projects) the Admin should set up privileges as follows: If a user gets a submodel and wants to add an entity, the user must have Create Entity permission in the model containing the submodel, as well as Add Member permission in that submodel. If the user deletes an entity, the user must have Remove Member permission in that submodel. In addition, if the user wants to select the "Delete From Model" check box, the user must have Delete Diagram Object permission in that model.
- Once you delete a role from the Repository, it will no longer be in the database.
- To access/create/update/delete user information, a Repository administrator needs to have the Access Security Info and Update Security Info Privileges applied to the Roles.
- Before you delete a role, you must unlink any diagrams that are assigned to it and delete any users assigned to it.
- The following lists the Repository Object Types, the permissions you can grant, and the common operations these permissions give access to:

Repository Object Types	Repository Object Type Permissions	Permitted Operations
Repository	Access Security Info	*Security Center*: View settings
	Update Security Info	*Security Center*: View and change settings
	Create Diagram	Add Diagram
	Update Diagram	Check Out Diagram, Check In Diagram Check Out Object(s), Check In Object(s) Undo Check Out Diagram, Undo Check Out Object(s) Redo Check Out Diagram, Redo Check Out Object(s)
	Delete Diagram	Delete Diagram If you delete a diagram in Repository, the file itself remains on the local disk.
	Create Enterprise Dictionary	Create Enterprise Dictionary
	Update Dictionary	Check Out Data Dictionary, Check Out Dictionary Object(s), Check In Data Dictionary, Undo Check Out Data Dictionary, Redo Check Out Data Dictionary

	Create Project	Create Project
	Delete Project:	Delete Project
Project	Add Project Member	Add Diagram to Project
	Remove Project Member	Remove Diagram from Project
Diagram	Bind Enterprise Dictionary	Create New Enterprise Data Dictionary, Bind Existing Enterprise Data Dictionary
	UnBind Enterprise Dictionary	Remove Enterprise Data Dictionary
	Compare Models	Run Compare/Merge Wizard
	Create Physical Model	Create Physical Model
	Delete Physical Model	Delete Model
	Set Named Release	Set Named Release
	Delete Named Release	Delete Named Release
	Rollback Diagram To Named Release	Rollback Diagram
	Update Diagram Properties	Edit Title Block Data, Edit Diagram Properties
	Create Data Flow	Create New Data Flow on the Data Lineage tab
	Delete Data Flow	Delete Data Flow from the Data Lineage tab

Data Dictionary	Create Dictionary Object	*Create*: Attachment Type, Attachment, Default, Rule, Data Movement Rules, Reference Value, User Datatype, Domain Folder, Domain, Reusable Trigger, Reusable Procedure, Library
	Update Dictionary Object	*Edit*: Attachment Type, Attachment, Default, Rule, Data Movement Rules, Reference Value, User Datatype, Domain Folder, Domain, Reusable Trigger, Reusable Procedure, Library
	Delete Dictionary Object	*Delete*: Attachment Type, Attachment, Default, Rule, Data Movement Rules, Reference Value, User Datatype, Domain Folder, Domain, Reusable Trigger, Reusable Procedure, Library
Logical Main Model	Create Diagram Object	*Create*: Entity, View, Relationship, View Relationship, Subtype Cluster, Subtype, Title Block
	Delete Diagram Object	*Delete*: Entity from Model, View from Model, Relationship from Model, View Relationship from Model, Subtype Cluster from Model, Subtype from Model, Title Block from Model
	Update Diagram Object	*Entity Editor*: Create/Modify/Delete: Attribute, Key, Key Attribute, Check Constraint *View Editor*: Modify View; Create/Modify/Delete: View Table, View Column *Key Editor*: Modify Key, Create/Modify/Delete Key Attribute *Relationship Editor*: Modify Relationship *Subtype Cluster Editor*: Modify Subtype Cluster *Edit Model*: Options, Properties
	Create Submodel	Create Submodel
	Delete Submodel	Delete Submodel
Logical SubModel	Add Member	*Submodel Editor*: Add to Submodel
	Remove Member	*Submodel Editor*: Remove from Submodel, Delete Entity from SubmodelRemove Database View *Delete*: Relationship from Submodel, View Relationship from Submodel, Subtype Cluster from Submodel, Subtype from Submodel
	Update Display Properties	*Move/Resize*: Entity/Table, View, Title Block, Text Block, Subtype Cluster *Color/Font Changes*: Entity/Table, View, Title Block, Text Block, Relationship Line, View Relationship Line, Subtype Cluster *Move*: Relationship Line, View Relationship Line *Create/Modify/Delete*: Text BlockChange Model Notation, Perform Layout, Zoom, Align Objects
Physical Model	Create Diagram Object	*Create*: Table, View, Relationship, View Relationship, Schema Object, Title Block
	Delete Diagram Object	*Delete*: Table, View, Relationship, View Relationship, Schema Object, Title Block

	Update Diagram Object	*Create/Modify/Delete*: Column, Index, Index Column, Check Constraint, View Table, View Column, Key Attribute, Key, Relationship, Subtype Cluster *Edit Model*: Options, PropertiesChange Database Platform
	Create Submodel	Create Submodel
	Delete Submodel	Delete Submodel
Physical SubModel	Add Member	*Submodel Editor*: Add to Submodel
	Remove Member	*Submodel Editor*: Remove from Submodel *Delete*: Table from Submodel, View from Submodel, Relationship from Submodel, View Relationship from Submodel, Schema Object from Submodel
	Update Display Properties	*Move/Resize*: Entity/Table, View, Title Block, Text Block, Physical Schema Object *Color/Font Changes*: Entity/Table, View, Title Block, Text Block, Relationship Line, Physical Schema Object *Move*: Relationship Line, View Relationship Line *Create/Modify/Delete*: Text BlockChange Notation, Perform Layout, Zoom, Align Objects
Data Flow Model	Create Data Flow Object	*Create*: Data Flow, Data Lineage Component, Data Stream, Source
	Update Data Flow Object	*Edit* Data Flow, Object, Transformation, Data Stream, Source *Check In*: Data Flow, Object, Transformation, Data Stream, Source *Check Out*: Data Flow, Object, Transformation, Data Stream, Source *Undo Check Out*: Data Flow, Object, Transformation, Data Stream, Source *Redo Check Out*: Data Flow, Object, Transformation, Data Stream, Source *Note*: The user must have permission to update the diagram in order to update data flow objects in the diagram.
	Delete Data Flow Object	*Delete*: Data Flow, Data Lineage Component, Transformation, Data Stream *Note*: Deleting a table\entity component from the Data Lineage window does not delete the table\entity from the model. *Note*: The user must have permission to delete the diagram in order to update data flow objects in the diagram.
Data Flow Display	Update Display Properties	*Move/Resize*: Transformation, Component, Data Flow *Color/Font Changes*: Data Lineage Background, Component, Transformation, Data Stream *Diagram And Object Display Options*: Change any option in this dialog to control how the data lineage diagram displays.Perform Layout, Zoom, Align Objects, Layout Data Stream, Straighten Data Stream, Remove All Bends

Assigning a Role to a User/Group

Assigning a role to a user/group grants the user/group all the permissions selected in the role.

1. Choose Repository > Security > Security Center.

2. Select a Repository item you want to assign permissions to.
    - **If the user/group is not currently assigned a role**, on the **Repository Security** tab, click the user name in the **Available Users** column and drag it onto a role name in the **Available Roles** column.
    - **If the user/group was already assigned a role**, on the **Repository Security** tab, click the user/group name the **Available Roles** column and drag it onto another role name in the **Available Roles** column.
3. Click **Apply** and then click *OK* to exit the **Security Center**.
4. Notify users/group members that they must log out and then log in again to receive the security updates.

See Also

- Creating and Managing Users
- Creating and Managing Groups

Creating and Managing Users

Managing users entails creating, changing, and deleting user account permissions and logins, checking in a users checked out documents under unusual circumstances. You can control the privileges granted to a user for a specific diagram, model, submodel or Enterprise Data Dictionary by associating the user with a role. Using Roles, you can choose the permissions granted a user for the Repository item.

- To access, create, update, and delete user information, a Repository administrator must have Repository Object Type Permission, Access Security Info and Update Security Info privileges.

Create, Edit, Delete, Deactivate, Reactivate, Log Out, and Check In Users

1. Choose Repository > Security > Security Center.
2. Click the **Manage Users** tab.
3. On the **Manage Users** tab, you can manage users by clicking their name in the list and then clicking a management option and following the prompts as required.
4. Continue making security changes and then click **OK** to exit the security center.
5. To assign permissions to users, see Assigning a Role to a User/Group.
6. Notify users that they must log out and then log in again to receive the security updates.

The following helps you complete the options of the *Manage Users* tab:

User Management area

Administrator User 'Force' Options area

Notes

User Management area

- **New**: Click **New** to add a new user through the **Create Repository User** dialog.
- **Directory Service User**: Select this option If you want the new user to login to the Repository using their Windows credentials. In this case ER Studio Data Architect accesses the LDAP server on the network to verify the user's credentials. Directory service users can check the **Log in using current Windows account** option on the Repository login dialog to use their Windows user ID and password to log into the Repository.
- **Delete**: Select a user and then click Delete to delete the user. You cannot delete the Admin user.
- Deleting a user also deletes version history for that user, which means that version history for all actions done on various objects by that user will also be deleted. It may be advisable to instead deactivate the user. Deactivating a user does not delete their history.

Administrator User 'Force' Options area

When you use the Check In User option, you can choose how to handle the checked out objects.

- Convert to non-exclusive: Enables the user to continue making changes to the checked out items. Other users can then check out the same items. When checking in non-exclusively checked out items, if the item was changed by another user since it was checked out, the user can resolve the differences and choose to override the other user's changes or to update his local version with the changes found in the Repository.
- Undo Check Out: When the user next connects to the Repository, the status of his items will be updated to show that they are not checked out.

Notes

- You cannot delete or deactivate the Admin user, but you should change the Admin password from the default, which is Admin.
- The new user appears in the user list with a star indicating that this user does not yet have any roles assigned to it for any Repository items.
- Users must log out and log in again to receive any security updates.
- The Admin does not need to know the user's password to create a new password for the user.
- The Admin can look to the Manage Users tab to find out which users are logged in to the Repository and whether or not they currently have items checked out. Knowing if users have items checked out is particularly useful when upgrading the Repository, because after before upgrading the Repository the users should check all their items in and then do a clean get and check out of the items when the upgrade is complete. Knowing which users are logged in is useful when changing security center settings because after making any setting changes, the Admin should request online users to log off and log in again to receive the security center updates.
- The Administrator may want to user log a user out to receive security center updates. For example, a new diagram has been added to a project and the diagram shouldn't be accessible by all the users who have access to the project. The Administrator can make the necessary security changes for the project and then log out the users, forcing them to log in again to receive the security updates. Other unusual circumstances may arise when it is necessary for the Administrator to log a user out of the Repository.

Assigning a Role to a User

Assigning a role to a user grants the user all the permissions selected in the role.

1. Choose Repository > Security > Security Center.
2. Select a Repository item you want to assign permissions to.

- **If the user is not currently assigned a role**, on the **Repository Security** tab, click the user name in the **Available Users** column and drag it onto a role name in the **Available Roles** column.
- **If the user was already assigned a role**, on the **Repository Security** tab, click the user name the **Available Roles** column and drag it onto another role name in the **Available Roles** column.

3. Click *Apply* and then click *OK* to exit the **Security Center**.
4. Notify users that they must log out and then log in again to receive the security updates.

See Also

- Creating and Managing Roles
- Creating and Managing Groups

Working with Objects in the Repository

The Repository performs version control and allows other users to retrieve a copy of the diagram to view or edit it on their local machines if they have permission. The user Roles dictate what a user can do with the diagram and the objects within it. Modelers can update or delete model information simultaneously by working with ER Studio Data Architect in the normal fashion and by merging their changes into a single model.

Before you add your diagram, if you want to place access restrictions on it, you need to have users and roles created. For more information, see 'Establishing Security for the Repository' in the ER/Studio Repository Installation and Administration Guide.

The Repository includes a robust features set for working with diagrams. Detailed procedures for using these features are provided in this sections.

This section covers the following topics:

- Repository Status Icons
- Adding a Diagram or Data Dictionary to the Repository
- Relating models to ER/Studio TS Data Sources
- Securing Repository Objects
- Changing the Diagram File Name
- Retrieving a Repository Diagram, Model, Submodel or Named Release
- Creating Special Search Strings (Using the Query Parser)
- Getting the Latest Version of a Diagram
- Saving Changed Items to the Repository (Checking In)
- Checking Out Repository Items
- Discarding Changes Made to a Checked Out Diagram
- Checking Out Objects When Not Logged In
- Deleting Diagrams from the Repository
- Forcing a Checkin
- Determining Who has an Object Checked Out
- Viewing the Checkin History of a Repository Object
- Working with Named Releases of Repository Diagrams
- Branching and Merging Diagrams
- Rolling Back a Diagram to a Previous Version

See Also

- Establishing Security for the Repository
- Working with Repository Projects

Repository Status Icons

The icons below show check-out status of diagrams and objects contained in Repository. For any one object, check-outs are allowed by multiple users simultaneously. The Local column illustrates the status of an object that appears on your PC only, whereas the Remote column illustrates how the same document status appears to other Repository users:

Remote Icon	Remote Status	Local Icon	Local Status
	Checked In		Checked In
	Checked Out		Checked In
	Exclusively Checked Out		Checked In
	Checked In		Checked Out
	Checked Out		Checked Out
	Checked In		Exclusively Checked Out
	Checked In		Delayed Check Out
	Checked Out		Delayed Check Out
	Exclusively Checked Out		Delayed Check Out
	Named Release		Named Release

See Also

- Checking Out Repository Items
- Working with Named Releases of Repository Diagrams

Adding a Diagram or Data Dictionary to the Repository

Using the Add Diagram operation you can add the active diagram and its data dictionary, if you choose, to the Repository.

- You cannot add data model objects to the Repository without first adding the diagram containing them.

1. Log in to the Repository.
2. Choose Repository > Diagrams > Add Diagram.
3. Enter or revise the options in the *Add Diagram to ER/Studio Repository* dialog as appropriate, and then click *OK* to exit the dialog.

The following describe options that require additional explanation:

Save As: Use to set the file name for a new, blank diagram or to change the file name for an existing diagram. This will also create a local DM1 file with the same name.

Add to Repository Project: Click the list to add the diagram to an established project. For more information, see Working with Repository Projects.

Bind Existing Enterprise Data Dictionaries: Select a data dictionary to bind to the diagram. This list is empty if no Enterprise Data Dictionaries have been created. Using the Repository enables you to have more than one data dictionary bound to your diagram, the local dictionary and as many of the Enterprise Data Dictionaries as you like.

Promote Local Data Dictionary to Enterprise: Creates an Enterprise Data Dictionary based on the selected DM1 file's data dictionary.

Data Dictionary Name: If Promote Local Data Dictionary to Enterprise is selected, lets you change the name for the new Enterprise Data Dictionary; otherwise, disabled.

Filename: When you add the diagram to the repository, the name of the local diagram file is automatically assigned to the respective file in the repository, however, you can change the name of the file in the repository, without affecting the local file name, by using the Change Diagram Filename command. For more information, see Changing the Diagram File Name.


Note: After you add your diagram, if you want to place access restrictions on it, you need to have users and roles selected and secure your diagram.

See Also

- Relating models to ER/Studio TS Data Sources
- Changing the Diagram File Name

Relating models to ER/Studio TS Data Sources

If you are connected to ER/Studio TS and the current model's parent diagram is in the Repository, a model can be related to a ER/Studio TS data source. This lets you see and document the scope or impact of a particular change.


Note: For information on ER/Studio TS data sources, see Logging In and Out of the Repository and ER/Studio TS.

To relate a model to a ER/Studio TS data source

1. Right-click a model and select **Data Source Mappings**. The **Model Data Sources** dialog opens.
2. Select an item from the left-hand list of available data sources and then click the >>>> button to relate the data source to the current model. A **Choose owner and database** dialog opens.
3. Select a **Database Type** and **Datasource**, provide a **User Name** and **Password**, and click **Connect** to connect to the target data source.
4. For each of the *Catalog* and **Owner** controls, click the associated browse button to open a dialog that lets you select catalogs and owners and click **OK** when ready.
5. Click **OK**.

See Also

- Securing Repository Objects
- Checking Out Repository Items

Securing Repository Objects

In the Repository Security Center, the user or administrator with Update Security Info privileges can restrict access to the data model at the Project, Diagram, Model, Submodel, and Data Dictionary levels.

- In order to impose security on Repository objects, Repository Users and Roles must be created. For more information, see 'Establishing Security for the Repository' in the *ER/Studio Repository Installation and Administration Guide*.

1. Choose Repository > Security > Security Center.
2. In the *Repository Object* area, navigate to and then click to select the object you want to secure.
3. **To give a user access to the selected object**, from **Available Users**, click and drag a user onto an **Available Role**. The user inherits the privileges of the role under which the user appears in the Available Role area.
   To revoke a user's access privileges, from **Available Roles**, click and drag a user onto an **Available Role**.
   - NOTE: The No Access role is available only at the Diagram and Project level.
4. Repeat **Step 1** and **Step 2** as required and then click *OK* to exit the security center.
5. Inform users that they must log out of the Repository and then log in again for them to receive Repository object security updates.

See Also

- Retrieving a Repository Diagram, Model, Submodel or Named Release
- Repository Status Icons

Changing the Diagram File Name

Renames the file in the Repository that is associated with a selected diagram. The file name is defined by the system when you first add the diagram to the Repository and is initially the same as the Diagram Name.

- Renaming the diagram using the Change Diagram Filename option, does not change the diagram name as displays in Diagram Title box. You can change the Diagram Properties and the content of the Diagram Title box by opening the diagram, choosing File > Diagram and then editing the name on the Information dialog.

1. Log in to the Repository.
2. Choose Repository > Diagrams > Change Diagram Filename.
3. In the tree, navigate to and then select the file whose name you want to change.
4. Click **Rename**.
5. In the **Change Filename** dialog, enter a new file name and document the file name change in the Comment box. **Change Filename** dialog as appropriate.
6. Click **OK**, verify the name change is as you expected, and then click **Close**.

Notes

- As with any Version Control Software (VCS), all users should check in all changes and perform a clean get after renaming.

See Also

- Saving Changed Items to the Repository (Checking In)
- Repository Status Icons

Retrieving a Repository Diagram, Submodel or Named Release

Retrieving repository objects downloads copies of selected diagrams, submodels, or named releases from the Repository onto the user's local machine. The data model, submodel or named release is copied from the Repository to your local system. If a local version of the object exists, the Get from Repository operation refreshes the local copy with the version from the Repository.

**Note:** Permissions assigned to your user account in the Repository Security Center determine what type of access you have to Repository objects. For more information, see Establishing Security for the Repository in the *ER/Studio Repository Installation and Administration Guide*.

1. Log in to the Repository.
2. Choose Repository > Diagrams > Get from Repository.
3. In the *Get From Repository* dialog, there are two ways to find a diagram.
    - Browse to and then select the object.
    - In the **Show diagrams that begin with**: field, enter the first few letters of the diagram name. As you type the list of diagrams updates to display just the diagrams that match the letter or letters entered.

    **Tip:** Holding *Shift* while expanding a node will expand only that node and make immediate child nodes collapsed. Holding *Ctrl* while expanding a node, all child nodes and their descendants will be expanded fully. Expanding without holding any keys will simply return the subtree to the state from the most recent collapse. This does not apply to diagram nodes that must be expanded individually to retrieve model and submodel data from the Repository.
4. If you plan to make changes to the diagram, you may select *Check Out Immediately* to immediately mark the objects in the file as editable.

    **Tip:** You can prevent other users from making changed to the object by selecting Check Out Exclusively. Otherwise, when you check in your changes you can resolve any differences between your version and the version in the Repository if another user checked in changes to the same object. For more information, see Checking In Objects.
5. Click **OK**.

The dm1 file will be placed in the Active Files directory as specified in Repository > Options > General.

Additional Information

- If a model with the same name already exists in the Models directory, as specified in Tools > Options > Directories > Models, then you will be alerted that proceeding with the Get from Repository operation will result in the local copy being overwritten with the version from the Repository. You can then choose to cancel the Get operation without affecting the contents of the local version.
- You can view a Named Release, but, in general, you may not modify it. A special Repository operation, Rollback Diagram, lets you Check In a Named Release as the latest version of a Diagram; once the Roll Back operation is completed, you can Check Out the Diagram (or a portion of it) to modify it.

See Also

- Securing Repository Objects
- Repository Status Icons

Creating Special Search Strings (Using the Query Parser)

Operator	Search criteria	Example
AND (&&)	Contains ALL the search terms entered	apples AND oranges AND bananas apples && oranges && bananas
OR (||)	Contains at least one of the words entered	cats OR dogs cats || dogs
" "	Contains the exact phrase entered	"exactly this"
	Does NOT contain some of the words entered	apples NOT oranges NOT bananas
+	Must contain a term and may contain another	+apples oranges
-	Must contain a term and must not contain another	+apples -oranges
?	Matches with a single character replaced	c? (matches cat but not coat
*	Matches zero or more characters	do* (matches do, dog, dogs, dogsled)
~	Terms are spelled similarly	land~ (matches land, sand, band)
" "~	Terms are found in close proximity to each other	"diagram update"~10 (matches objects where diagram and update appear within 10 words of each other)

ER/Studio Team Server uses Apache Lucene Query Parser to enable these search string operators. There are some special characters that when used within the syntax must be escaped. The following are special characters:

+ - && || ! ( ) { } [ ] ^ " ~ * ?: \

To escape these characters, enter a backslash (\) before the special character.

- For more details on the Lucene Query Parser, see http://lucene.apache.org and search for "query parser syntax."

See Also

- Getting the Latest Version of a Diagram
- Repository Status Icons

Getting the Latest Version of a Diagram

Get Latest Version refreshes the local version of the open diagram with the version in the Repository. For example, if multiple people check out the same diagram, and someone makes changes to the diagram you are working on, you can get the latest version to update your local version after they submit their changes.

Choose Repository > Diagrams > Get Latest Version.


Note: If conflicting changes are detected because multiple people changed the diagram, ER Studio Data Architect opens the Review Changes and Resolve Conflicts dialog, where you must resolve any conflicts between the local and remote diagrams before continuing.

See Also

- Determining Who has an Object Checked Out
- Forcing a Checkin

Saving Changed Items to the Repository (Checking In)

The check in process synchronizes the changes made to the local checked out item with the Repository version.

The check in process lets you check In an object, such as an entity, or an entire diagram, when you are done working with it. ER Studio Data Architect attempts to merge all of your modifications into the Repository. If there are any conflicts during the merge process, you are prompted to resolve them. After you have resolved all conflicts and the merge is complete, the merged results are added to the repository and your diagram. The merged results are added to your diagram so that it remains synchronized with the file in the repository.

Checking In Objects

When you check in an object, all the object's structural and referential integrity (RI) dependencies are checked in as well. You can check in a newly created object (i.e. an entity which you added to a checked-out diagram) or multiple, selected objects at one time to the repository.

The Repository enforces several rules when checking in individual objects to help maintain integrity throughout the diagram:

- If you have an object and one or more of its dependencies checked out separately, then when you check in the main object, all of its dependencies are also checked In.
- If you have deleted some objects, when you checks in a modified object, all of the deleted objects are also checked in to the repository, regardless of whether they were associated with the modified object.
- If you check in a data dictionary object that is bound to an attribute, and changes to the data dictionary object affect the attribute, then the attribute is marked for a delayed check out.

Checking In a Data Dictionary

When you check in a data dictionary, the Repository server may create new versions of objects that were bound to individual Data Dictionary objects. For example, if you check in a Data Dictionary after deleting a user Datatype, and the user Datatype was bound to an attribute, the server will create a new version of the attribute that does not contain the binding. This is because deleting a Data Dictionary object causes all bindings to it to be removed. The new version is reflected in the Version History for the attribute, with an automatically generated comment. When the user does a Get Latest Version on the affected attribute, the new version without the binding is placed in the user's copy of the Diagram. If the user modifies or deletes the affected attribute before Checking In the Data Dictionary, then a merge conflict will appear between the user's version of the attribute and the version of the attribute created by the Repository server. You should simply select the ER Studio Data Architect version of the attribute to keep your changes.

This behavior can also occur when you checks in a Data Dictionary with a modified Data Dictionary object. For example, say you change the data type of an Attachment, then checks in the Data Dictionary to which it belongs. Any objects bound to that Attachment must have the value override removed because the value override is only valid for a particular Attachment data type. This means that the objects (which can be in the Data Dictionary or the Diagram) that are bound to that Attachment get a new version in the Repository that reflects the removal of the object's value override.

If you made changes to the Data Dictionary, you need to check in the Data Dictionary before checking in the diagram. For example, if you create a domain in the Data Dictionary and use that domain in the diagram, if you check in the diagram before the Data Dictionary, ER Studio Data Architect produces errors.

- If conflicting changes are detected because multiple people changed the diagram, ER Studio Data Architect opens the Review Changes and Resolve Conflicts dialog, where you must resolve any conflicts between the local and remote diagrams before continuing.

Checking in to the Repository

The procedure to check in an object into the Repository is basically the same for all object types.

1. Save the changes you have made to the diagram; choose File > Save.
2. Click the object you want to check in or **CTRL-click** several objects to check in more than one.
3. Right-click and select **Check in**.
   Depending on what you have selected, the check-in option on the short menu may be: Check in Diagram, Check in Model, Check in Submodel, Check in Object(s), Check In Data Dictionary, Check In Data Dictionary Object(s), Check In Source/Target or Check In Data Flow.
   Note: Checking in the Data Dictionary also checks in any Data Movement Rules defined on the Data Lineage tab.
4. Complete the *Repository Check In* dialog as required and then click *OK* to start the check-in process.

If you selected Review changes before check in or if a conflict is detected, you will be prompted to resolve the differences between your version and the version on the Repository.

Change and Conflict Categories

The following provides some additional information about the icons used on the Review Changes and Resolve Conflicts dialog to differentiate the differences detected and their default resolutions.

Category Icons	Description
	Changes to the local diagram conflict with changes made by other users to the remote diagram in the Repository. These conflicts are changes that the user performed on items in the local diagram that were also changed by other users who already checked in their changes to the Repository. For more information on conflicts, see Resolving Check-in and Merge Conflicts.
	Additions to the local diagram to items were not otherwise modified in the Repository version by other users. These additions were made to the local diagram and do not conflict with any changes made to any remote diagrams.
	Deletions in the local diagram to items were not otherwise modified (in the Repository version by other users. These deletions were made to the local diagram and do not conflict with any changes made to any remote diagrams.
	Updates to the local diagram to items were not otherwise modified (in the Repository version by other users. These updates were made to the local diagram and do not conflict with any changes made to any remote diagrams.

- In general, the most effective and safest way to handle changes you do not recognize or remember making is to allow them to be checked in to the Repository. Only changes that you make but later decide against should be unchecked.

Resolving Check-in and Merge Conflicts

The following describes the logic behind conflict detection resolutions:

A conflict arises when there is a difference between the version in the Repository and the version checked out to a user. In these instances, you can resolve the data conflict by choosing to accept your version, the Repository version or a combination of both. A common scenario arises when multiple, simultaneous changes have been made to a model. There are three general types of Repository conflicts:

- An item was updated in a local copy but deleted in the Repository version.
- An item was deleted in a local copy but updated in the Repository version.
- An item was updated in a local copy and was updated differently in the Repository version.

Conflicts of type 1 and 2: Have a check box, which if selected indicates that the local version will be applied. (You can view the properties of the item or table updated in the local diagram by expanding the item. Only those properties that have been updated will be displayed).

If you select the check box of this conflict item, Table A will be applied to the local and the Repository version of the diagram; the table will not be deleted. However, if the user clears the check box, the remote version of the diagram will be applied to the local version - the table will be deleted in the local version of the diagram.

A converse example: an item is deleted locally but updated in the Repository. You can view the properties and the new table values (i.e. see how the table was updated by other users) by expanding the item. Only properties whose values changed will be displayed. If you select the check box of the conflict item, the local version will be applied - the table will be deleted in the local version and the Repository version of the diagram. If you do not select the check box, the table will be updated in the local version of the diagram.

Conflicts of type 3: the Review Changes and Resolve Conflicts dialog box will display each new name underneath the table conflict item. The first name property item (new name item) will display the local value for the table name (i.e. the name decided upon by the local user). The second item will display the new value for the table name as it appears in the Repository version of the diagram. Each item has an option button. If you click the option button of the ER Studio Data Architect version of the table name, the local name version will be applied to the Repository and the local ER Studio Data Architect version. If you select the Repository version, that name will go into effect in the local version of the diagram (and stay in effect in the Repository).

Common check-in conflicts and their resolution are summarized in the table below.

