

PUBLIC

SAP BusinessObjects Business Intelligence platform

Document Version: 4.3 – 2020-06-12

Universe Design Tool User Guide

© 2020 SAP SE or an SAP affiliate company. All rights reserved.

THE BEST RUN SAP

Content

1 Document History. 12

2 Introducing the Universe Design Tool. 13

2.1 Overview. 13

2.2 Universe design tool and universe fundamentals. 13

What is a universe?. 13

What is the role of a universe?. 14

What does a universe contain?. 14

About the universe window. 16

Universe design tool install root path. 16

2.3 How do you use the universe design tool to create universes?. 17

How do objects generate SQL?. 17

What types of database schema are supported?. 18

How are universes used?. 18

2.4 Who is the universe designer?. 19

Required skills and knowledge. 19

What are the tasks of the universe designer?. 20

2.5 The basic steps to create a universe. 20

2.6 Introducing the universe development process. 21

Universe design methodology. 21

Universe development cycle. 22

Optimizing universe planning and implementation time. 23

2.7 Multilingual universes. 24

2.8 Definitions of languages and locales. 25

2.9 The different locales. 26

2.10 Setting the product language for the universe design tool user interface. 26

2.11 Consuming multilingual universes. 26

2.12 Determining the fallback locale in linked universes. 27

2.13 The translation management tool. 27

2.14 Multilingual data. 27

2.15 Universe design tool example materials. 27

Demonstration databases. 28

Demonstration universes. 28

2.16 Using universes with the information design tool. 28

3 Doing basic operations. 29

3.1 Overview. 29

3.2 Starting the universe design tool. . . . 29
To start the universe design tool. . . . 30
Using the Quick Design wizard 30
3.3 Creating a basic universe with the Quick Design wizard. . . . 31
Why use the Quick Design wizard?. . . . 31
Using the Quick Design Wizard. . . . 31
Following up on a universe created with the Quick. . . . 39
3.4 Importing a universe. . . . 39
Importing a universe from the repository. . . . 39
What is the difference between opening and importing?. . . . 40
3.5 Opening a universe. . . . 40
To open a universe directly. . . . 40
3.6 Exporting a universe. . . . 41
How are universes organized on the repository file system?. . . . 41
Exporting a universe to the repository. . . . 41
What is the difference between exporting and saving?. . . . 42
3.7 Saving a universe. . . . 42
Universe file names as identifiers. . . . 43
Saving a universe. . . . 43
Saving a universe definition as PDF. . . . 43
3.8 Closing a universe. . . . 44
3.9 Working with multiple designers. . . . 45
Locking a universe. . . . 45
Revision number. . . . 45
3.10 Using the universe design tool user interface. . . . 45
The main components of the user interface. . . . 46
The universe design tool user interface. . . . 46
Manipulating windows 47
Using toolbars. . . . 47
Performing an action or operation in the universe design tool. . . . 49
3.11 Using Find and Replace. . . . 50
Using Find. . . . 50
Using Quick Find. . . . 53
3.12 Organizing the table display. . . . 53
How are tables represented?. . . . 53
Manipulating tables. . . . 54
Using List mode. . . . 54
Arranging tables automatically. . . . 55
Changing table display. . . . 56
3.13 Selecting schema display options. . . . 57
Setting graphic options for the Structure pane display. . . . 58

Viewing table and column values. . . . 59
Viewing the number of rows in database tables. . . . 62
3.14 Printing a universe. . . . 65
Setting print options. . . . 65

4 Creating a universe and setting the universe parameters. . . . 69
4.1 What are universe parameters?. . . . 69
4.2 Creating a new universe. . . . 70
Creating a new universe from scratch. . . . 70
4.3 Viewing and entering summary information. . . . 71
4.4 Setting universe parameters. . . . 72
Identifying the universe . . . 72
Defining and editing connections. . . . 74
Setting universe summary parameters. . . . 82
Selecting strategies. . . . 83
Indicating resource controls. . . . 87
What system resource options are available?. . . . 87
To enter resource control information. . . . 87
Limiting execution time for queries generating more than one SQL statement . . . 88
Indicating SQL restrictions. . . . 88
Indicating options for linked universes. . . . 90
Setting SQL generation parameters. . . . 90
About SQL Generation Parameters. . . . 92
Editing dynamic SQL generation parameters. . . . 93
SQL Parameters that you set in the user interface. . . . 93
SQL Parameters that you set in the PRM files. . . . 109

5 Creating a schema with tables and joins. . . . 123
5.1 Overview. . . . 123
5.2 What is a schema?. . . . 123
Schema design is the basis for a successful universe. . . . 124
Schema design and the universe creation process. . . . 124
What are the stages of schema design?. . . . 124
5.3 Inserting tables. . . . 124
Using the Table Browser. . . . 125
Arranging Tables in the Structure Pane. . . . 128
5.4 Using derived tables. . . . 128
Adding, editing, and deleting derived tables. . . . 129
5.5 Nested derived tables. . . . 131
Using the Derived Tables editor. . . . 131
To create a nested derived table. . . . 132
Renaming nested derived tables. . . . 132

5.6 Using tables that have input columns. . . . 133
To define a hard-coded list of values. . . . 133
To define a list of values for the user to enter or select. . . . 134
5.7 Defining joins. . . . 134
What is a join?. . . . 134
Why use joins in a schema?. . . . 135
What SQL does a join Infer?. . . . 135
What tables do not have to be joined?. . . . 136
Joining primary and foreign keys. . . . 136
Understanding the cardinality of a join. . . . 137
Creating joins. . . . 137
Join properties. . . . 142
Editing a join. . . . 144
ANSI 92 support for joins in a universe. . . . 147
Deleting joins. . . . 151
5.8 Defining specific types of joins. . . . 151
Creating Equi-joins. . . . 152
Theta joins. . . . 156
Outer joins. . . . 159
Shortcut joins. . . . 163
Self restricting joins. . . . 164
5.9 Using cardinalities. . . . 167
How are cardinalities used in the universe design tool?. . . . 167
Setting cardinalities manually. . . . 169
5.10 Checking the universe. . . . 176
Checking Universe Integrity Automatically. . . . 177

6 Resolving join problems in a schema. . . . 183
6.1 Overview. . . . 183
6.2 What is a join path problem?. . . . 183
What is a Lookup Table. . . . 183
What is a Fact Table. . . . 184
What Types of Join Paths Return Incorrect Results?. . . . 184
Detecting and Solving Join Problems. . . . 184
6.3 Defining aliases. . . . 185
How are Aliases Used in a Schema?. . . . 185
Creating Aliases. . . . 186
6.4 Defining contexts. . . . 189
How are contexts used in a Schema?. . . . 189
Creating a context. . . . 190
Editing a context. . . . 193
Deleting a context. . . . 194

Updating contexts. . . . 194
Join Paths that Prevent Context Detection. . . . 194
How do Contexts Affect Queries?. . . . 196
6.5 Resolving loops. . . . 199
What is a Loop?. . . . 199
How does a loop affect queries?. . . . 200
Visually identifying loops. . . . 208
Automatically Identifying and Resolving Loops. . . . 208
Tool features to detect and resolve loops. . . . 208
Examples of resolving loops. . . . 216
6.6 Resolving chasm traps. . . . 225
What is a chasm trap?. . . . 226
How does a chasm trap inflate results?. . . . 227
Detecting a chasm trap. . . . 229
Resolving a chasm trap. . . . 229
6.7 Resolving Fan Traps. . . . 231
What is a Fan Trap?. . . . 231
How do you detect a fan trap?. . . . 234
How do you resolve a fan trap. . . . 234
6.8 Detecting join problems graphically. . . . 237
Potential chasm trap. . . . 238
Potential fan trap. . . . 239
6.9 Checking the universe. . . . 240
Types of errors detected by Check Integrity. . . . 240
Checking universe integrity manually. . . . 242
Refreshing the Universe Structure. . . . 245

7 Creating universes. . . . 246
7.1 Overview. . . . 246
7.2 Introduction to universe building. . . . 246
What is an object?. . . . 247
What types of objects are used in a universe?. . . . 247
Using classes and objects. . . . 248
What is a class?. . . . 248
7.3 Using the Universe pane. . . . 248
Displaying classes and objects or conditions. . . . 249
7.4 Basic operations on classes, objects, and conditions. . . . 250
Cut, copy, paste. . . . 250
Moving classes, objects, or conditions. . . . 250
Showing or hiding classes, objects and conditions. . . . 250
7.5 Defining classes. . . . 251
Creating a class. . . . 251

Class properties. . . . 253
Modifying a class. . . . 253
Using subclasses. . . . 253
7.6 Defining objects. . . . 254
Creating an object. . . . 254
Object properties. . . . 256
Modifying an object. . . . 257
Object definition. . . . 257
Properties. . . . 260
Advanced. . . . 261
Defining index awareness. . . . 262
Source Information. . . . 266
Using the SQL editor to define an object. . . . 267
Defining an object format. . . . 269
Viewing the table used in an object definition. . . . 270
Defining a dimension. . . . 271
Defining a detail. . . . 271
Defining a measure. . . . 272
Defining restrictions for an object. . . . 277
Defining condition objects. . . . 281
Using self restricting joins to apply restrictions. . . . 287
Applying a restriction by inferring multiple tables. . . . 287
Concatenating objects. . . . 289
7.7 Defining hierarchies. . . . 291
What is multidimensional analysis?. . . . 291
How to identify a hierarchy. . . . 291
Setting up hierarchies. . . . 292
7.8 Using cascading lists of values for hierarchies. . . . 295
Creating a cascading list of values. . . . 296
7.9 Using lists of values. . . . 298
How is a list of values used?. . . . 298
Defining how a list of values is used with an object. . . . 300
List of values properties and options. . . . 301
Editing a list of values. . . . 304
Exporting a list of values. . . . 308
Refreshing values in a list of values. . . . 310
Using data from a personal data file. . . . 310
Administering lists of values in the universe. . . . 312
Optimizing and customizing LOV files. . . . 313
7.10 Linking universes. . . . 314
What are linked universes?. . . . 314

Different ways to link universes. . . . 316
Advantages of linking universes. . . . 317
Requirements for linking universes. . . . 318
Restrictions when linking universes. . . . 318
Creating a link between two universes. . . . 318
Editing a derived universe. . . . 322
Removing a link. . . . 322
Relocating the core universe. . . . 323
Derived universes and lists of values. . . . 323
Presenting objects in the order of the core universe. . . . 323
7.11 Including one universe within another. . . . 324
Copying a core universe into a derived universe. . . . 324
7.12 Creating stored procedure universes. . . . 324
Stored procedures in Java bean universes. . . . 325
Creating a universe based on stored procedures. . . . 325
7.13 Testing the universe. . . . 329
Testing objects in the Query Panel. . . . 329
Testing the integrity of the universe. . . . 330
Testing the universe with Web Intelligence. . . . 330

8 Optimizing universes. . . . 331
8.1 Overview. . . . 331
8.2 Using aggregate tables. . . . 331
What is aggregate awareness?. . . . 331
Applying aggregate awareness to Data Warehouses. . . . 332
Setting up aggregate awareness. . . . 332
Building the objects. . . . 333
Identifying all combinations of aggregate objects. . . . 334
Arranging objects in aggregate order. . . . 334
Defining aggregate objects with the @Aggregate_Aware function. . . . 334
Specifying the incompatible objects. . . . 337
Specifying incompatible objects. . . . 339
Resolving loops involving aggregate tables. . . . 342
Testing aggregate awareness. . . . 344
8.3 Using @Functions in the SQL of an object. . . . 344
Inserting an `@Function` in an object. . . . 346
@Aggregate_Aware. . . . 347
@Prompt. . . . 348
@Select. . . . 361
@Variable. . . . 362
@Where. . . . 369
8.4 Using external strategies to customize universe creation. . . . 371

Migrating external strategies to the universe design tool. 371
External strategies overview. 372
What is an external strategy?. 372
Creating Help text for external strategies. 373
Verifying that the external strategy file is declared. 375
Using example external strategies. 375
How is the strategy file (STG) structured?. 376
The output formats of strategies. 378
Creating an external strategy. 381
Creating a text file for data. 382
Applying external strategies in the universe design tool. 382
8.5 Using analytic functions. 384
What are analytic functions?. 384
What are the advantages of using analytic functions?. 384
Which analytic function families are supported?. 384
How are analytic functions used in the universe design tool?. 385
IBM DB2 UDB and Oracle. 385
RedBrick (RISQL functions). 389
Teradata (OLAP functions). 391
Inserting syntax automatically in Select statements. 393
8.6 Using the SQL prefix function. 395
To prefix SQL statements with the BEGIN_SQL universe parameter. 395
8.7 Optimizing the array fetch parameter. 396
Modifying the array fetch parameter. 396
8.8 Allocating table weights. 396
Modifying the PRM file to allocate table weights 397
8.9 Modifying the number of returned rows for a table 397
8.10 Using shortcut joins. 398

9 Working with OLAP universes. 399
9.1 About OLAP universes. 399
What is an OLAP universe?. 399
Which OLAP data sources can be used to create a universe?. 400
9.2 Defining connections to OLAP data sources. 406
About connections to OLAP data sources. 406
To start the New Connection wizard. 407
To select database middleware for an OLAP connection. 407
Login parameters for relational connections - UDT. 408
Login parameters for SAP BW OLAP connections. 410
Login parameters for MSAS OLAP connections 411
Defining login parameters for Essbase connections. 412
To select source cube or query for OLAP connections. 412

To define configuration parameters for OLAP connections. . . . 413
Defining custom parameters for Essbase connections. . . . 414
9.3 Customizing OLAP universes. . . . 414
Creating OLAP Universes with additional parameters. . . . 414
Defining OLAP options for your OLAP universe. . . . 415
Defining objects in OLAP universes. . . . 416
Universe design tool features supported for OLAP universes. . . . 416
Database delegated projection function. . . . 417
Setting delegated measures for OLAP universes. . . . 419
Setting aggregate projection for a measure. . . . 419
Calculated measures in OLAP universes. . . . 420
About MDX functions for cube queries. . . . 422
XML syntax for filter and WHERE statements. . . . 422
Predefined conditions in OLAP universes. . . . 423
Optional prompts in OLAP universes. . . . 428
To improve performance of certain queries on SAP BW universes. . . . 428
9.4 OLAP universe lifecycle management. . . . 429
About OLAP universe lifecycle management. . . . 429
Overview: relationship between universe object status and OLAP object status 430
To refresh an OLAP universe. . . . 432
Regenerating Level 00 for OLAP universes. . . . 434
Renaming level L00 to ALL. . . . 435
Replacing OLAP universe level prefixes. . . . 435
Synchronizing the universe and the OLAP cube. . . . 435
How dimensions are managed in OLAP universe updates. . . . 436
How hierarchies or characteristics are managed in OLAP universe updates 440
How levels are managed in OLAP universe updates 447
How SAP variables are managed in OLAP universe updates 450
How key figures or measures are managed in OLAP universe updates 453
How SAP key dates are managed in OLAP universe updates 457
9.5 How the different OLAP cubes are mapped to universes. . . . 459
How SAP BW objects are mapped and used in a universe. . . . 459
How Essbase cubes are mapped to universe components. . . . 467
How MSAS cubes are mapped to universe components 468

10 Deploying universes. . . . 469
10.1 Overview. . . . 469
10.2 How are universes deployed?. . . . 469
Identifying a universe in the repository. . . . 469
10.3 Giving all users access to a universe. . . . 470
10.4 Setting access restrictions on a universe. . . . 470
What is a restriction?. . . . 471

What restrictions can be applied in a universe? . . . 471
How are access restrictions managed? . . . 472
Creating a restriction . . . 473
Applying universe access restrictions . . . 475
Adding a user group to the list of available users for a universe . . . 476
Setting restriction group priority . . . 477
Viewing users and groups security restrictions . . . 478
10.5 Managing users and logins . . . 479
Managing logins . . . 479
Managing passwords . . . 480

11 Using the sample materials . . . 482
11.1 Overview . . . 482
11.2 The Club database . . . 482
The structure of the tables . . . 482

1 Document History

The following table provides an overview of the most important document changes.

Version	Date	Description
SAP Business Objects Universe Design Tool 4.3	June, 2020	Initial release

2 Introducing the Universe Design Tool

2.1 Overview

This chapter gives you a general introduction to the universe design tool, the tool you use to build universes. It describes universes, what they contain, how they are created, and the role that universes have in your business environment.

The typical universe development cycle is described, with best design practices recommended. The demonstration databases and universes shipped with this release are also described.

This chapter introduces the universe design tool, the devlopment process and the different language possibilities for the universe. The following topics are covered:

Related Information

Universe design tool and universe fundamentals [page 13]
How do you use the universe design tool to create universes? [page 17]
Who is the universe designer? [page 19]
Introducing the universe development process [page 21]
Multilingual universes [page 24]

2.2 Universe design tool and universe fundamentals

BusinessObjects Universe Design tool allows you to create universes for users using BI clients such as Web Intelligence.

2.2.1 What is a universe?

A universe is a file that contains the following:

- Connection parameters for one or more database middleware.
- SQL structures called objects that map to actual SQL structures in the database such as columns, tables, and database functions. Objects are grouped into classes. Objects and classes are both visible to Web Intelligence users.
- A schema of the tables and joins used in the database. Objects are built from the database structures that you include in your schema. The schema is only available to universe design tool users. It is not visible to the users using BI client tools such as Web Intelligence.

Web Intelligence users connect to a universe, and run queries against a database. They can do data analysis and create reports using the objects in a universe, without seeing, or having to know anything about, the underlying data structures in the database.

2.2.2 What is the role of a universe?

The role of a universe is to provide an easy to use and understand interface for non technical Web Intelligence users to run queries against a database to create reports and perform data analysis.

As the universe designer, you use the universe design tool to create objects that represent database structures, for example columns and database functions, that users need to access and query, to get the information necessary to meet their business requirements.

The objects that you create in the universe must be relevant to the end user business environment and vocabulary. Their role is to present a business focussed front end to the SQL structures in the database.

2.2.3 What does a universe contain?

A universe contains the following structures:

- Classes
- Objects

2.2.3.1 Classes

A class is a logical grouping of objects within a universe. It represents a category of objects. The name of a class should indicate the category of the objects that it contains. A class can be divided hierarchically into subclasses.

2.2.3.2 Objects

An object is a named component that maps to data or a derivation of data in the database. The name of an object should be drawn from the business vocabulary of the targeted user group. For example, objects used in a universe used by a product manager could be Product, Life Cycle, or Release Date. A universe used by a financial analyst could contain objects such as Profit Margin or Return on Investment.

2.2.3.3 Types of objects

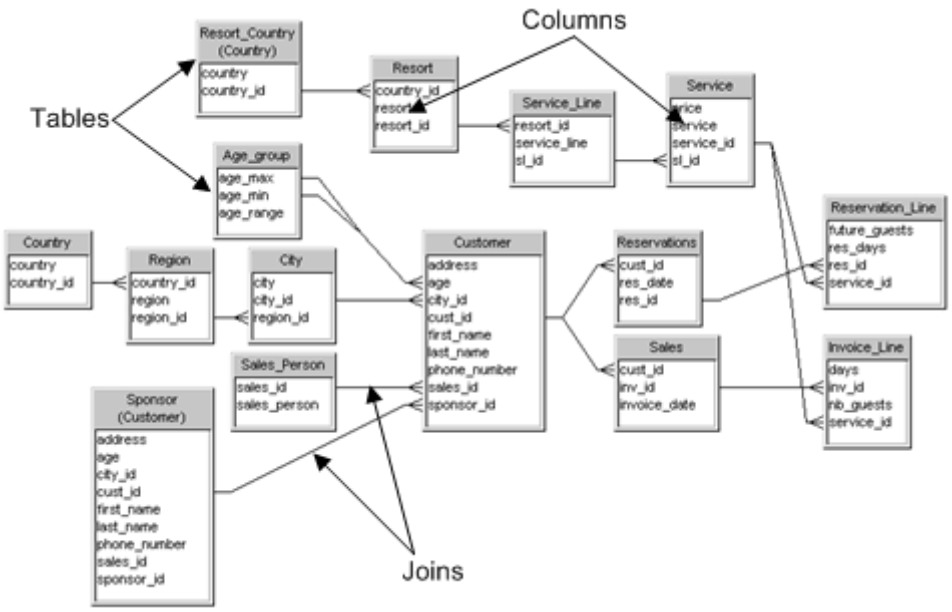
In the universe design tool, objects are qualified as one of three types: dimension, detail, or measure.

Object type	Description
Dimension	Parameters for analysis. Dimensions typically relate to a hierarchy such as geography, product, or time. For example. Last Name and City_Id
Detail	Provide a description of a dimension, but are not the focus for analysis. For example. Phone Number
Measure	Convey numeric information which is used to quantify a dimension object. For example, Sales Revenue

2.2.3.4 Objects infer SQL structures displayed in a schema

The objects that Web Intelligence users see in a universe infer SQL structures that you have inserted into a database schema. You, as the universe designer, create this schema based on the tables and joins that are required to return the data, needed by users for their analysis and report creation.

The schema is a part of the universe file, but is only visible and accessible in the universe design tool. You create the schema in the *Structure pane* of the *Universe window*. A schema is shown below for the sample universe Beach.unv.

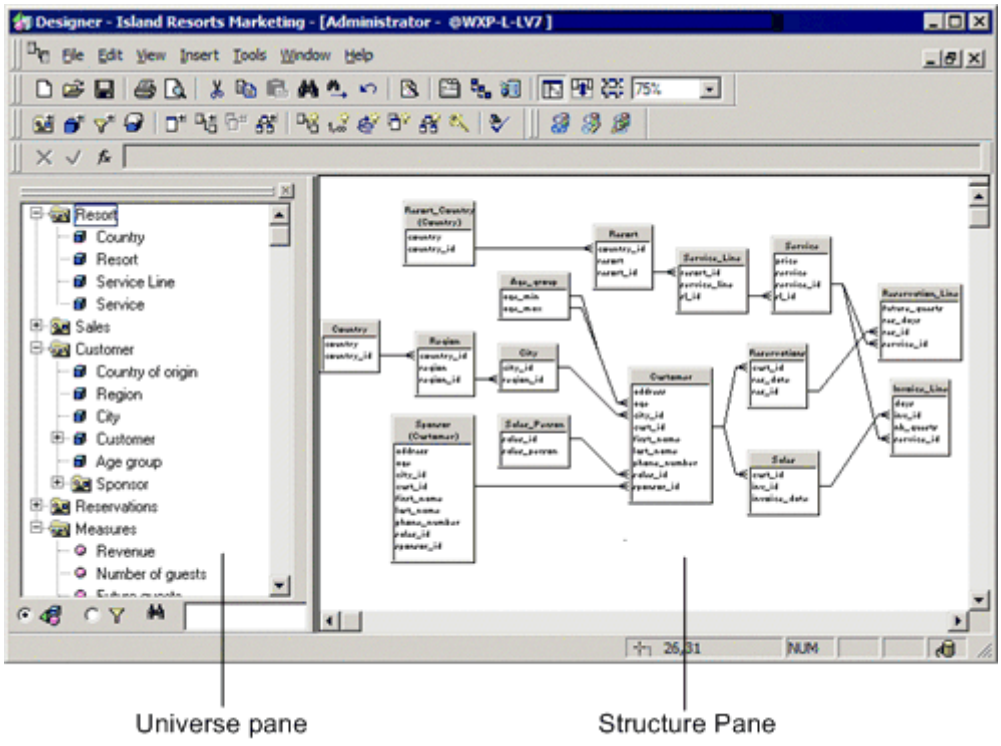


2.2.3.5 How are objects presented in a universe?

Objects are displayed as nodes in a *Tree explorer* view in the *Universe pane*. You use the object explorer to create, delete, copy, view, and move classes and objects.