Conflict		Resolution
ER Studio Data Architect	Repository	
Unchanged	Modified	Soft, does not show up in dialog, chooses Repository side.
Unchanged	Deleted	Soft, does not show up in dialog, chooses Repository.
Unchanged	Unchanged	Not a conflict, does not show up in dialog.
Modified	Unchanged	Soft, shows up in dialog with Review Changes enabled, defaults to ER Studio Data Architect.
Deleted	Unchanged	Soft, shows up in dialog with Review Changes enabled, defaults to ER Studio Data Architect.
Modified	Modified	Hard, shows up in dialog, defaults to ER Studio Data Architect.
Modified	Deleted	Hard, show up in dialog, defaults to Repository.
Deleted	Modified	No conflict, shows up in dialog with Review Changes enabled, defaults to ER Studio Data Architect
Does Not Exist	New	No conflict, does not show up in dialog, defaults to Repository.
New	New	Not indicated as conflicts in dialog and are resolved either by renaming one of the objects in ER Studio Data Architect, or by deleting one of the objects or properties.

Notes

- Soft conflicts are shown in the Resolve Conflicts Dialog box when the Review Changes option is selected.
- Hard conflicts are shown regardless of the setting of the Review Changes option.

Sequence Number Changes and Other Automatically-Resolved Differences

In large, complex models there can be minor inconsistencies in the way the data is presented. These consistencies do not have a major effect on the data in the model, but they can appear as changes in the review changes dialog. For example, when two foreign key columns exist in a table and have the same name and are not rolenames, they are unified by ER Studio Data Architect.

A column exists in the internal data for each relationship that propagates to the table, but only one is shown with that particular name. Which one is shown doesn't matter, they are assumed to refer to the same column in the database. Occasionally ER Studio Data Architect will change which column it displays based on which internal ID is lower. If the two columns were added at different times and given different sequence numbers within the table, it is possible that when ER Studio Data Architect switches which it displays, the ordering of the columns in the table changes. This can cause the sequence numbers of other columns to change as well. Those sequence number changes often show up in the dialog.

Although ER Studio Data Architect is designed to be able to handle the deselection of those extra changes, it is generally a better idea to let ER Studio Data Architect update itself automatically. It updates the data so that it is consistent and deselection changes in the dialog might undo some of the changes ER Studio Data Architect makes, but not all. In addition, there are some inconsistencies that could lead to corrupted data that cannot be handled by ER Studio Data Architect or its Repository Merge functionality.

Often, ER Studio Data Architect can identify and correct these inconsistencies before they cause major corruption, but if you deselect the changes, they are not allowed to be checked in, and they are never fixed in the Repository.

See Also

- Repository Status Icons
- Securing Repository Objects

Checking Out Repository Items

Checking out write enables items previously added to the Repository, such a diagram, model, submodel, data model object, Data Dictionary, Data Dictionary object, and data source/target. The procedure to check out an item from the Repository is basically the same for all object types.

Note: Checking out an item that is already checked out synchronizes the local version of the item with the Repository version.

1. Open a Repository diagram; a diagram that was previously added to the Repository.
2. Click the item you want to check out or *CTRL-click* several items to check out more than one.
3. Right-click and select **Check Out**.
   Depending on what you have selected; the check out option on the short menu may be Check Out Diagram, Check Out Model, Check Out Submodel, Check Out Object(s), Check Out Data Dictionary, Check Out Data Dictionary Object(s), Check Out Source/Target or Check Out Data Flow.
   Note: Checking out the Data Dictionary also checks out any Data Movement Rules defined on the Data Lineage tab.
4. Complete the **Repository Check Out** dialog as required and then click **OK** to initiate the check out process.
   The following describe options that require additional explanation:
   - **Synchronize with latest Repository diagram data**: Lets you synchronize your diagram data with the Repository's diagram data. For example, if an item is in a different location or has a different background color in the Repository version than in your version, you can update your diagram with the information from the Repository. In general, the check out operation finishes more quickly if you do not select this option; select this option only if you want to retrieve the very latest version of the Objects from the Repository.
   - **Check out Exclusively**: If selected, lets you check out an item with exclusive rights. This restricts any other user from checking out the item while you have the item checked out. You are the only who modify the item and submit the data to the Repository.

When you check out an item exclusively, other users have read-only and/or 'Delayed Check Out' access while the objects are checked out. If submodel A and submodel B share an object and user A exclusively checks out submodel A, then user B would not be able to exclusively checkout submodel B.

Notes

- If you check out a parent item that has a lot of relationships, the relationship-dependent items will also be checked out.

- You cannot check out an item exclusively if another user has it checked out. You can find out who has an item checked out, by right-clicking the item, selecting Version History, and then clicking the CheckOut Status tab.
- If you do not select check out exclusively, but attempt to add an object to a model, you will be prompted to check out the model exclusively.
- Working locally, if you check an object out of the Repository, ER Studio Data Architect saves a writable copy to your local machine. If you then disconnect from the Repository you can still make changes before checking back in your work. When you reconnect to the Repository, you will still have the objects checked out. For your changes to commit, you need to log in to the Repository and check in your changes.
- Instead of checking out an entire diagram, you can get single object or multiple objects.
- When you check out an object, object structural and Referential Integrity (RI) dependencies are also checked out. This means that you will get the latest versions of all dependencies as well.
- The Repository Check Out operation is the same for diagrams and objects. The menu varies depending on what you select in the Data Model Explorer.
- Multiple users can have the same object checked out simultaneously.
- If your modeling team has fairly segregated tasks, then an exclusive check out is recommended. For example, you may have several submodels of a model that are being worked on by different modelers. If you check out the entire submodel exclusively, other users will be prevented from using elements of that submodel that are present in other submodels. For example, if submodel A and B share the object "PARTY", submodel B cannot be checked out exclusively if submodel A was checked out exclusively.
- You can check out an entire diagram from the Repository. When you do so, the diagram data dictionary is also checked out.
- Checking out at the diagram level is faster than checking out individual objects because ER/Studio does not have to process the selected objects to ensure that related objects are also selected for checkout. This would be most obvious during an unsynchronized checkout. It is also true for synchronized checkouts as well as checkins, but because those operations perform a get latest option as part of the process, the effect is less obvious.
- You can check out an individual data dictionary object. If the data dictionary object is not bound to any attributes, you can check out or check in the data dictionary object without affecting the diagram. If the data dictionary object is bound to an attribute, and changes to the data dictionary object affect the attribute, then the attribute is marked for a delayed check out.

See Also

- Saving Changed Items to the Repository (Checking In)
- Changing the Diagram File Name

Discarding Changes Made to a Checked Out Diagram

If you have checked out a diagram, made changes to it, and want to undo all the changes you made you can undo the check out.

1. Log in to the Repository and open the target diagram.
    - *To discard the changes, without checking out the diagram again*, choose Repository > Diagrams > Undo Diagram Check Out.
    - *To discard the changes, and check out the diagram again*, choose Repository > Diagrams > Redo Diagram Check Out.

This refreshes the local copy of the diagram with the latest version from the Repository.

See Also

- Saving Changed Items to the Repository (Checking In)
- Viewing the Checkin History of a Repository Object

Checking Out Objects When Not Logged In

If you are not logged in to the Repository and try to modify a Repository item that is not checked out, you can check out the item locally and then run the delayed check out command later when you can connect to the Repository. This allows you to modify your local copy of a diagram from the Repository without connecting to the Repository, which is useful when you do not have a network connection, want to move objects in the diagram without checking them out, or get quicker access to large models.

When you are connected to the Repository and want to check out the diagram from the Repository, choose Repository > Diagrams > Run Delayed Checkout.

See Also

- Forcing a Checkin
- Determining Who has an Object Checked Out

Deleting Diagrams from the Repository

The Delete Diagram operation lets you delete a diagram from the Repository. This also renders the diagram on the local disk unusable.


Note: This is a destructive action. Before deleting a diagram, notify other users who may be currently accessing it. Save the diagram under a different name on your local disk in case you need to access it again.

1. Log in to the Repository.
2. Choose Repository > Diagram > Delete a Diagram.
3. In the **Select a Diagram to Delete** dialog, select the diagram to delete.
4. Select the *Leave Diagram Data behind and marked as Deleted* check box.
   Although not required, use this option because it will prevent total loss of the diagram. With this option selected, the diagram can later be retrieved if necessary.
5. Click *OK*.

Additional Information

- When you delete a Diagram from the Repository, the local copy of the Diagram is also deleted. If you want to save your local copy of the Diagram, you must rename the **.DM1* file.
- When deleting a diagram, ER Studio Data Architect leaves diagram data behind and marks it as deleted. If deleted, data is still present in the Repository but visible only when querying the Repository directly, not through ER Studio Data Architect client.

See Also

- Determining Who has an Object Checked Out
- Rolling Back a Diagram to a Previous Version

Forcing a Checkin

The Repository Administrator can check in objects a user has checked out. For more information, see "Creating and Managing Users" in the *ER/Studio Repository Installation and Administration Guide*.

See Also

- Determining Who has an Object Checked Out
- Viewing the Checkin History of a Repository Object

Determining Who has an Object Checked Out

At any time while logged in to the Repository, you can see who has any part of a diagram checked out; this includes viewing the checkout status of everything from attributes to submodels and data dictionaries to entire diagrams, and everything in between.

1. Open a Repository diagram and then log in to the Repository.
2. On the *Data Model*, *Data Dictionary*, *Data Lineage*, or *Macros* tab, navigate to the object you want to investigate.
3. Right-click the object and then click *Version History*.
4. On the Version History dialog, click CheckOut Status.

ER Studio Data Architect queries the Repository for the latest status. The following describes the fields of the dialog that require additional explanation:

- **User Name**: The Repository login ID of the user who has the object checked out.
- **Machine Name**: The name of the computer, as defined in the Windows System Properties, that was used when the object was checked out.

See Also

- Viewing the Checkin History of a Repository Object
- Saving Changed Items to the Repository (Checking In)

Viewing the Checkin History of a Repository Object

Using the Version History feature, you can view the checkin details to see who has been working on the object or determine which version you need, in the case where you need to rollback to a previous version.

**Note:** Each time you check in an object, ER Studio Data Architect increments the version number by 1. When you check out an individual object such as a domain, macro or data movement rule, modify it, and then check it in, the version number for just that object is incremented; if its parent objects were not checked out or not checked in at the same time, their version numbers are not incremented. However, when you check out an attribute, the entity that contains the attribute is also checked out along with all other attributes contained by that entity. So, when you check in the changes to an attribute, the version numbers for the changed attribute, the unchanged attributes, and the entity are incremented.

1. Open a Repository diagram and then log in to the Repository.
2. On the **Data Model**, **Data Dictionary**, **Data Lineage**, or **Macros** tab, navigate to the object you want to investigate.
3. Right-click the object and then click **Version History**.
4. In the **Repository Version History** dialog, click a version entry and then click **Details**. This displays the recorded version details, including the comments entered when the object was checked in.

See Also

- Saving Changed Items to the Repository (Checking In)
- Determining Who has an Object Checked Out

Working with Named Releases of Repository Diagrams

Named Releases are saved versions of a diagram, with a specified name. This version of the diagram can later be retrieved from the Repository for viewing or editing.

The following describe functions you may want to perform on named releases:

- Set a Named Release
- Get a Named Release or a Diagram
- Delete a Named Release

Set a Named Release

Saves the latest version of a diagram as a Named Release, with the specified name. This version of the diagram can later be retrieved from the Repository for viewing or editing.

1. Log in to the Repository.
2. Open the diagram that you want to set as a named release and then check it in to the Repository.
3. Choose Repository > Releases > Set Named Release.

ER Studio Data Architect adds the Named Release under the respective diagram. Named releases are indicated by a camera icon in the Explorer tree.

Notes

- When setting a Named Release, it is important to remember that uniqueness is enforced through the naming convention. You can determine a naming convention when setting a Named Release. If you want to associate the Diagram's file name with the Named Release, then you must enter the Diagram's file name as part of the name of the release.
- You can view a Named Release, but you cannot modify it. A special Repository operation, Rollback Diagram, lets you Check In a Named Release as the latest version of a diagram. Once the Rollback operation is completed, you can Check Out the diagram (or a portion of it) to modify it.
- When naming the Named Release, the name must be unique. We recommend using a naming convention that includes the name of the diagram in the release. The uniqueness is enforced at the Repository level, not the diagram level.

Get a Named Release or a Diagram

Using ER/Studio Repository, you can download named releases and diagrams from the Repository to a specified directory on the local machine. The Get From Repository operation lets you download diagrams, submodels, and named releases from the Repository. The Get From Repository dialog displays all diagrams and projects in the Repository and lets you expand each diagram node to show its submodels and named releases.

1. To get a named release, choose Repository > Releases > Get Named Release.
   To get a diagram, choose Repository > Releases > Get Diagram.
2. Complete the **Get From Repository** dialog as required.

The following describes options that are not self explanatory:

- *Repository tree*: Displays nodes for each Diagram in the Repository. Click the plus sign next to a node to expand and show all submodels and named releases. You can select individual objects to get from the Repository.
- *General Information*: Displays information about the repository, and the model or submodel selected. Information displayed includes Repository Version, Number of Models, Number of Nested Projects, and Users with checked out objects.
- *Check Out Immediately*: Available for submodels only. Immediately Checks Out to the local machine the items selected in the tree.
- *Check Out Exclusively*: Available for submodels only. If you have chosen Check Out Immediately, you can choose this option to immediately Check Out Exclusively to the local machine the items selected in the tree.

Notes

- You can make changes to objects you get, but changes do not reflect in the repository copy of the dm1 file until you check out and then check in the changes.
- You can view a named release, but, in general, you cannot modify it. A special repository operation, Rollback Diagram, lets you check in a named release as the latest version of a diagram. Once the Roll Back operation is completed, you can check out the diagram (or a portion of it) to modify it.
- If the local diagram has the same name as the diagram in the repository you want to get, you must rename the local copy of the diagram with a **.bak* file extension. If you do not rename the local copy, you cannot complete the Get From Repository operation.
- If a file exists in the active file directory with the same name as the selected diagram or the same name as the diagram that the selected submodel belongs to, then there are several possible outcomes:
- If the local file is the same as the selected diagram or submodel, then ER Studio Data Architect opens it and refreshes it with the latest Repository information.
- If the local file is from the same diagram, but contains a different submodel than the user selected (or the local file contains the entire diagram and the user selected a submodel, or vice versa), then ER Studio Data Architect checks the local file for checked out objects. If any objects are checked out, ER Studio Data Architect opens the local file instead of the selected diagram or submodel. This lets the user Check In their work. If no objects are checked out, the ER Studio Data Architect deletes the local file and replaces it with the chosen diagram or submodel.
- If the local file is not a DM1 file, is a DM1 file from a previous version of ER Studio Data Architect, is a DM1 file with a different diagram GUID, or is a DM1 file that has not yet been added to the repository, ER Studio Data Architect renames the file with a BAK

extension and ER Studio Data Architect "Gets" the target diagram or submodel to the correct location.

- When you choose to get a named release, ER Studio Data Architect gives you the option of specifying where to place the named release file. To see this option, you must select a named release in the tree (the UI is hidden and replaces the check out UI when the named release is selected).

Delete a Named Release

Lets you delete Named Releases of diagrams stored in the ER/Studio Repository. To delete a Named Release, you must already have Named Releases set in the server.

1. Choose Repository > Releases > Delete Named Release.
2. Click the **Select a Named Release** list and then select the target release.

See Also

- Viewing the Checkin History of a Repository Object
- Branching and Merging Diagrams
- Rolling Back a Diagram to a Previous Version

Branching and Merging Diagrams

Branches are typically needed for testing or development purposes and then, once stabilized, merged into the currently deployed model. You can branch and sub-branch as often as you want.

1. Open the target diagram for branching or merging.
2. Choose Repository > Diagrams > Branch/Merge Diagram.
3. To create a branch of the active diagram, from the Branch Tree, select the diagram to be branched and then click the Branch button.
   To merge the active diagram with another diagram, from the **Branch Tree**, select the diagram or branch of the diagram to be merged and then click **Merge**.

The following describes options that may not be self-explanatory.

Display deleted diagrams: Select this option to view all branches of the diagram, including those that were previously deleted. Deleted diagrams are unavailable and cannot be selected to use with the Branch or Merge functions.

Notes

- It is good engineering practice to stick with a single enumeration convention, be it alphanumeric or some other series, and to use descriptive names.
- You can select whether the *.dm1 file name or the diagram name is displayed by changing the display option: Choose Repository > Options > General and then change the Diagram Name Display Format.

See Also

- Rolling Back a Diagram to a Previous Version
- Working with Named Releases of Repository Diagrams

Rolling Back a Diagram to a Previous Version

Use the Rollback function to replace the most current version in the repository to a previous Named Release of the entire diagram. The Rollback Diagram operation is particularly useful when you want to return to a previous version of your diagram because a later version has changes you no longer need or want to use.

1. Log in to the Repository and get a Named Release.
2. Review the Diagram to ensure you want to rollback to a previous release.
3. Choose Repository > Diagrams > Rollback Diagram.

Notes

- When the rollback operation is completed, you can check out the diagram (or a portion of it) to modify it.
- We recommend that you review the diagram to be sure that you want to rollback to this version.

See Also

- Branching and Merging Diagrams
- Viewing the Checkin History of a Repository Object

Working with Data Dictionaries in the Repository

Enterprise Data Dictionaries (EDD) lets you share a single data dictionary amongst multiple diagrams in the same Repository. Instead of being stored in a particular DM1 file, an EDD is stored in the Repository, and bound to the DM1 files that use it. A change made to an EDD propagates to any DM1 to which dictionary is bound.

You can have multiple dictionaries per diagram. This means that you can have one all-encompassing dictionary or project-level dictionaries.

- It is a good idea to check in the Enterprise Data Dictionary after adding new objects. Otherwise, other users will not be able to use them or the current diagram will not check in successfully if the new dictionary objects have been used.

The following items are included in this section:

- Determining Where an Enterprise Data Dictionary is Used
- Creating an Enterprise Data Dictionary
- Associating a Data Dictionary with a Diagram
- Unbinding an Enterprise Data Dictionary from a Diagram
- Checking out the Data Dictionary
- Checking in the Data Dictionary
- Undoing a Data Dictionary Check Out
- Redoing a Data Dictionary Redo Check Out
- Associating the Enterprise Data Dictionary with Repository Diagrams
- Retrieving an Enterprise Data Dictionary

See Also

- Cancelling a Repository Operation
- Setting Repository and ER/Studio TS Options

Determining Where an Enterprise Data Dictionary is Used

Before changing an Enterprise Data Dictionary, you may want to know which diagrams use the dictionary.

Choose Repository > Data Dictionary > Enterprise Data Dictionary Bindings.

See Also

- Creating an Enterprise Data Dictionary
- Associating a Data Dictionary with a Diagram

Creating an Enterprise Data Dictionary

You can promote a regular Data Dictionary to an Enterprise Data Dictionary (EDD). Alternatively, you can create a new EDD and add objects to it, just as for a regular Data Dictionary.

Promote Local Data Dictionary to Enterprise Data Dictionary

1. Ensure the diagram containing the data dictionary you want to promote is the active diagram.
2. Log in to the Repository.
3. Choose Repository > Diagrams > Add Diagram.
4. On the **Add Diagram to ER/Studio Repository**, select **Promote Local Data Dictionary To Enterprise**.

Create a New, Empty Data Dictionary

1. Log in to the Repository.
2. Open a Repository diagram.
3. The Create Enterprise Data Dictionary command is disabled when you are not displaying a Repository diagram.
4. Choose Repository > Data Dictionary > Create Enterprise Data Dictionary.

Notes

- Once created, you can bind the EDD to a drawing and add objects just as you would to a local Data Dictionary.
- The new EDD is automatically bound to the current diagram.
- Unlike regular Data Dictionaries, EDDs do not include a Reusable Procedural Logic node, and therefore cannot contain triggers, procedures or libraries.

See Also

- Associating a Data Dictionary with a Diagram
- Determining Where an Enterprise Data Dictionary is Used

Associating a Data Dictionary with a Diagram

Binding an existing Enterprise Data Dictionary with a diagram enables you to reuse data dictionary objects such as domains, reference values, and attachments.

The Enterprise Dictionary Binding Dialog now includes attachments and reference values. The bindings are presented in a grid format and can be exported to *.csv files for reporting purposes.

1. Log in to the Repository.
2. Check out the diagram you want to bind the dictionary to.
3. Choose Repository > Data Dictionary > Bind Enterprise Data Dictionary and then select the dictionary you want to associate with.

Notes

- You can associate multiple data dictionaries with the same diagram.
- You must have the Enterprise edition of ER Studio Data Architect to have more than one data dictionary in a diagram.
- All changes to dictionary objects in an Enterprise Data Dictionary will propagate to diagrams where the object is used. If you want to use a domain or attachment only, for example, in one specific diagram, it can remain in the local dictionary. If a domain or attachment needs to be reused and updated across diagrams, then it should be created in an Enterprise dictionary.
- You can have multiple dictionaries per diagram. This means that you can have one all-encompassing dictionary or project-level dictionaries.
- You must check out the diagram before binding a data dictionary to it.
- You do not need to check in the diagram to commit the change to the Repository. The Bind Enterprise Data Dictionary operation automatically updates the Repository server with the appropriate information.

See Also

- Determining Where an Enterprise Data Dictionary is Used
- Creating an Enterprise Data Dictionary

Unbinding an Enterprise Data Dictionary from a Diagram

You can unbind an Enterprise Data Dictionary from a Diagram when you no longer need to use it. The Remove Enterprise Data Dictionary operation automatically updates the Repository server with the appropriate information. For example, a user has a Diagram that has an attribute bound to a domain in an Enterprise Data Dictionary called MyEnterpriseDD. When the user removes MyEnterpriseDD from the diagram, subsequent users who get the Diagram from the Repository will get a version of the Diagram that does not have MyEnterpriseDD in it; furthermore, the attribute is unbound from the domain.

1. Check out the diagram.
2. On the **Data Model Explorer**, select the **Data Dictionary** tab.
3. Choose Repository > Data Dictionary > Remove Enterprise Data Dictionary.

Notes

- You must check out the diagram before removing a data dictionary from it.
- You do not need to check in the diagram to commit the change to the repository.
- Any attributes that were bound to Data Dictionary items will be modified. If the attributes are not checked out from the repository before the data dictionary is removed, then ER Studio Data Architect automatically performs a delayed Check Out.

See Also

- Determining Where an Enterprise Data Dictionary is Used
- Checking out the Data Dictionary

Checking in the Data Dictionary

The Check In operation lets you Check In a Data Dictionary when you are done working with it. The Repository will attempt to merge all of your modifications into the Repository. If there are any conflicts during the merge process, you are prompted to resolve them. After you have resolved all conflicts and the merge is complete, the merged results are added to the Repository and your diagram. The merged results are added to your Data Dictionary so that it remains in sync with what is in the Repository.

When you Check In a Data Dictionary, it is possible that the Repository server will create new versions of Objects that were bound to individual Data Dictionary objects. For example, if a user checks in a Data Dictionary after deleting a user Datatype, and the user datatype was bound to an attribute, the server will create a new version of the attribute that does not contain the binding. This is because deleting a Data Dictionary object causes all bindings to it to be removed. The new version is reflected in the Version History for the attribute, with an automatically generated comment. When you do a Get Latest Version on the affected attribute, the new version without the binding is placed in your copy of the Diagram. If you modify or delete the affected attribute before Checking In the Data Dictionary, then a merge conflict will appear between your version of the attribute and the version of the attribute created by the Repository server. Select the ER Studio Data Architect version of the attribute to keep your changes.

This behavior can also occur when you check in a data dictionary with a modified data dictionary object. For example, say you change the data type of an attachment, then check in the data dictionary to which it belongs. Any objects bound to that attachment must have the value override removed because the value override is only valid for a particular attachment data type. This means that the objects (which can be in the data dictionary or the diagram) that are bound to that attachment get a new version in the Repository that reflects the removal of the object's value override.

On the **Data Model Explorer**, click the **Data Dictionary** tab and then click the target data dictionary object.


Note: If you made changes to the Data Dictionary, you need to check in the Data Dictionary before checking in the diagram. For example, if you create a domain in the Data Dictionary and use that domain in the diagram, if you check in the diagram before the Data Dictionary, ER Studio Data Architect produces errors.

Notes

- The Repository does not support the Check In of individual Data Dictionary objects. You can only Check In the entire Data Dictionary.
- If your version of the Diagram is in conflict with the Repository's version of the Diagram, the Merge Conflict Resolution dialog box opens. For more information about merge conflicts and their resolution, see *Branching and Merging Diagrams*.
- If you made changes to the Data Dictionary, you need to check in the Data Dictionary before checking in the diagram. For example, if you create a domain in the Data

Dictionary and use that domain in the diagram, if you check in the diagram before the Data Dictionary, ER Studio Data Architect produces errors.

See Also

- Determining Where an Enterprise Data Dictionary is Used
- Checking out the Data Dictionary
- Redoing a Data Dictionary Redo Check Out

Checking out the Data Dictionary

The Check Out operation lets you copy the latest version of a Data Dictionary or a Data Dictionary object so you can modify them.

1. On the **Data Model Explorer**, select the **Data Dictionary** tab and then click the target data dictionary object.
2. Choose Repository > Data Dictionary > Check Out Data Dictionary Object.

Notes

- You can check out an entire data dictionary from the repository.
- You can check out an individual data dictionary object. If the data dictionary object is not bound to any attributes, you can check out or check in the data dictionary object without affecting the diagram. If the data dictionary object is bound to an attribute, and changes to the data dictionary object affect the attribute, then the attribute is marked for a delayed check out.
- The repository check out operation is the same for data dictionaries and data dictionary objects. The menu depends on what you select in the Data Model Explorer.
- You can check out a data dictionary or data dictionary objects with 'exclusive' rights. This restricts any other user from checking out the data dictionary or data dictionary objects while you have either or both checked out, and means that you are the only who modify them and submit data to the Repository. When you check out something exclusively, other users have read-only and/or 'Delayed Check Out' access while the objects are checked out.
- If the data dictionary's objects are not bound to any attributes, you can check out the data dictionary without affecting the diagram.
- If the data dictionary object is bound to an attribute, and changes to the data dictionary object affect the attribute, then the attribute is marked for a delayed check out.

See Also

- Checking in the Data Dictionary
- Undoing a Data Dictionary Check Out

Undoing a Data Dictionary Check Out

The Undo Check Out operation transfers the latest version of the Diagram or Object from the Repository to your machine. You can view the Diagram, but is no longer able to modify it.

Choose Repository > Data Dictionary > Undo Check Out Data Dictionary.


Note: When you undo the check out of a data dictionary, any changes made to the Diagram before doing the Undo Check Out operation are discarded.

See Also

- Checking in the Data Dictionary
- Redoing a Data Dictionary Redo Check Out

Redoing a Data Dictionary Redo Check Out

The Redo Check Out operation transfers the latest version of the Diagram or objects from the Repository to your machine. Because you have the diagram or Objects Checked Out, you can view and modify them.

1. On the **Data Model Explorer**, select the **Data Dictionary** tab and then click the target data dictionary.
2. Choose Repository > Data Dictionary > Check Out Data Dictionary.

**Note:** Any changes made to the Diagram before doing the **Redo Check Out** operation are discarded.

See Also

- Undoing a Data Dictionary Check Out
- Checking in the Data Dictionary
- Checking out the Data Dictionary

Associating the Enterprise Data Dictionary with Repository Diagrams

Using Data Dictionary bindings, you can see and control the associations of Enterprise Data Dictionaries across all your diagrams in the Repository. The Enterprise Data Dictionary Bindings dialog provides a list of all your Enterprise Dictionaries and shows you where its objects are in use. For example, if you have a 'name' domain in an Enterprise Data Dictionary used across all or some of your diagrams, you can navigate to the domain to see what entities, tables, and models are used, which provides some impact analysis about which model objects will be affected by a change to the domain.

1. Open the Enterprise Data Dictionary Bindings Dialog
2. Choose Repository > Data Dictionary and then click **Enterprise Data Dictionary Bindings**.

See Also

- Associating a Data Dictionary with a Diagram
- Creating an Enterprise Data Dictionary

Retrieving an Enterprise Data Dictionary

To retrieve an enterprise data dictionary, you must either open a repository diagram and bind the enterprise data dictionary to it or open a Repository diagram that already has the data dictionary bound to it.

To bind an enterprise data dictionary to a repository diagram, see Associating a Data Dictionary with a Diagram.

To determine which repository diagrams have the enterprise data dictionary bound, see Associating the Enterprise Data Dictionary with Repository Diagrams.

See Also

- Determining Where an Enterprise Data Dictionary is Used
- Associating a Data Dictionary with a Diagram

Working with Repository Projects

Repository projects enable you to organize your models into project groups, which can be based on work groups, types of diagrams, or any other structure that suits your business environment. You can add Subprojects, and in nested hierarchies. You can create Subprojects as working copies of models to allow wider audiences access to them.

- To secure projects, Repository Roles are used, which can be assigned to projects that will cascade permissions to any diagrams managed within the project. For more information, see "Establishing Security for the Repository" in the *ER/Studio Repository Installation and Administration Guide*.

Create, Delete, Rename, and Define Repository Projects

Projects let you organize your diagrams in groups. You can create projects based on work groups, types of diagrams, basically any structure that suits your business environment.

1. Log in to the Repository.
2. Choose Repository > Project Center.
   The Repository Project Center dialog displays.
3. *To create a new project*, click **New**.
   To delete a project, on the projects tree, click the project you want to delete, and then click **Delete**. If necessary, in the **Delete Project Warning** dialog, choose how to deal with any nested projects.
   To rename a project, on the projects tree, click the project you want to rename, and then click **Rename**. In the **Repository Rename Project** dialog, provide a new name for the project and then click **OK**.
   To add a diagram to a project, on the projects tree, click the target project, from the **Available Diagrams** list, select the diagrams you want to include in the project, and then click the right arrows to move the diagrams to the **Selected Diagrams** list.
   To remove a diagram from a project, on the projects tree, click the target project, in the **Selected Diagrams** list, select the diagrams you want to remove from the project, and then click the left arrows to move the diagrams to the **Available Diagrams** list.
   - *To preserve the nested projects*, click **Preserve**. This causes the nested projects to be moved up a level.
   - *To delete all nested projects*, click **Delete All**. This causes the diagrams under the nested projects to be moved to the level above the deleted project.
4. Click **Apply** or **OK**.

Notes

- A diagram can either be in one project or not in any project at all.
- When you delete a project all underlying components of the project, such as the diagrams and roles, are not deleted. Diagrams and roles associated with deleted projects will remain available to the users with the appropriate privileges.

See Also

- Cancelling a Repository Operation
- Best Practices

Cancelling a Repository Operation

You can cancel a Repository operation, such as **Add Diagram**, **Check Out** or **Check In** Objects or Diagrams, at any time up to the point at which the client receives confirmation that the operation has completed.

Click the **Cancel** button on the **Status Bar** or click the **Cancel** button on the **Repository Operation Status** dialog box.

See Also

- Working with Repository Projects
- Best Practices

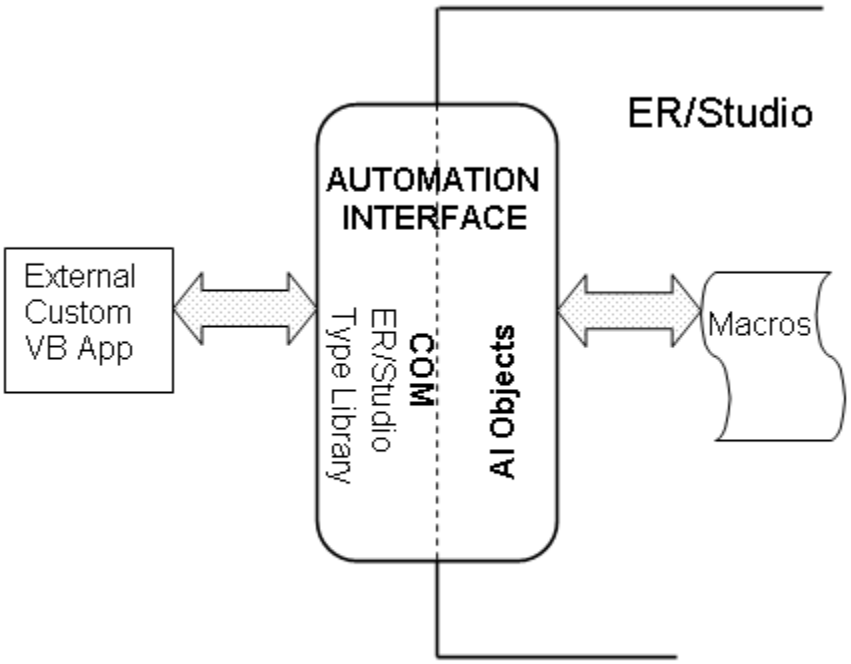
Automating ERStudio DA

Automation Interface objects are scriptable controls for most ER Studio Data Architectta Architect (ER Studio Data Architect) functions. You can create macros to automate repetitive tasks or to extend the functions already found in the application. A few days coding can save months of human resources in the long run as described in Why Use Automation Objects?.

This section is intended for programmers with a working knowledge of BASIC. You can learn it using ER Studio Data Architect's macro editor, but you will also need a textbook devoted specifically to beginner BASIC programmers. The macro language is SAX BASIC, similar to MS Visual BASIC. There are a few, but critical, syntactic differences from other BASIC languages. See the Introduction to the *Automation Interface Reference Guide*, which is accessible from the Help menu, to save some debugging time.

You can also create event handlers to run automatically every time you perform an action in the application, such as Add Entity. See Creating Event Handlers. You can also write handlers for update events, see Handling Update Events.

A number of example macros are included. To see them, open any ER Studio Data Architect **.dm1* file, switch to the Macro tab in the Data Model Explorer, and double-click any sample macro. The macro will open in the Macro Editor.



Interface Relationship to Internal Macros and External Applications

The diagram above shows the differences between using macros within ER Studio Data Architect and running your own COM application external to ER Studio Data Architect. In the external case, you will need to create a link to the ER Studio Data Architect Type Library and declare ERStudio object types. Example applications using MS Visual Basic 6.0 and.Net VB and

C# are in your installation directory under "..\Program Files\ERStudio Data Architect X.X\Readme\TableList.zip."

Internally, the only difference is that the namespace, type library, and top-level application objects are already created before you run or edit a macro. Otherwise the code for external and internal applications will be nearly identical. You can copy and paste code from your macros to an external application with few changes.

For example, if you want to let users automatically export ER Studio Data Architect model metadata to applications like Microsoft Excel, and simultaneously add the created file to a Microsoft Outlook message for distribution, you can write a macro to take advantage of this object interaction.

Topics covered in this section include:

- Why Use Automation Objects?
- Automation Objects Programmer's Guide
- Sample Macros Installed with ER Studio Data Architect
- Running ER Studio Data Architect Macros from Windows
- Creating Event Handlers
- Handling Update Events
- Adding Macros to GUI Modeling Menus
- Using the SAX Basic Macro Editor

See Also

- Why Use Automation Objects?
- Using the SAX Basic Macro Editor
- Access ER Studio Data Architect from Command Line
- Objects and Object Models
- Automation Objects Programmer's Guide

Why Use Automation Objects?

Automation Interface Objects are scriptable controls for most of the functionality in ER Studio Data Architect. You can create custom macros to automate repetitive tasks or to extend the functions already found in ER Studio Data Architect. For example:

- You might want to write an external program to control ER Studio Data Architect if you are integrating several applications into one business process, or to re-use a body of legacy code.
- Within ER Studio Data Architect you might want to add entities to twenty different models. You want each entity to be the same color, to have the same attributes and notes. You could do each one by hand, over and over. Or with the Automation Objects, you can automatically create the entities with the necessary attributes and notes. Each time you run the script, the operation is executed exactly the same way ensuring that there are no forgotten attributes or misnamed entities.
- You can create event handlers to run automatically every time you perform some GUI action within ER Studio Data Architect, such as, Add Entity. See Creating Event Handlers. You can also write handlers for update events, see Handling Update Events.

Many scripts begin as modifications to existing macros, so pick an ER Studio Data Architect macro that approximates what you would like to do and change it. The macros included with ER Studio Data Architect are described in Sample Macros Installed with ER Studio Data Architect. You can also connect with other users to share in their knowledge through the ER Studio Data Architect User Group and Online Forum. From the home page of www.embarcadero.com, choose **Services** > **Communities**.

Objects and Object Models

Object modeling is the central concept of modern object-oriented software. For further information on object modeling, see any object-oriented programming text, or try the UML (Unified Modeling Language) Web site. It is not necessary to understand UML to use the automation interface. The organization of controls can seem more intuitive to those familiar with object-oriented software. Otherwise, the summary below can be helpful to newcomers.

Object-Oriented Programming

An "object" is some code functions and the data used by those functions. Object-oriented models wrap the (presumably related) functions and data into a single package with one name, its "object" name. The word "function" essentially means "method." Similarly "data" is "property" or "attribute."

One analogy is an automobile engine, where we have an "Engine" Object. An object property might be the "Engine.Speed." A method would be the "Engine.Tachometer" which gets "Engine.Speed" and displays it.

Similarly, an entity object has an Indexes method that gets the current Indexes and displays them.

See Also

- Automation Objects Programmer's Guide
- Using the SAX Basic Macro Editor

Automation Objects Programmer's Guide

This section details specific coding tasks common to ER Studio Data Architect macros. There are a few, but critical, syntactic differences from other BASIC languages.

This is intended for programmers with a working knowledge of BASIC. You can learn it using ER Studio Data Architect's macro editor, but you will also need a textbook devoted specifically to beginner BASIC programmers.

Note on Properties Get/Let

A property can be defined as Get/Let, meaning you can read the property (Get) or change it (Let). However, many such properties are set by ER Studio Data Architect *and are in fact read-only*, that is Get but not Let. For example, most of the numeric IDs are Get-only.

In general, if you can edit a property using ER Studio Data Architect's GUI design controls, and there exists an automation object for it, it will exist as Get/Let. If you can't edit it in ER Studio Data Architect, it's read-only (Get -only).

Note on Use of "Set"

The code samples do not always use "Set." In the SAX BASIC Editor within ER Studio Data Architect, you must use the function *Set* on the first instance of the object in a script (or block of code within object scope). You must use it again if the object reference changes. This is distinct from coding for BASIC executable images where you can declare objects *Public* (etc.) and not have to *Set* them, as in Microsoft's VB.Net.

An ER Studio Data Architect Macro Editor Example:

Set MyDiagram = DiagramManager.ActiveDiagram

'some code that closes the active diagram and opens another diagram

Set MyDiagram = DiagramManager.ActiveDiagram

Note on Dim [what] As [ever]

SAX BASIC will not allow you to declare and initialize any variable in one statement. Example:

'No way:

Dim MyData As Integer = 30

'This is OK:

Dim MyData As Integer

MyData = 30

Related Topics

Object Model Hierarchy Diagram

Instantiating the Parent Object in the Macro Editor

Instantiating the Parent Object in an External Application

Accessing Collections of Objects

Assigning Object Appearance and Behavior Using Properties

Performing Complex Actions Using Methods

Sample Macros Installed with ER Studio Data Architect

Running ER Studio Data Architect Macros from Windows

Creating Event Handlers

Handling Update Events

Adding Macros to GUI Modeling Menus

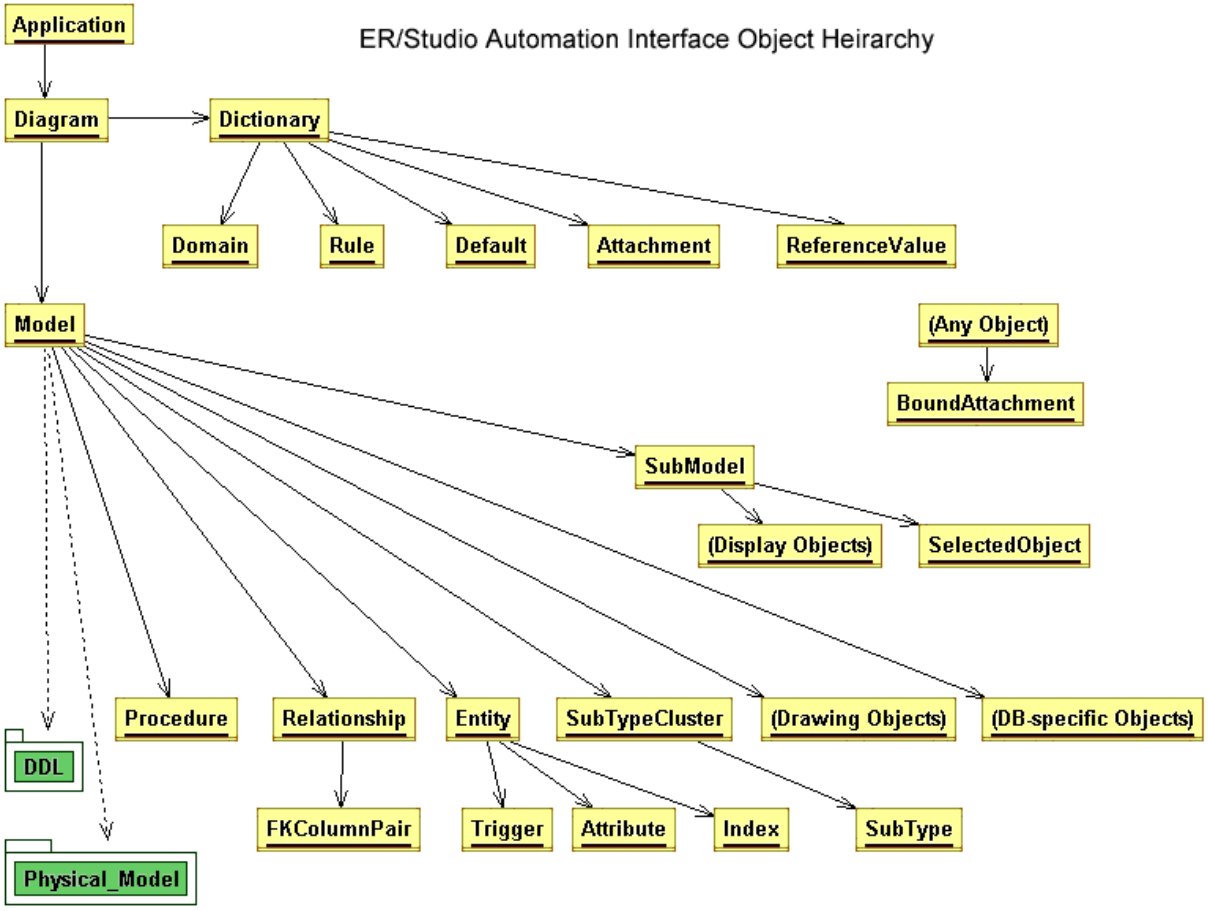
Automating ERStudio DA

See Also

- Sample Macros Installed with ER Studio Data Architect
- Running ER Studio Data Architect Macros from Windows

Object Model Hierarchy Diagram

The conceptual diagram below shows the hierarchical relationships among major objects. When writing macros you'll generally instantiate objects in the order shown. Collection objects, such as Lines and Entities or Indexes have been omitted for simplicity.



See Also

- Why Use Automation Objects?
- Sample Macros Installed with ER Studio Data Architect

Instantiating the Parent Object in an External Application

The namespace and context differ somewhat outside of ER Studio Data Architect's Macro Editor. External programs use the COM interface. For example, in an external Visual BASIC project, the base object is *ERStudio.Application*, whereas the base object in the ER Studio Data Architect Macro environment is *DiagramManager*. The object models and instantiation sequence are otherwise identical.

There is a working example VB application installed with ER Studio Data Architect. It is located in your installation directory under "..\Program Files\ERStudio Data Architect X.X\Readme\TableList.zip." Other examples of this type exist in MSVB6,.Net VB, and C#, and are in the same directory.

To run it, open the archive file TableList.zip and follow steps 1-3 below.

If you are familiar with COM programming, skip to step 3 and the code segments below to configure your VB project for ER Studio Data Architect.

1. Copy *Pubs.dm1* to *C:*.
   Pubs.dm1 is bundled with the application and is in:
   - *Windows XP*: C:\Documents and Settings\All Users\Application Data\Embarcadero\ERStudioDA_X.X\Sample Models
   - *Windows Vista*: C:\ProgramData\Embarcadero\ERStudioDA_X.X\Sample Models
   - *AppWave ION*: My Documents\ERStudio Data Architect XE\Sample Models
2. Using MS Visual Basic 6.0 (or later), open the VB project file TableList.vbp.
3. Make sure your project refers to the ER Studio Data Architect type library:
   a) Choose **Project > References...**
   b) Select ER Studio Data Architect *Type Library* from the list.
4. Add the following declarations at the appropriate scope. They are at module scope in the example TableList.bas: '

   ```
   Option Explicit 'Require object declarations

   Public app As ERStudio.Application 'Specify as ER/Studio types rather than "Object"

   Public diagm As ERStudio.Diagram

   Public mdl As ERStudio.Model

   Public ent As ERStudio.
   ```

5. And instantiate in this way:

```
Set app = CreateObject("ERStudio.Application") 'Start ER Studio Data Architect if not running

Set diagm = app.OpenFile("C:\Pubs.dm1") 'Instantiate a diagram

Set mdl = diagm.ActiveModel 'Instantiate a model
```

Macros which run inside ER Studio Data Architect's SAX BASIC shell will use the pre-initialized 'DiagramManager' object. There are numerous examples of this in ER Studio Data Architect's Macro tab and in the Code Samples section of help.

Good Programing Practice: Use the *DiagramManager.GetLastErrorCode()* or *DiagramManager.GetLastErrorString()* to check for errors after method calls. *To summarize:* Outside ER Studio Data Architect you begin with an instance of *ERStudio.Application*. Inside you begin with DiagramManager, and an instance is already running.

See Also

- Why Use Automation Objects?
- Using the SAX Basic Macro Editor

Instantiating the Parent Object in the Macro Editor

The DiagramManager is the parent-level object required to access all other ER Studio Data Architect objects. The DiagramManager object is instantiated when you start the Macro Editor, which means you can immediately start using this object; you do not need to create it. Begin your code with:

Dim ActiveDiagram as Diagram

Set ActiveDiagram = DiagramManager.ActiveDiagram

If you are writing an external application, the parent object must be instantiated as **ERStudio.Application**. See Instantiating the Parent Object in an External Application.

See Also

- Why Use Automation Objects?
- Using the SAX Basic Macro Editor
- Instantiating the Parent Object in an External Application

Accessing Collections of Objects

Collections are groups of Objects of the same type. For example, the Models object is a collection of all the Models in a Diagram.

Each Collection has four methods:

- Item (), this is the default method
- Add()
- Remove()
- Count.()

You can access objects by name (and sequence number or technical key, when appropriate). The following code sample illustrates default behavior: `Dim myentity as Entity`

`Set myentity = mymodel.Entities("Entity1")` In this example, the Item() method in the Entities collection is actually called to return the Entity1 object, but the script does not need to contain an explicit reference to the method because it is the default method.

The Add(), Remove(), and Count() methods are supported for all collections, except where noted. These methods modify the internal data in ER Studio Data Architect immediately. The Add() method adds a new object into the collection. Add() fails if the object is not properly initialized. The Count() returns the number of objects in the collection.

Iterating Through Collections

The most-often used way of retrieving each object in a collection is with the *For Each... Next* loop. Note the use of the variable *HowMany* to start and iterate the loop below.

' Sets the entity name for each entity in the given collection`Dim MyDiagram As Diagram`

```
Dim MyModel As Model

Dim MyEntity As Entity

Dim EntName As String

Dim HowMany As Integer

Set MyDiagram = DiagramManager.ActiveDiagram

Set MyModel = MyDiagram.ActiveModel

HowMany = 1

' Iterates through the Entities collection to rename all entities

For Each MyEntity In MyModel.Entities

' Uses CStr function to convert the Count variable to a string

EntName = "ShinyNewEntity" + CStr (HowMany)

MyEntity.EntityName = EntName
```

```
HowMany = HowMany + 1

Next MyEntity
```

See Also

- Why Use Automation Objects?
- Sample Macros Installed with ER Studio Data Architect

Assigning Object Appearance and Behavior Using Properties

Properties are Object attributes that determine Object appearance or behavior. Color, count, name and ID all are properties common to many objects. Some, such as ID, are usually read-only. Others can be changed. ER Studio Data Architect immediately reflects property modifications by updating internal data.

```
Dim MyDiagram As Diagram

Dim MyModel As Model

Dim MyEntity As Entity

Dim EntName As String

Dim EntID As Integer

Set MyDiagram = DiagramManager.ActiveDiagram

Set MyModel = MyDiagram.ActiveModel

EntID = 1 'We want the name of the Entity that ER Studio Data Architect has identified as ID =1

Set MyEntity = MyModel.Entities.Item(EntID)

EntName = MyEntity.EntityName

'Now we can display EntName in a MsgBox, write it to a text file, or
'change it and re-name the Entity.
```

See Also

- Why Use Automation Objects?
- Using the SAX Basic Macro Editor

Performing Complex Actions Using Methods

Methods perform actions more complex than the simple setting or reading of a property. For example, *OpenFile()* is a DiagramManager method that in turn invokes a number of other operations: it must navigate the directory/file system, allocate resources for file handling, keep track of the file handle and attributes and so on.

Dim MyDiagram As Diagram

Dim strFile As String

strFile = "C:\Documents and Settings\Owner\Application Data\Embarcadero\ERStudio\Model\MyModel.dm1 "

Set MyDiagram = DiagramManager.OpenFile(strFile)

See Also

- Why Use Automation Objects?
- Using the SAX Basic Macro Editor

Sample Macros Installed with ER Studio Data Architect

Sample macros are included with the installation of ER Studio Data Architect. An extensive set of working sample code is also included as part of the Automation Interface Reference Code Reference chapter in the on-line Automated Interface Reference.

The following provides brief descriptions of the functionality of the sample macros. The header of each macro includes more extensive usage details. Most installed macros are also in the help file in the Sample Code section for ease of cutting and pasting and for comparison purposes.

- Meta Data Management Macros
- Model Layout Macros
- Modeling Productivity Macros
- Physical Modeling Macros

Meta Data Management Macros

- *Attachment Example*: Exports attachment data for each bound attachment for every selected table in the active model.
- *Data Lineage Export to Excel*: Outputs the data lineage information to excel. If the format of the spreadsheet is unchanged you can update the mappings and import them back into the model.
- *Data Lineage Import from Excel*: Imports data lineage from Excel.
- *Domain Bindings Export to Excel*: Exports domain bindings for all the attributes in the current model.
- *Domain Bindings Import From Excel*: Imports domain bindings from Excel.
- *Export Domain Info to Excel*: Exports the domains and their properties to excel.
- *Export Index Column Info to Excel*: Generates an index column report for the active model in ER Studio Data Architect.
- *Export Model Meta Data to Excel*: Exports model meta data for the active model to Excel.
- *Export Model Meta Data to Word:* Generates a mini report for the selected entities in the active model.
- *Export Object Definitions and Notes to Excel*: Exports definitions and notes for tables, views, relationships and attributes to Excel.
- *Export Reference Value Info to Excel*: Exports the reference values and their properties to Excel.

- *Export Relationship Info to Excel*: Generates a foreign key column report for the active model in ER Studio Data Architect.
- *Import Domains from Excel*: Imports domains from the specified Excel spreadsheet.
- *Import Index Names from Excel*: Imports relationship names from Excel.
- *Import Object Definitions and Notes from Excel*: Imports definition and notes for entities, attributes, views and relationships from an Excel spreadsheet.
- *Import Reference Values from Excel*: Imports reference values from Microsoft Excel.
- *Import Relationship Names from Excel*: Imports relationship names from Microsoft Excel.
- *Submodel Report*: Generates a list of entities and their submodels for the active model. The output is an Excel spread sheet.

Model Layout Macros

The Model Layout macros all demonstrate how you can automate changing a model's appearance. You can use these macros to change a mode's appearance by aligning entities, changing the background or text color of entities and attributes, or combining several entities into one new entity.

- *Auto-Align Selected Entities Left*: Aligns all selected entities by their left-hand edges. This macro simulated the User selecting two or more entities in a model then clicking the Align Left button in the Alignment toolbar. Please note that you must select at least two entities before running this macro.
- *Auto-Color All Entities with FKs*: Illustrates how to use entity objects, entity display objects, and selected objects - and how to distinguish between them. The macro changes the background color of all entities with foreign keys to purple. First, all entities in the active submodel are selected. Next, the background color of all selected entities that contain foreign keys is changed to purple. Finally, all selected (highlighted) entities are deselected.
- *Auto-Combine Selected Entities*: Replaces all selected entities with one new entity that contains the non-foreign-key attributes from the selected entities. The new entity is placed in the diagram at the averages of the x and y coordinates of the selected entities. The new entity's name is a combination of the names of the selected entities, each separated by an underscore. Each attribute name is a combination of the originating entity name and the attribute name, separated by an underscore; attributes are renamed in this manner in order to ensure uniqueness in the new entity.
- *Change Submodel Display Properties*: Changes several display properties in the active submodel. In particular, it changes the font and color of non-key, non-inherited attributes. The font size and style for the attributes are also modified.

Modeling Productivity Macros

- *Add Base Attributes To Person Entity*: Adds base attributes to selected entities, which represent people. It will also introduce a primary key based ' upon the entity's name.

- *Add Definition to Type Entities*: Adds a definition to selected entities. The definition applied to the selected entity will also include the object's name automatically, as in "My definition text + entity name +(s)."
- *Add Parent Name Prefix to Propagating Key*: Adds the entity name as a prefix to all attribute role names. This macro demonstrates how to use the FKColumnPair object.
- *Add Table Name Prefix Globally*: Lets the user add a prefix string to the names of all entities in the active model. A dialog box prompts the User for the prefix. If the active model is a logical model, the prefix is assigned to all entity names. If the active model is a physical model, the prefix is assigned to all table names.
- *Add Table Name Prefix Selectively*: Lets the user add a prefix string to the names of the selected entities in the model. A dialog box prompts the User for the prefix. If the active model is a logical model, the prefix is assigned to the selected entities' names. If the active model is a physical model, the prefix is assigned to the selected tables' names.
- *Add Table Owner Globally*: For physical models, updates the owner field in the table editor for all tables in the active model.
- *Add Table Owner Selectively*: For physical models, updates the owner field 'in the table editor for all selected entities.
- *Add View Owner Globally*: For physical models, updates the owner field in the view editor for all tables in the active model.
- *Add View Owner Selectively*: For physical models, updates the owner field in the view editor for all selected entities.
- *Attribute Binding Macro*: Lists all the unbound attributes/columns of all the models (logical and all physicals) and domains. The attributes can then be bounded to any of listed domains by pressing the "Bind" button.
- *Auto-Convert Datatype*: Iterates through all selected entities to determine which attributes use the VARCHAR datatype, then changes the datatype to TEXT.
- *Auto-Create Data Dictionary and Bound Domain*: This macro follows several steps during its execution. First, it creates a new diagram and adds a rule, default, and domain to the Data Dictionary. Next, it binds the rule and the default to the domain. Then it adds an entity to the diagram, and adds an attribute to the entity. Finally, it binds the domain to the attribute.
- *Auto-Create Data Dictionary*: Creates a user-defined data dictionary quickly. The macro can be used as a template to create user-defined or business-specific data dictionaries. This macro can be inserted into ER Studio Data Architect's ERSBasicHandlers system (specifically in the '"CreateDiagramHandler(CurDiagram As Object)" section of ERSBasicHandlers) and if the Create Handlers option is checked on in ER Studio Data Architect's Automation Interface Options, this Data Dictionary will be created and populated any time a user creates a new diagram.
- *Auto-Create New Diagram*: Creates a new diagram that contains six entities in the logical model, then generates a physical model that uses the IBM DB/2 database platform.

- *Generate Constraints From Reference Values*: Generates constraints from the defined reference values.
- *Convert Name Case*: Converts the names of the selected tables or entities in the active model to all upper- or lower-case letters. A dialog prompts the User to decide if the names should be in upper- or lower-case letters.
- *Definition Editor*: Lists all the tables and allow 'the user to update the definition field for the table, by 'pressing the "update" button. There will also be a list of columns for the respective table that you can use to update the definitions for each column.
- *Domain Bindings*: Opens a dialog that shows all domains in the diagram's Data Dictionary, and all attributes that are bound to a selected domain. The User can view which attributes are bound to a selected domain; the User can also unbind specific attributes from the selected domain.
- *Example macro-Loop Through Current Submodel*: Demonstrates how to loop through the objects of the current submodel.
- *Get Related Entities*: This macro selects the related parents and/or child of the selected tables. To use the macro lasso a group of entities on the diagram or select them in the diagram tree, then right click on the macro to execute. Parents and children will be selected depending if the option is checked. This macro can be used to assist in submodel creation.
- *Import Data Dictionary*: Imports Data Dictionary objects from a specially formatted text file. The macro contains guidelines in its comments that outline and describe the file format that must be used.
- *Index Naming*: Applies naming conventions to all types of indexes. It provides an option to use the selected tables or all tables.
- *Name Foreign Constraints*: Prompts the user with a dialog to specify the naming convention for Foreign Constraints. It will then name all the constraints using the parent and child table names. It will also make sure the name is unique by adding an index for the last characters of duplicate names.
- *Name Primary Constraints*: Names all primary key constraints with the given naming conventions. The table name with either a prefix or suffix.
- *Notes Editor*: Lists all the tables and allow the user to update the notes field for the table, by pressing the "update" button. There will also be a list of columns for the respective table, that the user can use to update the notes for each column.
- *Selectively Output PK and FK DDL*: Outputs DDL for primary and foreign constraints for all selected tables. To operate, selected the desired tables, then right-click on the macro to execute. The DDL can be previewed or written to a file.
- *Switch Domain Bindings*: Scans all the columns and attributes in the active model or all models and switches the domain bindings from the source domain to the target domain. The information for each bound column will be updated with the target domain. Any domain overrides will be preserved.