2.2.4 About the universe window

The *Universe window* in the universe design tool is shown below. It contains both the *Universe* pane (also visible in Web Intelligence) and the *Structure pane* (visible only in the universe design tool).



2.2.5 Universe design tool install root path

In this guide the variable $INSTALLDIR is the install root path for the data access files used by the universe design tool and Web Intelligence. This is the Business Objects installation path with the operating system sub directory that contains the universe design tool executable and the data access drivers.

Under Windows, $INSTALLDIR = `\\...\Business Objects\BusinessObjects Enterprise 12.0\win32_x86`.

For example `C:\Program Files\Business Objects\BusinessObjects Enterprise 12.0\win32_x86`.

2.3 How do you use the universe design tool to create universes?

The universe design tool provides a connection wizard that allows you to connect to your database middleware. You can create multiple connections with the tool, but only one connection can be defined for each universe. This database connection is saved with the universe.

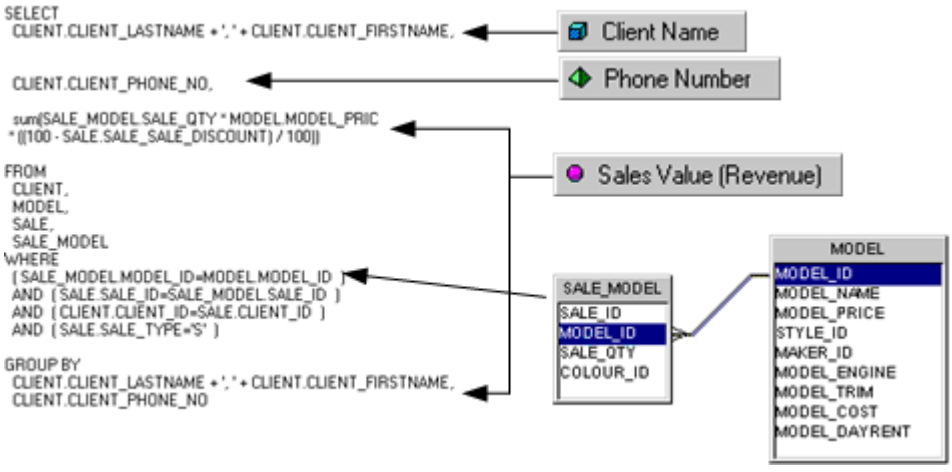
The universe design tool provides a graphical interface that allows you to select and view tables in a database. The database tables are represented as table symbols in a schema diagram. You can use this interface to manipulate tables, create joins that link the tables, create alias tables, contexts, and solve loops in your schema. Web Intelligence users do not see this schema.

The universe design tool provides an object explorer view, the *Tree explorer*. You use the *Tree explorer* to create objects that map to the columns and SQL structures that are represented in the schema view. Web Intelligence users manipulate these objects to run queries against a database.

The universe design tool allows you to distribute universes by importing and exporting universes to the Central Management System (CMS) repository.

2.3.1 How do objects generate SQL?

Web Intelligence users create queries by dragging objects into the *Query* work area. The definition of each object infers a SELECT statement. When a query is run, a SELECT statement and optional WHERE clause for all the objects is run against the target database.

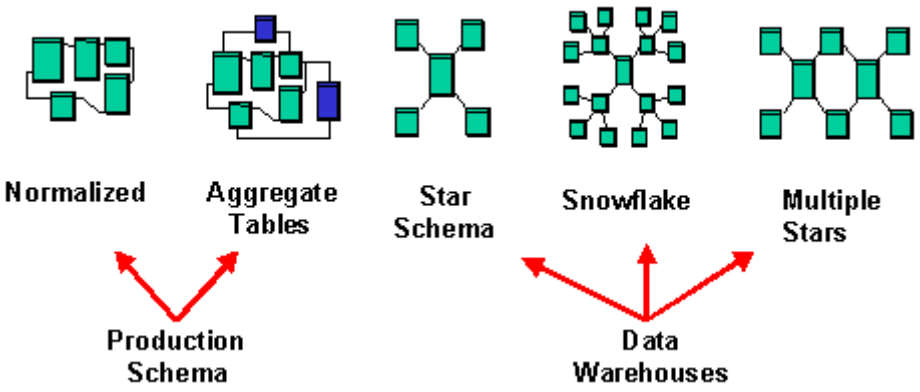


When a user chooses to include dimension and/or detail objects with a measure object in the *Query* work area, a GROUP BY clause containing the content of those dimension and detail objects is automatically added to the SELECT statement.

The tables that are included in the FROM clause and the Joins in the WHERE clause, are inferred from the table schema that you build in the *Structure* pane.

2.3.2 What types of database schema are supported?

The universe design tool can support most types of database schema, including all those shown below. You do not need to redefine or optimize your database before using the universe design tool.



2.3.3 How are universes used?

Universes are used by Web Intelligence users. The universes are stored in the Central Management System (CMS) repository. An end user connects to a universe from a web browser.

The connection to the database is defined in the universe, so by connecting to the universe, the end user automatically has access to the data. The access to data is in turn restricted by the objects that are available in the universe. These objects have been created by you, the universe designer, based on the user needs profile for a defined user group.

2.3.3.1 Representing a targeted data need

A universe can represent the data needs of any specific application, system, or group of users. For example, a universe can contain objects that represent the data needs of the Marketing or Accounting departments in a company.

A universe can also represent the data needs of a section within a department or any set of organized procedures such as a payroll or inventory system.

Examples of classes are Employee Information, Attendance Information, and Department Information.

2.3.3.2 Universes and the database schema

The database schema is used to create three universes; PERSONNEL, INVENTORY, and SALES. Each universe contains classes and objects. Each object maps to a part of the database structure.

2.3.3.3 Who uses universes?

Web Intelligence users use universes for reporting and analysis. The universe should provide them with classes and objects relevant to their business domain.

2.4 Who is the universe designer?

Universes are created by a universe designer using the universe design tool. There is no standard profile for a universe designer. Within a company, the person designated as the universe designer may be the database administrator, an applications manager or developer, a project manager, or a report creator who has acquired enough technical skills to create universes for other users.

There can be more than one universe designer in a company. The number of universe designers depends on the company's data requirements. For example, one universe designer could be appointed for each application, project, department or functional area.

When several people create universes, it is important to define a set of rules or guidelines for terminology, so that the objects are represented consistently.

2.4.1 Required skills and knowledge

A universe designer should have the following skills and level of technical knowledge:

Skill/Knowledge	Description
Ability to analyze user needs	Universes are created to meet a user need for data. The universe designer must have the skills to conduct user needs analyses to create classes and objects that are relevant to the user vocabulary, and to develop universes that meet the needs of the user community. These needs include report creation and query results that are suitable for analysis
Database knowledge	Universe designer needs to have a good working knowledge of the company's database management system (DBMS), how the databases are deployed, the logical database structure, and the type of data stored in company databases

Skill/Knowledge	Description
Stuctured Query Language (SQL)	A working knowledge of SQL is necessary

2.4.2 What are the tasks of the universe designer?

The universe designer is normally responsible for the following tasks:

- Conducting user needs analysis
- Designing and creating the universe
- Distributing the universe
- Maintaining the universe

2.5 The basic steps to create a universe

The universe is the part of the .unv file that is used by end-users who use reporting tools to create reports. It is the only part if the .unv file that is visible to the end-users. When you create a universe, you perform the following steps:

1. Create a new (empty) universe file and set the connection and parameter details.
2. Create a schema of the database tables that you want to use, and define the joins between tables. The schema is not visible to end-users who use reporting tools to create reports.
3. Resolve any join problems in the schema.
4. Create the classes and objects that make up the universe. The universe is used by end-users who use reporting tools to create reports.

Related Information

Doing basic operations [page 29]
Creating a universe and setting the universe parameters [page 69]
Creating a schema with tables and joins [page 123]
Resolving join problems in a schema [page 183]
Creating universes [page 246]
Optimizing universes [page 331]

2.6 Introducing the universe development process

The following sections give an overview of how you manually create a universe, and describe how universe creation fits into a typical universe development cycle.

2.6.1 Universe design methodology

The universe design methodology described in this manual consists of one planning stage, and three implementation phases:

- Analysis of business problem and planning the universe solution
- Designing a schema
- Building the universe
- Distributing the universe to users

Each implementation phase is based on an assumption that you have completed an initial planning phase. The planning phase can be done without using the universe design tool, and is the decisive phase for the success or failure of your universe. A poorly planned universe that is not based on a study of user reporting needs will be difficult to design, implement, maintain, and will not be useful to your target users.

Each of these phases is described as follows:

2.6.1.1 Plan the universe before you start using the universe design tool

Before starting the first phase, you should spend up to eighty percent of the time allotted for the universe creation project, planning the universe. You should note the following points:

- You must analyze the data analysis and reporting needs of the target audience for the universe. The structures that you use to create the schema should be based on a clearly defined user need to access the data contained in those tables and columns.
- You should have a clear idea of the objects that you need to create before you start using the universe design tool. Do not create objects by looking at the columns available in the database, but identify columns that match an object that you have already identified from your user needs analysis.

2.6.1.2 Designing a schema

You create a schema for the underlying database structure of your universe. This schema includes the tables and columns of the target database and the joins by which they are linked. You may need to resolve join problems such as loops, chasm traps, and fan traps, which may occur in the structure by using aliases or contexts. You test the integrity of the overall structure. In this guide, the designing a schema phase is described in the chapters Creating a schema with tables and joins [page 123] and Resolving join problems in a schema [page 183].

2.6.1.3 Building the universe

You create the objects that infer SELECT statements based on the components of your schema. You organize these objects into classes. These are objects that you have identified from an analysis of user reporting needs. You can create many types of objects to enhance user reporting capabilities, multidimensional analysis, and optimize query performance.

You test the integrity of your universe structure. You should also perform tests by running reports in Web Intelligence.

The building phase is described in the chapter Creating universes [page 246].

2.6.1.4 Distributing the universe

You can distribute your universes to users for testing, and eventually for production, by exporting them to the Central Management System (CMS) repository. This phase is described in the chapter Deploying universes [page 469].

2.6.2 Universe development cycle

Universe development is a cyclic process which includes planning, designing, building, distribution, and maintenance phases. You use the universe design tool to design and build a universe, however, the usability of any universe is directly related to how successfully the other phases in the development cycle interact with each other.

This section presents an overview of a universe design methodology that you can use to plan and implement a universe development project.

The table below outlines the major phases in a typical universe development cycle:

Development phase	Description
Prepare	• Identify the target data source and become familiar with its structure. • Know what data is contained within each table of each of the target databases. • Understand the joins. • Identify the cardinality. • Know what is possible.

Development phase	Description
Analyze	• Identify the user population and how it is structured; for example is the user group structured by department or by task. • Identify what information the users need. • Identify what standard reports they require. • Familiarize yourself with their business terminology so that you can name objects sensibly.
Plan	Identify a project strategy. For example, how many universes should be created and which ones should have the capacity to be linked and to what level.
Implement	• Build the universe using the tool. This manual covers this part of the universe development cycle, the actual use of the universe design tool. • Test frequently during the build process for validity and reliability of inferred SQL.
Test	Form a small group of Web Intelligence power users who have some knowledge of what information they expect to get from the universe. Ask the users to perform thorough tests simulating live usage of the universe(s).
Deploy	Distribute the universe by exporting universe to the Central Management System (CMS) repository, where it can be accessed by end users.
Evolve	Update and maintain the universe as the data sources and user requirements change and grow.

> **i** Note
>
> Universe design should always be driven primarily by user requirements and NOT the data source structure.

2.6.3 Optimizing universe planning and implementation time

The analysis of user requirements and design are the most important stages in the process. Users must be heavily involved in the development process if the universe is going to fulfil their needs both with the business language used to name objects and the data that can be accessed.

Implementation will be very quick and easy if the first three stages are carried out properly.

You can spend up to 80% of the time allocated to the development of a universe on the first three stages:

- Preparing

- Analyzing
- Planning

If you have spent the time in the laying the foundation for your universe, the other 20% of the time spent actually using the universe design tool to build your universe will be much more productive than if you have not spent the necessary time in planning and analysis.

2.7 Multilingual universes

One of the key features of SAP BusinessObjects Enterprise XI4 is the ability to produce multilingual metadata and reports from the same universe. This feature enables the user to have a one-step multilingual reporting solution that is locale sensitive, supported by a single metadata universe model, and provides full Unicode support. Reports can then be built once from the same universe and displayed in several languages based on user preferences.

The universe metadata that can be translated are:

- Universe name
- Universe description
- Class names
- Object names
- Object names, descriptions and formats
- Custom hierarchies names
- Prompt and input column questions

> **i** Note
>
> Only prompts defined in the universe metadata can be translated. Prompts defined using the @Prompt function cannot be translated.

A universe may contain translations in several locales. When creating reports on top of the universe, the metadata are displayed in a locale that depends on the user's Preferred Viewing Locale.

A universe also defines a fallback locale that is a locale to use if no locale is available.

Only locales that have their status set to *Ready for use* can be seen by the user that creates reports on top of the universe.

In these visible locales, only metadata whose status is in the *Translation Visible* category can be seen by the user who creates reports on top of the universe. The *Translation Visible* category contains metadata that have the following statuses:

- NEEDS_REVIEW_TRANSLATION
- NEEDS_REVIEW_LOCALIZATION
- NEEDS_REVIEW_ADAPTATION
- TRANSLATED
- FINAL
- SIGNED_OFF

Translating universe metadata, setting the universe locales, and and metadata statuses are done through the translation management tool tool. Translations and locales parameters are stored in an additional XML stream and saved in the .unv file format.

The universe design tool user interface can also be displayed in different languages. The following text describes the multilingual universe features of the tool.

2.8 Definitions of languages and locales

Some languages are associated to several countries. For example French (fr) is a language spoken in the following countries: France (FR), Belgium (BE) and Switzerland (CH). In this example fr-FR, fr-BE, fr-CH means French (fr) as the language in France (FR), in Belgium (BE) and in Switzerland (CH).

Language	Country
French	France
French	Belgium
French	Switzerland

At the same time a country can be associated to several languages (fr-CH, de-CH, it-CH). For instance, in Switzerland German, French and Italian are all spoken.

Language	Country
French	Switzerland
German	Switzerland
Italian	Switzerland

A locale defines the combination of a language and a geographical area, and the way data is sorted. Dates and times are often formatted in specific formats. When you combine language and country (for example, French language in France), the locale appears as follows depending on Operating System or applications:

Operating System	Locale format
Windows	Français (France) Where the locale is taken from your system settings (country)
Java	fr_FR
Sun Solaris	fr_FR.ISO8859-1

To simplify the documentation and the application, the same term "language" may be used for both language and locale meanings.

2.9 The different locales

Terms	Definition
Product language	This is the universe design tool user interface language. The menu and messages appear in that language.
Preferred viewing locale (PVL)	Your preferred viewing language settings. This defines the locale in which strings, text and format - the parts of the resource (document or universe) content or attribute list - appear in the application on InfoView or Web Intelligence Rich Client.
Substitution (Fallback) locale	The locale used when your preferred viewing locale is not available.
Source language	The locale in which the document has been created.

2.10 Setting the product language for the universe design tool user interface

In the General tab of the *Tools* > *Options* settings of the universe design tool, choose a *Language* from the list of available languages, this is the product language, also known as the User Interface Language (UIL). There is no impact on the universe metadata: object names, context names, and classes appear in the original language of the database elements. To translate the universe metadata, use the translation management tool.

2.11 Consuming multilingual universes

When a user creates a report on top of a multilingual universe, in Web Intelligence for example, the locale in which metadata is displayed depends in the locale's availability/status and the metadata category:

- A metadata translation in a locale is displayed if and only if its status is in the Visible category and this locale has been defined as Ready for use.
- The locale in which to display the translation metadata is, by priority order:
  - The user's Preferred Viewing Locale.
  - If the Preferred Viewing Locale is not available, the fallback locale if it has been defined in this universe.
  - If no fallback locale has been defined in this universe, then the dominant locale of the user's Preferred Viewing Locale.
  - If it is not available, the original content is displayed. This original content is the metadata as it is defined in the universe design tool.

2.12 Determining the fallback locale in linked universes

A derived universe may re-use metadata from different core universes. When the derived universe and the core universes define different fallback locales:

- If a fallback locale is defined at derived universe level, then use this fallback locale.
- If no fallback locale is defined at derived universe level, then use the fallback locale of the first core universe defined in the derived universe, if any.
- If no fallback locale has been defined in any core universes, then the universe has no fallback locale.

2.13 The translation management tool

The universe design tool cannot be used to translate universe metadata or even display metadata translations. In the universe design tool, only the original universe content is displayed. For translation tasks, the BusinessObjects Enterprise suite provides the translation management tool. This tool is a stand-alone application available only on Windows platforms.

Through the translation management tool, the person designing the universe can:

- Add new locales in a universe and set them as *Ready to Use*.
- Define the universe's fallback locale.
- Translate universe metadata in the added locales through the interface itself.
- Set the status of the translation of the metadata in the different locales.
- Export universe metadata into XLIFF files for external translation and import them back into the application.

Once the universe metadata has been translated, it can be re-saved and multilingual reports may benefit from these translations. Refer to the *Translation management tool* user guide for more information.

2.14 Multilingual data

Through the PREFERRED_VIEWING_LOCALE and the DOMINANT_PREFERRED_VIEWING_LOCALE variables, the designer can customize the universe in order to filter multilingual data and retrieve only data in the user's Preferred Viewing Locale at query time. This can be done by using the @Variable function.

2.15 Universe design tool example materials

The following samples are shipped with the universe design tool:

2.15.1 Demonstration databases

Most of the examples in this guide are based on the Club database built with Microsoft Access 2000. This database is used by the sales manager of the fictitious business, Island Resorts, to perform sales and marketing analysis. You can find the database file, Club.mdb, in the Databases subfolder in the Business Objects installation path.

For more information on the structure of this database, refer to the appendix at the back of this guide.

The efashion database is also shipped with this release. This MS Access 2000 database tracks 211 products (663 product color variations), sold over 13 stores (12 US, 1 in Canada), over 3 years.

The database contains:

- A central fact table with 89,000 rows of sales information on a weekly basis.
- A second fact table containing promotions.
- Two aggregate tables which were set up with aggregate navigation.

2.15.2 Demonstration universes

A complete demo universe called beach.unv is delivered in the Universes subfolder of the Samples folder in the Business Objects installation path. It was built with the Club database described above.

You can use this universe to learn how to build specific objects and classes with the universe design tool.

The universe design tool also comes with the efashion universe built using the efashion database.

2.16 Using universes with the information design tool

You can use the information design tool to work with .unv format universes that were created by the universe design tool, universe design tool desktop edition, Universe Designer or Universe Designer Personal. These files cannot be used directly by the information design tool, you must first convert them, or upgrade and then convert them, depending on the version of the file. See the information design tool for information about the steps to perform in order to use different versions of .unv universe files, and which features are supported after the files are converted.

> **i** Note
>
> Once a .unv file has been converted for use with the information design tool, the file cannot be opened with the tool that was originally used to create it.

3 Doing basic operations

3.1 Overview

This chapter describes the basic operations you perform in the universe design tool to create, modify, and update universes. The following topics are covered:

- Starting the universe design tool [page 29]
- Importing a universe [page 39]
- Opening a universe [page 40]
- Exporting a universe [page 41]
- Saving a universe [page 42]
- Creating a universe and setting the universe parameters [page 69]
- Using the universe design tool user interface [page 45]
- Using Find and Replace [page 50]
- Organizing the table display [page 53]
- Selecting schema display options [page 57]
- Printing a universe [page 65]

3.2 Starting the universe design tool

The universe design tool can only be used with a Central Management System (CMS) repository. You must log in to the repository before starting the tool.

If you are starting the tool for the first time and want to work on an existing universe, you need to open the universe directly first, save it with a secure connection, and export it to the repository. You then import the universe to make updates and export updated versions. This ensures that the CMS and the local universe versions are synchronized.

Once you start the universe design tool you can open a universe in one of the following ways:

- Create a new universe
- Import a universe from the CMS repository
- Open a universe directly from the file system

A universe is only available to Web Intelligence users when it is exported to the repository. Importing a universe, making changes, then exporting the updated universe is the most common way of working with the universe design tool. It ensures that the CMS (repository) version is synchronized with the file version.

> **i** Note
>
> You can save a universe to the file system. You do this when you want to share the universe with other users who may not have a connection rights to the target CMS. See the section Saving a universe [page 42] for more information.

You start the universe design tool from the task bar by clicking the tool icon in the group of installed Business Objects products for this release. You are prompted to log into the CMS before the tool starts.

3.2.1 To start the universe design tool

To start the universe design tool :

1. Click the *Start* button on the taskbar.
2. Point to the *Programs* menu.
3. Click the *universe design tool* program from the *BusinessObjects* command.
   The login box for the CMS appears.
4. Type the following information. This information is normally provided for you by the BusinessObjects administrator.

Login information	Description
System	Name of the CMS server.
User Name	Your repository user name.
Password	Your repository password.
Authentication	Your security level

5. Click *OK*.
   The universe design tool startup screen appears, and an empty session opens.
   The user name and CMS name appear in the title bar.

Depending on options set for the universe design tool, the *Quick Start* universe design wizard can start automatically when you start the universe design tool. Click *Cancel* to close the wizard. For more information on disabling other wizard options, see the section Disactivating the Quick Design wizard [page 38]. If you want to use the Quick Design wizard, then you can refer to the section Using the Quick Design wizard [page 30].

3.2.2 Using the Quick Design wizard

When you start a session for the first time, a *Quick Design* wizard appears by default. You can use the wizard to quickly create a universe, or to familiarize yourself with the universe design tool, however, it is not an appropriate tool for creating a complete universe that responds to end user reporting requirements.

It is recommended that you disable the *Quick Design* wizard, and use it only as a means to familiarize yourself with the universe design tool , and not use it to design universes. All the universe design, building, and maintenance information and procedures in this manual assume that you have disabled the *Quick Design* wizard, except for the chapter Using the Quick Design wizard [page 30] which deals specifically with using the wizard. For information on disabling other *Quick Design* wizard options, see the section Disactivating the Quick Design wizard [page 38].

3.3 Creating a basic universe with the Quick Design wizard

For a demonstration or quick test universe based on a simple relational schema, use the *Quick Design* wizard for creating a basic yet complete universe. You can use the resulting universe immediately, or you can modify the objects and create complex new ones. In this way, you can gradually refine the quality and structure of your universe.

If you are designing a production universe, you should create the universe manually. All other chapters of this guide are based on showing you how to manually create a universe. This is the only section that deals with automatic universe creation.

3.3.1 Why use the Quick Design wizard?

The Quick Design wizard assists you throughout the creation of a universe. It guides you in establishing a connection to the database and then lets you create simple classes and objects. The wizard also provides built-in strategies for the automatic creation of objects, joins, and tables.

Using Quick Design has the following benefits:

- If you are new to the universe design tool, it can help you get familiar with the user interface and basic universe design.
- If you are creating a demonstration universe, it saves you time by automating much of the design process. With the wizard, you can quickly set up a working model of your universe, and then you can customize the universe to suit the needs of your target audience.

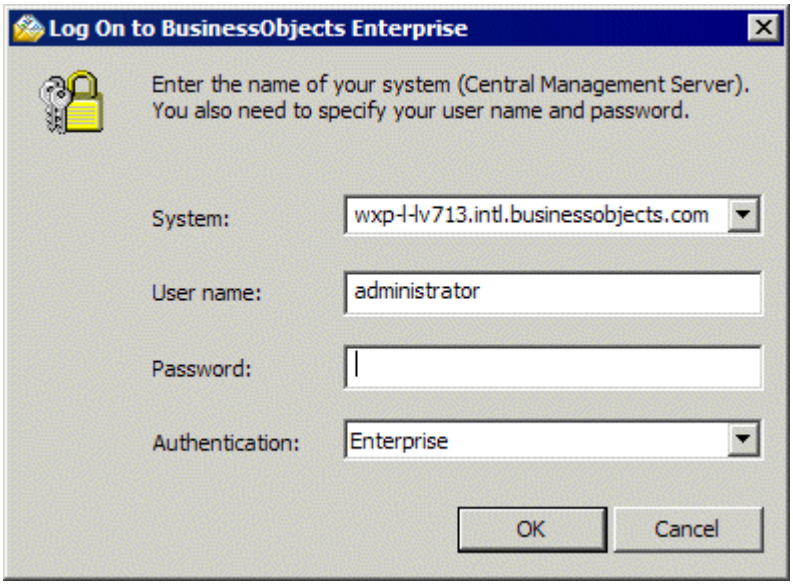
3.3.2 Using the Quick Design Wizard

Quick Design is the name of the wizard that you use to automatically create a universe. Each step in the wizard is described in each of the following sections.

3.3.2.1 Starting the Quick Design wizard

To start the Quick Design wizard:

1. Start the universe design tool .
   The User Identification dialog box is displayed.



2. In the User Identification dialog box, enter your user name and password.
3. Click *OK*.
   The welcome screen of the Quick Design wizard appears.

> **i** Note
>
> If you do not want the wizard to appear the next time you launch a session, clear the check box Run this Wizard at Startup. In addition, you can find two options relating to the display of the wizard in the General tab of the Options dialog box: Show Welcome Wizard and File/New Starts Quick Design wizard (Tools menu, Options command).

3.3.2.2 The welcome screen

The welcome screen displays an overview of the four steps necessary to create a basic universe. It also provides a check box: Click here to choose strategies. If you click this check box, you will be able to select the strategies for creating the universe; otherwise, the universe design tool applies the default built-in strategies.

In each dialog box that follows, Quick Design prompts you for the information needed to carry out the action.

To move from one dialog box to the next, click *Next*. You can return to the previous dialog box by clicking *Back*. You may end the process and quit Quick Design at any time by clicking the Cancel button.



When you select the Click here to choose strategies check box, a dialog box appears listing strategies. This dialog box is described in Choosing the strategies [page 34]. You can select a strategy, or accept the default strategies.

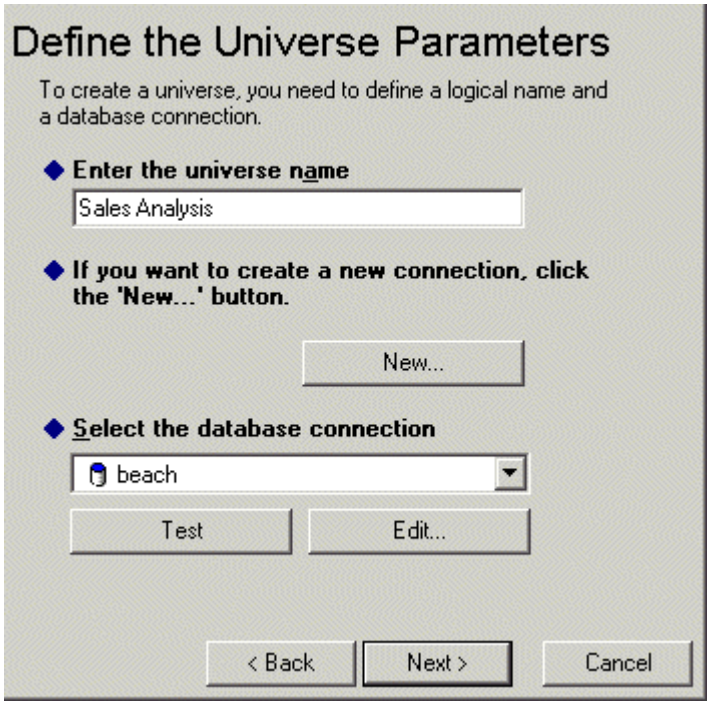
Click *Begin* to start the creation process.

3.3.2.3 Defining the universe parameters

In this step, you define the universe parameters: the universe name and a database connection.

You can enter a long name of up to 35 alphanumeric characters for the universe.

> ⚠ Caution
>
> Don't use the following special characters in the universe name: \ / : * ? " < > | &



You can either create the connection, or select an existing one. To create a connection, click the New button, and specify the necessary parameters in the dialog boxes that follow. For more instructions on these dialog boxes, refer to the section Modifying universe identification parameters [page 73].

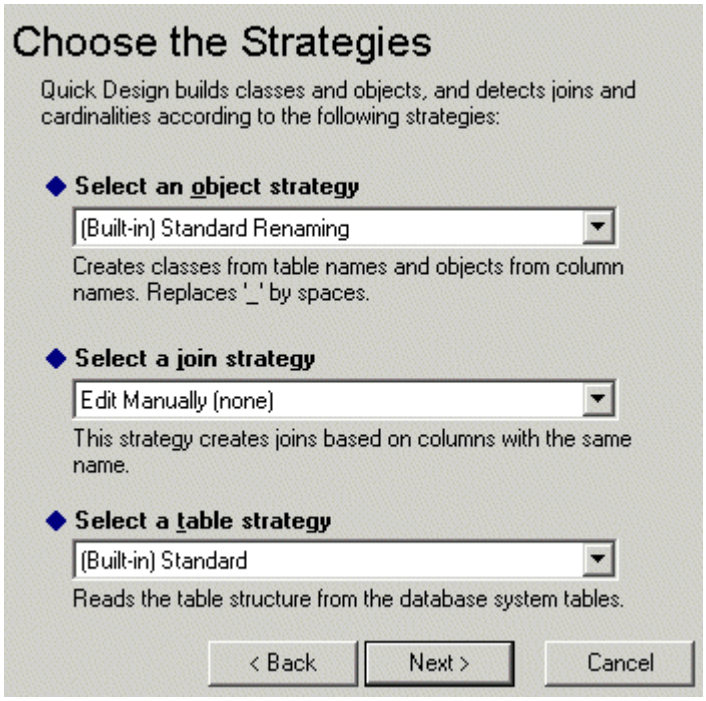
To check whether your connection is valid, click the Test button. The Edit button lets you modify the parameters of the connection.

Click the Next button to proceed to the next step.

3.3.2.4 Choosing the strategies

If you clicked the check box for strategies in the welcome screen, Quick Design prompts you to specify strategies for the creation of objects, joins, and tables.

A strategy is a script that reads structural information from a database or flat file. The universe design tool uses these scripts to create objects, joins, and tables automatically.



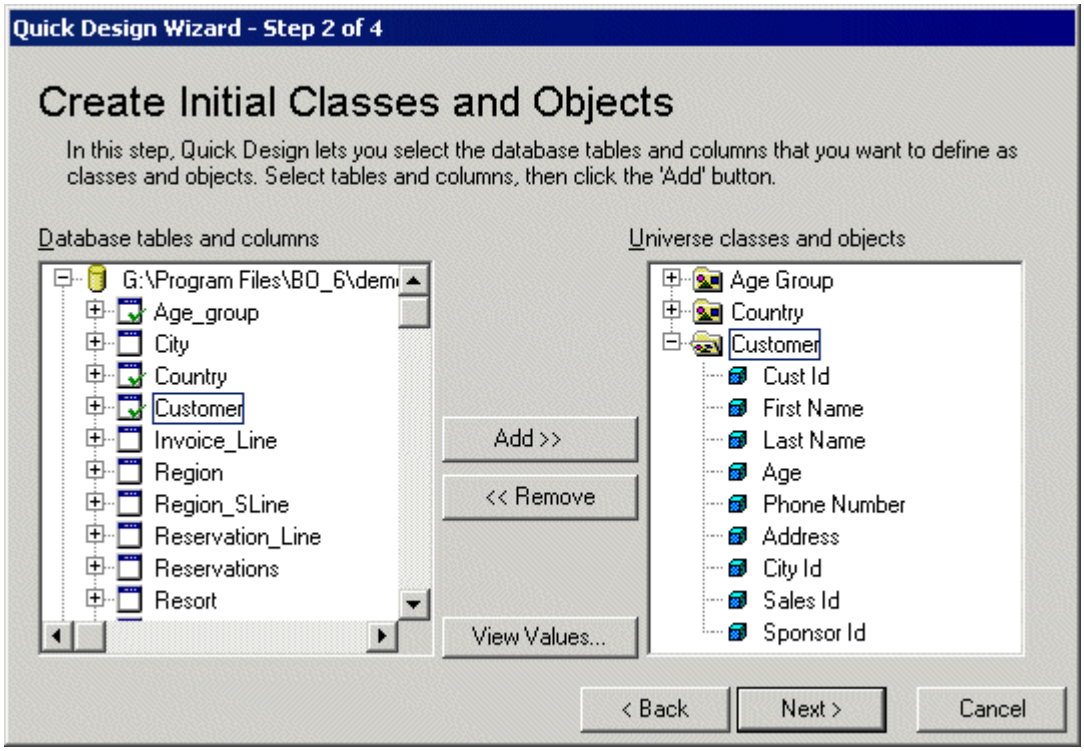
From a list box, you can select another strategy, or none at all. Brief descriptions of the current strategies appear just below the list boxes.

In addition to the built-in internal strategies provided by the universe design tool , you can also create your own external strategies. Refer to the section Using external strategies to customize universe creation [page 371].

Click the Next button to proceed to the next step.

3.3.2.5 Creating the initial classes and objects

Based on the parameters of your database connection, the wizard presents you with a list of database tables and columns. You create the initial classes and objects by selecting tables and columns from the left pane, and adding them to the Universe classes and objects pane on the right.



By default, the left pane shows only the names of the tables.You can use the following methods to navigate through the file trees, and add classes and objects to the right pane:

- To view the columns of any table, click the plus sign (+) to the left of the table name.
- To view the data values of any table or column, click it and then click the View Values button.
- To select one table, click the table, and then click the Add button.
- To select several contiguous tables, hold down the Shift key, then click the first table and last table. All the tables between the selected tables will be highlighted. Then click the Add button.
- To select several tables that are not contiguous, click each table while holding down the Ctrl key. Click the Add button.
- Another way to select tables is to drag and drop them from the left pane to the right pane.
  When you insert a table, the universe design tool includes all of its columns.

In the right pane, the names of classes are displayed beside a folder icon. Click the plus sign (+) beside the class name to view the objects. You can rename a class or object by double-clicking it and entering a new name in the dialog box.

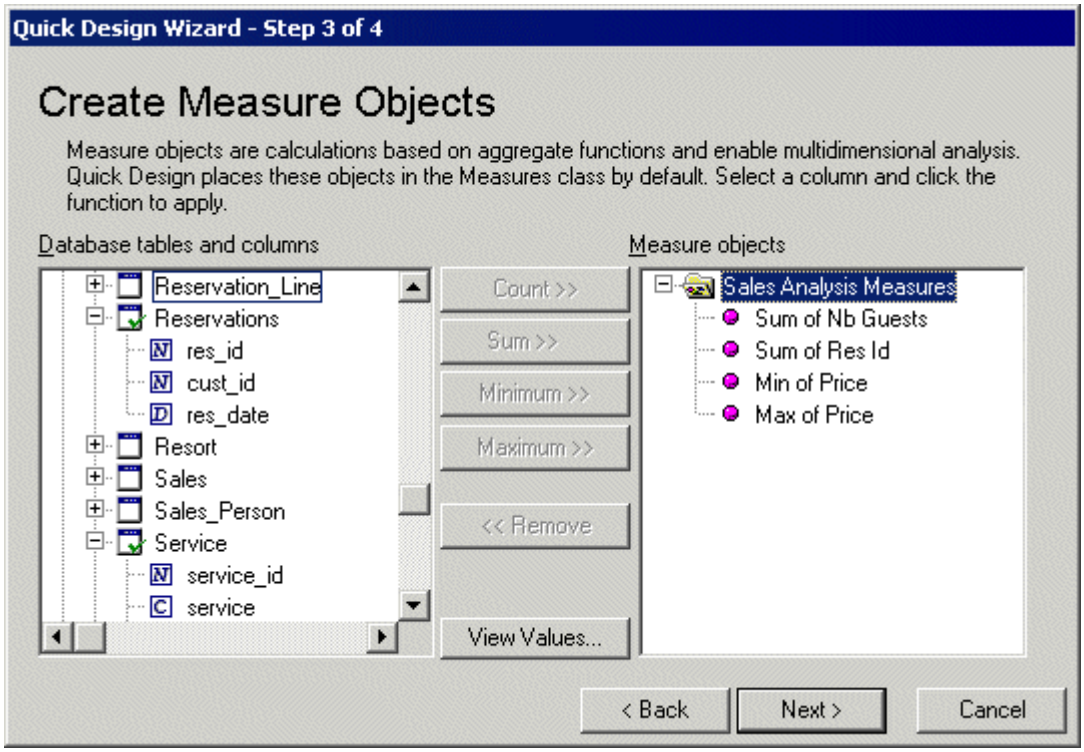
By default, an object is qualified as a dimension object, which is indicated by the cube symbol that precedes the object's name.

To remove a class or object, click it and then click the Remove button.

Click *Next* to move to the next step.

3.3.2.6 Creating measure objects

A measure object is derived from an aggregate function: Count, Sum, Minimum, or Maximum. This type of object provides numeric information. Examples of measure objects are shown in the right pane of the dialog box below:



If you wish to view the data values associated with an object, click it and then click the View Values button.

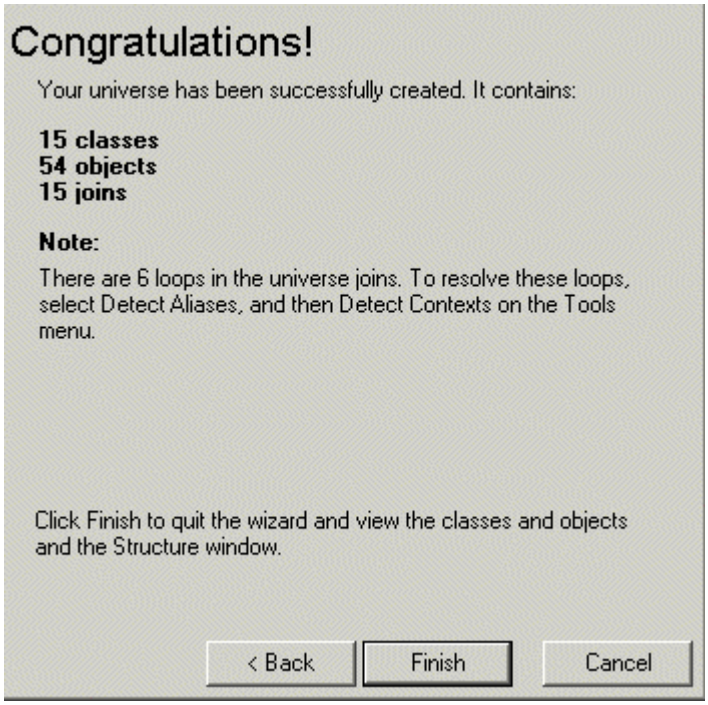
To create a measure object, click the appropriate object in the left pane, and then click the aggregate button. You can rename any measure object you create.

Grouping measure objects in one or more measures classes improves the organization of the universe. It also facilitates the end user's ease of navigation. For more information on measure objects, refer to the section Defining a measure [page 272].

When you click *Next*, Quick Design begins creating your universe.

3.3.2.7 Generating the universe

Quick Design automatically generates your new universe based on the parameters you specified. It indicates the number of classes, objects, and joins created in your universe.



In the dialog box above, a message states that loops exist within the joins of the universe. The universe design tool enables you to resolve loops with aliases and contexts. Refer to the Designing a Schema chapter for more information.

When you click the *Finish* button, the Universe pane and the Structure pane of your new universe appear.

3.3.2.8 Ending a Work Session

Select File > Save As to save the universe, then File > Close to close the universe.

When you save the universe, the universe design tool prompts you to enter a file name. A universe file name can contain the maximum number of characters allowed by your operating system. It has a .unv extension. By default, the universe design tool stores these files in the Universe subfolder of the BusinessObjects folder. In Windows 2000, this folder appears under the Local Data folder for your user profile.

To quit the universe design tool, select File > Exit.

3.3.2.9 Disactivating the Quick Design wizard

When you first start a session, a *Quick Design* wizard appears by default. You can prevent the wizard appearing automatically when you create a new universe as follows:

To disactivate the *Quick Design* wizard:

1. Select *Tools* > *Options*.
   The *Options* dialog box opens to the *General* page.

2. Clear the *Show Welcome Wizard* check box. (This check box is already cleared if you have cleared the *Run this Wizard at Startup* check box from the *Startup Wizard* Welcome page.)
3. Clear the *File/New Starts Quick Design Wizard* check box.
4. Click *OK*.

You can activate the *Quick Design* wizard at any time by selecting the above check boxes from the *General* page of the *Options* dialog box. Using the *Quick Design* wizard is covered in the section Using the Quick Design wizard [page 30].

3.3.3 Following up on a universe created with the Quick

Once you have created a basic universe with the Quick Design wizard, you may find it necessary to edit joins, and to resolve all loops using aliases or contexts. In addition, you can choose to enhance your universe with more complex components using the various universe design tool features. For the appropriate information, you should refer to the relevant section in this manual.

3.4 Importing a universe

You can import one or more universes stored in a universe folder in the repository. You can only import a universe that has already been exported to the repository.

When you import a universe, the CMS checks the universe version on the repository file system. If the version is identical, the universe is made available to the universe design tool. If the universe version on the repository file system is more recent than the CMS version, a message box appears asking if you want to replace the universe in the folder. If you answer *Yes*, then the universe on the repository file system is replaced by the version in the CMS.

3.4.1 Importing a universe from the repository

3.4.1.1 To import a universe from the repository

1. Select *File* > *Import*.

   The *Import Universe* dialog box appears.
2. Select a universe folder from the drop down list box.

   Or

   Click the *Browse* button and select a universe using the folder browser.

   You want to import a universe from this folder.
3. If you want to lock the universe, double-click the universe name.

   A locked universe appears with a padlock symbol. To unlock a universe, double-click it again.

4. Click a universe name.

   This is the universe that you want to import.
5. Verify the file path for the import folder in the *Import Folder* box.

   The universe is imported to this folder.
6. Click *OK*.

3.4.2 What is the difference between opening and importing?

You can open a universe directly from the file system. When you save this universe, it is saved only to the file system, it is not updated in the CMS. Updates in this universe are not available to Web Intelligence users.

When you import a universe, the current version available in the repository is made available to the universe design tool. When you have finished modifying the universe, you export it to the repository. The CMS is updated with the latest changes.

3.5 Opening a universe

You open a universe using the menu commands or by clicking *Open*. When you open a universe directly without importing, you are opening a version on the local file system, that may not correspond to the latest version in the CMS.

3.5.1 To open a universe directly

1. Select *File* > *Open*.

   A *Open* box opens to the directory designated as the default universe file store. You can set this directory in the *Save* page of the *Options* dialog box (*Tools* > *Options* > *Save*).
2. If necessary, browse to the directory that contains the universe file (.UNV).
3. Select a universe file and click *Open*

   Or

   Double-click the universe file.

The Universe opens in the current universe design tool window.

3.6 Exporting a universe

You make a universe available to Web Intelligence users and other designers by exporting a universe to the repository.

When you export a universe the universe is:

- Moved to the selected universe folder on the repository file system
  and
- Created in the Central Management System (CMS)

Each time the universe is exported to the repository, the universe version in the CMS is updated. This is the version that is available to Web Intelligence users.

> **i** Note
>
> Saving a universe is not the same as exporting a universe. Saving updates the universe on the repository local file system, but not the CMS repository version of the universe. See the section What is the difference between exporting and saving? [page 42] for more information.

3.6.1 How are universes organized on the repository file system?

The repository stores universes on the local file system and in the CMS server. You work on the universe version on the local file system. The local file system is the server on which the tool is installed. Your universes are saved by default in the universes folder in your user profile path as follows:

```
\\Documents and Settings\<user>\Application Data\Business Objects\Business Objects
12.0\universes\@<repository name>\universe folder\<universe>.unv
```

The universes stored in the CMS server are used for version control. When you export an updated universe to the repository, the updated universe is copied to the CMS server.

3.6.2 Exporting a universe to the repository

3.6.2.1 To export a universe to the repository

1. Select *File* > *Export*.

   The *Export Universe* dialog box appears.
2. Select a universe folder from the folder drop down list box.

   Or

   Click the *Browse* button and select a universe folder in the folder browser.

You want to export the universe to this folder.

3. If you want to lock the universe, double-click the universe name.

   A locked universe appears with a padlock symbol. To unlock a universe, double-click it again.
4. Click a group in the *Groups* list box. This is the user group that uses the exported universe.
5. Click a universe in the *Universes* list box. The *Universes* list box shows the names of the active universes.
6. If you want to export other universes that are not open, click the *Add Universe* button, and then use the browser to select the other universes.
7. Click *OK*.