Physical Modeling Macros

The physical modeling macros are divided into folders for specific database platforms. The following describes the Physical Modeling Macros:

IBM DB2

- *Generate Partitions for OS390 Clustered Indexes*: Provides and interface to add multiple partitions to clustered indexes. A list of tables is provided. Selecting a table will load the clustered indexes to be partitioned. The storage parameters are uniform across partitions.
- *Selectively Add DB2 Permissions to PostSQL*: Adds permissions to the PostSQL of any selected table in a DB2 physical model.
- *Selectively Update OS390 Index Storage*: Provides a list of tables from the active physical model. The tables in the right box will be updated with the specified storage parameters. Tables can be added and removed from the updated list.
- *Selectively Update OS490 Table Storage*: Provides a list of tables from the active physical model. The tables in the right box will be updated with the specified storage parameters. Tables can be added 'and removed from the updated list.
- *Update DB2 OS390 Index Storage Parameters*: Provides a list of tables from the active physical model. The tables in the right box will be updated with the specified storage parameters. Tables can be added and removed from the updated list.
- *Update DB2 OS390 Table Storage Parameters*: Provides a list of tables from the active physical model. The tables in the right box will be updated with the specified storage parameters. Tables can be added and removed from the updated list.

MS SQL Server

- *Selectively Add MS SQL Server to PostSQL*: Adds permissions to the PostSQL of any selected table in an SQL Server physical model.
- *SQL Server Storage Update*: Opens a dialog that lets you update table parameters for a user-specified file group, or update index parameters for a user-specified file group and fill factor.

Oracle

- *Insert Synonym in PostSQL*: Opens a dialog that prompts the user to specify a synonym for a specific table or view. The synonym code is inserted into the PostSQL of the specified table or view. The active model must be an Oracle physical model in order to run this macro.
- *Selectively Add Oracle Permissions to PostSQL*: Adds permissions to the PostSQL of any selected table in an Oracle physical model.

- *Selectively Add Oracle Sequence*: Presents a dialog that allows the user to specify the naming conventions and options for an Oracle sequence. The sequence is added to the PreSQL of the selected tables.
- *Selectively Update Index Storage Parameters*: Updates the Oracle index storage parameters for the selected tables.
- *Selectively Update Oracle Storage*: Opens a dialog that lets you set Oracle-specific properties to selected entities in a model. Some of the properties that you can set are the tablespace, initial extent size, and next extent size.

Sybase

- *Selectively Add Sybase Permissions to PostSQL*: Adds permissions to the PostSQL of any selected table in a Sybase physical model.

See Also

- Why Use Automation Objects?
- Using the SAX Basic Macro Editor

Running ER Studio Data Architect Macros from Windows

You can run ER Studio Data Architect macros from the Windows command shell (similar to the old DOS command line console). You can do it with a batch file or from the Windows Desktop.

Access ER Studio Data Architect from Command Line

To execute as a batch file (*.bat file) command, use the *-m* option and specify the macro you want to run:

Start /wait /min "erstudio" erstudio.exe -m MyMacro.bas

Access ER Studio Data Architect from Windows

1. Choose Start > Run.
2. Type *cmd* and press *Enter*.
3. Enter *ERStudio.exe* -m MyMacros.bas.

Notes

- You can use the ER Studio Data Architect command-line to access macros written in ERStudio's Sax Basic Macro editor. Run the command from the directory where ERStudio.exe is located. The default location is:

...\Program Files\Embarcadero\ERStudio Data Architect X.X

You cannot run the ER Studio Data Architect command-line when running the ER Studio Data ArchitectAppWave ION. You must have ER Studio Data Architect installed locally on your machine to access this command.

- You must specify the macros directory in Options Editor > Application Tab. The default location is:
  Windows XP: C:\Documents and Settings\All Users\Application Data\Embarcadero\ERStudioDA_X.X\Macros
  Windows Vista*: C:\ProgramData\Embarcadero\ERStudioDA_X.X\Macros*
  AppWave ION*: My Documents\ERStudio Data Architect XE\Macros*
- Macros may not be located in a subdirectory. Macros with long names or spaces need to be enclosed in quotes.
- You must include the.*bas* extension in the command line syntax.
- Any errors that occur during the execution of the macro are written to an error file (*.err*) in the macros directory in *Options Editor - Directories* tab. The file name includes the date of the error. ER Studio Data Architect lists each error with a timestamp in the file.

- You can use dialogs in macros executed from the command line, but you must write the macro code to handle the opening of diagrams. Macros that work on the active diagram may not work.

See Also

- Why Use Automation Objects?
- Using the SAX Basic Macro Editor

Creating Event Handlers

ER Studio Data Architect supports programmatic event handling. In order to use the events, you must have the option enabled. To do this, choose Tools > Options > Automation Options and then select *Create Handlers*.

You can also write handlers for update events, see Handling Update Events.

You can customize templates to create database objects, including the following:

- Entity
- Attribute
- Relationship
- Index
- Model
- Submodel
- Domain
- Default
- User Datatype
- Rule
- View
- View Relationship
- Trigger
- Stored Procedure

When an entity is created, ER Studio Data Architect calls the *CreateEntityHandler( )* function in the *ERSBasicHandlers.bas* file.

When the function is called, ER Studio Data Architect passes to the handler the newly created entity (the current entity) and the current diagram. The bodies of all the functions in the file are currently empty. You can extend the creation behavior of any of the objects listed above by implementing the bodies of these functions.

- ER Studio Data Architect does not call the creation event functions for attributes and indexes until after you exit from the Entity Editor.

Example: `Sub CreateEntityHandler(CurEntity As Object, CurDiagram As Object)`

```
Dim Prefix as String

Dim EntityName as String

Dim NewEntityName as String

Prefix = "ERS"

EntityName = CurEntity.EntityName

NewEntityName = Prefix + EntityName

CurEntity.EntityName = NewEntityName
```

`End Sub` In this example, the CreateEntityHandler function is modified so that a prefix of ERS is attached to the default entity name each time you create a new entity.

Here is a more detailed explanation:

When you use the Entity Tool to create an entity, the default name for the entity is EntityX, for example, Entity1, Entity2. By modifying the body of the CreateEntityHandler function in the manner shown above, the default entity name becomes ERSEntityX (e.g. ERSEntity1, ERSEntity2).

Notes

- When ER Studio Data Architect starts, it reads in all the creation event handler functions from the file ERSBasicHandlers.bas. Modifications to these functions while the application is running will have no effect. You must restart ER Studio Data Architect for your modifications to take effect.
- You cannot declare a specific object type (for example, Entity) in the creation event handler functions. All objects must be declared as the generic Object type.

See Also

- Handling Update Events
- Why Use Automation Objects?
- Using the SAX Basic Macro Editor

Handling Update Events

You can customize the Entity update template subroutines.

To enable this option, choose Tools > Options > Automation Options and then select *Update Handlers*.

You can customize templates for the following objects:

- Name
- Table Name
- Definition
- Note
- PostSQL
- PreSQL
- Storage Location
- Initial Extent
- PCT Free
- No Logging
- PCT Increase
- Next Extent
- PCT Used

When you edit an entity with the Entity Editor, ER Studio Data Architect immediately calls your UpdateEntityHandler function in the ERSUpdateHandlers.bas file.

The function is called, ER Studio Data Architect passes the entity object as the first parameter, the diagram object as the second parameter, and the UpdateType as the third parameter. This distinguishes which property has been updated. The body of the function in the file contains empty case statements for the different entity properties. These empty statements do nothing. You can customize/extend the update behavior of any of the object's properties listed above by implementing the bodies of the property case statements in the function.

- ER Studio Data Architect does not call the update event functions for entities until after you exit from the Entity Editor.

Example

```
Sub UpdateEntityHandler(CurEntity As Object, CurDiagram As Object, UpdateType
As Integer)

Dim Prefix as String

Dim EntityName as String

Dim NewEntityName as String

Prefix = "ERS"

Select Case UpdateType

Case UPDENTITYNAME

EntityName = CurEntity.EntityName

NewEntityName = Prefix + EntityName
```

```
CurEntity.EntityName = NewEntityName

Case UPDENTITYTABLENAME

Case UPDENTITYDEFINITION

Case UPDENTITYNOTE

Case UPDENTITYPOSTSQL

Case UPDENTITYPRESQL

Case UPDENTITYSTORAGELOCATION

Case UPDENTITYINITIALEXTENT

Case UPDENTITYPCTFREE

Case UPDENTITYNOLOGGING

Case UPDENTITYPCTINCREASE

Case UPDENTITYNEXTEXTENT

Case UPDENTITYPCTUSED

End Select
```

`End Sub` In this example, the UpdateEntityHandler function is modified so that a prefix of ERS is attached to the entity name each time you change the entity name.

Here is a more detailed explanation. When you edit the entity name and change it to Entity1, you also automatically add a prefix to the name. By modifying the case statement in the body of the UpdateEntityHandler function in the manner shown above, the entity name becomes ERSEntity1.

Notes

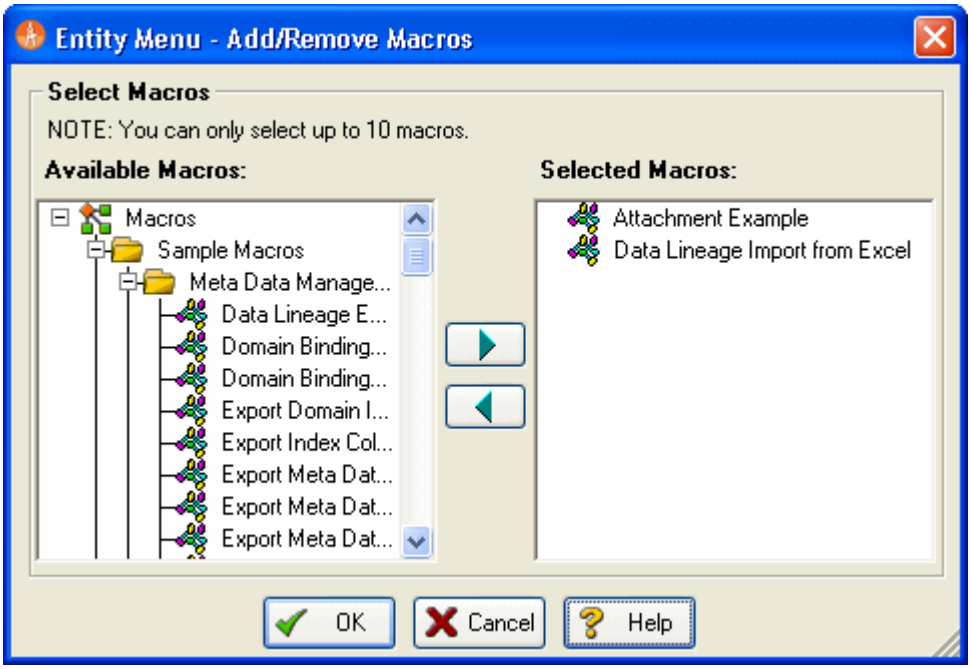
- When ER Studio Data Architect starts, it reads in all the update event handler functions from the file ERSUpdateHandlers.bas. Modifications to these functions while the application is running have no effect. You must restart ER Studio Data Architect for the modifications to take effect.
- A user cannot declare a specific object type (for example, Entity) in the update event handler functions. All objects must be declared as the generic Object type.

See Also

- Why Use Automation Objects?
- Using the SAX Basic Macro Editor

Adding Macros to GUI Modeling Menus

The Add/Remove Macros dialog box lets you add shortcuts to macros that you want to quickly access from various places in ER Studio Data Architect. This dialog box is supported from the Main Menu, as well as the diagram, Entity, View, and Relationship shortcut menus. You can add up to ten macros for each object.



Notes

- The point at which you open the dialog box, is where ER Studio Data Architect adds the shortcut. For example, if you right-click an entity and then click Add/Remove Macro Shortcuts, ER Studio Data Architect opens the Add/remove Macros Dialog box and indicates Entity in the title bar.
- You can only add ten macros for each object.

The table below describes the options and functionality available on the Add/Remove Macros dialog box:

Option	Description
Available Macros	Lists all Macros available to set in the shortcut menu.
Selected Macros	Displays macros currently on the shortcut menu.

Add or Remove Macros

1. Choose Macro Shortcuts > Add/Remove Macro Shortcuts.
2. In the *Available Macros* box, click the target macros and then click the right-arrow.

**Tip:** You can double-click any macro to add or remove it.

3. To remove a Macro, in the *Selected Macros* box, click the target macros and then click the left-arrow.

   **Note:** You can only select up to ten macros. If you try to select more than ten, ER Studio Data Architect returns an error. You must remove some macros before you can add new ones.

4. When you are finished adding or removing macros, click *OK*.

See Also

- Using the SAX Basic Macro Editor
- Sample Macros Installed with ER Studio Data Architect

Using the SAX Basic Macro Editor

You can write and execute automation scripts using the Macro Editor. The ER Studio Data Architect Macro language consists of the SAX BASIC language extended with ER Studio Data Architect object types. SAX BASIC is very similar to Visual Basic but there are some differences, so be sure to note the differences described in the *Automation Interface Reference*.

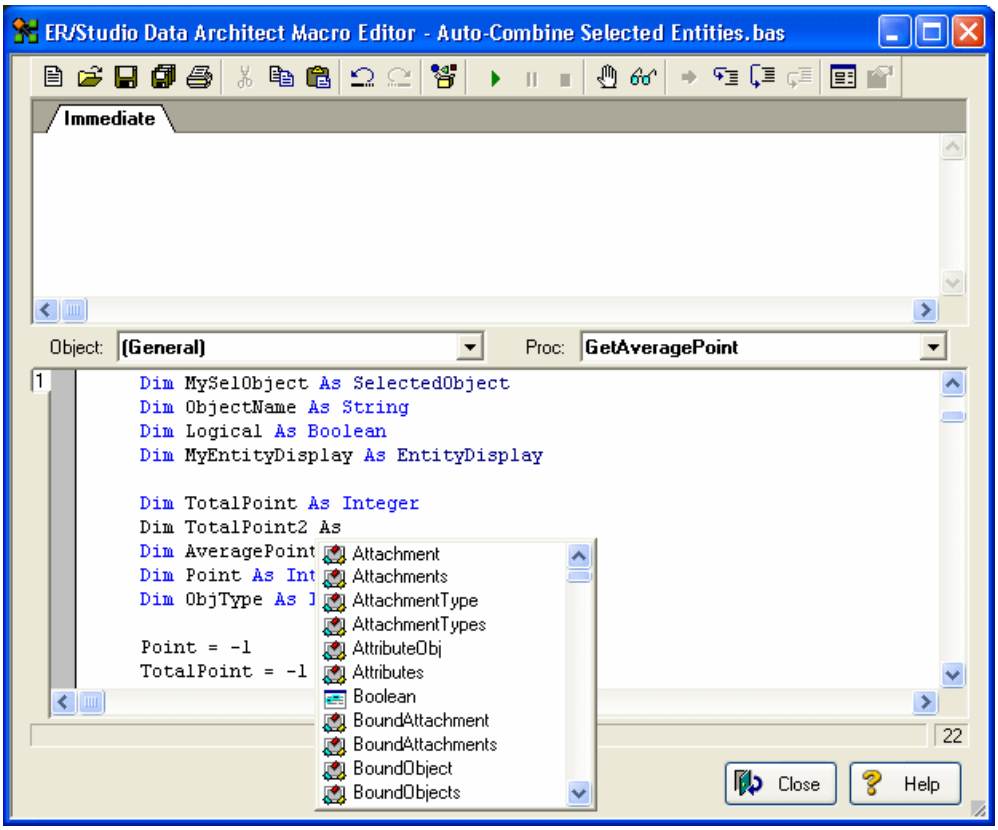
To launch the Basic Macro Editor, choose Tools > Basic Macro Editor.

Note: You can have a maximum of nine macro editor instances open at one time.

Code Auto-Completion

You can save time typing code using the context-sensitive editing assistant. Use the short cut menus and pop-up menus to help you write macros. When you start the Macro Editor, a reference to the ER Studio Data Architect library is automatically set and then you can use the menus to complete regular expressions and so on. The Editor offers the menus and pop-ups while you write your code. Where a choice must be made, the Editor either gives you a list of choices or the necessary syntax.

The scrolling menus let you select from the available objects or properties and methods. As an example:



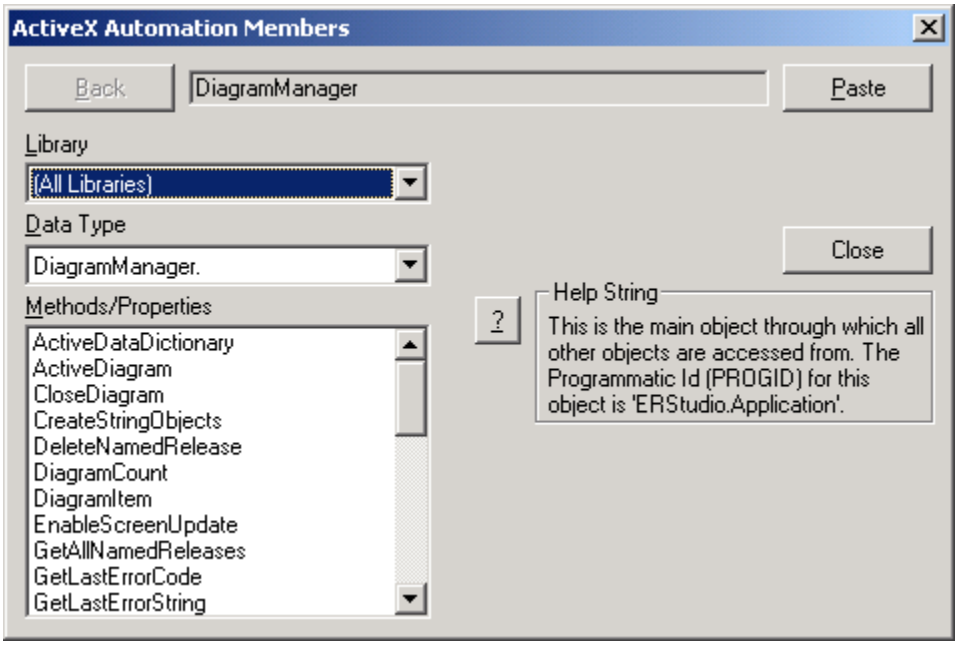
ActiveX Automation Members Browser (OLE Viewer)

The ActiveX Automation Members Browser displays the ER Studio Data Architect Type Library automation objects. You can copy and paste function templates into your scripts.

To browse through the objects, select an object from the Data Type list. This menu lists all automation objects used specifically in ER Studio Data Architect, along with several Visual Basic objects denoted by Vb<DataType>. The Methods/Properties list box is populated with methods and properties specific to the selected object. When you select one of the methods or properties, other information appears in the dialog box.

The Browser lets you paste a function into your macro. You can also follow the result before pasting the function. Following the result adds other parameters to the function. The help string describes the selected property or method. This is the same information used by other typelib viewers, such as Microsoft's Visual Studio OLE Viewer or typelib utilities in any code editor.

Click the ? button to open the Automation Interface Reference for the object to give more detailed information about its property or method.



For more information about the ActiveX Automation Members Browser, see the Sax Basic Help.

Add, Edit, and Run BASIC Macros

ER Studio Data Architect contains a SAX BASIC editor, interpreter and debugger ("editor" hereafter).

To use the SAX BASIC editor

On the **Data Model Explorer**, click the **Macros** tab.

- For more information on the SAX BASIC language, click the editor's *Help* button.

Add a Macro

1. On the **Data Model Explorer**, select the **Macros** tab.
2. Right-click a macro or macro folder, and select **Add Macro** to open the Macro editor.
3. You may need to refresh the file list to see it displayed after you have saved it.

Edit a Macro

To edit a macro, double-click it.

Rename a Macro

1. Right-click the macro and select **Rename Macro**.
2. Edit the macro name, or type in a new name.
3. Press *Enter* to save the new name.

**Note:** You may need to refresh the file list to see it displayed.

Delete a Macro

If you no longer need a macro, you can delete it. To delete a macro, do the following:

1. Right-click the macro you want to delete and then select **Delete Macro**.
2. Click **Yes**.

Run a Macro

Once you have created and saved a macro, you can run the macro.

Right-click the macro and then select **Run Macro**.

**Tip:** To run a macro from within the ER Studio Data Architect Basic Macro Editor, click **Start** or click the green arrow button.

See Also

- Adding Macros to GUI Modeling Menus
- Sample Macros Installed with ER Studio Data Architect

Tutorials

The tutorials are intended to help you get started using Embarcadero's data modeling and database design solution, ER Studio Data Architect.

After completing these tutorials, you'll have the foundation you need to explore the many features and benefits of ER Studio Data Architect. You will have learned how to do the following:

- Create a new data model.
- Work with logical and physical diagrams.
- Leverage productivity-focused features such as powerful reporting engines.
- Use common tasks and commands to make you more productive.

The tutorials are divided into three sessions. Do them all at once or complete them individually as your time permits.

You can use this basic tutorial as a road map of product highlights, but also to help you find your own path in exploring ER Studio Data Architect.

Once you have started, you can click Help from the Main menu to find many additional resources that complement and build on many of the activities shown in this brief guide. Or visit the Contents section of http://docwiki.embarcadero.com

See Also

- What's New
- Upgrading ER Studio Data Architect
- Licensing ER Studio Data Architect
- Release Notes

Introduction

ER Studio Data Architect is a visual modeling application used for platform-independent logical data architecture analysis and design, in addition to platform-specific physical database design and construction. The powerful, multi-level design environment addresses the everyday needs of database administrators, developers, and data architects who build and maintain large, complex database applications and strive to consolidate, report, and re-use metadata across the enterprise.

ER Studio Data Architect's progressive interface and simplicity has been designed to effectively address the ease-of-use issues which have plagued data modeling and CASE tools for the past decade and more. The application equips you to create, understand, and manage the life-cycle of mission-critical database designs and business metadata within the enterprise.

ER Studio Data Architect is rich and customizable, offering:

Strong logical design capabilities

- The ability to spawn many physical designs from a corporate logical design.
- Bi-directional model comparison and information synchronization.
- Visual-Basic for Applications API for product customization.
- Powerful DDL reverse engineering and generation.
- Metadata import and export capabilities.
- Data lineage documentation.
- Sophisticated XML, HTML, and RTF-based documentation and reporting facilities.

Notice for Developer Edition Users

The Developer Edition of ER Studio Data Architect provides the user with all the standard functionality of ER/Studio except for the following limitations:

- No access to the ER/Studio Repository and ER/Studio Team Server
- Limited import and export functionality; the Import and Export Metadata Model Bridges are not supported.
- Limited database support.

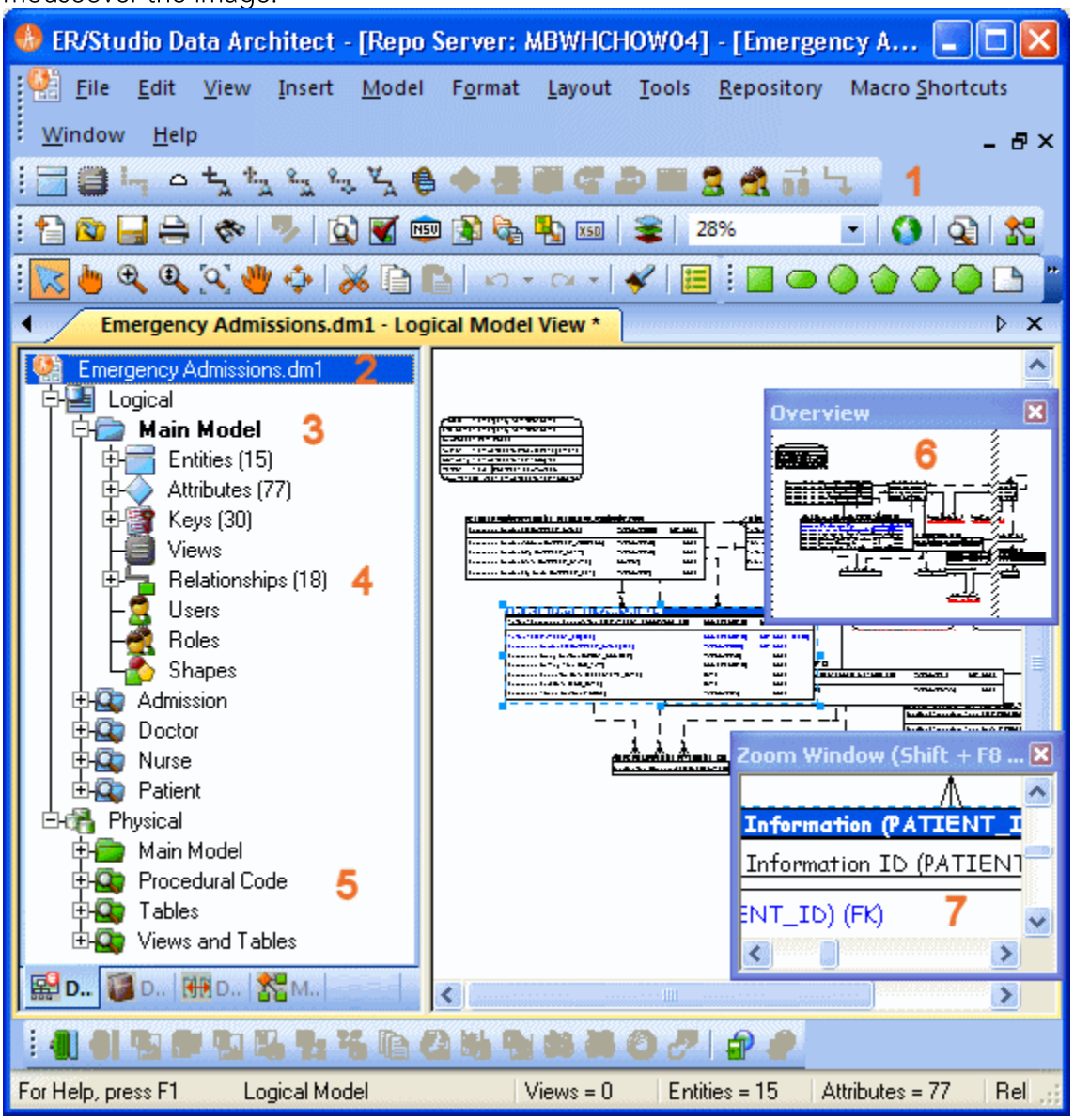
Application Interface

The application interface is divided into several tabbed windows that let you navigate and customize your workspace. The application includes context-sensitive toolbars, menus, intuitive icons, keyboard commands, and other resources to help you work efficiently. ER Studio Data Architect user interface has a standard Windows look and feel.

For a detailed description of the internal data structure of the ER Studio Data Architect application and the Repository, you can look at Metadata models (DM1 files), usually found in the following directories:

- *Windows XP*: C:\Documents and Settings\All Users\Application Data\Embarcadero\ERStudioDA_X.X\Sample Models
- *Windows 8, Windows 7, and Windows Vista*: C:\ProgramData\Embarcadero\ERStudioDA_X.X\Sample Models

For information on enhanced UI features, please see the corresponding numbers below or mouseover the image:



Interface Features

ER Studio Data Architect has an enhanced user interface that is similar to Microsoft Windows with intuitive icons.

1. You can dock toolbars anywhere in the ER Studio Data Architect user interface.
2. The name of the active model, with or without the full path to the file, can be displayed on the title bar of the main application window. This is an option you can set in Tools > Options > Application > Application Defaults.
3. ER Studio Data Architect displays the active model in bold type, so that you can easily see which model or submodel you are working on.
4. The *Data Model* tab of the Data Model Explorer displays indexes, relationships, and attributes as separate nodes.
5. The *Data Model* tab of the Data Model Explorer displays schema objects like packages, procedures, functions, materialized views, auxiliary tables, and triggers as separate nodes.
6. The *Overview Window* lets you navigate large Data Models.
7. The *Zoom Window* helps you focus on the details of a specific area of a large, reduced diagram.

The Data Model Explorer is comprised of five tabs that offer easy access to important functionality and also includes an Explorer tree that lets you efficiently navigate the objects of your data models. The ER Studio Data Architect interface to the Data Model is divided into two main areas. On the left is the Data Model Explorer and on the right is the Data Model Window. This convention is carried out on other tabs of the user interface, such as the Repository and Data Lineage tabs, where you'll find an explorer on the left and a graphical representation on the right. The Data Model Window also provides a complete workspace for creating physical and logical data models and smaller, movable windows to help you navigate large data models.

See Also

- What's New
- Upgrading ER Studio Data Architect
- Licensing ER Studio Data Architect
- Release Notes

Installing Embarcadero ER Studio Data Architect

This section will help you ensure your environment meets the technical requirements and helps you install ER Studio Data Architect or ER Studio Data Architect XE.

You can download a 14-day trial version of ER Studio Data Architect, from *http://www.embarcadero.com/downloads*. Embarcadero will send you a license registration code by email.

System Requirements

Review the requirements before you install ER Studio Data Architect. Adhering to these requirements optimizes ER Studio Data Architect performance.

ER Studio Data Architect Requirements		
Hardware	Processor	Pentium III, 600 MHz or higher
	RAM	1 GB
	Disk Space	340 MB
		175 MB in the System TEMP directory for temporary file storage during installation.
	DVD Reader	
Operating System	Any of the following: • Windows 8 • Windows 7 (32-bit and 64-bit modes) • Windows Server 2008 • Windows Vista • Windows XP Professional • Windows 2003 (32-bit mode) • Windows 2000 Pro Server • Advanced Server with Service Pack 2	
Software	To use an Open Data Base Connectivity (ODBC) database as a data source, install the ODBC driver on your client computer. To connect to the DBMS using native connectivity, install the native client libraries available from the DBMS vendor.	
Privileges	Local administrator privileges (required for installation only). DBMS privileges to create objects on the server: EXECUTE	

	permission as the user needs to execute stored procedures for example Create_Temp_Tables.	

Installing ER Studio Data Architect

Note: To ensure compatibility, the release dates of ER/Studio Repository and ER Studio Data Architect must be the same.

Log on to Windows with local administrator privileges.

1. Start the ER Studio Data Architect installation program.

   Note: To do a silent install, you can run the installation program from the command prompt with the -s parameter.

2. Walk through the installation wizard following the onscreen prompts.

The following provides additional information for some options that are not self-explanatory:

- *Default Notation*: Choose the notation type you want ER Studio Data Architect to use by default to display the model.
- *IDEF1X*: Data modeling technique used by many branches of the United States Federal Government.
- *IE (Martin/Finkelstein)*: Information Engineering (IE) as developed by Martin and later revised by Finkelstein.
- *IE (Crow's Feet)*: Uses IE notation and represents relationships with connecting lines between entities, and symbols at the ends of those lines to represent the cardinality of the relationship. This is the notation used in Oracle texts and in other applications such as Visio. Foreign keys are displayed in entity boxes.
- *IE (Hide Foreign Keys)*: Uses standard IE notation, but foreign keys are not displayed in entity boxes.

Note: You can change the notation setting at a later time. To change the notation, choose Tools > Options from the ER Studio Data Architect Main menu, and then click the *Logical* or *Physical* tab where you can choose the notation type.

- *Shared Application Data Location*: By default, ER Studio Data Architect installs shared directories and application files such as the DatatypeMapping, Macros, and Sample Models directories in the following locations:
- *Windows XP:*C:\Documents and Settings\All Users\Application Data\Embarcadero\ERStudio_X.X
- *Windows 8, Windows 7, and Windows Vista:*C:\ProgramData\Embarcadero\ERStudio_X.X

If you prefer to store these files in another directory after installation, follow the procedure in Changing the Location of Shared Files.

- For step-by-step instructions on how to use ER Studio Data Architect, see the Tutorials section.

Installing ER Studio Data Architect via Command Line

ER Studio Data Architect can also be installed from the command line as follows: *erda_<ver>.exe* Optional parameters:

- /s (specifies to install silently without prompts, using default installation settings.
- TARGETDIR=<install dir> (specify an alternate installation directory)
- DESKTOPICON=TRUE (enable the desktop icon option)

For information on installing ER/Studio with Embarcadero AppWave, please see the *AppWave Admin Quick Start* and the *AppWave Administrator Guide* at http://docwiki.embarcadero.com

See Also

- What's New
- Upgrading ER Studio Data Architect
- Uninstalling ER Studio Data Architect
- Licensing ER Studio Data Architect
- Release Notes

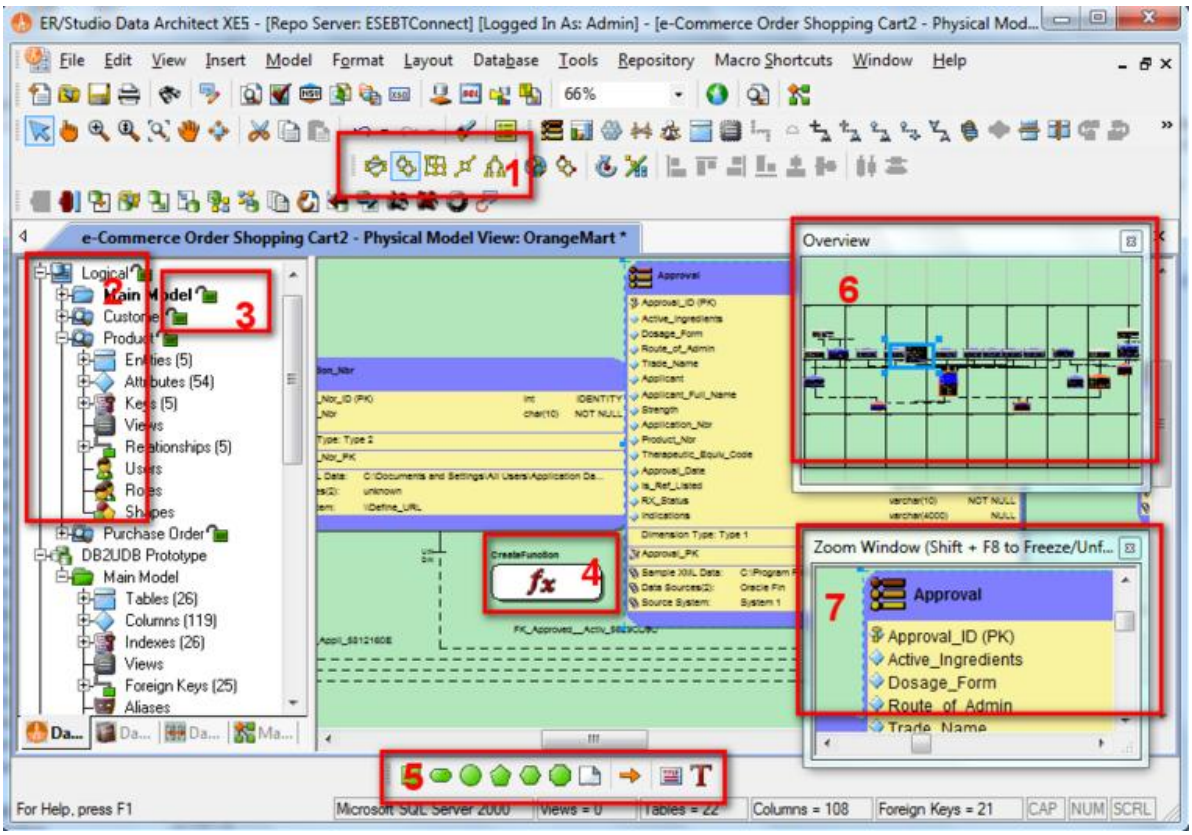
Tutorials: Getting Started

Interface Features

ER Studio Data Architect has an enhanced user interface that is similar to Microsoft Windows with intuitive icons.

The graphic below names and describes the functionality of some key elements of the ER Studio Data Architect user interface.

For information on enhanced UI features, please see the corresponding numbers below or mouseover the image:



1. Sophisticated diagram auto-layout tools provide single-click clean up of diagram objects.
2. Data Model Explorer displays information about logical and physical models, submodels and nested submodels.
3. Repository object status icons display real-time user access information.
4. Complex schema objects like functions can be displayed to illustrate dependencies.
5. Toolbars are dockable anywhere in the ER Studio DA application window.

6. The Overview Window lets you navigate large Data Models.
7. The Zoom Window helps you focus on the details of a specific area of a large, reduced diagram.

Starting to Data Model with ER Studio Data Architect

On the Windows Start > Programs menu, choose Embarcadero > ERStudio Data Architect.

1. Choose File > New > Draw a new data model.

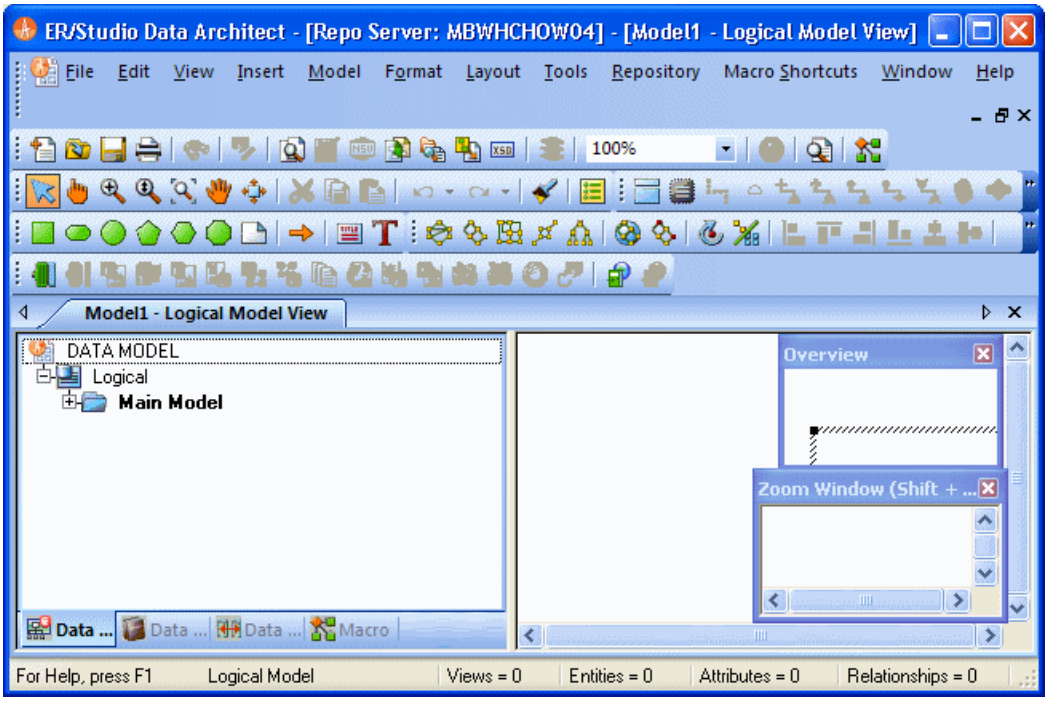
As you can see in the Create a New Model dialog, there are a number of ways to begin modeling with ER Studio Data Architect:

- Build a new design from the ground up by drawing a new data model.
- Build a data model from an existing database through live reverse engineering.
- Import designs from other modeling products such as Computer Associate's ERwin or SQL files.

Tip: You can select an initial layout style for your model before the SQL import takes place.

- Ensure *Relational* is selected for the new model type and then click *OK*.

After selecting Draw a new data model and clicking OK, ER Studio Data Architect will resemble the image below:



See Also

- Application Interface
- Upgrading ER Studio Data Architect
- Uninstalling ER Studio Data Architect
- Licensing ER Studio Data Architect

Tutorials: Logical and Physical Modeling

ER Studio Data Architect supports both logical (non-DBMS or technology-specific) modeling and physical (DBMS-specific) modeling. ER Studio Data Architect is designed to allow organizations the flexibility to analyze and design a business problem or application logically and generate as many different physical interpretations from the logical model as required. Multiple physical models can be generated from the logical model for the same DBMS (for example, Oracle) or other DBMSs (such as Oracle, SQL Server and DB2). Generating logical and physical models will be discussed in detail in the following sessions.

- Using Data Dictionary Domains to Populate New Entity
- Establishing Relationships Between Entities
- Creating and Working with Submodels
- Generating Physical Models from a Logical Model
- Denormalizing the Physical Model
- Finding out How an Entity Maps to the Physical Model

See Also

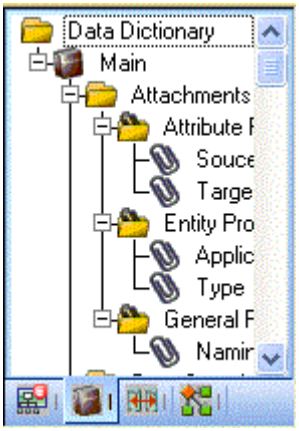
- Logical Design Concepts
- Data Modeling Concepts

Tutorials: Using Data Dictionary Domains to Populate New Entity

As instructed in Getting Started, you have chosen to draw a new data model to begin a logical model from the ground up. Before we begin to add entities, we will populate ER Studio Data Architect with some sample domains (Domains are re-usable attributes).

1. Choose File > Import Data Dictionary.
2. Next to the **File Location** box, click the ellipsis and browse to the **Sample Models** folder, which is located at:
   - **For Windows XP**:
     C:\Documents and Settings\All Users\Application Data\Embarcadero\ERStudioDA_X.X\Sample Models
   - **For Windows 8, Windows 7, and Windows Vista**:
     C:\ProgramData\Embarcadero\ERStudioDA_X.X\Sample Models
3. Double-click the *Orders.dm1* sample model and then click **OK**.
   This model contains a pre-populated, sample data dictionary.

   Note: Under *Resolve Imported Objects with Duplicate Names*, you can choose between a couple of options to determine how the dictionary objects are imported. This is important when importing into a diagram that already has dictionary objects in it.
   Once opened, you will see that the ER Studio Data Architect Model Explorer has automatically switched to the **Data Dictionary** tab to allow immediate drag and drop access to domains.



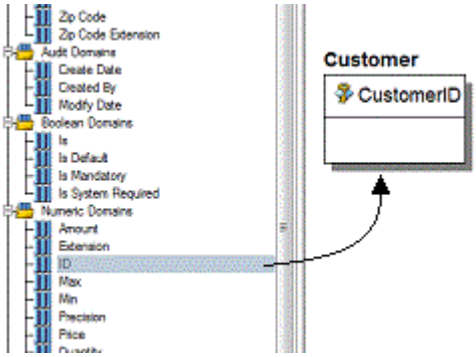
4. Now, to add an entity to the Data Model Window, click the *Entity* tool on the *Modeling Toolbar*,



   and then click in the Data Model Window to drop the entity.

Tip: The cursor will change to an entity symbol once the *Entity* tool is clicked, so you can click to drop as many entities on the Data Model Window as you want.

5. Right-click to return your mouse to the selection cursor.
6. Right-click the entity and open the *Entity Editor*. In the entity name field, type *Customer*, replacing the default entity name, Entity1 and then click **OK**..
7. In the **Domains** folder of the **Data Dictionary** tab, locate the **ID** domain in the **Numeric Domains** folder.



8. Click the **ID** domain (do not release your mouse), drag it onto the **Customer** entity, and then release it just below the entity's name, which is the entity's Primary Key field.

Tip: You can edit or rename an entity and insert, edit or rename attributes by holding down the **Shift** key and then clicking the name or attribute. Pressing the **Tab** key cycles between the entity's name, and primary key and non-primary key fields. After naming the field, press **Return** to insert a new field.

Tip: If you need to zoom in on the entity to read the entity name and attributes, press **F8** to view the **Zoom Window**, then use the scroll bars of the **Zoom Window** to center the entity on the Data Model Window. Press **Shift** while rolling the mouse wheel forward to increase the view magnification. You can use the **Pan** tool to reposition the view to better see the entity.

9. In the entity, click **ID**, the name of the attribute we just created from the ID domain, and change its Attribute Name and Default Column Name properties to **CustomerID**, as seen in the illustration above. Then click **OK**.
10. Repeat the process in step 7 and step 8 to populate the Customer entity with the following domains:
    - **Name** and **Phone** from the **Property Domains** folder
    - **Address**, **City**, **State**, **Zip Code** from the **Address Domains** folder.
11. Drop another entity on the **Data Model Window** and call it **Order**.
12. Drag the ID domain onto the Order entity's Primary Key field, change the Attribute and Column name to**OrderID**.
13. Right-click the Order entity and select **Comments**.

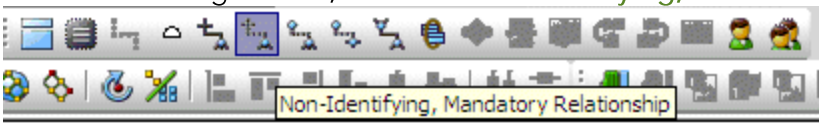
14. Enter some text in the **Comments** editor, click **Add** and then click **OK**. You can add comments to any database object in the same manner. This can be very useful when you want to provide feedback on a model to other users.
15. Save your data model. We will use it in the next session of this tutorial.

See Also

- Ensuring Consistent Domain Definitions Using User Datatypes
- Reusing Attribute Definitions Using Domains

Tutorials: Establishing Relationships Between Entities

1. On the Modeling toolbar, click the Non-Identifying, Mandatory Relationship tool.



Note: The screen shots of this tutorial were taken using a model where the notation was set to IE (Crow's Feet). Depending upon the notation your model is currently set to, the icons for each relationship will be slightly different.

Tip: You can change the model notation by choosing Model > Model Options, and then selecting another notation option in the Notation area.

2. To establish a relationship between Customer and Order, click the parent entity, *Customer* and then click the child entity, *Order*.

Note: ER Studio Data Architect supports sound design practices by automatically propagating the primary key, from parent to child entities. If there are candidate alternate keys that should also be propagated to the child, in the *Relationship Editor* you can choose all available parent entity keys in the Parent Key list. Deleting a relationship will remove a non-native propagated attribute. However, if you want to keep the child columns of the relationship or foreign constraint, when you delete the relationship you can check the Make Foreign Keys Native option. In this case, if the relationship between Customer and Order is deleted, the CustomerID will be left in the Order entity.

What is a Domain?

A Domain is a valuable tool in establishing standard, re-usable Attributes/Columns. They allow data modelers to create a data element once (such as an ID field you require all of your entities to leverage as its primary key) which has the same data type, definition, rule, and constraint no matter where the data element is distributed and bound. Read more about Domains in ER Studio Data Architect User Guide.

See Also

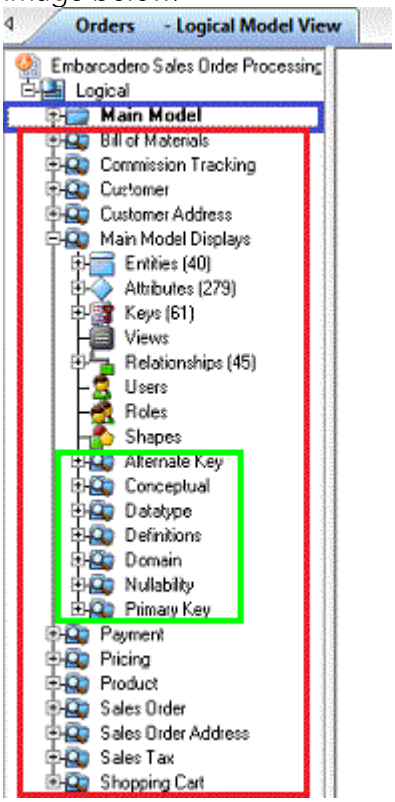
- Reusing Attribute Definitions Using Domains
- Ensuring Consistent Domain Definitions Using User Datatypes

Tutorials: Creating and Working with Submodels

Now that you have a general understanding of how to build logical models from the ground up in ER Studio Data Architect, it is important to understand how to work with an important navigation feature of ER Studio Data Architect, called Submodels. Submodels and nested submodels are designed to break down large, complicated views of a data model in order to focus on a specific area. An important aspect of Submodels to understand is that any changes made in the submodel, other than layout, color, display settings, notation or similar items which can be unique to the submodel, will occur automatically in the Main Model view. In other words, change or add an attribute to an object in a Submodel and the change is automatically propagated to its Main Model counterpart.

Close the current sample model, and open a more mature model. Use this exercise to learn more about submodeling.

1. Choose File > Open.
2. Select *Orders.dm1* and then click *Open*.
3. To preserve this sample model for future use, choose File > Save As and then save the *Orders.dm1* file with a new name.
   In this exercise, we will be modifying this model.
4. Collapse the folders in the *Data Model* tab of the Data Model Explorer to look like the image below:

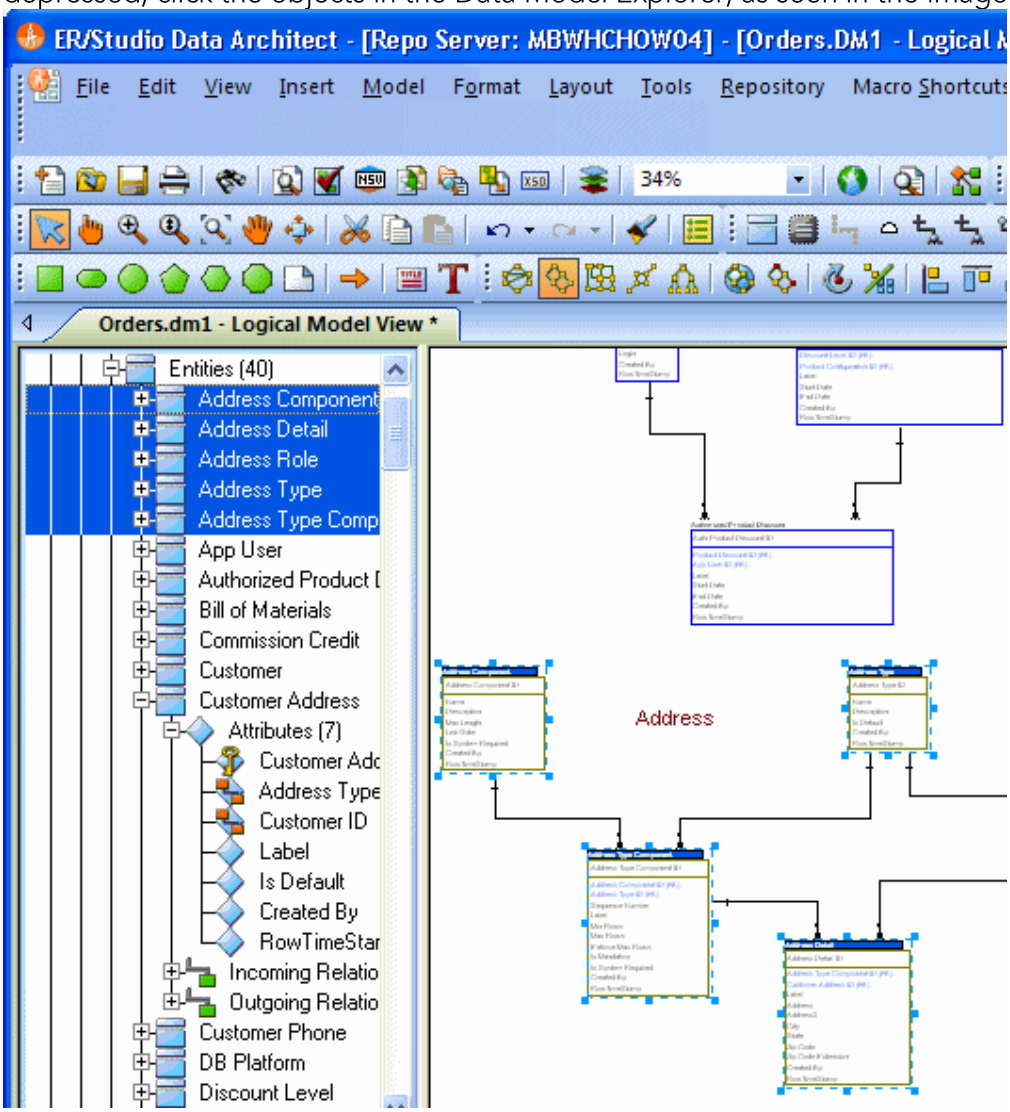


In the *Orders.DM1* sample model, there are no physical models. This model includes several submodel folders that help to describe the logical model:

- Main Model – This is the entire collection of all logical objects in the Orders.DM1 file. Note the absence of the magnifying glass on the folder icon which designates it as the main model.
- Bill of Materials throughShopping Cart – These are submodels, which are smaller collections of entities derived from the Main Model that help to describe specific areas of the Main Model free from other entities.
- Alternate Key through Primary Key – These are nested submodels, which can go 'n' levels deep and are literally submodels of submodels.

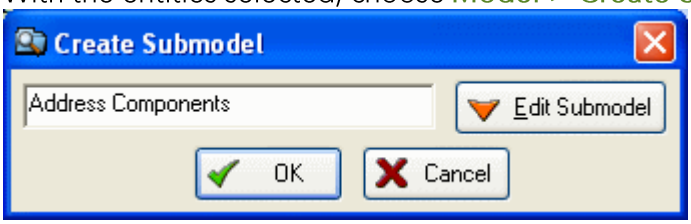
Feel free to explore. Click the plus sign (+) to expand these folders.
Let's create a submodel with all the objects related to the *Orders.DM1* Address components.

5. To make a new submodel, navigate to Logical Main model and with the CTRL key depressed, click the objects in the Data Model Explorer, as seen in the image below.



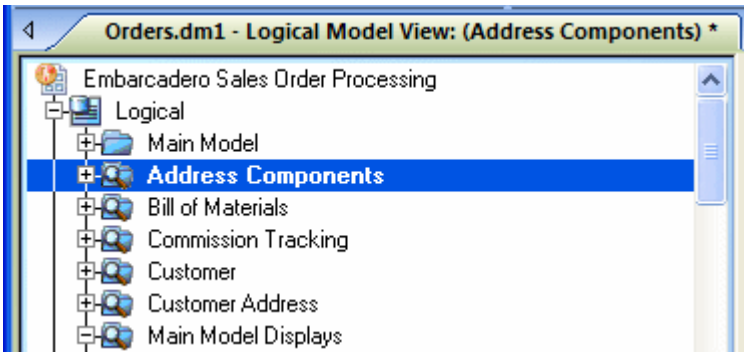
Note: Any objects selected in the Data Model Explorer will also be selected on the Data Model Window as well (also seen here). You can also select entities by lassoing them on the Data Model Window.

6. With the entities selected, choose Model > Create Submodel.



7. Below **Create Submodel**, enter **Address Components** as the name for the submodel.
8. Click **OK**.

ER Studio Data Architect creates the **Address Components** submodel.



What do the results look like and how do I navigate to the submodel?

Once created, you will see the new submodel listed in the Data Model Explorer, denoted as a submodel by the magnifying glass over its folder, as in the case with Bill of Materials and the other submodels.

See Also

- Retrieving a Repository Diagram, Model, Submodel or Named Release
- Creating and Editing Submodels

Tutorials: Generating Physical Models from a Logical Model

ER Studio Data Architect can generate as many physical models from a single logical model as desired. There are many ways to leverage multiple physical models in ER Studio Data Architect to help the design process. Examples of how multiple physical models are used are:

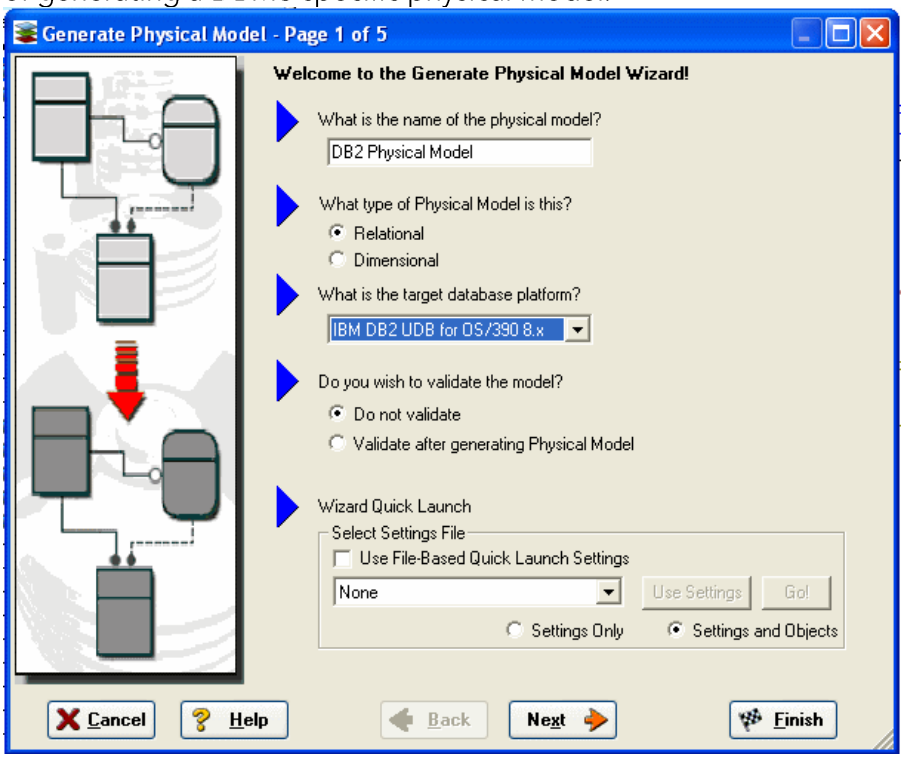
- Managing change in an existing application: Maintain independent development, test, and production physical model diagrams that represent specific databases.
- Migrating database applications: Use ER Studio Data Architect as an analysis and design hub for migrating database applications. Manage a physical model of the legacy source database application in addition to its new target physical model, which can be for an entirely new DBMS than originally maintained in the legacy database.