3.6.3 What is the difference between exporting and saving?

When you save a universe, you update the version in the repository file system. This does not update the CMS version.

When you export a universe, the update of the version in the repository file system is synchronized with the update of the universe in the CMS.

If you save a universe and do not export the updated version, the CMS is not updated. The saved universe is not available to other users.

Each universe in the repository is assigned a system identifier. Refer to the section Identifying a universe in the repository [page 469] for more information in identifiers.

You cannot export a universe if it has been locked in the repository by another designer.

You can export only a universe defined with a secured connection.

3.7 Saving a universe

You should regularly save your universes throughout a work session. When you save a universe, the universe design tool stores it as a file with a `.unv` extension on the local file system.

In Web Intelligence, a user identifies the universe by the universe name (long name).

When you save a universe, the changes are not saved to the CMS. You must export the universe to the CMS when you have completed updating a universe.

You can use the following maximum characters in the universe name (the long name) and `.unv` file name:

Name type	Maximum number of characters
Universe name	100 (\, /, :, *, ?, ", <, >, |, and & characters are not supported)
.unv name	Operating system maximum

3.7.1 Universe file names as identifiers

You should not change the universe filename `.unv` after reports have been created based on that universe. If you change the filename, any report built on the universe with the old name, will not point to the universe once its name has been changed.

3.7.2 Saving a universe

The universe name can be different from the `.unv` name.

When you use *Save As* to save the universe under new name, the new universe is not associated in the CMS. You must export the new universe to the CMS to create a version of the new universe.

You can use the following methods to save a universe:

- Select *File* > *Save* from the menu bar
- Click the *Save* icon
- Press *CTRL+S* from the keyboard

3.7.3 Saving a universe definition as PDF

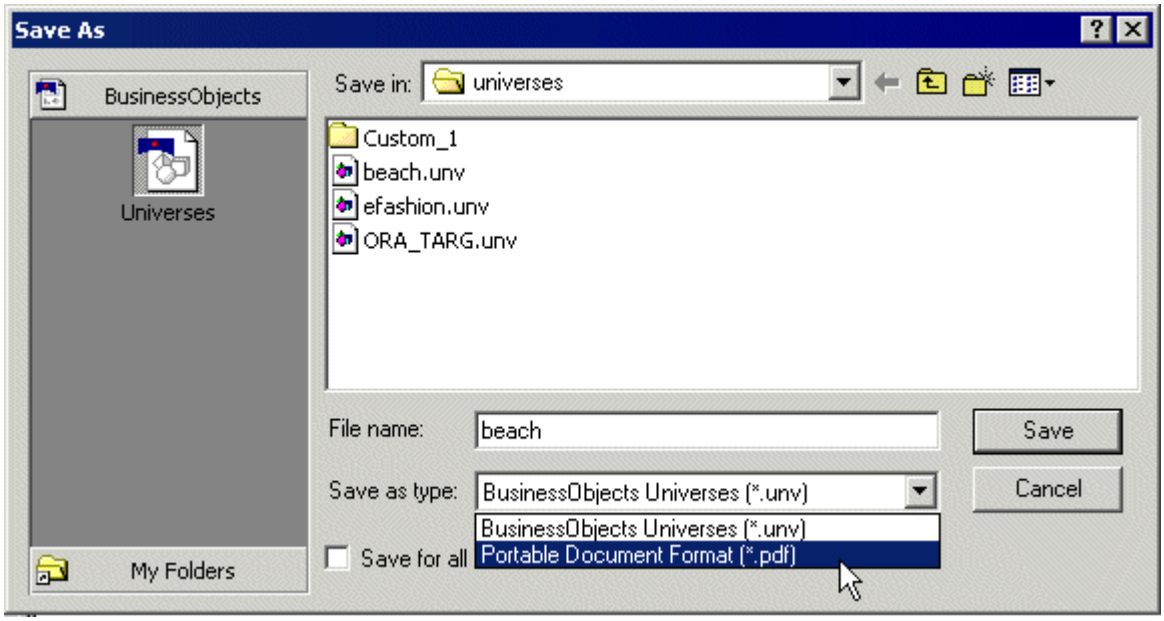
You save the universe information as an Adobe PDF file. You can save the same information that you can print out for a universe. This information includes:

- General information: parameters, linked universes, and the graphical table schema.
- Component lists: lists of components in the universe including objects, conditions, hierarchies, tables, joins, and contexts.
- Component descriptions: descriptions for the objects, conditions, hierarchies, tables, joins, and contexts in the universe.

You can select what components that you want to appear in the PDF from the Print Options dialog box (*Tools* > *Options* > *Print*). These options are described in the section Printing a universe [page 65].

To save universe information as a PDF file:

1. Select *File* > *Save As*
2. Select *Portable Document Format (*.pdf)* from the *Save as type* drop down list box.



3. Click *Save*.

3.7.3.1 Setting default save options

By default, the universe design tool stores the files that you save in the Universe subfolder in the Business Objects path. You can specify another default save folder as follows:

1. Select *Tools* > *Options*.
   The *Options* dialog box appears.
2. Click the *Save* tab.
   The *Save* page appears.
3. Type a file path in the *Default Universe Folder* text box.
   Or
4. Browse to a folder that contains `.unv` files.
5. If you want to specify an automatic save time, select the *Save Automatically* check box and select or type a time period number from the *Minutes value* select box.
6. Click *OK*.

3.8 Closing a universe

You can use the following methods to close a universe.

To close a universe:

- Select *File* > *Close* from the menu bar

- Click the *Close window* button at the top right corner of the universe window
- Press *CTRL+W* from the keyboard.

3.9 Working with multiple designers

You can use the universe design tool in a multiple user environment in which several designers can work on the same universes without causing conflicts between versions.

You can lock a universe so that only one designer at a time can make modifications on the universe, and a universe can also be assigned a version number to keep track of changes.

3.9.1 Locking a universe

When stored in a universe folder, a universe can be shared by several designers provided that they have the necessary user rights.

Only one designer can work on a given universe at a time. A designer who wants to work on a universe, can do so only if the universe has not been locked by another designer.

> **i** Note
>
> You lock a universe from the Import or Export dialog box. When a universe is locked, a padlock symbol is displayed next to the universe name. When another designer locks the universe, the padlock symbol appears dimmed.

3.9.2 Revision number

Each time you export a universe to a universe folder, the universe design tool increments the revision number of the universe. This allows you to determine which is the latest version of the universe. The revision number appears on the Summary tab of Universe Parameters (File > Universe Parameters > Summary).

3.10 Using the universe design tool user interface

The universe design tool interface user interface complies with Microsoft Windows standards. It features windows, menus, toolbars, shortcut keys, and online help.

3.10.1 The main components of the user interface

Each universe is contained within a single universe window, which is contained within the main window.

You also use an independent window called a *Table Browser* which shows all the tables available in the connected database.

3.10.1.1 Universe window

The *Universe* window is divided into two panes:

Pane	Displays
Structure	Graphical representation of the underlying target database of the universe. It includes the tables and joins to which you map objects that end users use to run their queries.
Universe	Classes and objects defined in the universe. These are the components of the universe that Web Intelligence users see and use to create their queries.

3.10.1.2 Table browser

The *Table* browser is a window that displays the tables available in the connected database. You can insert tables into the *Structure* pane by selecting the table and dragging it into the *Structure* pane, or by double-clicking the appropriate table in the *Table* browser.

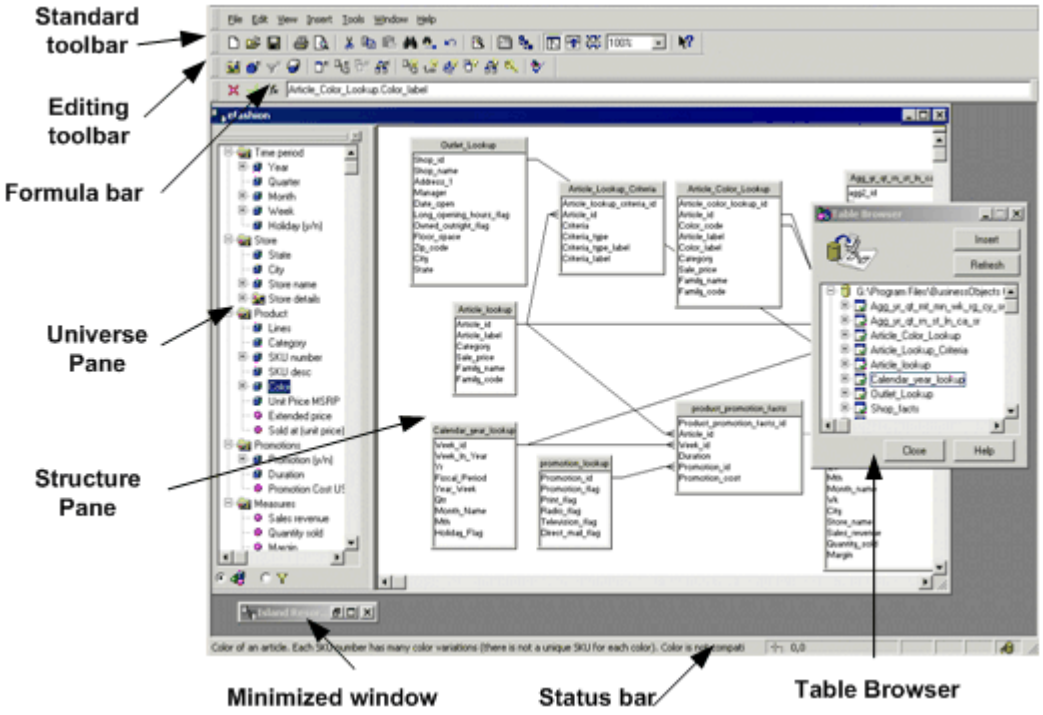
You can display the *Table* browser by any of the following methods:

- Double-click the *Structure* pane background.
- Right-click the *Structure* pane background and select *Insert Table* from the contextual menu.
- Select *Insert* > *Tables*.

> **Note**
>
> Using the table browser is described fully in the Designing a Schema chapter.

3.10.2 The universe design tool user interface

The main components of the interface are labeled below:



3.10.3 Manipulating windows

You can use the windows in the user interface in the following ways:

- In a work session, you can work on more than one universe at a time. the tool displays each universe in one *Structure* pane and in one *Universe* pane.
- Recently opened universes are listed at the bottom of the *File* menu. You can modify the number of universes listed by selecting *Tools* > *Options* > *General*, and indicating the number of universes in the *Recent File* list.
- You can move, resize, or minimize any window within the universe design tool window.
- You can position these windows in the way you find most convenient by selecting *Window* > *Arrange*, and selecting *Cascade*, *Tile Horizontally*, or *Tile Vertically*.
- You can line up all windows that were minimized in the tool window by selecting *Window* > *Arrange Icons*.

3.10.4 Using toolbars

The universe design tool window contains two sets of toolbars: the *Standard* toolbar and the *Editing* toolbar.

For either toolbar, the buttons that you can select depend on which pane is active the *Universe* pane or the *Structure* pane. Buttons that are not available are displayed as dimmed.

The toolbars are dockable. You can drag a toolbar and position it anywhere in the universe window.

3.10.4.1 Moving a toolbar

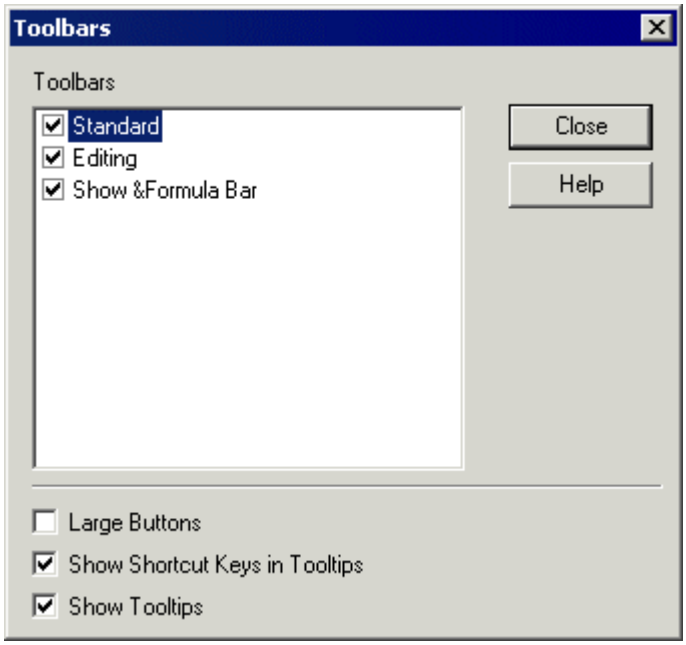
To move a toolbar:

1. Click in an area within the rectangle containing the toolbar.
   The area is shown for both toolbars in the illustration above.
2. While keeping the left mouse button pressed, drag the toolbar to the desired location.
3. Release the mouse button.
   The toolbar is displayed independently.

3.10.4.2 Hiding and showing toolbars

To display or hide either toolbar alternately:

1. Select *View* > *Toolbars*.
   The *Toolbars* dialog box appears.



2. Select or clear check boxes corresponding to toolbars.
3. Select or clear options for the display of the toolbar buttons, tooltips, and shortcut keys listed at the bottom of the dialog box.
4. Click *OK*.

3.10.5 Performing an action or operation in the universe design tool

In the universe design tool, you perform an action or operation in the following ways:

- Select a command from a menu.
- Press the Alt key and enter a shortcut key from the keyboard.
- Click a button on the toolbar.

3.10.5.1 Using the mouse

You can use single and double mouse clicks as follows:

Single click

You use a single click for the following actions:

- Performing a standard action (selecting a command or clicking a button)
- Selecting an element from the *Universe* pane, the *Structure* pane, or the *Table Browser*.
- If you select one or more components within the universe design tool window, a single-click with the right mouse button causes a pop-up menu to be displayed. It contains commands related to the components you selected.

Double-click

You can double-click the following universe structures to affect display changes or modify properties:

Double click...	Result...
An empty space in the *Structure* pane	*Table Browser* appears.
A table in the *Structure* pane	Modifies table display. A table and its columns can be displayed in one of three views. Refer to the section Using List mode [page 54] for more information.
A join in the *Structure* pane	*Edit Join* dialog box for the join appears. You can modify join properties from this dialog box.
A class in the *Universe* pane	*Edit Properties* dialog box for the class appears. You can modify class properties from this dialog box.

Double click...	Result...
An object in *Universe* pane.	*Edit Properties* dialog box for the object appears. You can modify object properties from this dialog box.
A Condition object in the *Condition* view of *Universe* pane	*Edit Properties* dialog box for the condition object appears. You can modify object properties from this dialog box.

3.10.5.2 Undoing an Action

You can undo a previously performed action in two ways:

- Select *Edit* > *Undo*.
- Click *Undo*.

3.11 Using Find and Replace

You can use *Find* to locate characters or a text string in both the universe and structure panes. You can use *Find and Replace* to locate and replace characters or text in the names and descriptions for any structure in the universe.

3.11.1 Using Find

You can search for text contained in universe structures in the universe and structure panes.

3.11.1.1 Setting Find options

The *Find* options available are dependant on whether the *Universe* pane or the *Structure* pane is active.

You can set the following search options to locate a string:

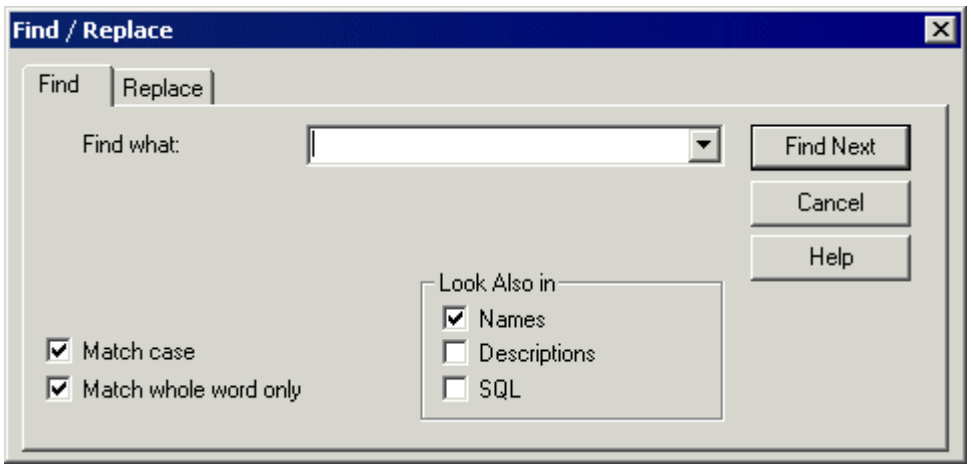
Option	Option is available	Description
Find What	When *Universe* or *Structure* pane is active	Text string to search.
Match Case	When *Universe* or *Structure* pane is active	Include upper and lower case character match in search.

Option	Option is available	Description
Match whole word only	When *Universe* or Structure pane is active	Match on entire string.
Look also in names	When *Universe* pane is active	When selected, searches class and object names or predefined condition names only. When cleared, class, object or predefined condition names are not included in search.
Look also in descriptions	When *Universe* pane is active	When selected, includes all descriptions of universe structures in search.
Look also in SQL	When *Universe* pane is active	When selected, includes SQL definitions of objects, joins, and other universe structures in search.

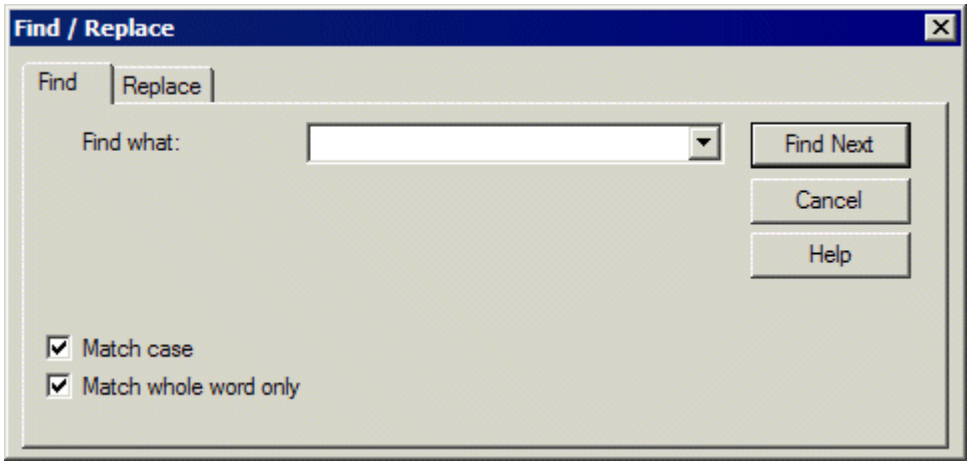
3.11.1.2 Searching in a universe

To search in a universe:

1. Click in the *Universe* or *Structure* pane.
   You want to find a string in this pane.
2. Select *Edit* > *Find*.
   The *Find and Replace* box appears. The box for an active *Universe* pane is below.



The box for an active *Structure* pane appears below.

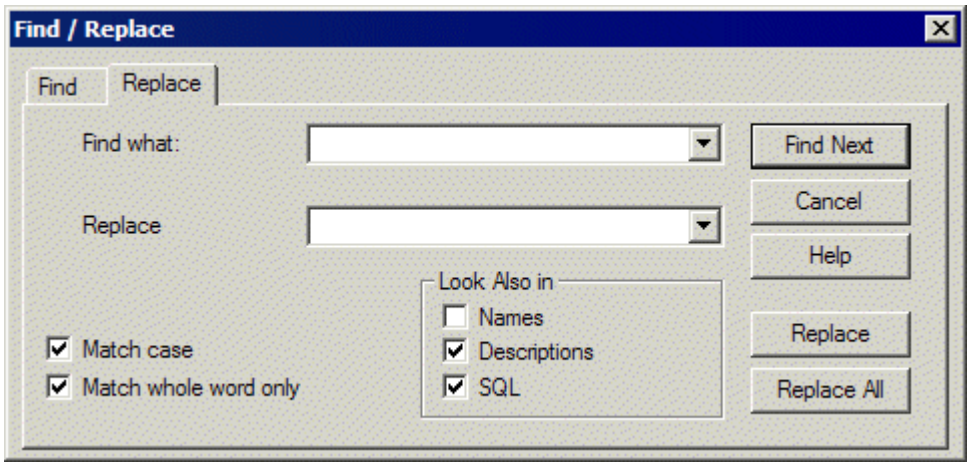


3. Type a character or a string in the *Find what* text box.
4. Select or clear search option text boxes.
5. Click *Find Next*.
   When a character or string is found in the universe pane, the object is highlighted. When an instance is found in an object description, or SQL definition, the object properties box is opened automatically, and the character or string highlighted.
6. Click *Find Next* to search for another instance of the search string.
7. Click *Cancel* to close the *Find and Replace* box.

3.11.1.3 Searching and replacing in a universe

To search and replace a character or string in a universe:

1. Select *Edit* > *Replace Next*.
   The *Find and Replace* box appears.
2. Type a character or a string in the *Find what* text box.



3. Type a character or a string in the *Replace* box. This is the text item that you want to replace an instance of the contents of the *Find what* box.

4. Select or clear search option text boxes.
5. Click *Replace* if you want to replace a text item each time an instance is found.
   Or
   Click *Replace All* to automatically replace all instances in the universe.
   If you replace found items individually, the object properties box automatically opens and becomes the active box when an item appears in an object description. You need to click the *Find and Replace* box to continue the search.

3.11.2 Using Quick Find

You can search the active pane by typing the first letter of the search string in the search box at the bottom of the Universe pane.

If the Universe pane is active, the search is performed on class and object names.

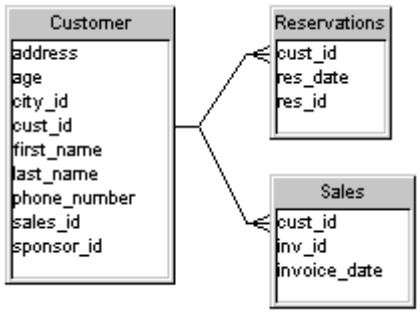
If the Structure pane is active, the search is performed on table names.

3.12 Organizing the table display

This section describes the graphic features that you can use to organize and manipulate tables in the structure pane. The design methodology that you use to design the schema, and what you need to know to create a successful schema in the *Structure* pane, is described in the chapter Creating a schema with tables and joins [page 123].

3.12.1 How are tables represented?

In the *Structure* pane, tables are represented graphically as rectangular symbols. The name of the table appears within a strip in the upper part of the rectangle. The list of items within the rectangle represents the columns of the table. The lines connecting the tables are the joins.



3.12.2 Manipulating tables

You can perform the following actions to manipulate tables in the *Structure* pane:

3.12.2.1 Selecting tables

You can select tables as follows:

To select	Do the following...
One table	Click the table.
Several tables	• Hold left mouse button down while drawing a selection border around the tables. • Click multiple tables while holding down the SHIFT key.
All tables at once	Select *Edit* > *Select All*.

To undo a selection, place the pointer away from the tables and click again.

3.12.2.2 Deleting tables

To delete a table:

1. Select a table.
2. Do one of the following actions:
   - Click the *Cut* button on the *Standard* toolbar.
   - Select *Edit* > *Cut*.
   - Press Delete.

3.12.3 Using List mode

You can use List Mode to list the tables, joins, and contexts used in the active universe. In List Mode, the universe design tool adds three panes above the *Structure* pane. These panes are labeled *Tables*, *Joins*, and *Contexts*.

You can use List Mode in the following ways:

Action	Result
Click a listed component in any of the List mode panes.	Component is highlighted in *Structure* pane.
Select a table, join, or context in the Structure pane.	Corresponding listed component in *List* pane is highlighted.
Double-click a table name in the Table pane.	Rename Table box appears. You can rename the table and depending on the database, edit table owner and qualifier.
Double-click a join name in the *Joins* pane.	*Edit Join* box for the join appears. You can edit join properties.
Double-click a context name in the *Contexts* pane.	*Edit Context* box appears. You can add joins to the selected context by pressing CTRL and clicking joins in the list.
Click a component then click a triangle between two *List* panes.	Components in neighboring list pane related to original component are displayed. All non-related components are filtered out.
Click on separator line between *List* pane and *Structure* pane, then drag line up or down.	*List* pane enlarges or decreases size depending on drag direction.

3.12.3.1 Using the triangles between panes to filter listed components

The small triangles that appear between the panes act as filters on the display of the components. For example:

- You click a table name in the *Tables* pane, and then click the triangle pointing to the *Joins* pane. The Joins pane now shows only the joins of the selected table.
- You click a join name in the *Joins* pane, and then click the triangle pointing to the Tables pane. The *Tables* pane now only shows the tables linked by the join.

3.12.3.2 Returning to normal view from List Mode

You can remove *List* view and return to normal view in two ways:

- When in List Mode, select *View* > *List Mode*.
- When in List Mode, click the *List Mode* button.

3.12.4 Arranging tables automatically

You can automatically arrange the tables in the structure pane in two ways:

- Select *View* > *Arrange tables*.
- Click the *Arrange* button.

3.12.5 Changing table display

You can display three different views of a table. Each type of view acts as a filter on the amount of information shown in the table symbol.

Each view is described as follows:

Table view	Description
Default	Each table is displayed with up to eight columns. You can modify this value. Refer to the section Selecting schema display options [page 57] for more information.
Name only	Only table names are displayed in the table symbols. This reduces potential clutter in the *Structure* pane when you have many tables.
Join columns	Only columns involved in joins between tables are shown in each table symbol. These are usually key columns.

Each table view is shown as follows:

3.12.5.1 Default table view

A table symbol with the first eight columns is shown below.



The ellipsis (...) appears after the last column when there are more then the default number of columns in a table. The scroll bar appears when you click the table once. You can enlarge a table by dragging the lower border of the table downward.

3.12.5.2 Table name only view

You can display only table names in a table symbol as follows:

- Double-click a table.

Only the name of the table is displayed.

3.12.5.3 Join columns table view

You can display only join columns in a table symbol as follows:

- In the *Structure*, double-click a table that is already in name only view. The table shows only the join columns.

3.12.5.4 Changing the display for all tables

To change the view of all selected tables simultaneously:

- Select *View* > *Change Table Display*.

3.13 Selecting schema display options

You can customize the shape or appearance of the tables, columns, joins, and cardinalities in the *Structure* pane.

You have the following graphical options for the display of components in the structure pane:

Option	Description
Join shape	Joins can be represented as different types of simple lines, or as lines that include cardinality indicators such as crows feet ends, or cardinality ratios.
Best Side	When selected the join linking two tables is automatically evaluated as being better displayed on the left or right side of one table, ending on the left or right side of another table, and having the shortest length.

Option	Description
Tables	Tables can have 3D effect, show an aliased name, or show the number of rows. To display the number of rows in each table, you also need to refresh the row count by selecting View > Number of Rows in Table. This is described in the section Modifying the default value for number of returned rows [page 61].
Columns	A column data type can be displayed next to the column. Key columns can be underlined, and columns can also be shown left justified in the table symbol, or centered.
Default number of columns	You can type the default number of columns that are shown in a table symbol. If a table has more than the default number, the table symbol appears with an ellipsis (...) at the end of the column list. When you click the table once, a scroll bar appears at the side of the table.
Center on selection	The view of the *Structure* pane based on a calculated center point.

3.13.1 Setting graphic options for the Structure pane display

You can set graphic options for the components of the *Structure* pane as follows:

1. Select *Tools* > *Options*.
   The *Options* dialog box appears.
2. Click the *Graphics* tab.
   The *Graphics* page appears. It lists graphic options for components in the *Structure* pane.
3. Select or type graphic display options.
4. Click *OK*.

3.13.1.1 Examples of graphical options

The following are some examples of the possible graphical representations of components in the structure pane using the graphical options available in the *Options* dialog box (*Tools* > *Options* > *Graphics*.

Aliased name

When selected an aliased table in the *Structure* pane is displayed both with its name and the name of the table from which it is derived, in parentheses.

Show Row Count and Show Format

When *Show Row Count* is selected the number of rows in each table appears at the bottom of each table symbol. You need to select *View* > *Number of rows in Table* to refresh row numbers for all tables before the row count is displayed.

When *Show Format* is selected, a letter representing the column type appears beside the column name. The column type can be:

- C for character
- D for date
- N for number
- T for long text
- L for blob (binary large object).

In the *Structure* pane, the numbers appear below the lower left corner of the tables, the data types are shown next to the column names.

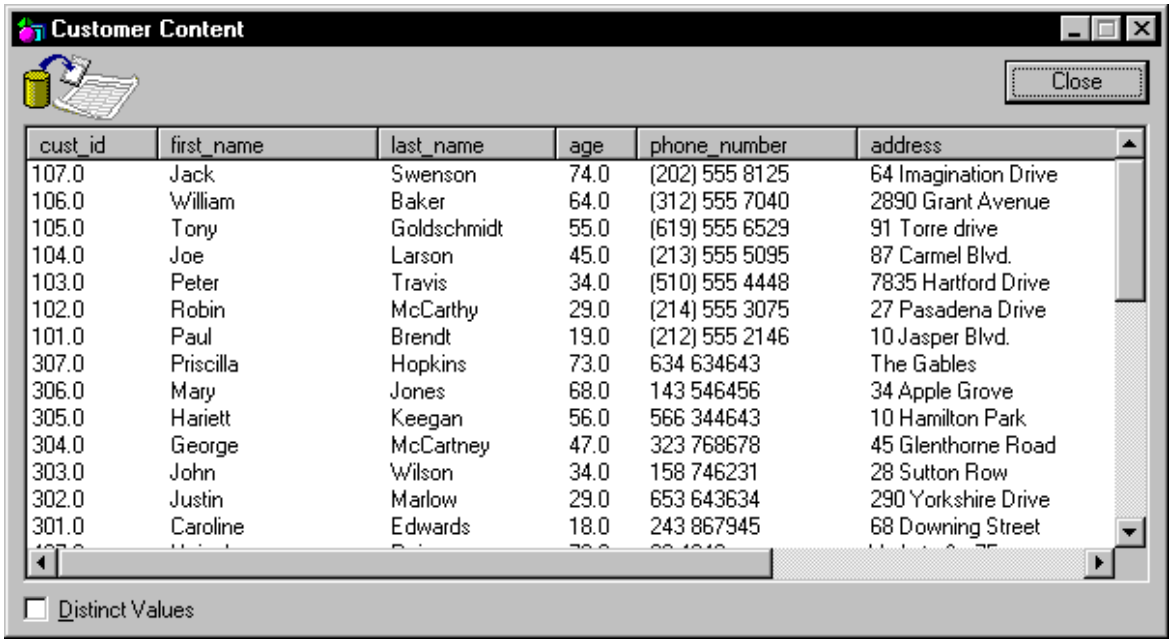
3.13.2 Viewing table and column values

You can view the data values of a particular table or column. The default number of rows that you can view for any table is 100. You can change this value to return more or less rows depending on your needs.

3.13.2.1 Viewing the values of a table

To view the values in a table:

1. Click the table in the *Structure* pane.
2. Select *View* > *Table Values*.
   A content dialog box for the table appears listing the values for each column in the table.



3. Select the *Distinct Values* check box if you want to show only distinct values.
4. Click *Close*.

3.13.2.2 Viewing the values of a column

When viewing column values you can enlarge the view of the columns by selecting *View* > *Zoom In*. This makes it easier to select a column.

You can view the values for an individual column as follows:

1. Place the pointer over a table column in the *Structure* pane.
   The pointer is transformed into a hand symbol.
2. Right-click the column and select *View Column Values* from the contextual menu.
   A content dialog box for the column appears listing the column values.



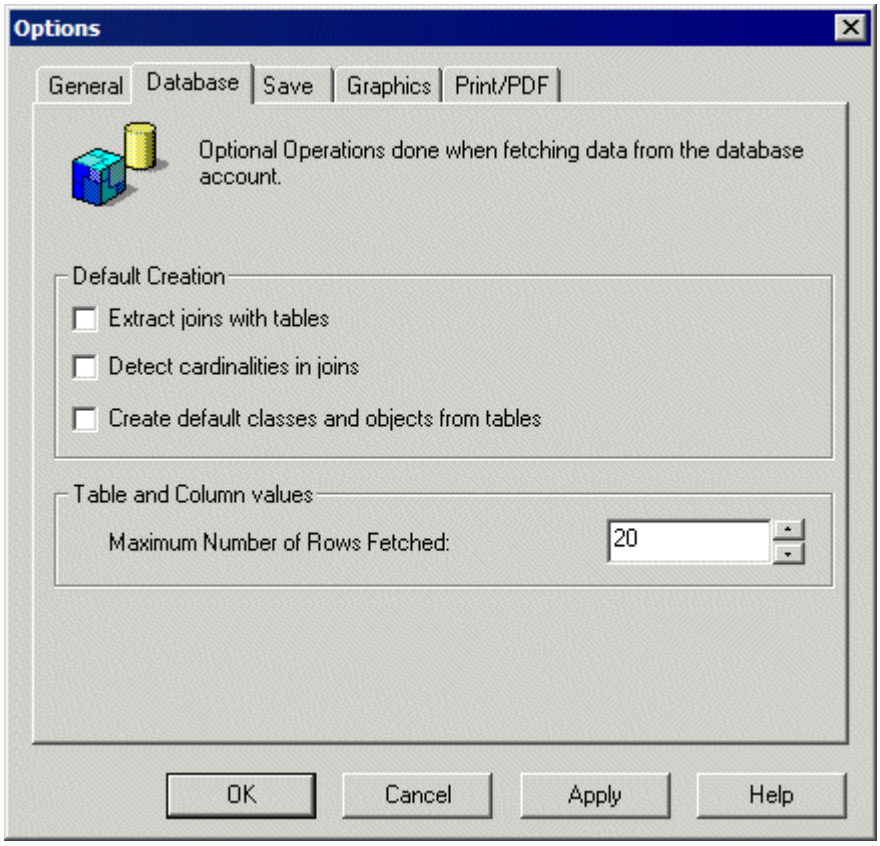
3. Select the *Distinct Values* check box if you want to show only distinct values.
4. Click *Close*.

3.13.2.3 Modifying the default value for number of returned rows

You can modify the default value for the number of rows returned when you view table or column values. This can be useful if you only want to view a small sample of the values in a table, so you can restrict the returned values to a smaller number.

To modify the number of rows fetched for a table:

1. Select *Tools* > *Options*.
   The *Options* dialog box appears.
2. Click the *Database* tab.
   The *Database* page appears.
3. Type or select a number using the up and down arrows from the *Table and Column values* list box.
   The *Database* page below has 20 rows specified to be returned when values are viewed for a table or column.



4. Click *OK*.

3.13.3 Viewing the number of rows in database tables

You can display the number of rows in each table. You do this in two stages:

- Activate the graphic option *Show Row Count* (*Tools* > *Options* > *Graphics*),
- Refresh the row count for all tables by selecting *View* > *Number of Rows in Table*.

You can display the number of rows in each table in the database, or you can set a fixed number of rows for a selected table to optimize query performance. This allows you to control the order of tables in a FROM clause, which is based on table weight. This is described in the section Modifying the row count of a table [page 64].

> **i Note**
>
> Displaying the number of rows in a table is not the same as setting the number of rows that are returned to view table or column values.

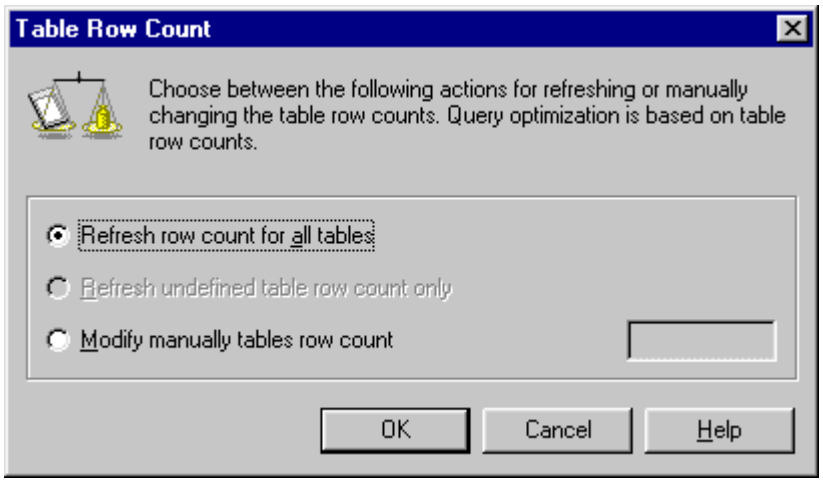
3.13.3.1 Displaying number of rows in tables

To display the number of rows in each table:

1. Select *Tools* > *Options*.
   The *Options* dialog box appears.
2. Click the *Graphics* tab.
   The *Graphics* page appears.
3. Select the *Show Row Count* check box.
4. Click *OK*.
5. Select one or more tables.
   Or
   Click anywhere in the *Structure* pane and select *Edit* > *Select All* to select all the tables in the structure pane.

> **Note**
>
> When you click in the *Structure* pane, you activate the menu items that relate to the components in the *Structure* pane. If you do not click in the *Structure* pane before selecting a menu item, only the menu items that apply to the *Universe* pane are available.

6. Select *View* > *Number of rows in Table*.
   The *Table Row Count* box appears.



The options in this dialog box are described below:

Option	Description
Refresh row count for all tables	Refreshes the display of the row count for selected tables, or all the tables in the Structure pane.
Refresh undefined table row count only	Displays the row count of tables that were previously not selected. As a result, all the tables in the *Structure* pane appear with their row count.

Option	Description
Modify manually tables row count	Lets you modify the row count for either selected tables or all the tables in the Structure pane. Enter the new value in the text box beside the option. This option is used for optimizing queries, a topic covered in the next section.

7. Select the *Refresh row count for all tables* radio button.
8. Click *OK*.
   The row count for each selected table appears under the bottom left corner of each table symbol in the *Structure* pane.

3.13.3.2 Modifying the row count of a table

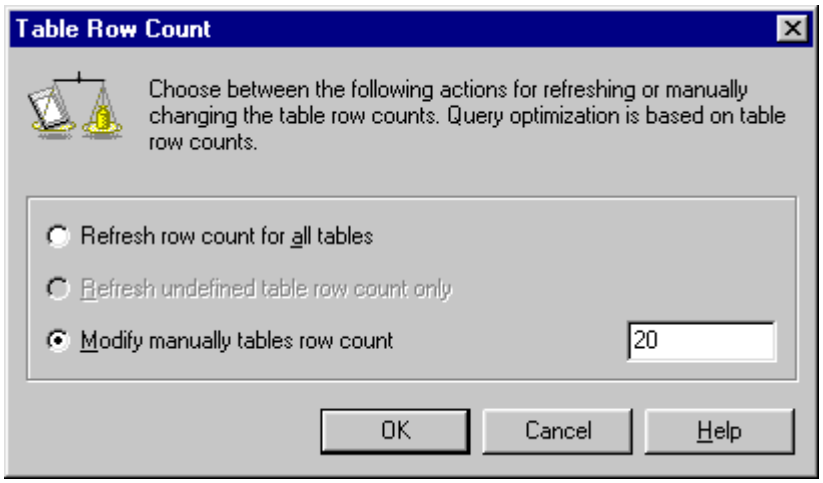
You can modify the row count of tables. Two reasons for doing this are as follows:

Modify row count to...	Description
Optimize queries	Query optimization is based on the order of the tables in the FROM clause of the generated SQL. Tables with many rows appear before tables with fewer rows. This order can be important especially for RDBMS that lack an optimizer feature. By modifying the row count of tables, you can change their order in the FROM clause.
Adapt row count to a subsequent change in data capacity	You can modify the row count of a table when the row count does not reflect the number of rows a table is to hold. For example, you can work with a test table having a row count of 100 even though the table will contain 50,000 rows.

To modify row count of one or more tables:

1. Select *Tools* > *Options*.
   The *Options* dialog box appears.
2. Click the *Graphics* tab.
   The *Graphics* page appears.
3. Select the *Show Row Count* check box.
4. Click *OK*.
5. Select one or more tables.
   Or
   Click anywhere in the *Structure* pane and select *Edit* > *Select All* to select all the tables in the structure pane.
6. Select *View* > *Number of rows in table*.
   The *Table Row Count* box appears.
7. Select the *Modify manually tables row count* radio button.

8. Type the number of rows that you want to display for the table.



9. Click *OK*.
   The row count for each selected table appears under the bottom left corner of each table symbol in the *Structure* pane.

3.14 Printing a universe

The universe design tool provides all standard Windows print facilities. You can print out the schema, as well as lists of the tables, columns, and joins in the *Structure* pane. You can also control the way the components and information appear on a printed page.

> **i** Note
>
> You can print out a PDF version of the universe definition and schema by saving the universe as a PDF file, then printing the PDF file. See the section Saving a universe definition as PDF [page 43] for more information.

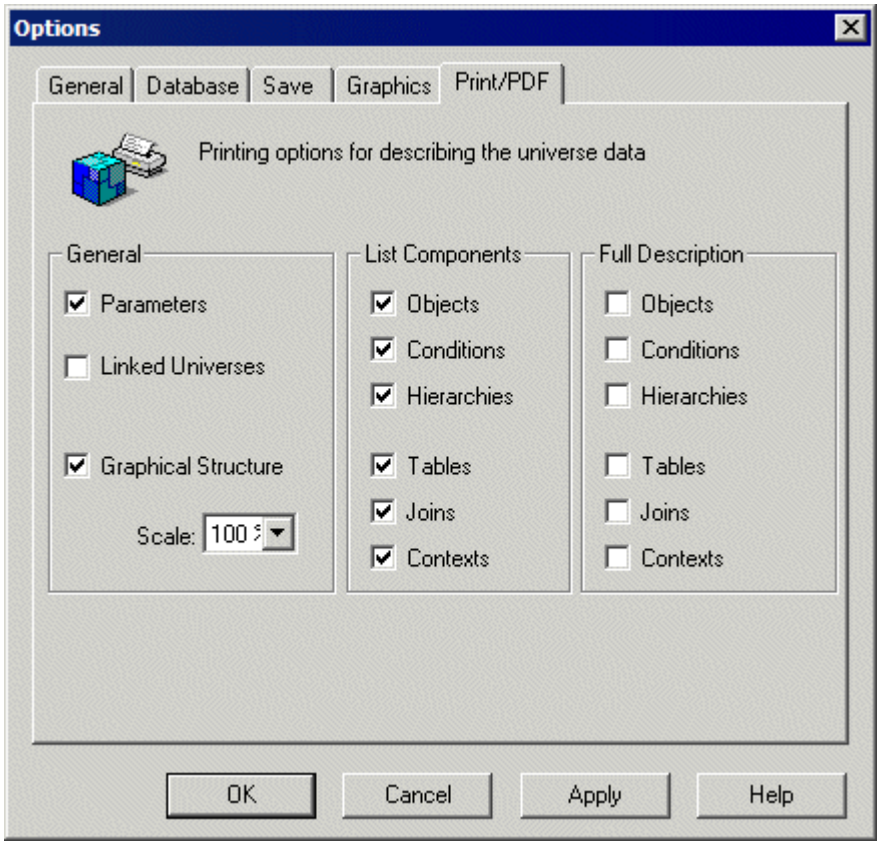
3.14.1 Setting print options

You can select print options from the *Print* page of the *Options* dialog box (*Tools* > *Options* > *Print*). The Print options that you set, also apply to the options that are saved to a PDF file when you save the universe definition as PDF. You can select the following print and PDF options:

Print option	Prints out...
General information	Information on the following: • Universe parameters • Linked universes The graphical structure of the schema in the *Structure* pane. You can select the scale for this graphic.
Component lists	Lists of components in the universe grouped by one or more of the following types: objects, conditions, hierarchies, tables, joins, and contexts.
Component descriptions	Descriptions for the following components: objects, conditions, hierarchies, tables, joins, and contexts. The description includes detailed information on the properties of the component. For an object, this information can include the SQL definition, qualification and security access level.

To set print options for a universe:

1. Select *Tools* > *Options*.
   The *Options* dialog box appears.
2. Click the *Print/PDF* tab.
   The *Print* page appears.



3. Select print option check boxes as required.
4. Click *OK*.

3.14.1.1 Specifying page setup

To specify page setup options:

1. Select *File* > *Page Setup*.
   The *Page Setup* sheet appears.
2. Select or type page setup options.
3. Click *OK*.

3.14.1.2 Using Print Preview

You can preview your universe before printing in two ways:

- Select *File* > *Print preview*.
- Click the *Print Preview* button.

3.14.1.3 Printing the Universe

You can print your universe in two ways:

- Select *File* > *Print*.
- Click the *Print* button.

4 Creating a universe and setting the universe parameters

Before you can build a universe, you must firstly create a new universe file.

When you create a new universe file, you must define a connection parameter to allow the universe to access your database middleware. You can also define other parameters that determine how the universe design tool creates objects, links from the current universe to other universes, and query restrictions.

You save the new universe as a `.unv` file. The new universe contains no classes and objects. You create these during the universe development process by designing a table schema and then creating objects that map to database structures.

4.1 What are universe parameters?

Universe parameters are definitions and restrictions that you define for a universe that identify a universe and its database connections, specify the type of queries that can be run using the universe, and set the controls on the use of system resources.

You define universe parameters from the *Universe Parameters* dialog box (*File* > *Parameters*) when you create a universe. The database connection is the only parameter that you must manually select or create when you create a new universe.