Let's generate a new physical model from a logical model in order to build a database. We will use the *Orders.DM1* sample model.

1. Open your version of the Orders.DM1 sample model.

   Tip: Use the steps shown in the last session to do so.

2. Select the **Main Model** and then click Model > Generate Physical Model.
   ER Studio Data Architect invokes a step-by-step wizard to walk you through the process of generating a DBMS-specific physical model.



3. Name the new physical model, **DB2 Physical Model** and then select **DB2 UDB for OS/390 8.x** as the target DBMS to generate.

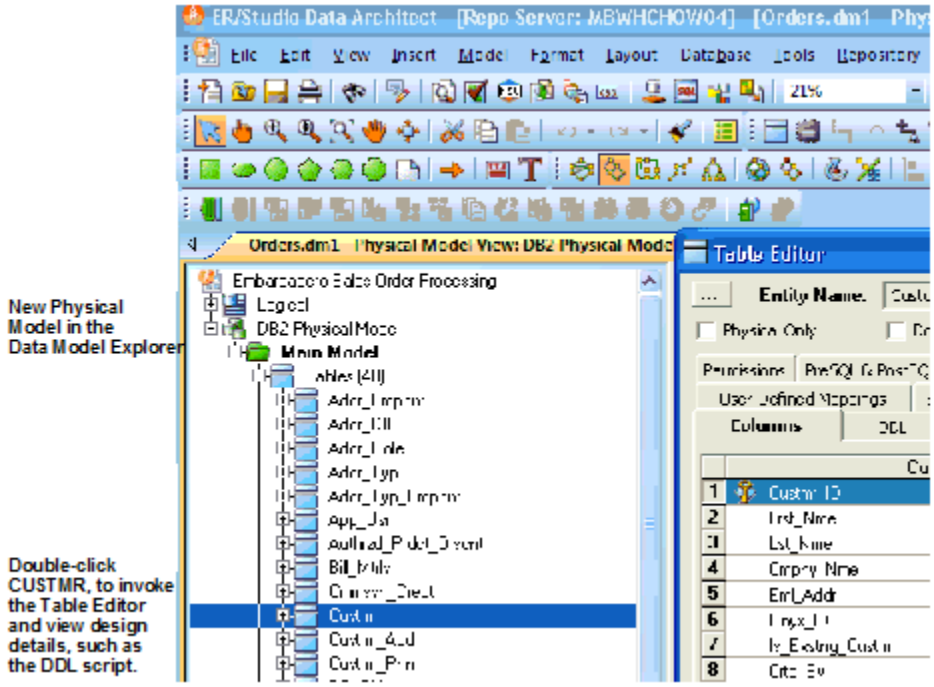
4. Continue through the **Generate Physical Model** wizard, which prompts several questions about how you want your physical model to be generated.

   Note: The wizard prompts you to customize items such as individual object selection, index assignment, default storage parameters, resolution of many-to-many relationships that may be in the logical model, and naming conventions. In Step 3 of the wizard, you can add prefixes or suffixes to the object name. For example, it can be useful to add an indicator of the object type in the name. A DBMS-specific validation check is also provided in this wizard.

   Tip: The *Quick Launch* can store common settings so that an operation can be reused on this model or on any other models. You can reuse the settings on another model by choosing the *Use File-Based Quick Launch Setting* option when saving the Quick Launch information on the last page of the wizard.

5. To generate the new Physical Model, on the last page of the wizard, click *Finish*.

Now that a physical model has been generated from the logical model, feel free to navigate to specific objects via the Data Model Explorer, such as the CUSTMR table selected here. Double-click and view the physical details of the object such as DDL, Indexes, Partitions, and Storage.



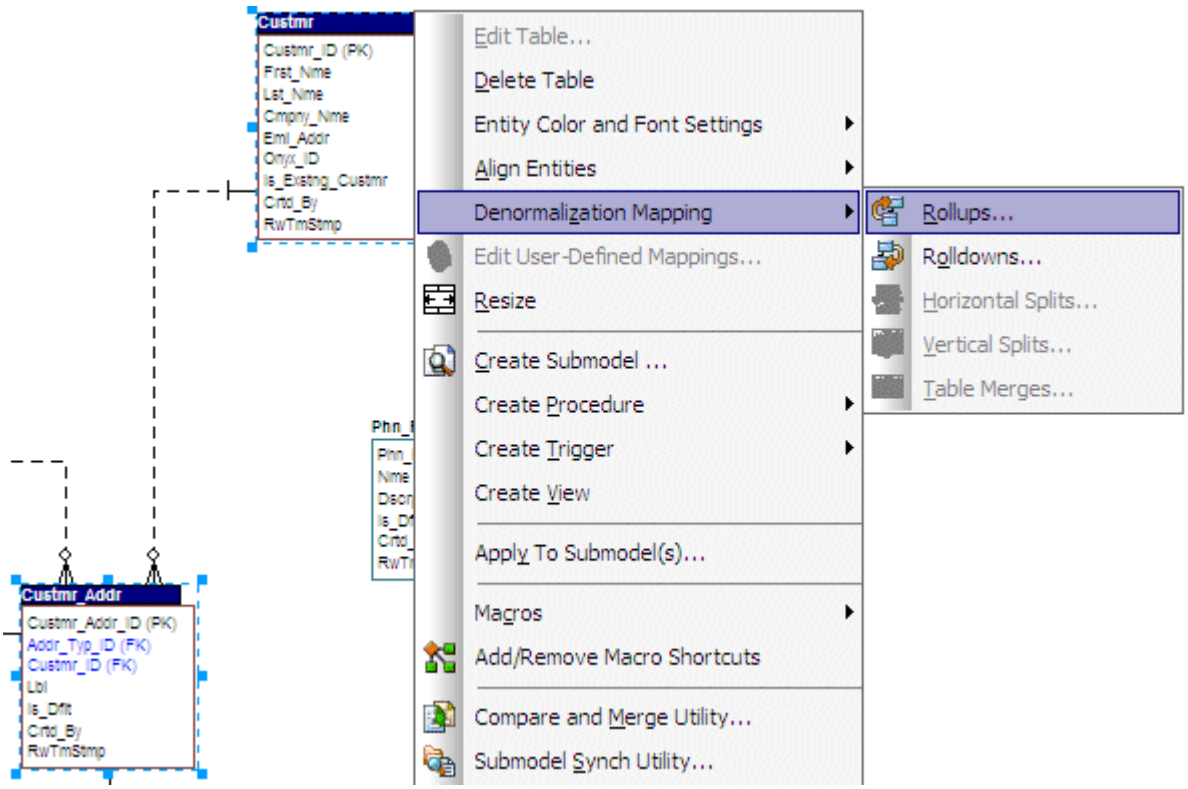
See Also

- Developing the Physical Model
- Transforming Model Objects from Logical to Physical

Tutorials: Denormalizing the Physical Model

ER Studio Data Architect comes equipped with denormalization wizards to help you optimize the physical design once the physical model is generated. The wizards help automate the process and keep the ties between the physical tables and the logical entities.

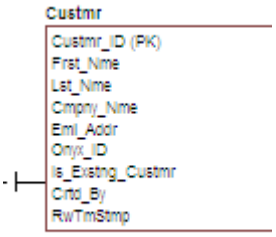
The active, denormalization wizards available depend on which tables are selected in the physical model when you select Denormalization Mapping. For example, if two tables that are related to each other are selected, the valid operations would be Rollups or Rolldowns.



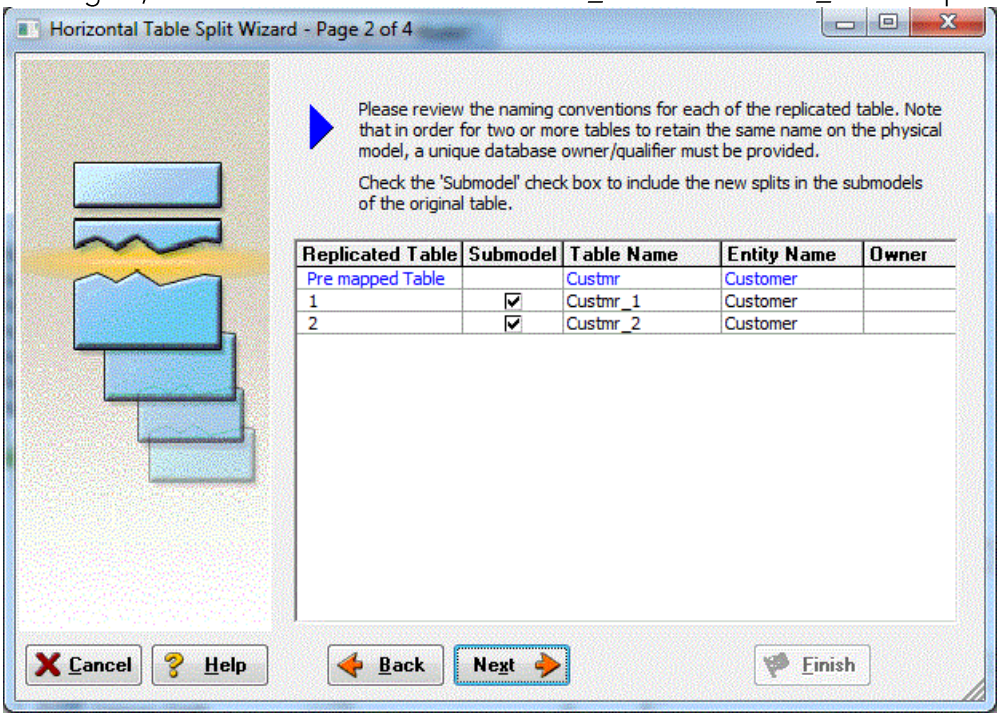
When only one table is selected, the option to split the tables becomes available. The Table Merge option is available when two unrelated tables are selected.

Let's walk through an example of a denormalization operation using the generated physical model in a previous session of this tutorial. We may want to reduce the overhead on the Custmr table by splitting it into two physical tables, Custmr_East and Custmr_West. Splitting the table can reduce the query time and provide opportunities to store the tables at different locations which could further reduce lookup time.

Before the operation, the Custmr table should look like:

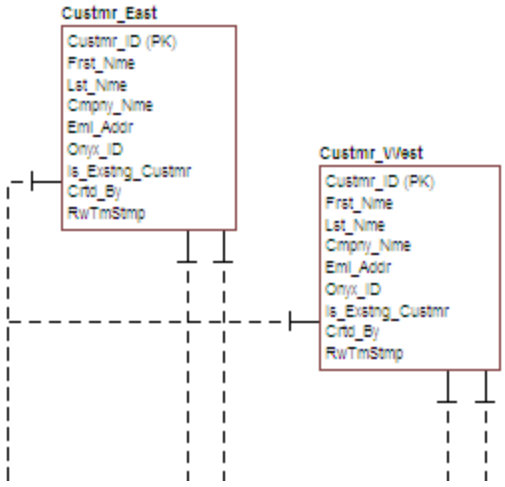


1. Open the Orders1.dm1 model you modified and saved in the last session.
2. In the Data Model Explorer, right-click the **Custmr** table in the Physical Model.
3. Choose Denormalization Mapping > Horizontal Splits.
   Notice that since only Custmr is selected, the only possible mappings are vertical and horizontal splits.
   The **Horizontal Table Split Wizard** launches.
4. On Page 1, type *2* for the number of splits.
5. On Page 2, rename Custmr1 and 2 to *Custmr_East* and *Custmr_West* respectively.



6. On Page 3, click **Next**.
   We will keep all the relationships.
7. On Page 4, type a name and definition for the denormalization operation, and then click **Finish**.

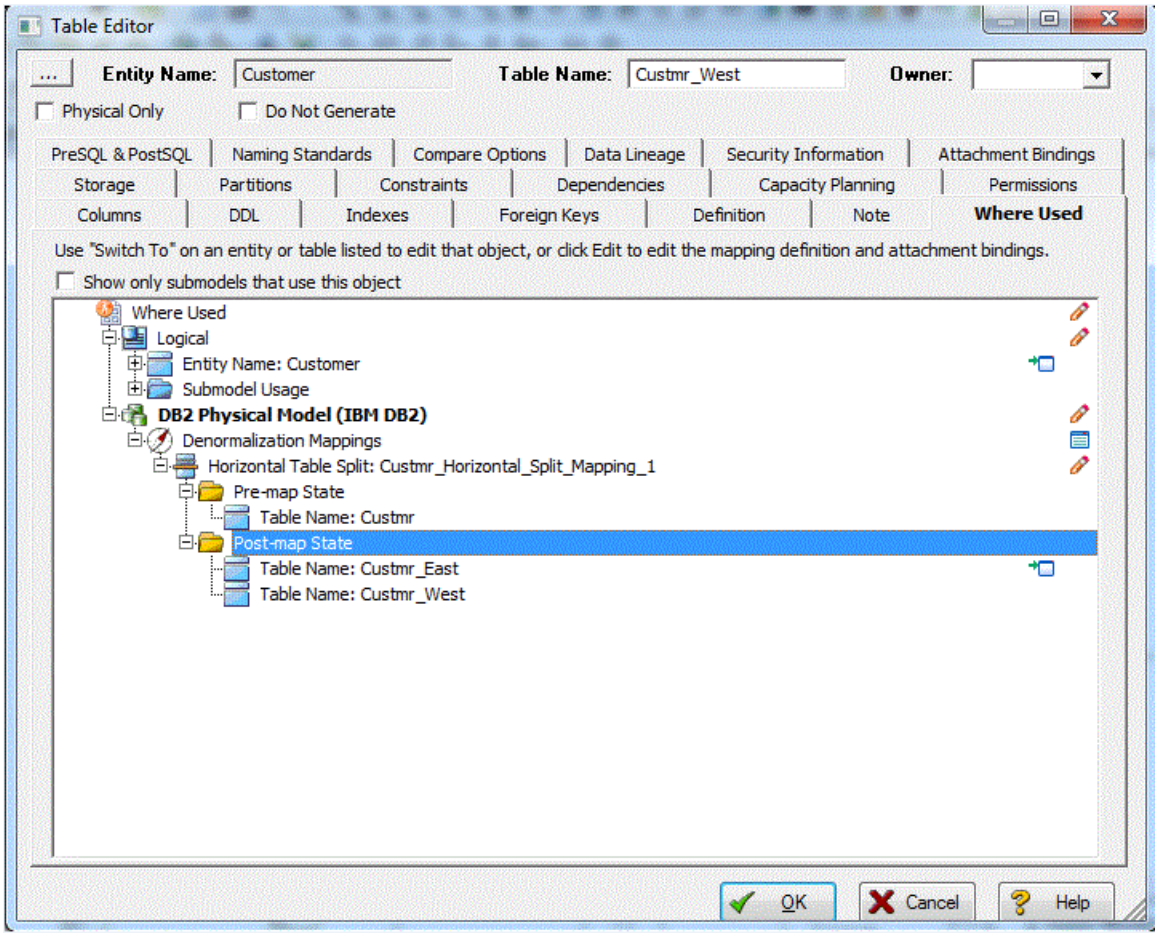
Finished! After the split the Custmr table will be two physical tables that look like this:



The two tables are identical except for the name.

You can selectively choose which attributes are included in the resultant tables by using a vertical split.

The denormalization mapping is stored with each of the tables.



You can use the denormalization information to undo the operation or see the history of what happened. ER Studio Data Architect tracks the before and after states of these operations. This comes in handy in the next section where we discuss the **Where Used** analysis that can be performed between the logical and physical models.

See Also

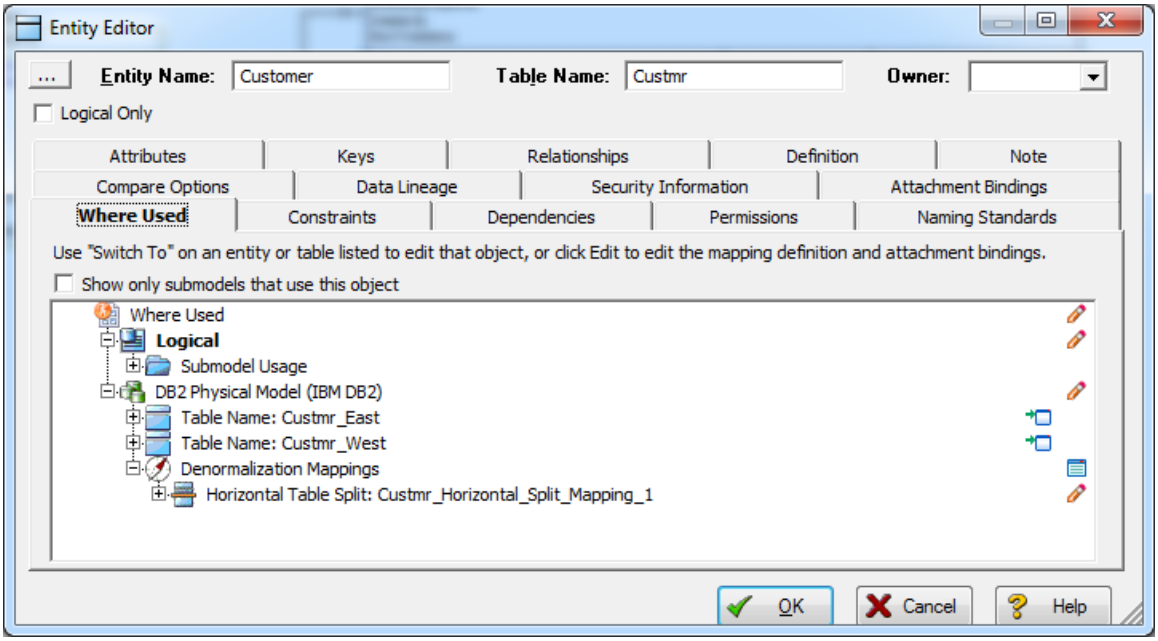
- Optimizing Query Performance on the Physical Model (Denormalizing the Physical Model)
- Upgrading Diagrams with Denormalization Mappings

Tutorials: Finding out How an Entity Maps to the Physical Model

Now that we have performed a denormalization operation, the logical entity, Customer, essentially has become two physical tables, Custmr_East and Custmr_West. The ties between the logical and physical models are not lost. ER Studio Data Architect allows you to see what Customer in the logical model maps to in the DB2 physical model.

Let's take a look at the Customer entity in the logical model.

1. In the Data Model Explorer, navigate back to the Customer entity in the Logical model.
2. To start the Entity Editor, double-click the **Customer** entity.
3. Click the **Where Used** tab.



Once the tree is expanded, you can see the lineage of what has happened to the object. Notice that *Custmr_East* and *Custmr_West* are listed as physical implementations of the *Customer* entity. The denormalization mapping object shows how the end result was achieved.

The **Where Used** tab also displays the submodel usage of a particular entity within the logical or physical model. This allows you to see which business areas the entity belongs to.

Note:Where Used information is also available for attributes and columns.

Conclusion

In this session, you have seen how incredibly quick and easy it is to:

- Build a logical data model from scratch.
- Create a new submodel view to understand how to model on specific parts of a larger Main Model.
- Generate a physical model from a logical database in preparation for building a new database.
- Denormalize objects in the physical model.
- View the mappings between the logical and physical models using the Where Used tab.

See Also

- Developing the Logical Model
- Developing the Physical Model
- Synchronizing Physical and Logical Models
- Generating a Script File or Database

Tutorials: Documenting an Existing Database

One of ER Studio Data Architect's most powerful applications is that of a documentation generator to communicate complex databases and associated metadata to the Enterprise. ER Studio Data Architect is equipped with extensive report generation capabilities:

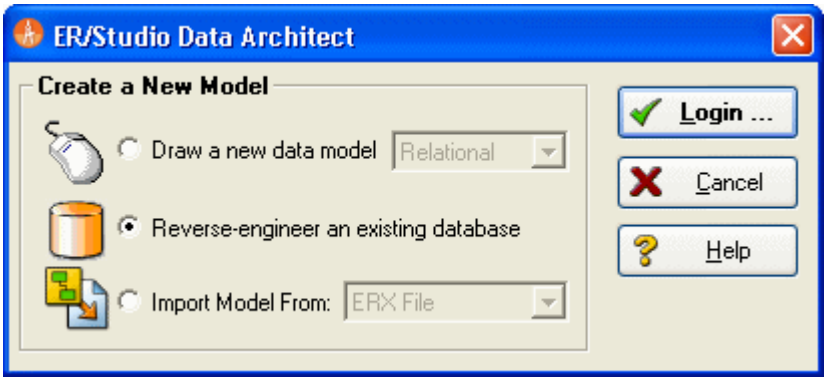
- HTML Report Generation: Instantaneous generation of an HTML-based Web site designed to provide simple navigability through data models and model metadata using standard browsers such as Google Chrome or Mozilla Firefox.
- RTF Report Generation: Instantaneous documentation generation compatible with applications like Microsoft Word.

In the exercise below, we will reverse-engineer an existing database and generate an HTML report for distribution and navigation to those who depend upon the information about the data model, but who may not be permitted to connect to the database for security or organizational reasons.

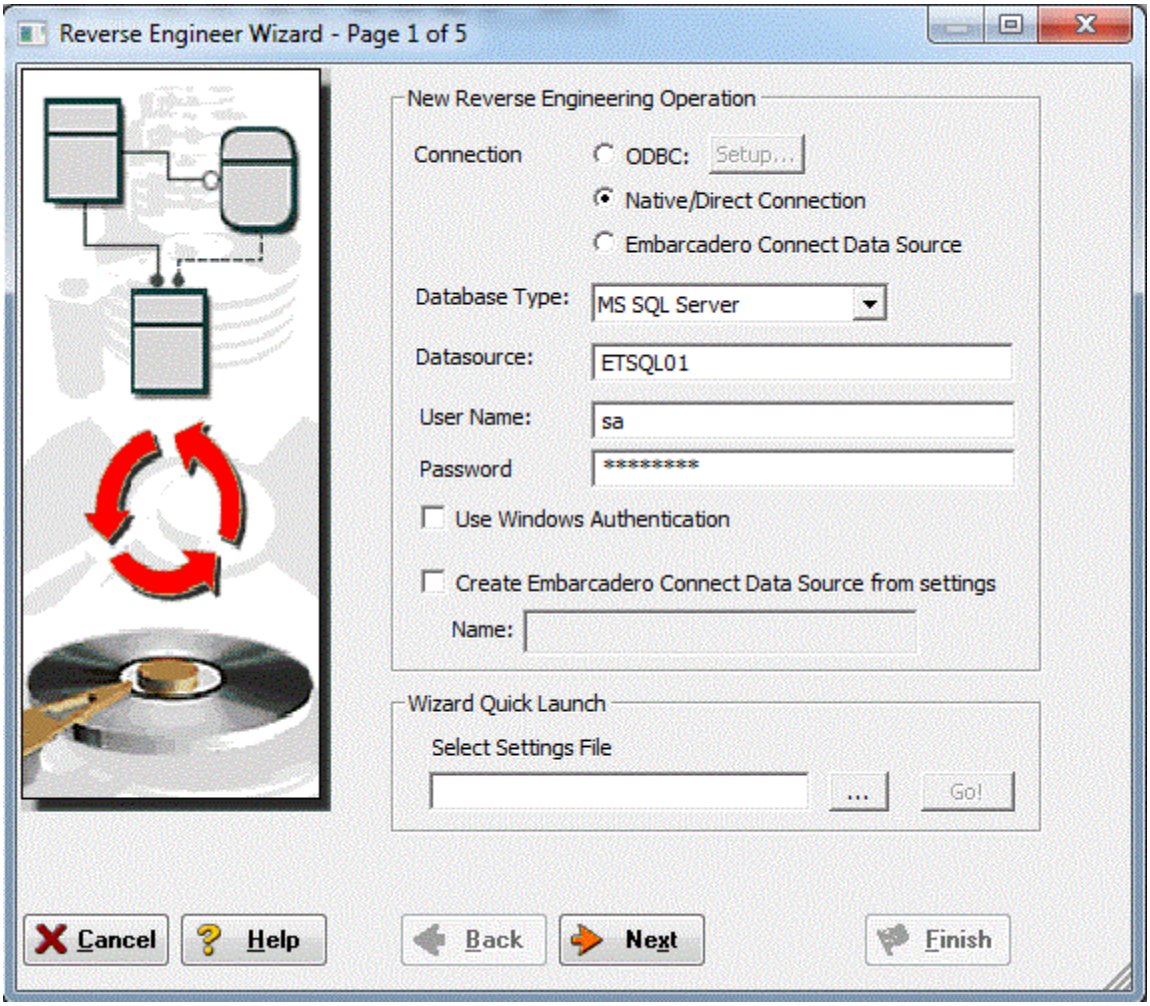
PRE-REQUISITE:

This exercise assumes that you can connect to an existing database in order to document it. If you cannot connect to an existing database, you can still generate documentation from the installed sample models; skip steps 1 through step 8 below which relate to in reverse-engineering and begin at step 9 after opening a sample model included with ER Studio Data Architect.

1. Choose File > New.

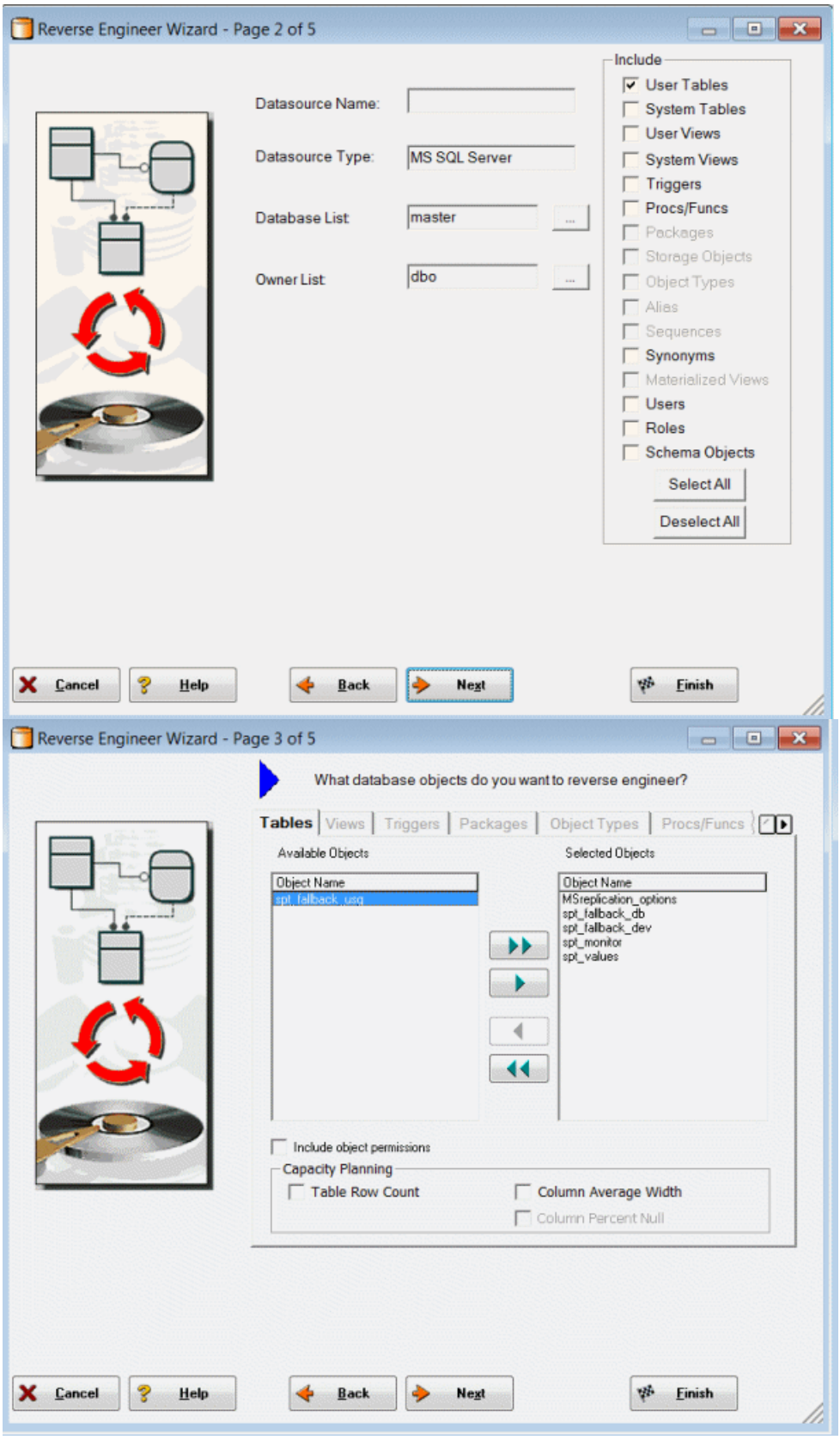


2. Select *Reverse-engineer an existing database*.
3. Click *Login*.
    - You can reverse engineer the database from either an ODBC datasource or via Native RDBMS client connectivity. In this example, Native Connectivity to Microsoft SQL Server will be demonstrated.
4. Type the relevant connectivity information such as the data source name, and user name and password and then click *Next*.

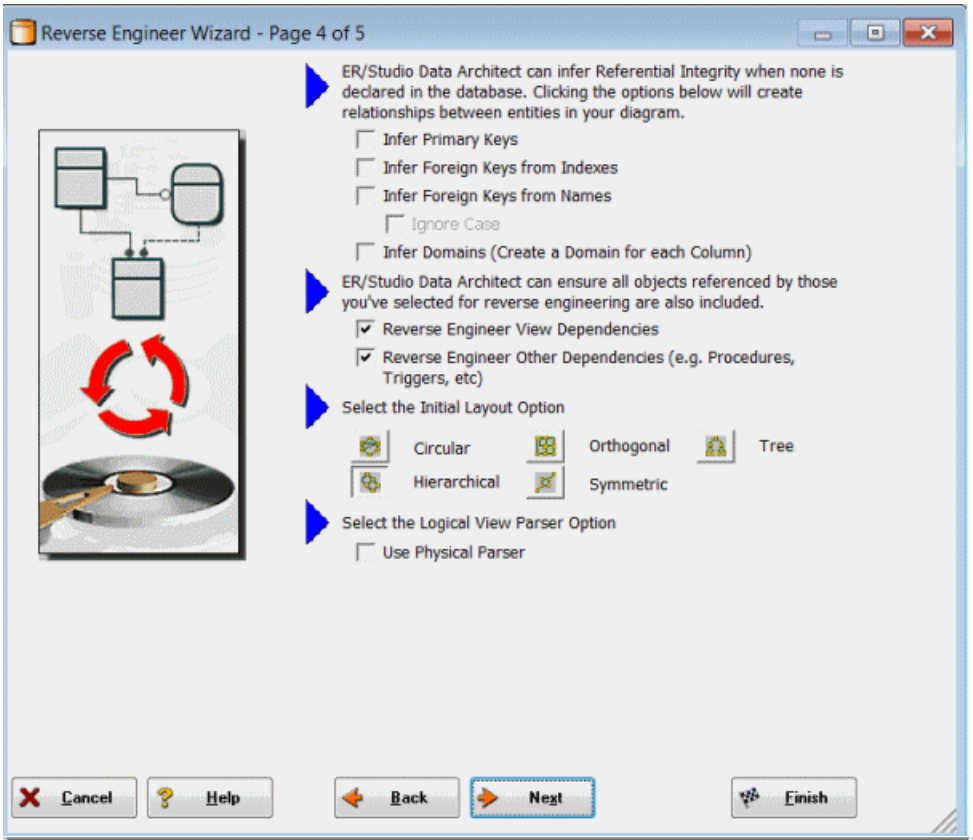


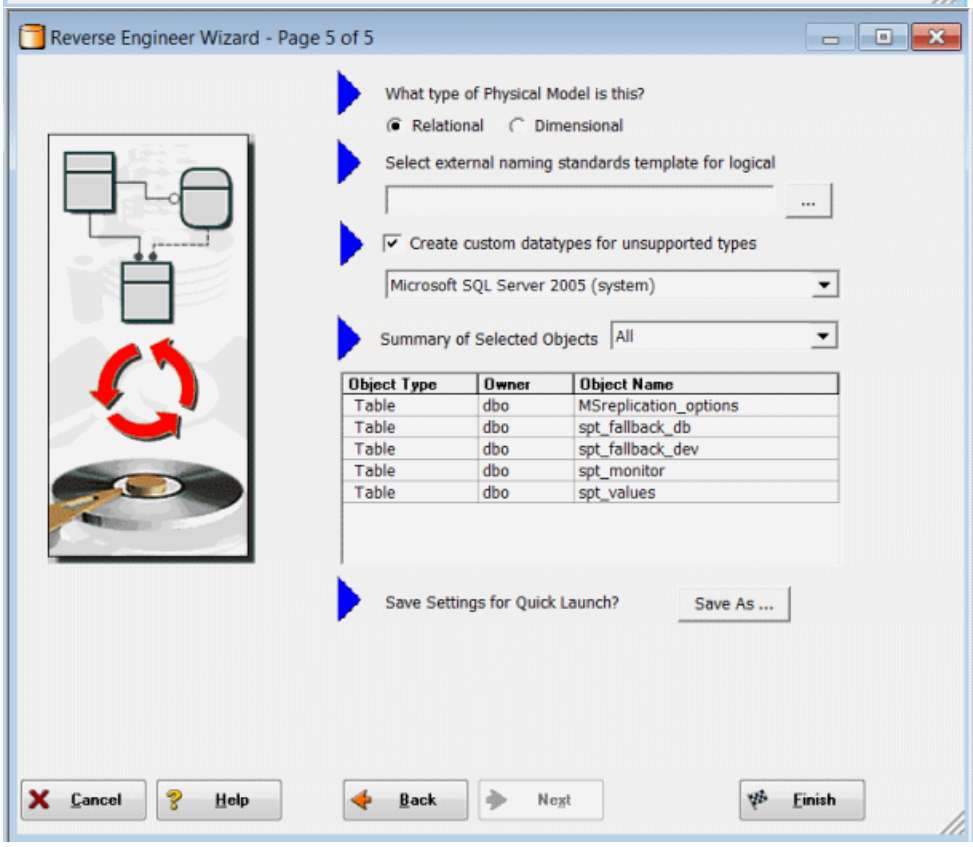
5. Walk through the *Reverse Engineer Wizard* selecting the objects, options, and layout preferences for the model.

   Note: A new feature using ER/Studio Team Server data sources is available in the Reverse Engineer Wizard. Please see ER/Studio TS documentation for more details.



6. Continue through the wizard to select layout styles and other preferences.

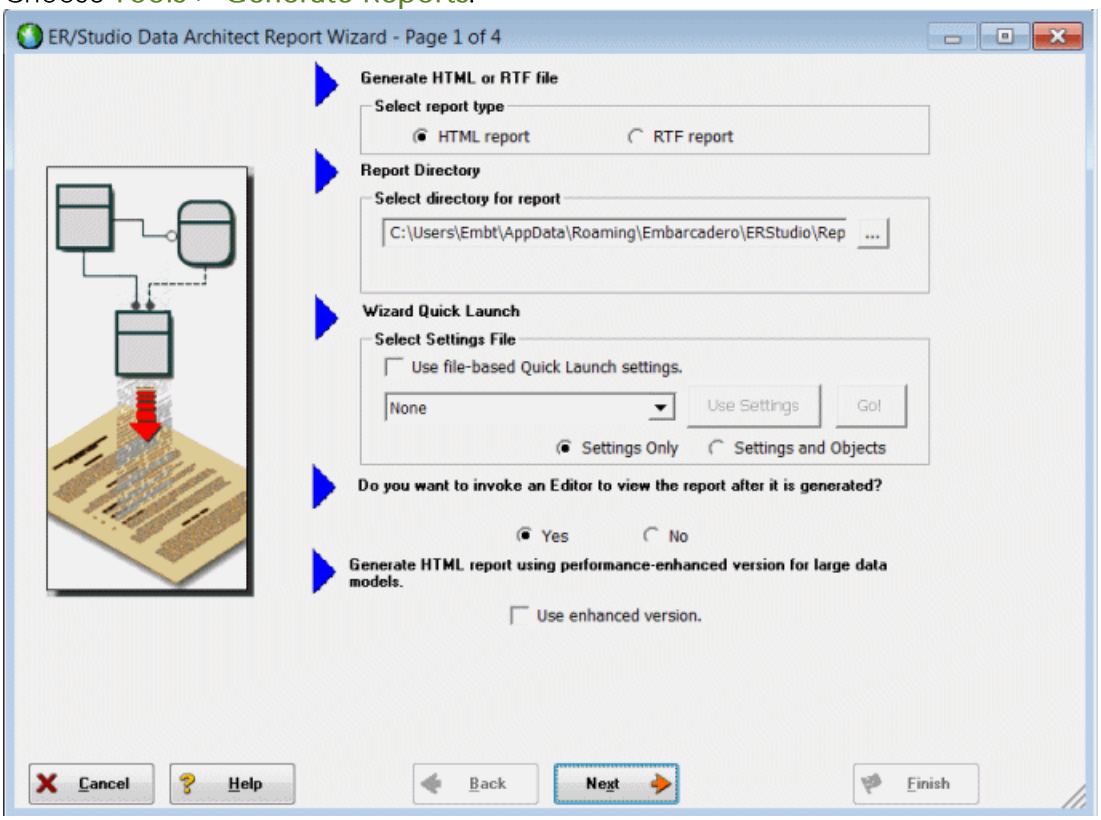




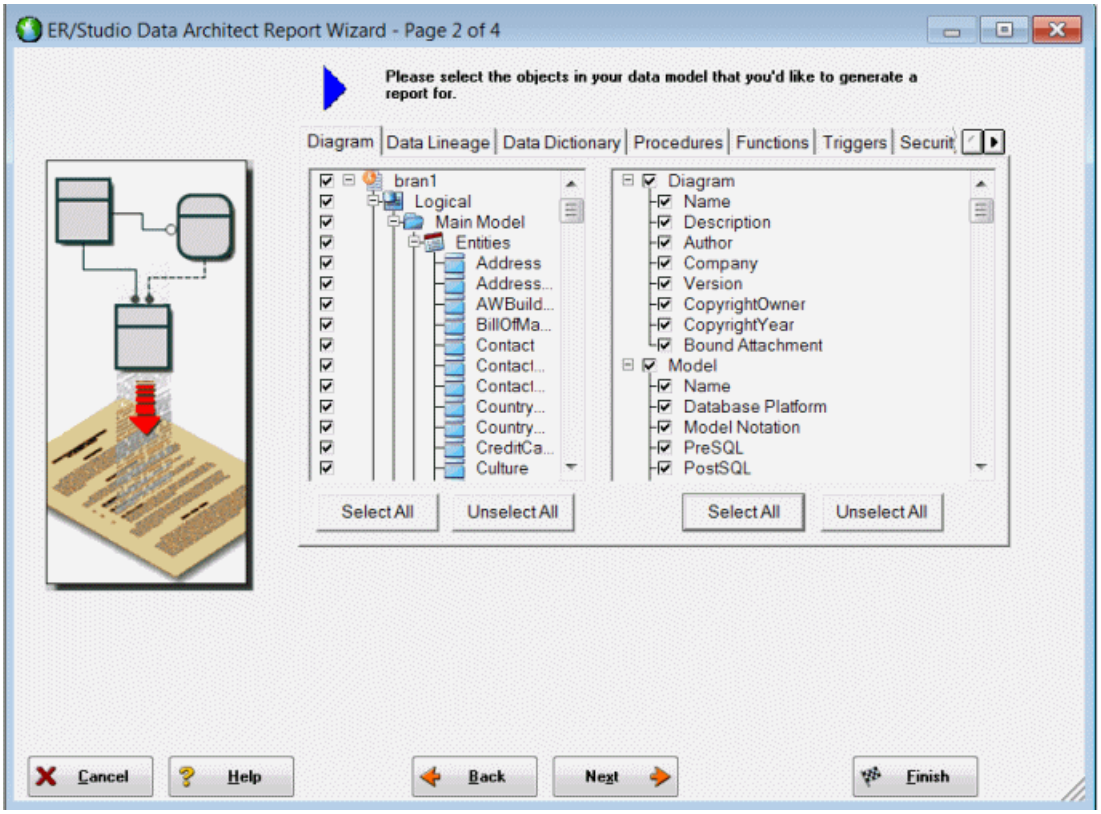
7. Click **Finish** and ER Studio Data Architect reverse engineers your database!

Once reverse engineering of your database is complete, we will generate a complete HTML report of the database for others in your organization to review.

8. In the Data Model Explorer, select the **Physical Main Model**.
9. Choose Tools > Generate Reports.



10. On the first page of the wizard, for the report type, select **HTML**.
11. On page 2, click **Select All** in both areas of the **Diagram** tab.

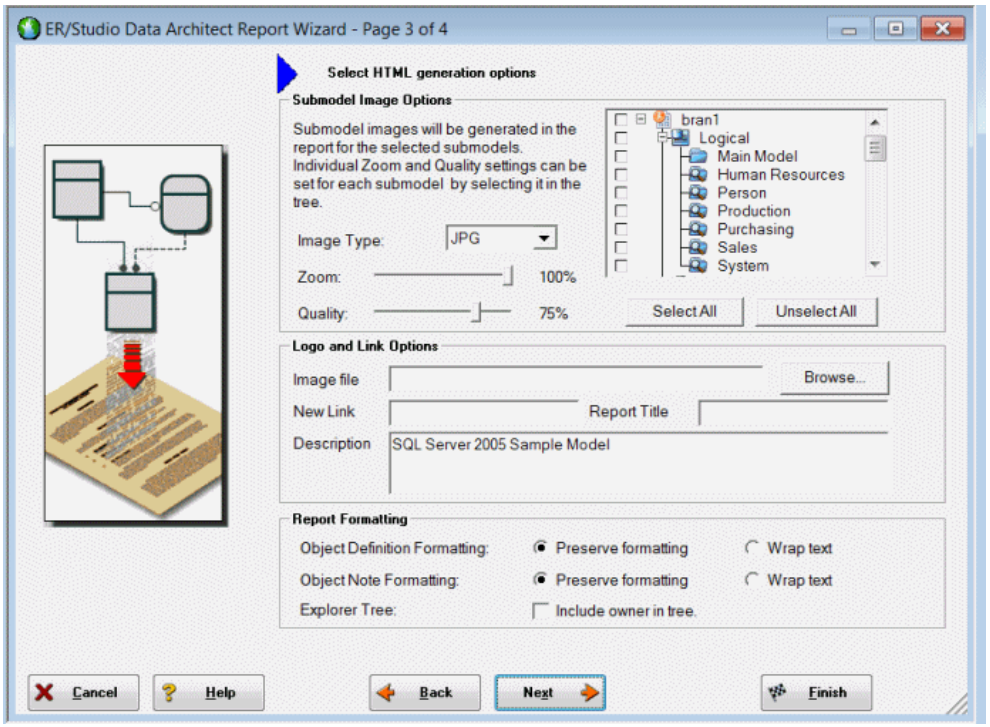


12. Click **Select All** in both areas of the **Data Dictionary** and the **Procedures** tabs, and then click **Next**.

    Note: The tabs available on page 2 depend on what objects are supported by the database platform of the selected model. Some databases support more database objects than Microsoft SQL Server 2005, such as Oracle11g and IBM DB2 LUW 9.x for which there are also tabs on this page for procedures, functions, triggers, packages, and tablespaces. If the model was previously denormalized, a **Denormalization Mappings** tab would also appear.

13. On page 3, in the **Submodel Image Options** area, click **Select All**.

    Tip: In the *Logo and Link Options*, you can choose to replace the ER Studio Data Architect default Embarcadero Technologies logo in favor of your own corporate logo (and Hyperlink).



Because HTML formatting can be included in object definitions, you can also choose to preserve the formatting specified on the *Definitions* tab of the various object editors.

14. Click **Next** to advance to Page 4 of 4, and then click **Finish**.
    ER Studio Data Architect then begins the report publication process and launches the default browser so you can review the report.

15. Finished!

Start navigating the report via your browser. Navigation will perform exactly as it does when you are using ER Studio Data Architect! Expand the tree to find Model Image and click on it (see below). You will see a read-only version of your data model (as seen below). Use the Explorer to navigate to any metadata you want or select the entities and relationships in the model image to jump to their information.

Conclusion

In this session, you learned how to:

- Connect to and reverse-engineer an existing database with ER Studio Data Architect.
- Document a database in seconds by using ER Studio Data Architect's automatic HTML documentation publication facility.

See Also

- Generating RTF and HTML Model Reports
- [ER/Studio TS documentation]

Tutorials: Documenting Data Lineage

The Data Lineage feature of ER Studio Data Architect enables you to document the movement of data from point A to point B (and any intermediate steps in between). This movement is sometimes referred to as Extraction, Transformation and Load (ETL). Points A and B can be anything from flat files, high-end databases such as Oracle and DB2, XML, Access databases, and Excel worksheets. This is sometimes referred to as source and target mapping. A model produced in ERStudio can represent any point along the way. Data Architects need the ability to specify the *source* or *target* of data down to the column-level. Along with the metadata that defines the source and target mapping are rules for how the data is manipulated along the way.

The next section will help you document the data lineage of your systems.

It is comprised of the following tasks which correspond to the general ETL workflow:

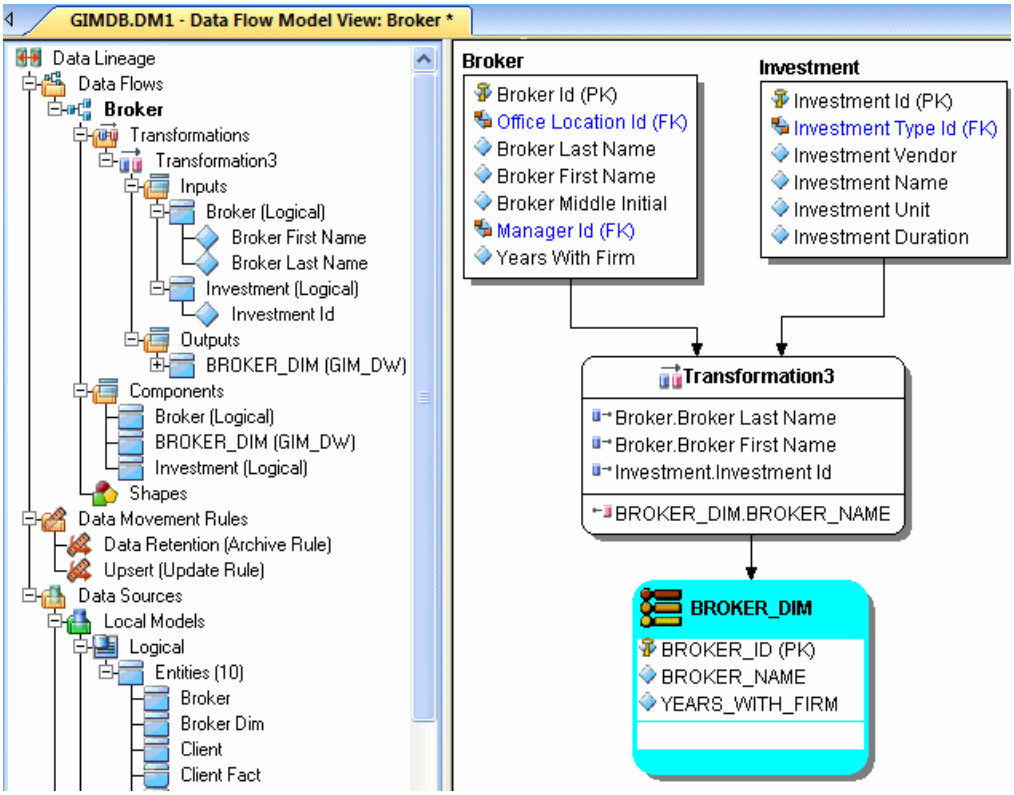
- Creating a Data Flow
- Creating a Data Movement Rule
- Defining External Source and Target Systems
- Creating a Data Lineage and Transformation Visualization

See Also

- Data Lineage Workflow
- Documenting Data Extraction, Transformation, and Load

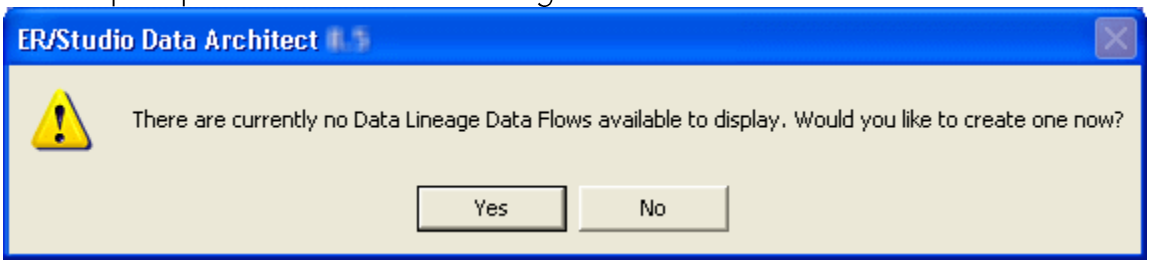
Tutorials: Creating a Data Flow

The Data Flow organizes and encapsulates one data transformation and the source tables and columns used in the transformation to produce the target data. Multi-tiered mappings are possible and there can be multiple transformations involving different columns between two tables as illustrated below.



Create a Data Lineage Data Flow

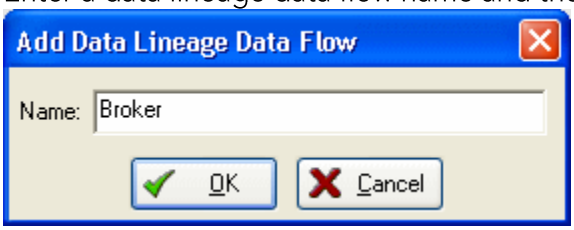
1. Choose File > Open and select the GIMB.DM1 diagram in the Sample Models directory.
2. Click the **Data Lineage** tab at the bottom of the application window.
   You are prompted to create a *Data Lineage Data Flow*.



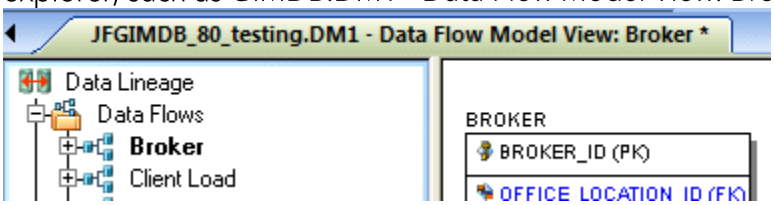
3. Click *Yes*.
   If this is not the first time you click the Data Lineage tab after opening a diagram, from the **Data Lineage Explorer,** right-click the **Data Flows** node and then click **Create Data Flow**.

If you would like to keep your version of `GIMB.DM1` unchanged, *'save as'* a different filename before proceeding past the Undo/Redo warning dialog.

4. Enter a data lineage data flow name and then click *OK*.



Note: The name that appears in the diagram title tab at the top of the application window is appended with*: data flow* name, when you click a task in the *Data Lineage* explorer, such as *GIMDB.DM1 - Data Flow Model View: Broker**.



The Data Flow has been created.

See Also

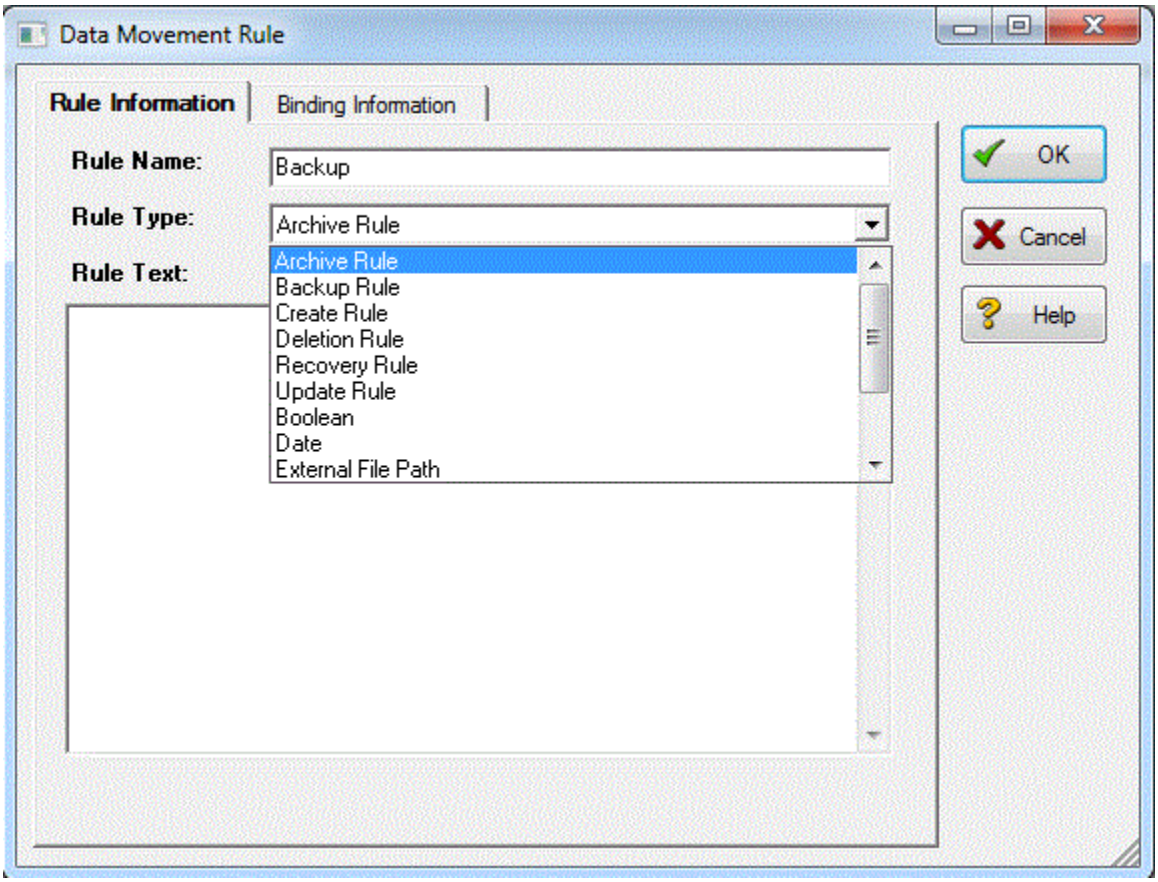
- Data Lineage Workflow
- Introduction to Data Lineage

Tutorials: Creating a Data Movement Rule

Data Movement rules describe how source and target tables and entities are related. You can relate source data to one or more tables and entities in the same model, the active diagram, or to tables imported from external systems. The rules defined here are used at the table level on the Data Lineage tab of the entity and table editors.

Create a data movement rule

1. On the Data Lineage tab, right-click Data Movement Rules and choose New Data Movement Rule.



2. Complete the **Data Movement Rule** editor as required and then click **OK** to exit the editor.

   Tip: Once created, you can edit the Data Movement rule by double-clicking it to launch the **Data Movement Rule** editor.

The following describes options that require additional explanation:

Rule Information tab

- **Rule Name**: Enter a name that indicates the operation and objects acted on, depending on the specifics of your binding definition.
- **Rule Type**: Select a generic movement rule type that best describes the data movement.
- **Rule Text**: Document your data movement plan here, perhaps adding instructions or contingency plans.

Binding Information tab

Select the object classes and/or specific objects to which you want to bind this attachment. You can override this setting using the **Data Lineage** tab of the entity or table editor.

See Also

- Assigning Model Data Movement Properties
- Data Lineage Workflow

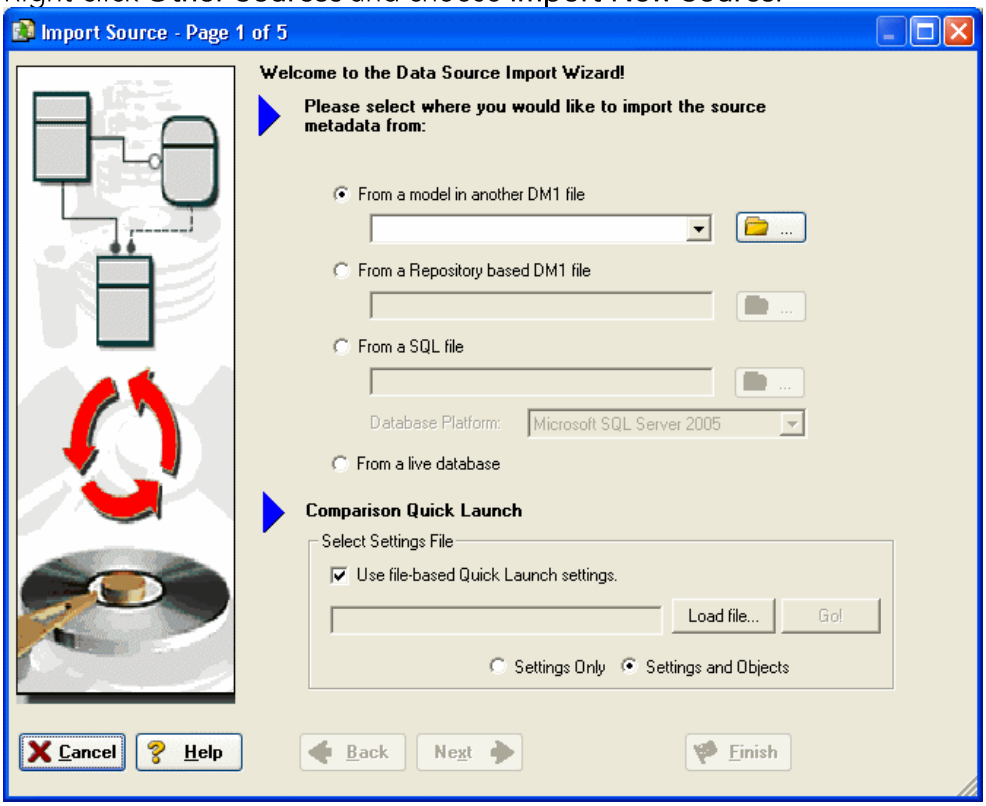
Tutorials: Defining External Source and Target Systems

Data sources can originate from models in the active diagram (local models) or from external sources that are either imported into the active diagram or created on the Data Lineage tab. A data source can be imported from **.dm1* files, **.dt1* files, database or from SQL files, flat files, and other common application files. The following describes how to import metadata from an external source.