You can modify these parameters at any time.You can define the following universe parameters:

Parameter	Description
Definition	Universe name, description, and connection parameters and information. These are the parameters that identify the universe. Refer to the section Identifying the universe [page 72] for information on defining and modifying this parameter.
Summary information	Version and revision information, designer comments, and universe statistics. Refer to the section Viewing and entering summary information [page 71] for information on defining and modifying this parameter.
Strategies	Indicates the strategies used by the universe. A strategy is a script used to extract structural information from a database. Refer to the section Selecting strategies [page 83] for information on defining and modifying this parameter.

Parameter	Description
Controls	Indicates the limitations set for the use of system resources. Refer to the section Indicating resource controls [page 87] for information on defining and modifying this parameter.
SQL	Indicates the types of queries that the end user is allowed to run from the Query pane. Refer to the section Indicating resource controls [page 87] for information on defining and modifying this parameter.
Links	Indicates the settings defined for linked universes. Refer to the section Entering SQL restriction options [page 90] for information on defining and modifying this parameter.

4.2 Creating a new universe

The following procedure describes how you can create a new universe from scratch by defining universe parameters then saving the universe. The procedure provides an overview of all the pages available from the Parameters dialog box.

For more detailed information on each step you should refer to the respective section for the parameter in this chapter.

Defining all the parameters at universe creation may not be necessary. You must select a connection, but you can accept the default values for other parameters, and then modify them as appropriate when necessary.

4.2.1 Creating a new universe from scratch

To create a new universe from scratch:

1. Select *File* > *New*.
   The *Universe parameters* dialog box opens to the *Definition* page. See the section Identifying the universe [page 72] for information on this page.

   > **i Note**
   >
   > When you select any option for a universe parameter, the option *Click here to choose stored procedure universe* is greyed out. It cannot be selected or deselected. To change the type of universe you want to create, click *Cancel* and start again.

   - Type a name and description for the universe.

     > **⚠ Caution**
     >
     > Don't use the following special characters in the universe name: \ / : * ? " < > | &

- Select a connection from the *Connection* drop-down list box.

Or

- Click the *New* button if you want to define a new connection that is not listed in the drop-down list. See the section Modifying universe identification parameters [page 73] for information on defining a new connection.

2. Click the *Summary* tab.
   The *Summary* page appears. See the section Viewing and entering summary information [page 71] for information on this page.
   - Type universe information in the *Comments* box.
3. Click the *Strategies* tab.
   The *Strategies* page appears. It displays the strategies available for your connected data source. See the section Selecting strategies [page 83] for information on this page.
   - Select a strategy from each of the Objects, Joins, and Tables drop-down list boxes.

   Depending on the RDBMS for the connection, there can be more than one strategy available from each drop-down list box.
4. Click the *Controls* tab.
   The *Controls* page appears. See the section Indicating resource controls [page 87] for information on this page.
   - Select or clear check boxes in the *Query Limits* group box.
   - Enter values for the check boxes that you select.
5. Click the *SQL* tab.
   The SQL page appears. See the Indicating SQL restrictions [page 88] for information on this page.
   - Select or clear check boxes as appropriate.
6. Click the *Links* tab, if you want to link the new universe with an existing universe.
   The Links page appears. See the section Entering SQL restriction options [page 90] for information on this page.
   - Click the *Add Link* button to select a universe to link with the new universe.
7. Click the *Parameters* tab.
   The *Parameters* page appears. It lists SQL parameters that can be set to optimize SQL generation. See the section Setting SQL generation parameters [page 90] for information on this page.
8. Click *OK*.
   The universe and structure panes open up in the universe design tool.
9. Select *File* > *Save*.
   - Type a name for the universe file.
   - Click *Save*.

4.3 Viewing and entering summary information

The *Summary* page displays universe administration information. You can use this information to help you keep track of the development of the active universe.

The *Summary* page displays the following information:

Information	Description
Created	Universe creation date and the name of the creator.
Modified	Date of last modification and the name of the modifier.
Revision	Revision number which indicates the number of times the universe has been exported to the CMS.
Comments	Information about universe for yourself or another designer. This information is only available in th universe design tool. You should include information about the universe for users in the *Description* field on the *Identification* page.
Statistics	List of the number of classes, objects, tables, aliases, joins, contexts, and hierarchies contained in the universe.

4.4 Setting universe parameters

You can set universe parameters for the following purposes:

- Identifying the universe [page 72]
- Modifying universe identification parameters [page 73]
- Viewing and entering summary information [page 71]
- Selecting strategies [page 83]
- Indicating SQL restrictions [page 88]
- Entering SQL restriction options [page 90]
- Setting SQL generation parameters [page 90]

Each type of parameter is contained on a page in the *Parameters* dialog box (*File* > *Parameters*). Each group of parameters is described in its respective section below.

4.4.1 Identifying the universe

Each universe is identified by the following parameters:

Identifier	Used by
File name (8 characters)	File system, and Web Intelligence to reference the universe.
Long name (35 characters)	Web Intelligence users.
Description	Web Intelligence users.

Identifier	Used by
Unique numeric ID	CMS to identify universe. This number is assigned to the universe when it is first exported to the CMS.

The name and description parameters are defined at universe creation from the *Definition* page of the *Universe Parameters* dialog box. You can modify the universe identification parameters at any time.

You also define the database connection from this page.

For information on defining a new connection, you can refer to the section Modifying universe identification parameters [page 73].

You can define the following identification parameters for a universe:

Identification parameter	Description
Name	Universe name. Identifies the universe to Web Intelligence users. The name characters supported by the registry are defined by the General Supervisor. Character support is RDBMS dependent. **⚠ Caution** Don't use the following special characters in the universe name: \ / : * ? " < > | &
Description	Description of universe purpose and contents. Optional field. This description is viewable by Web Intelligence users, so information in this field can provide useful information about the role of the universe.
Connection	Named set of parameters that defines how Web Intelligence accesses data in a database file. All available connections appear in the Connections drop-down list box. You can also create new connections.

4.4.1.1 Modifying universe identification parameters

To modify universe identification parameters:

1. Select *File* > *Parameters*.
   Or
   Click the *Universe Parameters* button in the toolbar.
   The *Universe Parameters* dialog box opens to the *Definition* page.
2. Type a name and a description.

   ⚠ Caution

   Don't use the following special characters in the universe name: \ / : * ? " < > | &

3. Select a connection from the *Connection* drop-down list box.
4. Click *Test* to verify that the connection is valid.
   If you receive a message informing you that the server is not responding, the connection is not valid. You can correct connection parameters by clicking *Edit* and editing connection properties. If the error persists, refer to the section of the RDBMS documentation relating to error messages.
5. Click *OK*.

4.4.2 Defining and editing connections

A connection is a named set of parameters that defines how a Business Objects application accesses data in a database file. A connection links Web Intelligence to your middleware. You must have a connection to access data.

You must select or create a connection when you create a universe. You can modify, delete, or replace the connection at any time.

> **i Note**
>
> Connection objects have an additional administrator-defined security right called "Download connection locally". The administrator will need to define the security associated with the Connection, e.g. define who can download locally the (sensitive) connection information.

> **i Note**
>
> See the Data Access Guide for complete information on creating, modifying, and optimizing connections.

You can manage connections in the following ways:

How to manage the connection	Description
From the *Connection panel*	From the command in the menu: *Tools* > *Connections*. This panel displays the list of connections you can access: Personal, Shared and Secured in the CMS where you are logged. If the universe design tool has been started in Standalone mode, only Personal and Shared connections are displayed. The connections can be displayed as a flat list or as a hierarchy, where connections are displayed in sub-folders. You can delete, edit, and create new connections from this page.
From the *Universe parameters* dialog	From the *Definition* page of the *Universe Parameters* dialog (*File* > *Parameters* > *Definition*). You create a new connection when there is not an existing connection appropriate to the current universe. You can also edit connections.

A connection contains three elements:

- Data Access driver
- Connection and login parameters
- Connection type

Each element is described in the following sections:

4.4.2.1 About managing connection folders

You can manage connection folders with the universe design tool. Connection folders are displayed in any location where a secured connection is displayed or managed. You can manage connections in the following ways:

How to manage the connection	Description
The *Connection Panel*	From the command in the menu: *Tools* > *Connections*. This panel displays the list of connections you can access: Personal, Shared and Secured in the CMS where you are logged. If the universe design tool has been started in Standalone mode, only Personal and Shared connections are displayed. The connections can be displayed as a flat list or as a hierarchy, where connections are displayed in sub-folders. You can delete, edit, and create new connections from this page. Once validated, any actions run from the *Connection Panel* are automatically committed in the CMS.
Universe parameters dialog	From the *Definition* page of the *Universe Parameters* dialog box (*File* > *Parameters* > *Definition*). You create a new connection when there is not an existing connection appropriate to the current universe. You can also edit connections.

The actions you can perform depend on your user rights, but the available actions are as follows:

- Create a new connection
- Create a new connection folder
- Edit a connection or folder
- Delete a connection or empty folder
- Rename a connection or folder
- Edit a description of a connection
- View tehe properties of a connection
- Cut, copy and paste in edit mode.

4.4.2.2 Data Access driver

A Data Access driver is the software layer that connects a universe to your middleware.

Data Access drivers are shipped with Business Objects products. There is a Data Access driver for each supported middleware. When you install the universe design tool, your Data Access key determines which Data Access drivers are installed.

When you create a new connection, you select the appropriate Data Access driver for the RDBMS middleware that you use to connect to the target RDBMS.

4.4.2.3 Connection and login parameters

You configure the Data Access driver by specifying the following connection and login parameters.

Parameter	Description
Type	Type of connection, personal, shared, or secured.
Name	Identifying name for the connection.
User name	Your database user name. This is normally assigned to you by the database administrator.
Password	Your database password. This is normally assigned to you by the database administrator.
Use Single Sign-On when refreshing reports at view time	When selected, the user name and password used to access the CMS are automatically used as database login parameters. See the BusinessObjects Enterprise Administrator's Guide for information on setting up Single Sign-On.
Use database credentials associated with BusinessObjects user account	When selected the user is forced to enter a database user password associated with their BusinessObjects account to refresh a report. This is set at the Central Management Console level. Refer to BusinessObjects Enterprise Administrator's Guide for information on setting up this option.
Data source/Service	Data source or database name. If you are using an ODBC driver the data source name identifies the target database. If you are using a native driver, the database name identifies the target database.

> **i Note**
>
> You can create connections through ODBC to Excel files, and to text files in .csv format. So that Web Intelligence can use a universe based on a text file or an Excel file accessed through ODCB, you must edit the msjet.prm file for the connection.
>
> This file is located in the folder: `$INSTALLDIR$/BusinessObjects Enterprise 12.0/win32_x86/dataAccess/connectionserver/odbc` where `$INSTALLDIR$` is the directory in which your Business Objects applications are installed. In the `msjet.prm` file, change the `DB_TYPE` parameter as follows:
>
> From: `<Parameter Name='DB_TYPE'>MS Jet Engine</Parameter>`
>
> To: `<Parameter Name='DB_TYPE'>MS Jet</Parameter>`
>
> You must stop and restart the Business Objects Enterprise server after making this change. Note: If you are running the universe design tool on the same machine as your Web Intelligence server and you want to create additional universes based on text or Excel files after changing this value, you must reset the value to `<Parameter Name='DB_TYPE'>MS Jet Engine</Parameter>`

4.4.2.4 Connection type

The type of connection determines who can use the connection to access data. The universe design tool automatically stores all the connections that you create during a work session. The next time you launch a session, these connections will be available to you.

You can create three types of connections with the tool:

- Personal
- Shared
- Secured

Each connection type is described as follows:

Personal connections

Restricts access to data to the universe creator and the computer on which it was created.

Connection parameters are stored in the PDAC.LSI file located in the LSI folder in the Business Objects 12.0 folder in you user profile directory, for example:

```
C:\Documents and Settings\<user name>\Application Data\Business Objects\Business Objects 12.0\lsi
```

These parameters are static and cannot be updated.

Personal connections are unsecured in terms of Business Objects products security.

You do not use personal connections to distribute universes. You could use personal connections to access personal data on a local machine.

Shared connections

Allows access to data for all users. These connections are unsecured in terms of Business Objects products security.

Connection parameters are stored in the SDAC.LSI file located in the `lsi` folder in the Business Objects 12.0 folder in you user profile directory, for example:

```
C:\Documents and Settings\<user name>\Application Data\Business Objects\Business Objects 12.0\lsi
```

Secured connections

- Centralizes and controls access to data. It is the safest type of connection, and should used be to protect access to sensitive data.

- You can create secured connections with the universe design tool.
- You must use secured connections if you want to distribute universes through the CMS.
- Secured connections can be used and updated at any time.

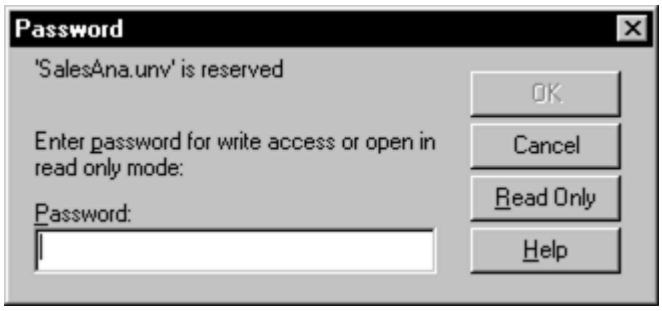
4.4.2.5 Setting passwords with personal and shared connections

You can set a password on any universe that has a personal or shared connection type. Using passwords, you can protect the universe from unauthorized users in an environment without a repository.

> **i** Note
>
> If you forget a password, you can not recover the universe file. You should keep a backup file of universe passwords.

There are two different options available for the password you can set:

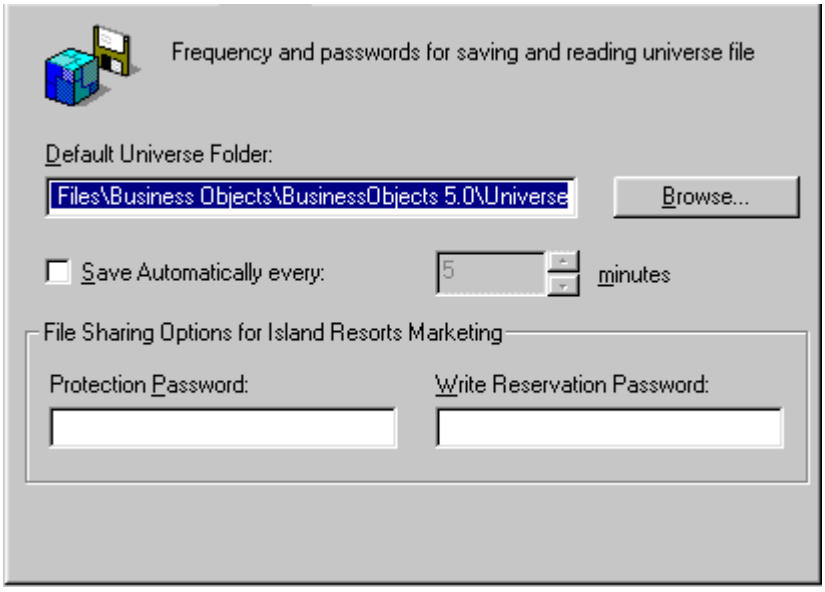
- Protection Password causes a dialog box to appear; it simply prompts the user to enter the password. If the password is correct, the universe is opened.
- Write Reservation Password causes the following dialog box to appear:



The user can then open the universe in read only mode, or in read-write mode by entering the correct password.

To set a password when using personal or shared connections:

1. Select *Tools* > *Options*
   The *Options* dialog box appears.
2. Click the *Save* tab.
   The *Save* page appears.



3. Type a password in the *Protection Password* or the *Write Reservation Password* text boxes. You can enter up to 40 alphanumeric characters.
4. Click *OK*.

4.4.2.6 Accessing the database after the user DBPass has changed

The BusinessObjects administrator can let a BusinessObjects user login (name and password) continue to access data after the database user password has been changed.

When the following parameters are set, a BusinessObjects user can continue to access data without contacting the BusinessObjects administrator, after the database password has been changed:

- In the *Central Management Console*, the *Enable and update user's Data Source Credentials at logon time* check box must be selected.
- In the universe design tool, on the *Define a new connection* page of the *New Connection* wizard, the *Use Database Credentials associated with BusinessObjects user account* and the *Use Single Sign On when refreshing reports at view time* check boxes must be selected.

When the check boxes described above are selected, the updated `DBUser` and `DBPass` parameters are automatically associated with the BusinessObjects user account.

> **i Note**
>
> `DBUser` and `DBPass` are static parameters, and must be updated in the Central Management Console. If the Database password is changed, it is recommended that the password is updated for each user account in the Central Management Console.

4.4.2.7 Defining a new connection

You can define a new database connection using the *New Connection* wizard. You access the wizard from:

- *Definition* page of the *Universe Parameters* dialog box (*File* > *Parameters* > *Definition*). You normally define a new connection when there is not an existing connection available for the data that the universe needs to access.
- *Connections* list (*Tools* > *Connections*).

You can use the connection wizard to set advanced and custom parameters for a connection. Refer to the Data Access Guide for complete information on creating, editing, and optimizing connections.

When you create the connection from the *Universe Parameters* dialog box, the *Universe Parameters* dialog box appears with the new connection listed in the *Connection* box.

When you create the connection from the Connections dialog box, the connection appears in the list.

Related Information

To start the New Connection wizard [page 407]

4.4.2.8 Viewing available connections

You can view all available stored connections in the Connections list. You can edit existing connections, and create new connections.

To view available connections:

1. Select *Tools* > *Connections*.
   The *Connections* list appears. It displays all the connections available to the current universe.
2. Click *Cancel* to close the dialog box.

You can edit connections from the *Connections* dialog box.

You can edit a secured connection only if you are working in online mode. Personal and Shared connections can be modified in any mode.

You cannot modify the name of an existing connection.

4.4.2.9 Editing a connection

To edit a connection:

1. Select *Tools* > *Connections*.
   The *Connections* list appears.

2. Click a connection name in the list of available connections.
3. Click *Edit*.
   The *Login* page for the connection appears.
4. Type a new data source, or database name in the Data Source or Service box if required.
5. Type modifications to login parameters as required.
6. Click *Next*.
   The *Perform a Test* page appears.
7. Click *Test Data Source* to verify the modified connection.
8. Click *Next* to move to the *Advanced* and *Custom* pages. You can modify parameters as required. You can also accept the default or existing values.
9. Click *Finish* from the *Custom* page to apply the changes to the connection.

4.4.2.10 Deleting a connection

You can delete connections from the Connections list. You can delete a secured connection only if you are working in online mode. Personal and Shared connections can be deleted in any mode.

To delete a connection:

1. Select *Tools* > *Connections*.
   The *Connections* list appears.
2. Select a connection name in the list.
3. Click *Remove*.
   A confirmation box appears.
4. Click *Yes*.
   The connection is removed from the list.

4.4.2.11 Adding a new connection

You can add a new connection from the *Connections* page by selecting *Tools* > *Connections*, clicking *Add*, and following the *Define a new connection* wizard. Full Instructions for following the connection wizard are in the section Editing a connection [page 80].

Using DSN connections

If the Information Design Tool or Universe Design Tool is using ODBC DSN for universe data sources, DSN with the same names should be created on the physical machine where server components are installed for the SAP BusinessObjects reporting applications that use the published universes.

Using DSN-less connections

The SAP BusinessObjects Business Intelligence platform 4.3 Connection Server drivers now support only the 64-bit applications. In order to use UDT, users have to install the Business Objects clients in 64-bit system instead of 32-bit system.

If the Universe Design Tool is using ODBC DSN-less connection, when the user chooses to connect using a connection string, the user must declare the driver and connection string in the connection wizard.

For more information on DSN connections and ODBC drivers, please refer to Installation of ODBC Drivers in the Data Access guide.

4.4.3 Setting universe summary parameters

The Summary page displays universe administration information. You can use this information to help you keep track of the development of the active universe.

The Summary page displays the following information:

Information	Description
Created	Universe creation date and the name of the creator.
Modified	Date of last modification and the name of the modifier.
Revision	Revision number which indicates the number of times the universe has been exported to the repository.
Comments	Information about universe for yourself or another designer.
Statistics	List of the number of classes, objects, tables, aliases, joins, contexts, and hierarchies contained in the universe.

4.4.3.1 To view and enter summary information

1. Select File>Parameters

   or

   Click the Parameters tool.

   The Universe parameters dialog box appears.
2. Click the Summary tab.

   The Summary page appears.
3. Type a comment in the Comment textbox.
4. Click OK.

4.4.4 Selecting strategies

A strategy is a script that automatically extracts structural information from a database or flat file. Strategies have two principle roles:

- Automatic join and cardinality detection (Join strategies)
- Automatic class, object, and join creation (Objects and Joins strategies)

Strategies can be useful if you want to automate the detection and creation of structures in your universe based on the SQL structures in the database.

> **i** Note
>
> Strategies that automate the creation of universe structures are not necessarily an essential part of universe design and creation. They can be useful if you are creating a universe quickly, allowing you to use metadata information that already exists in a database or database design tool. However, if you are building a universe by creating objects and joins that are based on relationships that come directly from a user needs analysis, then you will probably not use the automatic creation possibilities that strategies offer.

In the universe design tool you can specify two types of strategies:

Strategy	Description
Built-in strategy	Default strategy shipped with the tool. Built-in strategies can not be customized.
External strategy	User defined script that contains the same type of information as a Built-in strategy, but customized to optimize information retrieval from a database.

4.4.4.1 Selecting a strategy

To select a strategy:

1. Select *File* > *Parameters*.
   Or
   Click the *Parameters* tool.
   The *Universe Parameters* dialog box appears.
2. Click the *Strategies* tab.
   The *Strategies* page appears.
3. Select a strategy from the *Objects*, *Joins*, or *Tables* drop-down list boxes.
4. Click *OK*.

4.4.4.2 Using built-in strategies

Built-in strategies are default strategies that are shipped with the universe design tool. There are built-in strategies for all supported databases. These cannot be modified. Built-in strategies appear by default before external strategies in the strategy drop-down lists.

You can use built-in strategies for the following purposes:

Strategy	Used for...
Objects	Automatic creation of default classes and objects when tables are created in the table schema.*
Joins	• Automatic extraction of default joins when tables are created in the table schema.* • Automatic insertion of cardinality at join creation.* • Automatic detection of joins in table schema. When you select *Tools* > *Automated Detection* > *Detect Joins*, the universe design tool uses the strategy to automatically detect candidate joins. You can choose to implement the joins or not. • Automatic detection and insertion of cardinalities for existing joins in the table schema. When you select *Tools* > *Automated Detection* > *Detect Cardinalities*, the universe design tool uses the strategy to detect cardinalities for joins selected in the table schema.
Tables	Filtering information available for tables in the table browser.

* These automatic creation uses for strategies must be activated from the *Database* page of the *Options* dialog box.

Using the Objects strategy

The Objects strategies are used only for creating classes and objects automatically when you add a table to the table schema. To use this strategy you must activate it from the *Database* page of the *Options* dialog box. For more details see the section Using the automatic creation functions of a strategy [page 85].

Using the Joins strategy

The selected Joins strategy determines how the universe design tool automatically detects cardinalities and joins in your table schema.

Depending on your database, there can be one or more Join strategies in the list. For example, when using Oracle databases, you can specify a Join strategy to automatically detect joins based either on matching column names, or matching column number names.

If you do not select a strategy, the universe design tool uses the default Joins strategy which matches columns names to detect joins. The use of the selected join strategy to detect joins does not have to be activated. The strategy is always used when you choose to detect the joins or cardinalities in your table schema.

The Joins strategy is also used to automatically create joins and implement cardinality when joins are created. To use the automatic default creation functions of this strategy you must activate it from the *Database* page of the *Options* dialog box. For more details see the section Using the automatic creation functions of a strategy [page 85].

Using the Tables strategy

The selected table strategy reads the structure of database tables. Depending on the strategy, the strategy could determine what sort of information is shown in the table browser. For example, column data types and descriptions.

4.4.4.3 Using the automatic creation functions of a strategy

The automatic creation and insertion functions of strategies are not activated by default. To use these functions, you must select the *Default Creation* check box that corresponds to the strategy that you want to apply at object or join creation. These are listed on the *Database* page of the *Options* dialog box (*Tools* > *Options* > *Database*).

Each default creation option on the *Database* page is described as follows:

Option	When cleared	When selected
Extract joins with tables	Joins must be created manually. If you select *Tools* > *Automated Detection* > *Detect Joins*, then the universe design tool uses the strategy to detect joins and proposes candidate joins. You can choose to implement the candidate joins or not.	Retrieves tables with the joins that link them according to the selected Join strategy.

Option	When cleared	When selected
Detect cardinalities in joins	Cardinalities must be manually defined. If you select *Tools* > *Automated Detection* > *Detect Cardinalities*, then the universe design tool uses the strategy to detect and implement cardinalities for selected joins.	Detects and implements the cardinalities inherent in the joins at join creation.
Create default classes and objects from tables	Classes and objects must be created manually, either by creating directly in the *Universe* pane, or by dragging a table or column from the *Structure* pane to the *Universe* pane.	Default classes and objects are created in the Universe pane automatically when a table is added to the *Structure* pane. A class corresponds to the table name, and objects correspond to column names. It replaces all underscore characters (_) with spaces

To select default creation options for strategies:

1. Select *Tools* > *Options*.
   The *Options* dialog box appears.
2. Click the *Database* tab.
   The *Database* page appears.
3. Select the check box that corresponds to the default creation function for which you want to use the strategy.
4. Click *OK*.

4.4.4.4 Setting the number of rows to be viewed

From the *Database Options* dialog box, you can also indicate the maximum number of rows to be viewed from each table of the database. You can not restrict the physical number of rows retrieved from the database, but this modifies the default value for the number of rows that can be viewed at any one time when you view table or column values. This only applies to the rows returned in the universe design tool, and not for queries run in Web Intelligence.

To set the number of rows that can be viewed:

- Enter a value in the text box of the *Maximum Number of Rows Fetched* option. You can also click one or more times on the up or down arrow to increase or decrease the default value (100).

4.4.4.5 Using external strategies

An external strategy is a user defined SQL script that follows a defined output structure to perform customized automatic universe creation tasks. External strategies are stored in an external XML strategy file (<RDBMS>.STG). SQL scripts in this file appear in the drop down list on the Strategies page with the other strategies.

External strategies contain the same type of information as the built-in strategies, but are often customized to allow the universe design tool to retrieve a specific type of database information, or to optimize how information is retrieved from the database.

For complete information on defining external strategies, see the section Using external strategies to customize universe creation [page 371].

4.4.5 Indicating resource controls

The universe design tool offers a number of options that let you control the use of system resources.

> **i** Note
>
> If you are viewing this tab from the Restriction Preview dialog box, then the modified parameters that apply to the restriction appear in red.

4.4.6 What system resource options are available?

You can specify the following limitations on system resources:

Query limits	Description
Limit size of result set to a specified value	The number of rows that are returned in a query are limited to the number that you specify. This limits the number of rows returned, but does not restrict the RDBMS from processing all rows in the query. It only limits the number once the RDBMS has started to send rows.
Limit execution time to a specified value	Query execution time is limited to the number of minutes that you specify. This limits the time that data is sent to WebIntelligence, but does not stop the process on the database.
Limit size of long text objects to a specified value	You specify the maximum number of characters for long text objects. When this check box is not selected, the parameter is not activated. It is automatically set to the default maximum value (1000). To ensure that you allow results larger than the default, the check box must be selected, and a value entered.

4.4.7 To enter resource control information

1. Select File > Parameters.

   Or

Click the Parameters tool.

The Universe parameters dialog box appears.

2. Click the *Controls* tab.

   The *Controls* page appears.

3. Select a check box in the *Query Limits* group box.

   Type a value in the text box that corresponds to the selected Query Limit option. You can click the up and down arrows at the end of the text boxes to increase or decrease the value entered.

4. Click *OK*.

4.4.8 Limiting execution time for queries generating more than one SQL statement

The time limit that you specify for query execution is the total execution time for a query. If the query contains multiple SQL statements, then each statement is given an execution time equal to the total query execution time divided by the number of statements, so each statement in the query has the same execution time.

If one statement requires a lot more time than others to run, it may not complete, as its execution time will not correspond to its alloted execution time within the query.

When you specify an execution time limit for multiple SQL statements, you need to take into account the normal execution time of the single statement that takes the longest time to run, and multiply this value by the number of statements in the query.

4.4.9 Indicating SQL restrictions

You can set controls on the types of queries that end users can formulate from the *Query Pane* in Web Intelligence.

You can indicate controls for the following areas of query generation:

- Use of subqueries, operators, and complex operands in individual queries.
- Generation of multiple SQL statements.
- Prevent or warn about the occurrence of a cartesian product.

Each of these sets of controls is described in the following sections:

4.4.9.1 Query controls

You can set the following controls for individual queries:

Option	Description
Allow use of union, intersect and minus operators	Enables end users to combine queries using data set operators (union, intersect, and minus) to obtain one set of results.

4.4.9.2 Multiple SQL statements controls

You can set the following controls to determine how multiple SQL statements are handled:

Option	Description
Multiple SQL statements for each context	Enables end users to create queries that contain multiple SQL statements when using a context. Select this option if you have any contexts in the universe.
Multiple SQL statements for each measure	Splits SQL into several statements whenever a query includes measure objects derived from columns in different tables. See the section Using Multiple SQL Statements for Each Measure [page 230] for more information on using this option. If the measure objects are based on columns in the same table, then the SQL is not split, even if this option is checked.
Allow selection of multiple contexts	Enables end users to create queries on objects in more than one context and to generate one set of results from multiple contexts. If you are using contexts to resolve loops, chasm traps, fan traps, or any other join path problems, then you should clear this check box.

4.4.9.3 Cartesian product controls

A Cartesian product is a result set which contains all the possible combinations of each row in each table included in a query. A Cartesian product is almost always an incorrect result.

You can set the following controls for the production of a Cartesian product.

Option	Description
Prevent	When selected, no query that results in a cartesian product is executed.

Option	Description
Warn	When selected, a warning message informs the end user that the query would result in a Cartesian product.

4.4.9.4 Entering SQL restriction options

To enter SQL restriction options:

1. Select *File* > *Parameters*.
   Or
   Click the *Parameters* tool.
   The *Universe Parameters* dialog box appears.
2. Click the *SQL* tab.
   The *SQL* page appears.
3. Select or clear options in the *Query* and *Multiple Paths* group boxes.
4. Select a radio button in the *Cartesian Product* group box.
5. Click *OK*.

4.4.10 Indicating options for linked universes

The *Links* tab is used with dynamically linked universes, a subject covered in the Deploying universes [page 469] chapter.

4.4.11 Setting SQL generation parameters

In the universe design tool, you can dynamically configure certain SQL parameters that are common to most RDBMS to optimize the SQL generated in Web Intelligence products using the universe.

4.4.11.1 Using parameter (PRM) files in previous versions of the universe design tool

In versions prior to Designer 6.5, the SQL generation parameters used by a universe were maintained and edited in a separate file called a parameters (PRM) file. The values set in the PRM file applied to all universes using the associated data access driver defined for a connection.

Many of the SQL parameters that are used to optimize query generation are now controlled within an individual universe file. The PRM file is now no longer used for the query generation parameters that you can set in the universe design tool PRM files are still used for parameters that are database specific.

> **Note**
>
> See the *Data Access Guide* for more information on the PRM file for your data access driver. You can access this guide by selecting *Help* > *Data Access Guide*.

4.4.11.2 Setting the SQL parameters dynamically in the universe design tool

Many of the parameters common to most supported RDBMS middleware are available for editing in the *Parameters* tab in the universe parameters dialog box (*File* > *Parameters* > *Parameter*).

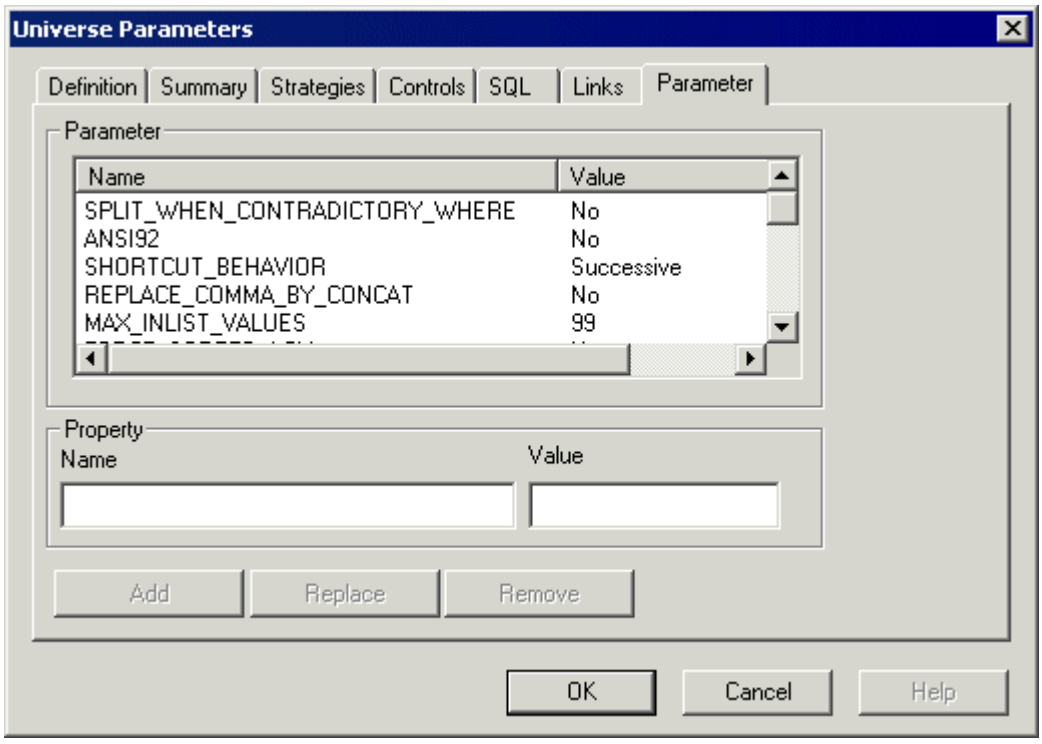
These parameters apply only to the active universe, and are saved in the UNV file. When you modify an SQL parameter for a universe in the universe design tool, the value defined in the universe design tool is used, and not the value defined in the PRM file associated with the data access driver for the connection.

4.4.11.3 Editing SQL generation parameters

You can modify the values for SQL parameters that determine SQL generation in products using the universe.

To edit SQL generation parameters:

1. Select *File* > *Parameters*.
   The *Parameters* dialog box appears.
2. Click the *Parameter* tab.
   The *Parameter* page appears.



3. Edit, add, or remove parameters as follows:

To...	then do the following:
Add a new parameter	○ Click any parameter in the list. ○ Type a name in the Name box ○ Type a value in the Value box. ○ Click *Add*. ○ The new value appears at the bottom of the list
Change name or value	○ Click a parameter in the list. ○ Type a new name in the Name box ○ Type a new value in the *Value* box. ○ Click *Replace*. The value is replaced by the new definition.
Delete a parameter	○ Click the parameter that you want to remove from the list. ○ Click *Delete*.

4. Click *OK*.

> **i** Note
>
> The SQL generation parameter values that you set in a universe, are only available to products using that universe.

4.4.12 About SQL Generation Parameters

The following reference describes the parameters that affect the generation of the query script. The parameters are listed alphabetical order in two groups:

- SQL parameters that you set in the user interface of the universe design tool or the information design tool. These are SQL parameters that are common to most data access drivers. Each parameter is valid for the universe in which it is set.
- SQL parameters that you set in the PRM files. These are connection-specific parameters that are listed in the data access parameter (PRM) file for the target data access driver.

Related Information

SQL Parameters that you set in the user interface [page 93]
SQL Parameters that you set in the PRM files [page 109]

4.4.13 Editing dynamic SQL generation parameters

1. Select File > Parameters.

   The Parameters dialog box appears.
2. Click the Parameter tab.

   The Parameter page appears.
3. Edit, add, or remove parameters as follows:

To...	Do the following...
Add a new parameter	Click any parameter in the list. Type a name in the Name box Type a value in the Value box. Click Add. The new value appears at the bottom of the list
Change name or value Click a parameter in the list.	Type a new name in the Name box Type a new value in the Value box. Click Replace. The value is replaced by the new definition.
Delete a parameter	Click the parameter that you want to remove from the list. Click Remove.

4. Click OK.

The SQL generation parameter values that you set in a universe, are only available to products using that universe.

See the Data Access Guide for more information on dynamic SQL parameters, and editing the PRM file for your data access driver. You can access this guide by selecting Help > Data Access Guide.

4.4.14 SQL Parameters that you set in the user interface

4.4.14.1 ANSI92

`ANSI92` = Yes|No

Values	`Yes/No`
Default	`No`

Description	Specifies whether the SQL generated complies to the ANSI92 standard. Yes: Enables the SQL generation compliant to ANSI92 standard. No: SQL generation behaves according to the PRM parameter OUTER_JOIN_GENERATION.

4.4.14.2 AUTO_UPDATE_QUERY

AUTO_UPDATE_QUERY = Yes|No

Values	Yes/No
Default	No
Description	Determines what happens when an object in a query is not available to a user profile. Yes: Query is updated and the object is removed from the query. No: Object is kept in the query.

4.4.14.3 BACK_QUOTE_SUPPORTED

BACK_QUOTE_SUPPORTED = Yes|No

Values	YES: the SQL uses backquotes. NO: the SQL does not use backquotes.
Default	YES No for OpenAccess database.
Description	Specifies if the SQL uses backquotes to enclose table or column names containing spaces or special characters.
Result	Table name=`My Table`

4.4.14.4 BEGIN_SQL

BEGIN_SQL = <String>

Values	String
Default	Empty string

Description	This is used to prefix SQL statements for accounting, prioritization, and workload management. This parameter applies to any SQL generation, including document generation and LOV queries.
	It is supported in Web Intelligence , LiveOffice, and QaaWS. But it is ignored by Crystal Reports.
	Example for Teradata:
	`BEGIN_SQL=SET QUERY_BAND='string' for transaction;`
	This parameter requires a string that contains one or more name-value pairs, separated by a semicolon, all inside single quotes. All SQL statements are prefixed with the parameter that follows BEGIN_SQL. The name-value pairs entered in this parameter are written in the GetQueryBandPairs system table.
	Example of three name-value pairs:
	`BEGIN_SQL=SET QUERY_BAND='UserID=Jones;JobID=980;AppID=TRM' for transaction;`
	You can also use the @Variable function as the value in the name-value pair, the returned value is enclosed in single quotes: `BEGIN_SQL=SET QUERY_BAND='USER='@Variable('BOUSER');Document='@Variable('DPNAME')';' for transaction;`

4.4.14.5 BLOB_COMPARISON

`BLOB_COMPARISON` = Yes|No

Values	`Yes/No`
Default	`No`
Can be edited?	No
Description	Species if a query can be generated with a `DISTINCT` statement when a BLOB file is used in the `SELECT` statement. It is related to the setting `No Duplicate Row` in the query properties. `Yes`: The `DISTINCT` statement can be used within the query. `No`: The `DISTINCT` statement cannot be used within the query even if the query setting `No Duplicate Row` is on.

4.4.14.6 BOUNDARY_WEIGHT_TABLE

BOUNDARY_WEIGHT_TABLE = Integer 32bits [0-9]

Values	Integer 32bits [0-9, or a negative integer]
Default	-1
Description	Allows you to optimize the FROM clause when tables have many rows. If the table size (number of rows) is greater than the entered value, the table is declared as a subquery: `FROM (SELECT col1, col2,....., coln, ,...., FROM Table_Name WHERE simple condition).` A simple condition is defined as not having a subquery. -1, 0, or any negative number means that this optimization is not used.
Limitations	Optimization is not implemented when: • The operator OR is in the query condition • Only one table is involved in the SQL • The query contains an outer join • No condition is defined on the table that is being optimized • The table being optimized is a derived table.

4.4.14.7 COLUMNS_SORT

COLUMNS_SORT = Yes|No

Values	Yes/No
Default	No
Description	Determines the order that columns are displayed in tables in the Structure pane. Yes: Columns are displayed in alphabetical order No: Columns are displayed in the order they were retrieved from the database

4.4.14.8 COMBINE_WITHOUT_PARENTHESIS

COMBINE_WITHOUT_PARENTHESIS= Yes|No

Values	Yes/No
Default	No
Description	Specifies whether or not to encapsulate a query with parentheses when it contains UNION, INTERSECT or MINUS operators. Used with RedBrick. Yes Removes the parentheses. No Leaves the parentheses.

4.4.14.9 COMBINED_WITH_SYNCHRO

COMBINED_WITH_SYNCHRO = Yes|No

Values	Yes | No
Default	No
Description	Specifies whether to allow a query to execute that contains UNION, INTERSECTION, or EXCEPT operators, and whose objects in each subquery are incompatible. Yes: Specifies that you do allow a query to execute that contains UNION, INTERSECTION and EXCEPT operators, and whose objects in each subquery are incompatible. This type of query generates synchronization (two blocks in the report). No: Specifies that you do not allow a query to execute that contains UNION, INTERSECTION and EXCEPT operators, and whose objects in each subquery are incompatible. When the query is executed the following error message is displayed: "This query is too complex. One of the subqueries contains incompatible objects." This is the default value.

4.4.14.10 COMPARE_CONTEXTS_WITH_JOINS

COMPARE_CONTEXTS_WITH_JOINS = Yes|No

Values	Yes | No
Default	Yes

Description	Specifies how contexts are compared. `Yes`: The system verifies that the contexts give the same joins. `No`: The system verifies that the contexts give the same sets of tables. This is the default value.

4.4.14.11 CORE_ORDER_PRIORITY

`CORE_ORDER_PRIORITY` = Yes|No

Values	`Yes|No`
Default	`No`
Description	This parameter applies to classes or objects that you add to a linked derived universe. This parameter does not apply to the classes or objects in the core universe or in the original derived universe. This parameter specifies in how you want the new classes and objects to be organized in the universe design tool. See also the `FIRST_LOCAL_CLASS_PRIORITY` parameter. `Yes`: Specifies that classes and objects are organized as follows: • First core universe class Core universe objects Any derived universe objects belonging to first core universe class • Second core universe class Core universe objects Any derived universe objects belonging to second core universe class • Other core universe classes... • Derived universe classes and objects `No`: Specifies that classes and objects follow the original order defined in the derived universe. This is the default value.

4.4.14.12 CORRECT_AGGREGATED_CONDITIONS_IF_DRILL

> ⚠ Caution
>
> This parameter is now deprecated.

`CORRECT_AGGREGATED_CONDITIONS_IF_DRILL` = Yes|No

Values	`Yes` | `No`
Default	`No`
Description	Applies to Desktop Intelligence only. Specifies whether Desktop Intelligence can aggregate measures in queries and conditions. `Yes`: Desktop Intelligence can aggregate measures separately in the main query and the condition, if the query is drill enabled. `No`: Desktop Intelligence cannot aggregate measures separately in the main query and the condition, if the query is drill enabled.

4.4.14.13 CUMULATIVE_OBJECT_WHERE

`CUMULATIVE_OBJECT_WHERE` = Yes|No

Values	`Yes` | `No`
Default	`No`

Description	This parameter applies to filtered objects only. Specifies how to combine the objects `WHERE` clause with the query condition on those objects.
	`Yes`: Specifies that `WHERE` clauses are combined with the main query condition with the `AND` operator.
	`No` : Specifies that the object's `WHERE` clause is combined with the condition for this object.
	Example:
	If the condition is find all French clients different from John or American cities different from New York, the SQL is:
	`Yes`:
	`(customer.first_name <> 'John') OR (city.city <> 'New York AND customer_country.country = 'France' AND city_country.country = 'USA'`
	`No`:
	`(customer.first_name <> 'John' AND customer_country.country = 'France' ) OR (city.city <> 'New York' AND city_country.country = 'USA' )`

4.4.14.14 DECIMAL_COMMA

DECIMAL_COMMA = Yes|No

Values	Yes|No
Default	No
Description	Specifies that Business Objects products insert a comma as a decimal separator when necessary. Yes: Business Objects products insert a comma as a decimal separator when necessary. No: Business Objects products do not insert a comma as a decimal separator. This is the default value.

4.4.14.15 DISABLE_ARRAY_FETCH_SIZE_OPTIMIZATION

DISABLE_ARRAY_FETCH_SIZE_OPTIMIZATION = Yes|No

Values	Yes/No
Default	No
Description	An optimization algorithm can be used to optimize the size of the returned arrays instead of using the default setting. No: All queries run on the universe will benefit from the optimization. Yes: Queries use the default value set. This parameter also applies to OLAP connections.