Note: Source data imported through the Data Lineage tab only includes information such as table and column name, datatype, nillability, primary key, and column definitions. To obtain more details, reverse engineer the database or import it into ER Studio Data Architect using the Metadata Wizard.

Import external source or target data

1. From the Data Lineage tab, expand the Data Sources node.
2. Right-click **Other Sources** and choose **Import New Source**.



3. Complete the **Import Source** wizard as required and then click **Finish** to import the source.
   The new source will appear under the *Other Sources* node.

The following describes options that require additional explanation:

Page 1 - Please select where you would like to import the source metadata from

- **From a Repository based DM1 file**: Lets you obtain source from data models and Named Releases managed within the ER/Studio Repository. When you select this option, ER Studio Data Architect opens the Repository Operation Status dialog box and the **Get From Repository** dialog box. This process connects to the current Repository Server defined in the Repository settings. The **Import Source** wizard automatically gets the diagram.
- **From an SQL file** ER Studio Data Architect imports the SQL file.
- **From a live database:** If you select this option, a page appears where you can select the database and connection type. The connection type can be either ODBC or Native/Direct Connection. For information about connecting to databases, including troubleshooting information, see Connecting to Database Sources and Targets.
- **Comparison Quick Launch**: The Compare Quick Launch data is saved as an *.rvo file. For information on using the *Quick Launch* option in the wizard, see Saving and Using Quick Launch Settings.

Page 5 - Results

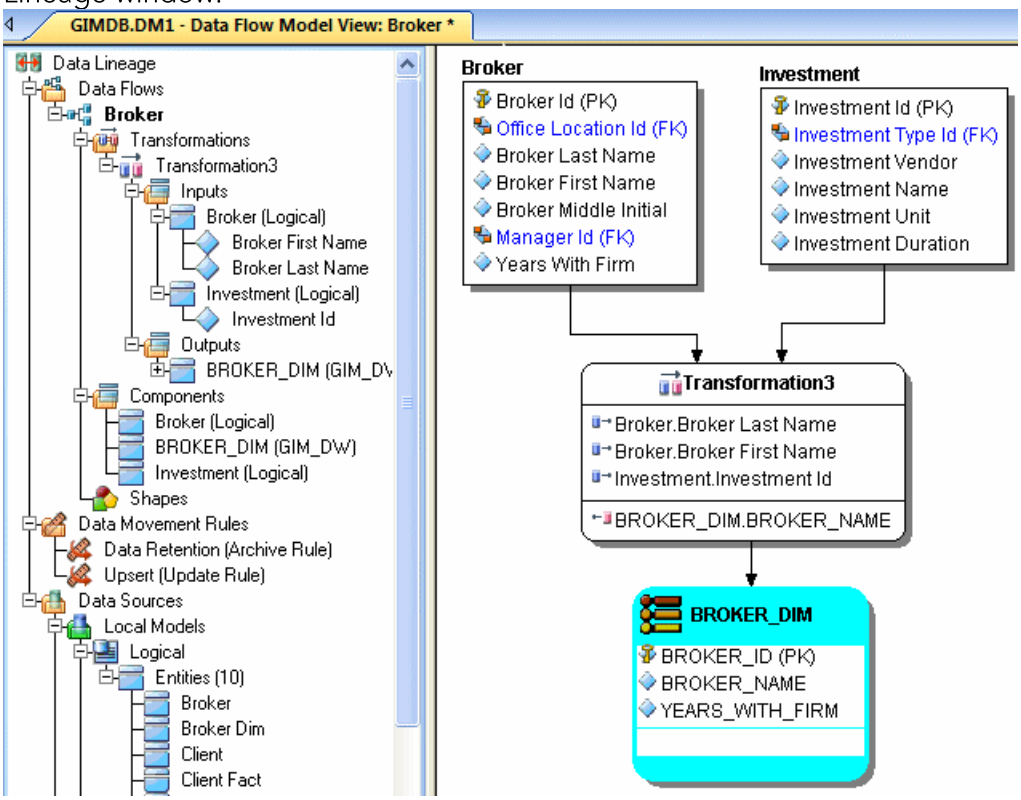
- **Current and Target Model Display Grid**: Between the Source and Target models is a Resolution column. The default merge decision is Merge the data into the new source file. You can click on any item in the **Resolution** column to enable the decision list. If you want to change the decision, click the list and then click the new resolution. When you change the default resolution of an object, the decisions of their dependent properties and objects are automatically updated. You can also click the category folders, like the **Tables Resolution** column to change all the decisions for all the underlying objects in that object category. And, you can use the **CTRL** key to select multiple items, and then right click to enable the decision list.
- **SQL Difference**: To enable the SQL Difference utility, select any difference that is a long text field, such as a Definition, Note, or DDL, and then click SQL Difference to view the differences between the SQL of the models. This utility only allows you to view the differences; difference resolutions are performed on the **Results** page of the **Compare and Merge Utility**.
- **Filter Report on Results:** Create a report of the source content and your chosen resolutions. You can choose to create an HTML or an RTF report.
- **Tip:** You can modify the default display using the options at the bottom of the page.

See Also

- Importing a Model from External Metadata
- Attaching External Documents to the Data Model

Tutorials: Creating a Data Lineage and Transformation Visualization

1. To create the data source or transformation input tables, expand Data Sources > Local Models > Logical > Entities and then drag and drop the **Broker and Investment** tables onto the Data Lineage window.
2. To create the data target or transformation output tables, navigate to Data Sources > Local Models > GIM_DW and then drag and drop the **Broker** table onto the Data Lineage window.

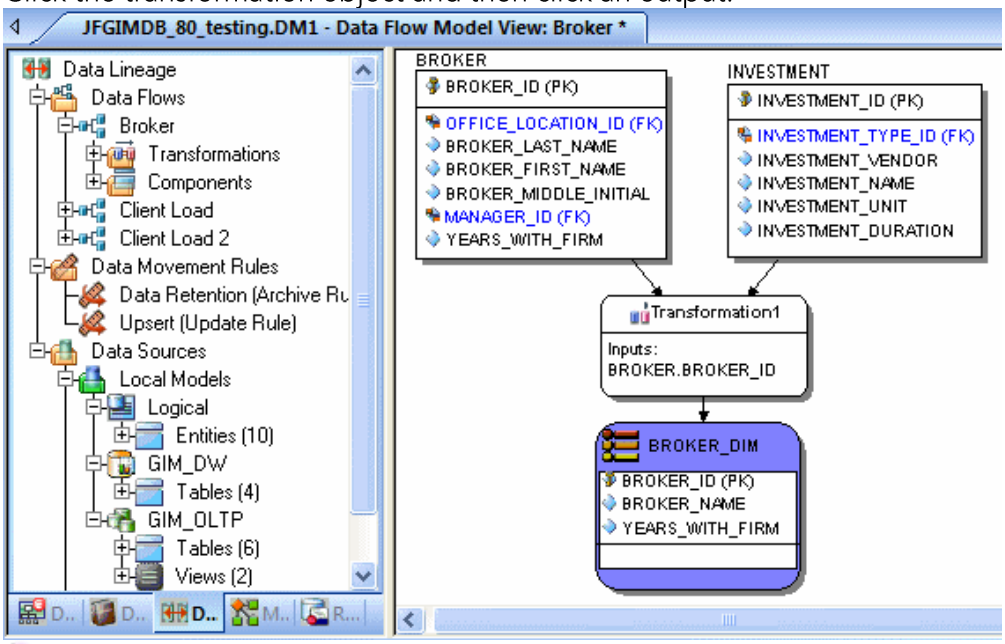


3. To obtain the Transformation Insertion tool, right-click an empty space in the **Data Lineage** window and then click **Insert Transformation**.
4. To insert the transformation, click in the **Data Lineage** window between the source and target data sources and then right-click to drop the **Transformation Insertion** tool.
5. Reposition and resize the transformation object to suit your needs.
6. Right-click an empty space of the **Data Lineage** window and then click **Insert Data Stream**.


Tip: Transformation and Data Flow tools are also available on the toolbar. Hover the mouse over the tools to find the tool you need.

7. Click an input and then click the transformation object. Repeat as many times as necessary to link all the inputs to the transformation object.

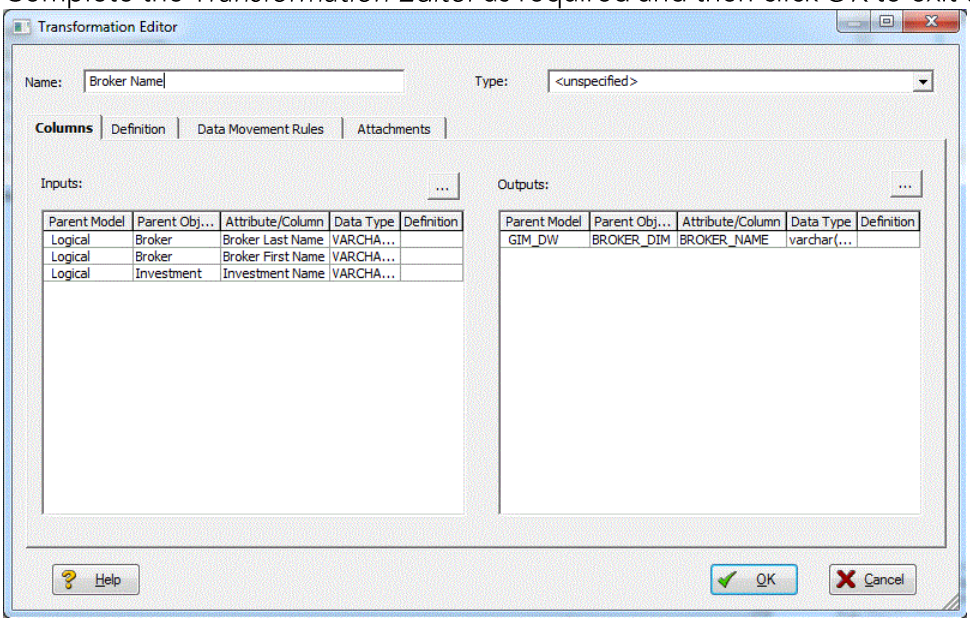
8. Click the transformation object and then click an output.



Tip: If the Inputs and Outputs do not display in the diagram as they do in the illustration above, choose View > Diagram and Object Display Options > Transformation and then click *Inputs and Outputs Columns*.

9. To define which columns should be used in the transformation and any transformation rules, double-click the new transformation to open the **Transformation Editor**.

10. Complete the *Transformation Editor* as required and then click *OK* to exit the editor.



11. You are done! Now you can more easily share your ideas with your colleagues!

**Tip:** Once the Data Flow is created, you can double-click it to change its name, or double click a transformation or component to change its properties.

The following describes options in the Transformation Editor that require additional explanation, for more options that are not required for this tutorial, please read the User Guide:

Columns tab

- **Inputs**: Click the ellipsis (...) button to choose the inputs to be transformed in this task.
- **Outputs**: Click the ellipsis (...) button to choose the outputs resulting from the transformation.

Definition tab

- **Business**: Describe the transformation for your audience.
- **Code**: Enter the code that will perform the transformation, such as a SELECT statement, or a VBBasic or Java Script function or procedure.

Data Movement Rules tab

These are the rules from the Data Movement Rules node of the Data Lineage Explorer.

**Note:** You can delete or edit an input or output column by double-clicking the transformation in the **Data Lineage** window, clicking the ellipsis in the *Transformation Editor* and then deselecting the column you want to remove.

Attachments tab

Bind an external piece of information or attachment to the transformation. You can also remove an attachment from an object, override the default value of an attachment, or change the position of a bound attachment. To override the value of the attachment you have moved to the **Selected Attachments** grid, double-click the **Value** field of the target attachment. ER Studio Data Architect opens the **Value Override Editor** or a list, depending on the attachment datatype. Attachments are created in the **Attachments** folder of the **Data Dictionary** and must be applied to the default before they will display on this tab.

See Also

- Documenting Data Extraction, Transformation, and Load
- Introduction to Data Lineage

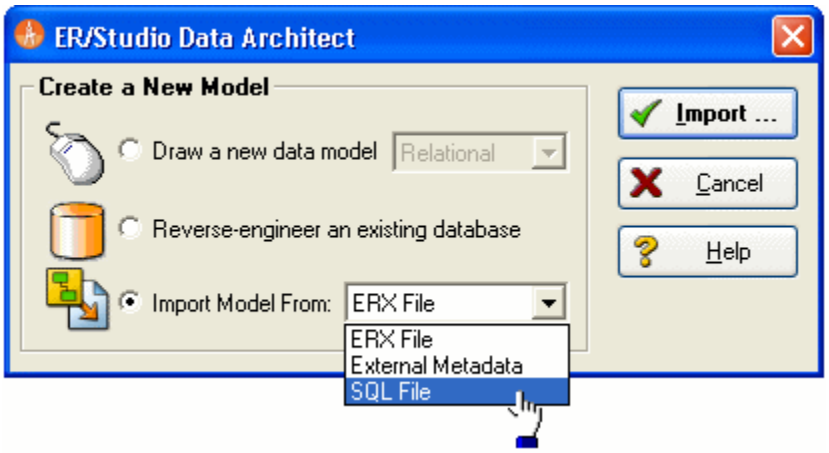
Tutorials: Diagram Navigation and Aesthetics

To assist with the creation of presentation-quality diagrams that are easy to navigate and are aesthetically pleasing, ER Studio Data Architect offers progressive diagram Auto Layout and Navigation utilities that also help you to clean up complex diagrams. Modelers should spend time solving complex database or business data model problems, not forcing boxes and lines to look a certain way.

Navigating the Diagram

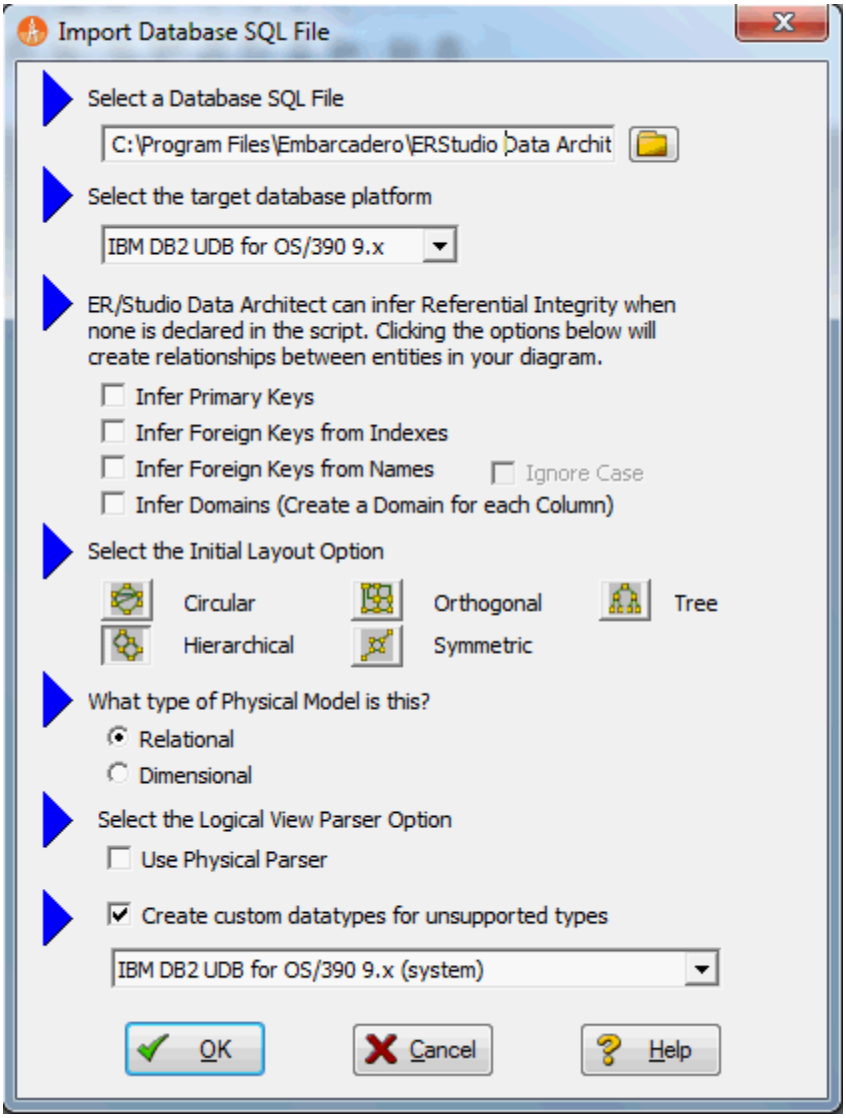
To demonstrate some of ER Studio Data Architect's layout and navigation utilities, we will import a sample SQL script provided with ER Studio Data Architect.

1. Close any files you have open.
2. Choose File > New.



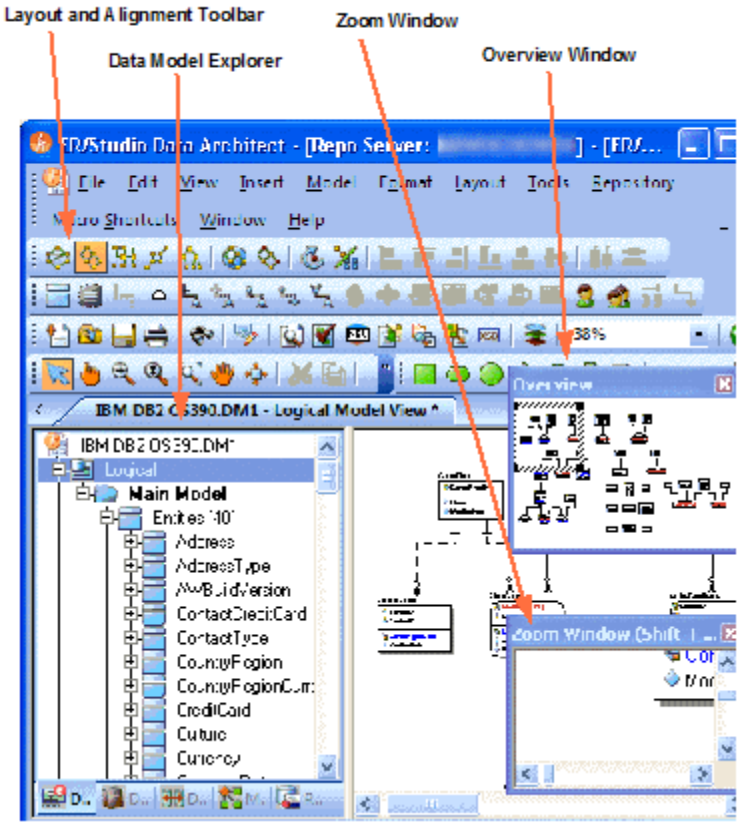
3. Select **Import Model From:** and then in the import list, click **SQL File**.

   Note: The ERX File choice enables you to import Computer Associates ERwin 3.5.2 ERX files. Selecting *From External Metadata* launches the MetaWizard to import from alternative sources.
   The '*Import Database SQL File* dialog appears:



4. To the right of **Select a Database SQL File** click the folder icon, click **Sample DDL (DB2 7.x for OS390).SQL**, and then click **Open**.
    - The full path to this file is:
      For Windows XP:
      C:\Documents and Settings\<user>\Application Data\Embarcadero\ERStudio\SQLCode
      For Windows 8, Windows 7, and Windows Vista:
      C:\Program Files\Embarcadero\ERStudio Data Architect XE4\SQLCode
5. In the **Select the target database platform** list, click **IBM DB2 UDB for OS /390 9.x**.
6. Click **OK**.

Finished! Once the SQL Script is finished importing (as depicted below) the following items will assist you in leveraging a variety of Auto Layout and Navigation Features.



- Layout and Alignment Toolbar: Use any of the four Auto Layout styles to change the layout of the diagram with the click of a button. The auto layout styles are all entirely customizable styles. You can also customize the diagram layout via the Layout Properties pages that can be launched by clicking Layout > Layout Properties. In addition, you can have multiple layout styles in a model. Select some object and then choose the layout style. If you don't have any objects selected when you click the layout button, the layout style chosen will be applied to the entire model.
- Data Model Explorer: Click on any object in the Data Model explorer and it will automatically be selected in the diagram and focused in both the *Zoom Window* and *Overview* window.
- Overview Window: Use this window as a thumbnail of your model to pan the entire model or zoom in and out. It can also pan and zoom the diagram if grabbed or sized. If the Overview window is not already visible, press F9 to activate it.
- Zoom Window: Use this window as a magnifying glass to enlarge any diagram objects under your mouse cursor. You can also press SHIFT+F8 to freeze the zoom window to keep a single object frozen while you continue to pan around the diagram. If the Zoom window is not already visible, press F8 to activate it.

Diagram Aesthetics

One of the tremendous benefits of building data models is the wide range of audiences that can realize value from them. Part of this relies on what information is displayed in the diagram.

Depending on the audience you may want to limit or expand what is displayed. For example, developers may benefit from looking at a model that displays data type, null option, and unique and non-unique index information, while business analysts may just need the entity name and the definition. ER Studio Data Architect offers many display properties that can be customized exactly for this purpose.

Continuing with the previous section, we will use the DB2 model that was built to demonstrate some of the ways to customize the appearance of the model.

We will use the Diagram and Object Display Options dialog on the Diagram toolbar to further customize the view of the logical and physical models.

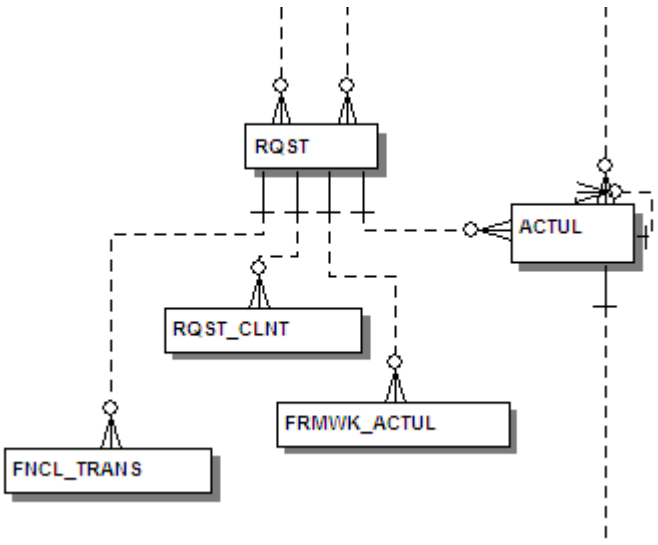
- You can use the Colors & Fonts tool to customize the look and feel further of each model.

Setting the Logical Model Display

1. Select the *logical model* and then on the Diagram Toolbar, click the *Diagram and Object Display Options* tool.



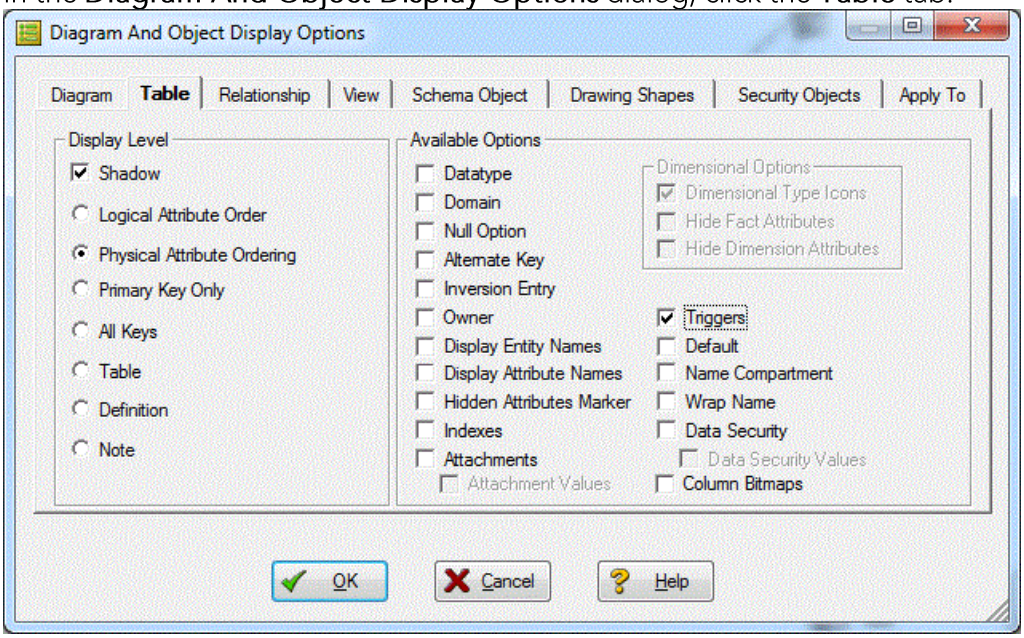
2. In the *Diagram and Object Display Options* dialog, click the *Entity* tab, and then in the *Display Level* area, select *Entity*.
3. Click *OK*.



Note: Only entity names are displayed for each entity. You may also want to re-layout the diagram since the entity sizes have changed.

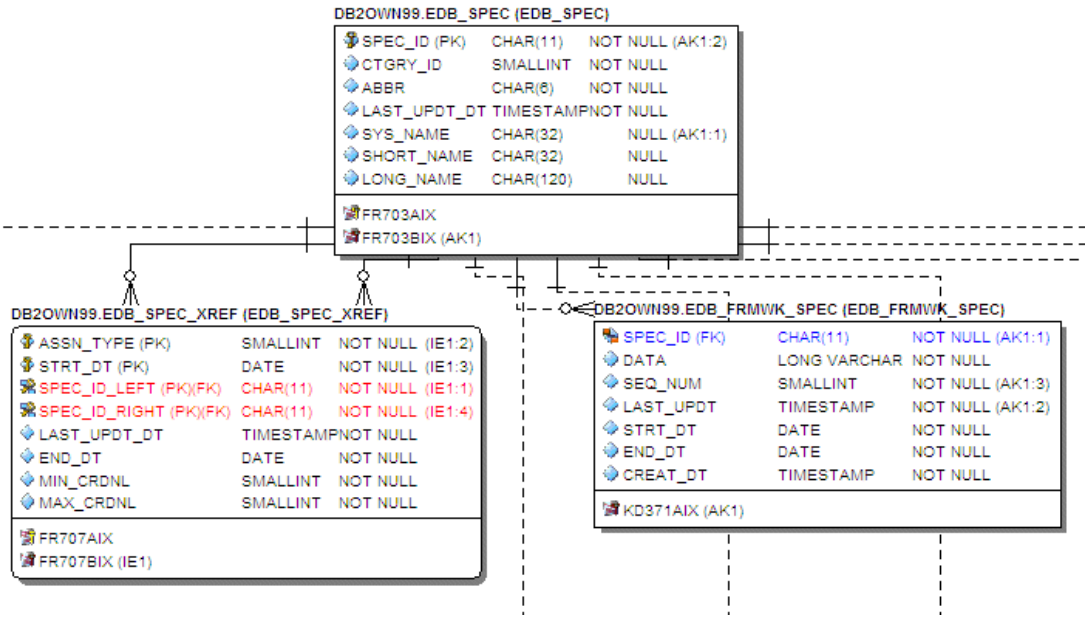
Setting the Physical Model Display

1. Select the **physical model** and then choose View > Diagram and Object Display Options.
2. In the **Diagram And Object Display Options** dialog, click the **Table** tab.



3. In the **Display Level** area, select **Physical Attribute Ordering**.
4. In the **Available Options** area, select the specific properties you want to display.
5. Click **OK**.

The model should now display more details for the physical model, as seen below.




Note: Since the sizes of the objects changed, you may want to change the model layout using one of the ER Studio Data Architect advanced layout engines. You can also customize the default display properties for new models by choosing Tools > Options, and then selecting the desired options on the **Display** tab.

Conclusion

In this session, you have learned how to:

- Import an SQL file and allow ER Studio Data Architect to automatically create a diagram.
- Use a variety of auto layout and navigation tools to enhance the aesthetic experience of the diagram and to improve the data model navigability.
- Customize the display of both the logical and physical models.

Thank you for completing the tutorials section. Please visit the User Guide for more examples and information on using ER Studio Data Architect.