4.4.14.16 DISTINCT_VALUES

DISTINCT_VALUES = GROUPBY|DISTINCT

Values	GROUPBY|DISTINCT
Default	DISTINCT

Description	Specifies whether SQL is generated with a `DISTINCT` or `GROUP BY` clause in a list of values and Query pane when the option "Do not retrieve duplicate rows" is active. `DISTINCT`: The SQL is generated with a `DISTINCT` clause, for example; `SELECT DISTINCT` cust_name `FROM Customers` `GROUPBY`: The SQL is generated with a `GROUP BY` clause, for example; `SELECT` cust_name `FROM` Customers `GROUP BY` cust_name

4.4.14.17 END_SQL

`END_SQL` = String

Values	String
Default	<empty string>
Description	The statement specified in this parameter is added at the end of each SQL statement.
Example	• For a SAP HANA data source, you can pass the user name of a user executing a query from the universe to SAP HANA by inserting the @Variable('BOUSER') parameter as follows: `END_SQL=-- @Variable('BOUSER')` • For IBM DB2 databases, you can use the following: `END_SQL=FOR SELECT ONLY`. The server will read blocks of data much faster. • Another example: `END_SQL='write ' UNVID To Usage_Audit.Querieded_universe` writes a universe id to an audit table, which can be used to record other data such as user and tables queried.

4.4.14.18 EVAL_WITHOUT_PARENTHESIS

`EVAL_WITHOUT_PARENTHESIS` = Yes|No

Values	`Yes|No`
Default	`No`

Description	By default, the function `@Select`(Class\object) is replaced by the `SELECT` statement for the object <Class\object> enclosed within brackets. For example, when combining two `@Select` statements, `@Select`(objet1) *`@Select`(objet2). If the SQL(object1) = A-B and SQL(object2) =C, then the operation is (A-B) * (C). You avoid the default adding of brackets by setting `EVAL_WITHOUT_PARENTHESIS = Yes`. The operation is then A - B * C. `Yes`: Brackets are removed from the `SELECT` statement for a function `@Select`(Class\object) `No`: Brackets are added around the Select statement for the function `@Select`(Class \object).

4.4.14.19 FILTER_IN_FROM

`FILTER_IN_FROM` = Yes|No

Values	`Yes` | `No`
Default	`No`
Description	Determines if query conditions are included in the `FROM` Clause. This setting is only applicable if the other universe parameter setting `ANSI92` is set to `Yes`. `Yes`: When editing an outer join, the default behavior property selected in the drop down list box of the Advanced Join properties dialog box in the universe design tool, is set to "All objects in `FROM`". `No`: When editing an outer join, the default behavior property selected in the drop down list box of the Advanced Join properties dialog box in the universe design tool is set to "No object in `FROM`".

4.4.14.20 FIRST_LOCAL_CLASS_PRIORITY

`FIRST_LOCAL_CLASS_PRIORITY` = Yes|No

Values	`Yes` | `No`
Default	`No`

Description	This parameter only applies to Web Intelligence and Crystal Reports. Only taken into account when `CORE_ORDER_PRIORITY=Yes`. `Yes`: Classes in the derived universe are listed first. `No`: Objects and sub classes from the derived universe appear after those of the core universe.

4.4.14.21 FORCE_SORTED_LOV

`FORCE_SORTED_LOV` = Yes|No

Values	`Yes` | `No`
Default	`No`
Description	Retrieves a list of values that is sorted. `Yes`: Specifies that the list of values is sorted. `No`: Specifies that the list of values is not sorted.

4.4.14.22 INNERJOIN_IN_WHERE

`INNERJOIN_IN_WHERE` = Yes|No

Values	`Yes` | `No`
Default	`No`. You must manually add the parameter to activate it.
Description	Allows you to force the system to generate SQL syntax with all the inner joins in the `WHERE` clause when `ANSI92` is set to yes. This is only possible if a query contains only inner joins (Does not contain FULL OUTER, RIGHT OUTER, or LEFT OUTER joins). `Yes`: If `ANSI92` is set to yes, the system generates `ANSI92` join syntax in the `FROM` clause except when the query contains only inner joins. In this case, the inner joins go into the `WHERE` clause. `No`: If `ANSI92` is set to `Yes`, the system generates `ANSI` 92 join syntax in the `FROM` clause.

4.4.14.23 JOIN_BY_SQL

JOIN_BY_SQL = Yes|No

Values	Yes | No
Default	No
Description	Specifies how multiple SQL statements are handled. Multiple statements can be combined (provided that the database permits this). Yes: Specifies that multiple SQL statements are combined. No: Specifies that multiple SQL statements are not combined. This is the default value.

4.4.14.24 MAX_INLIST_VALUES

MAX_INLIST_VALUES = [0-99]

Values	Integer: min-1, max depends on DB
Default	-1
Description	Allows you to set the maximum number of values you may enter in a condition when you use the IN LIST operator. 99: Specifies that you may enter up to 99 values when you create a condition using the IN LIST operator. The maximum authorized value you may enter depends on your database. The value of -1 means that there is no restriction on the number of values returned, except that imposed by the database.

4.4.14.25 OLAP_UNIVERSE

OLAP_UNIVERSE = Yes|No

Values	Yes | No
Default	No default value.

Description	Indicates if an OLAP universe is used. When the universe design tool uses an OLAP universe, the value is set to `Yes` and the parameter is visible in the SQL parameters list. When the universe is not an OLAP universe, the parameter is not visible in the SQL parameters list. `Yes`: The universe is an OLAP universe. `No`: The universe is not an OLAP universe.

4.4.14.26 PATH_FINDER_OFF

Parameter is not listed by default. You must add the parameter manually to the list and set a value.

`PATH_FINDER_OFF`= Yes|No

Values	`Yes` | `No`
Default	No default. You must manually enter the parameter.
Description	Used for HPIW because the join generation is done by the database. `Yes`: Joins are NOT generated in the query. `No`: Joins are generated in the query. This is the default behavior.

4.4.14.27 REPLACE_COMMA_BY_CONCAT

`REPLACE_COMMA_BY_CONCAT`= Yes|No

Values	`Yes` | `No`
Default	`No`
Description	In previous versions of the universe design tool, a comma could be used to separate multiple fields in an object Select statement. The comma was treated as a concatenation operator. For universes that already use the comma in this way you can set `REPLACE_COMMA_BY_CONCAT` to `No` to keep this behavior. In the current version of the universe design tool, this parameter is set to `Yes` by default, so that any expressions using a comma in this way are automatically changed to use concatenation syntax. `Yes`: Comma is replaced by the concatenation expression when multi field object is found. `No`: Keep the comma as it is.

4.4.14.28 SELFJOINS_IN_WHERE

SELFJOINS_IN_WHERE = Yes|No

Values	Yes|No
Default	No
Description	Self-joins are usually included in the FROM clause. This allows you to force the system to generate SQL syntax with all the conditions of a self-join in the WHERE clause. The ANSI92 parameter must be set to Yes for this parameter to be taken into account. You must manually add the parameter to the list to activate it. Yes: The conditions of a self-join go in the WHERE clause of the SQL query. No: The syntax for self-joins is generated according to the ANSI 92 convention, and conditions for a self-join go in the ON clause of the table join definition in the FROM clause of the SQL query.

4.4.14.29 SHORTCUT_BEHAVIOR

SHORTCUT_BEHAVIOR = Global|Successive

Values	Global|Successive
Default	Successive
Description	Specifies how shortcut joins are applied. This parameter was formerly listed as GLOBAL_SHORTCUTS in the PRM files. The values have been changed to Global for Yes, and Successive for No. Global: Specifies that shortcut joins are considered one by one. A shortcut join is applied only if it really bypasses one or more tables, and if it does not remove a table from the join path used by a following shortcut join. Successive: Specifies that all shortcut joins are applied. Note: If it generates a Cartesian product, no shortcut joins are applied.

4.4.14.30 SMART_AGGREGATE

SMART_AGGREGATE = Yes|No

Values	`Yes` | `No`
Default	`No`
Description	Determines how aggregate tables are used for smart measures that are based on an aggregate tables. This ensures that a universe object based on a ratio is correctly aggregated. By default the system takes the advantage of the pre-calculated values from the aggregated tables, if these table are not consistent during time (different time periods), you use this parameter to ensure the most detailed aggregate tables are used. This parameter is not visible in the universe parameter list (by default not activated). The universe designer must manually insert it in the parameter list before activating it (value `Yes`). `Yes`: Any additional grouping set query should be based on the aggregate table of the initial query for the smart measure based on aggregate table. `No`: The system takes the most appropriate aggregate table.

4.4.14.31 STORED_PROC_UNIVERSE

`STORED_PROC_UNIVERSE` = Yes|No

Values	`Yes` | `No`
Default	`No`
Description	This value is automatically set to `Yes` when you create a universe that contains stored procedures. Do not change this value manually. `Yes`: The universe you are creating/editing contains stored procedures. `No`: The universe does not contain stored procedures.

4.4.14.32 THOROUGH_PARSE

`THOROUGH_PARSE` = Yes|No

Values	`Yes` | `No`
Default	`No`

Description	Specifies the methodology used for default Parsing in the Query pane and individual object parsing. Yes: `PREPARE, DESCRIBE`, and `EXECUTE` statements are used to parse SQL for objects. `Prepare+DescribeCol+Execute` `No: PREPARE` and `DESCRIBE` statements are used to parse SQL for objects.

4.4.14.33 TRUST_CARDINALITIES

`TRUST_CARDINALITIES` = Yes|No

Values	`Yes|No`
Default	`No`
Description	Allows you to optimize the SQL in case of inflated results. `Yes`: For queries that include a measure, all conditions that inflate the measure and do not appear in the Result Objects, are transformed to sub queries to ensure that tables that may return false results for the measure are not included in the query. `No`: No optimization is implemented.

4.4.14.34 UNICODE_STRINGS

`UNICODE_STRINGS` = Yes|No

Values	`Yes|No`
Default	`No`

Description	Specifies whether the current universe can manipulate Unicode strings or not. Only applies to Microsoft SQL Server and Oracle 9. If the database character set in the SBO file is set as Unicode, then it is necessary to modify the SQL generation to handle specific Unicode column types like NCHAR and NVARCHAR. Yes: Conditions based on strings are formatted in the SQL according to the value for a parameter UNICODE_PATTERN in the PRM file, for example for Microsoft SQL Server (sqlsrv.prm) :UNICODE_PATTERN=N$ The condition Customer_name='Arai ' becomes Customer_name=N'Arai'. Note: When you create a prompt with @Prompt syntax based on Unicode value, the data type should be 'U' not 'C' No: All conditions based on strings are formatted in the standard SQL. For example the condition Customer_name='Arai ' remains Customer_name='Arai'

4.4.15 SQL Parameters that you set in the PRM files

4.4.15.1 CASE_SENSITIVE

```
<Parameter Name="CASE_SENSITIVE">NO</Parameter>
```

Description	Specifies if the database is case-sensitive. This parameter is used with Oracle.
Values	YES: the database is case-sensitive. NO: the database is not case-sensitive.
Default	NO

4.4.15.2 CHECK_OWNER_STATE

```
<Parameter Name="CHECK_OWNER_STATE">NO</Parameter>
```

Description	Specifies if the SQL checks if the database supports table classification by owner name.

Values	YES: the SQL checks if the database supports table classification by owner name. NO: the SQL does not check if the database supports table classification by owner name.
Default	YES

4.4.15.3 CHECK_QUALIFIER_STATE

```
<Parameter Name="CHECK_QUALIFIER_STATE">NO</Parameter>
```

Description	Specifies if the SQL checks if the database supports table classification by qualifier.
Values	YES: the SQL checks if the database supports table classification by qualifier. NO: the SQL does not check if the database supports table classification by qualifier.
Default	YES

4.4.15.4 COMMA

```
<Parameter Name="COMMA">||' '||</Parameter>
```

Description	Specifies what database concatenation operator should be used to replace a comma for objects that have the following syntax: Tab.Col1, Tab.Col2. This parameter is used with all data access drivers.
Values	||' '|| +' '+
Default	||' '||
Result	Tab.Col1||' '||Tab.Col2

4.4.15.5 CONCAT

```
<Parameter Name="CONCAT">||</Parameter>
```

Description	Specifies the concatenation operator. The parameter is used with all data access drivers.
Values	double pipe (`||`) or plus sign (+)
Default	`||`

4.4.15.6 DATE_WITHOUT_QUOTE

```
<Parameter Name="DATE_WITHOUT_QUOTE">YES</Parameter>
```

Description	Specifies if dates are surrounded with single quotes in the SQL syntax. This parameter is used with MS Access.
Values	`YES`: dates are not surrounded by single quotes. `NO`: dates are surrounded by single quotes.
Default	`YES`

4.4.15.7 DELIMIT_LOWERCASE

```
<Parameter Name="DELIMIT_LOWERCASE"></Parameter>
```

Description	Specifies if lowercase identifiers are delimited with quotes.
Values	`YES`: the lowercase identifiers are delimited with quotes. `NO`: the lowercase identifiers are not delimited with quotes.

4.4.15.8 EXTERN_SORT_EXCLUDE_DISTINCT

```
<Parameter Name="EXTERN_SORT_EXCLUDE_DISTINCT">YES</Parameter>
```

Description	Specifies if the application generates a SELECT DISTINCT when a query contains an ORDER BY clause.

Values	YES: a SELECT DISTINCT is not generated when the query contains an ORDER BY clause. NO: a DISTINCT is generated when the query contains an ORDER BY clause.
Default	YES

4.4.15.9 GROUPBY_WITH_ALIAS

```
<Parameter Name="GROUPBY_WITH_ALIAS">YES</Parameter>
```

Description	Specifies if the database can create a GROUP BY clause that contains aliases in the SELECT statement.
Values	YES: it allows you to create a GROUP BY clause with aliases in the SELECT statement. NO: it does not let you create a GROUP BY clause with aliases in the SELECT statement.
Default	YES

4.4.15.10 IDENTIFIER_DELIMITER

```
<Parameter Name="IDENTIFIER_DELIMITER">"</Parameter>
```

Description	Specifies the following features: • Table or column names that contain spaces or special characters are enclosed within quotes if BACK_QUOTE_SUPPORTED parameter is activated. • Tables or column names regardless of their characters are enclosed within quotes if DELIMIT_IDENTIFIERS parameter is activated. To use this parameter, either BACK_QUOTE_SUPPORTED or DELIMIT_IDENTIFIERS must be set to YES. This is the default value of both parameters.
Values	" (double quote): table or column names that contain spaces or special characters are enclosed in double quotes. ' (single quote): table or column names that contain spaces or special characters are enclosed in single quotes. This value can only be used with Microsoft Access.
Default	"
Result	Table name="My Table"

4.4.15.11 IF_NULL

```
<Parameter Name="IF_NULL">NO</Parameter>
```

Description	Specifies a function that takes two parameters. If the first parameter returns NULL, the second parameter value is used.
Values	Database-dependent.
Default	Database-dependent.

4.4.15.12 OUTERJOINS_COMPLEX

```
<Parameter Name="OUTERJOINS_COMPLEX"></Parameter>
```

Description	Along with `OUTERJOINS_GENERATION`, this parameter controls the generation of outer join queries.
Values	`YES` `NO`

4.4.15.13 OUTERJOINS_GENERATION

```
<Parameter Name="OUTERJOINS_GENERATION">ANSI92</Parameter>
```

This parameter controls the default outer join generation behavior.

- Outer join generation conforms to the ANSI92 specification.
- Outer join generation remains the same as for previous versions of theuniverse design tool.

> **i Note**
>
> The PRM file `OUTERJOINS_GENERATION` parameter relates to the universe `ANSI92` setting in the following way:
>
> - If the PRM file `OUTERJOINS_GENERATION` parameter is set to `ANSI92` and the universe `ANSI92` setting is set to `NO`, the PRM parameter overrides the universe setting and outer joins conform to ANSI92 behavior.
> - If the PRM file `OUTERJOINS_GENERATION` parameter is set to `USUAL`, then the universe `ANSI92` setting takes precedence, and outer joins conform to ANSI92 depending on whether the universe `ANSI92` setting is `YES` or `NO`.

→ Remember

The ANSI92 value makes REVERSE_TABLE_WEIGHT parameter not useful for optimization of SQL generation. Outer joins that conform to ANSI92 behavior leads the order of the tables in the SQL sentence.

Description	Specifies the SQL syntax for outer joins. The value ANSI 92 generates an outer join in the FROM clause. Other values generate the outer join in the WHERE clause. When you modify this setting, you should check join properties to verify that the outer join expression is valid, and that the cardinalities are correct. ANSI92 does not support any manual customization in the join syntax.
Values	The primary values for OUTERJOINS_GENERATION are: • ANSI92: the default outer join behavior conforms to the ANSI92 standard no matter what the ANSI92 parameter value of the universe design tool. • No: outer joins are not supported. • USUAL: the default outer join behavior is the same as with previous versions of the universe design tool . This behavior is overridden if ANSI92 parameter of the universe design tool is set to YES. Other settings are available depending on the database. See the defaults below.
Default	ANSI_92: default value for Oracle, MS SQL Server 2005 and Sybase. DB2: default value for IBM DB2. FULL_ODBC: default value for Microsoft SQL Server. INFORMIX: default value for IBM Informix. INGRES: default value for Teradata. NO: default value for ODBC. USUAL: default value for HP Neoview, Netezza, IBM Red Brick and MS SQL Server 2000.

Examples of OUTERJOINS_GENERATION parameter settings

Setting = USUAL:

```
FROM T1, T2
WHERE T1.col1(+) = T2.col2
```

Setting = DB2:

```
FROM T2 LEFT OUTER JOIN T1
  ON T1.col1 = T2.col2
```

Setting = ODBC:

```
FROM {oj T1 LEFT OUTER JOIN T2 ON T1.col1=T2.col2}
 Where (T2.col3 = T3.col1)
```

Setting = INFORMIX

```
FROM T2
OUTER T1
WHERE T1.col1=T2.col2
```

Setting = FULL-ODBC

```
FROM {oj T1 RIGHT OUTER JOIN T2 ON T2.col2=T1.col1
 T2 INNER JOIN 3 on T2.col3 = T3.col1}
```

Setting = ANSI_92:

```
SELECT DISTINCT
  t1.col1,
  t2.col2
FROM
  (t1 RIGHT OUTER JOIN t2 ON (t1.col1=t2.col2) )
```

Using OUTERJOINS with Oracle

The default OUTERJOINS_GENERATION setting can affect the behavior of existing universes irrespective of the universe-level setting for the ANSI92 parameter.

To set that your existing Oracle universes behave as with the previous universe design tool versions:

1. In the PRM file, ensure that the OUTERJOINS_GENERATION parameter is set to USUAL.
2. In the PRM file , set the LEFT_OUTER and RIGHT_OUTER parameters to $(+)

For more information about universe SQL parameters and PRM files in previous versions of the universe design tool, see the *Designer Guide*.

4.4.15.14 OVER_CLAUSE

```
<Parameter Name="OVER_CLAUSE">YES</Parameter>
```

Description	Allows SAP BusinessObjects applications to include RISQL functions when generating SQL. The supported RISQL functions for the database are listed in the `ANALYTIC_FUNCTIONS` parameter.
Values	`YES`: applications can include RISQL functions when generating SQL. `NO`: applications cannot include RISQL functions when generating SQL.
Default	`YES`

4.4.15.15 OWNER

```
<Parameter Name="OWNER">YES</Parameter>
```

Description	Specifies if the database supports the owner name as prefix for tables.
Values	`YES`: the database supports prefixing tables with the owner name. `NO`: the database does not support prefixing tables with the owner name.
Default	`YES`

4.4.15.16 PREFIX_SYS_TABLE

```
<Parameter Name="PREFIX_SYS_TABLE">RBW_</Parameter>
```

```
<Parameter Name="PREFIX_SYS_TABLE">MSys</Parameter>
```

Description	Specifies if the system tables are displayed in the universe design tool .
Values	`MSys`: the MS Access system tables are hidden in the universe design tool table browser. `RBW_` : the IBM Red Brick system tables are hidden in the Universe Designer table browser. no value: the database system tables are displayed in the universe design tool table browser.

Default	MSys: default value for MS Access RBW_: default value for IBM Red Brick

4.4.15.17 QUALIFIER

```
<Parameter Name="QUALIFIER">NO</Parameter>
```

Description	Specifies if the database supports the qualifier name as prefix for tables.
Values	YES: the database supports prefixing tables with the qualifier name. NO: the database does not support prefixing tables with the qualifier name.
Default	RDBMS-dependent.

4.4.15.18 QUOTE_OWNER

```
<Parameter Name="QUOTE_OWNER">YES</Parameter>
```

Description	Specifies if an owner name should be in single quotes. Used by IBM Informix only.
Values	YES: table names are prefixed by an owner name in single quotes. This is mandatory for an ANSI-compliant IBM Informix database. If not, IBM Informix converts the owner name to upper case characters. NO: table names are not prefixed by an owner name in single quotes.
Default	YES
Result	SELECT Alias.col (<Alias> is a local Alias) FROM 'Owner'.table.col Alias

4.4.15.19 REFRESH_COLUMNS_TYPE

```
<Parameter Name="REFRESH_COLUMNS_TYPE">O</Parameter>
```

Description	Specifies how columns are refreshed.
Values	O: columns are refreshed by owner name. Q: columns are refreshed by qualifier name. T: columns are refreshed by table name.
Default	O: default value for Oracle Q: default value for IBM Red Brick, Sybase, MS SQL Server and MS Access

4.4.15.20 REMOVE_SEMICOLONS

```
<Parameter Name="REMOVE_SEMICOLONS"></Parameter>
```

Description	Instructs the Query Panel of SAP BusinessObjects applications whether to remove semicolons in freehand SQL.
Values	YES: the Query Panel removes semicolons. NO: the Query Panel does not remove semicolons.

4.4.15.21 REVERSE_TABLE_WEIGHT

```
<Parameter Name="REVERSE_TABLE_WEIGHT">YES</Parameter>
```

Description	Specifies in which order tables are to be generated. This parameter is used with Oracle. This parameter can also be used with some other databases, possibly with the `YES` and `NO` reversed. **i Note** This parameter is not supported by Teradata. **→ Remember** If `OUTERJOINS_GENERATION` parameter is set to `ANSI92` or if the universe `ANSI92` setting is set to `YES`, then `REVERSE_TABLE_WEIGHT` parameter does not affect the optimization of SQL generation.
Values	`YES`: tables are generated from the smallest to the largest. `NO`: tables are generated from the largest to the smallest.
Default	`YES`

4.4.15.22 UNICODE_PATTERN

```
<Parameter Name="UNICODE_PATTERN">UNISTR($)</Parameter>
```

Description	Only applies when the universe SQL generation parameter `UNICODE_STRINGS` is set to `YES`. All conditions based on strings are then formatted with this string value. This is used with Microsoft SQL Server and Oracle only.
Values	`N$`: for Microsoft SQL Server `UNISTR($)`: for Oracle

4.4.15.23 USER_INPUT_DATE_FORMAT

```
<Parameter Name="USER_INPUT_DATE_FORMAT">'dd-MM-yyyy HH:mm:ss'</Parameter>
```

Description	Specifies the default date and hour formats generated in the WHERE clause of a SQL statement.
Values	`{\d 'yyyy-mm-dd'}`: default date format with ODBC. `'DD-MM-YYYY HH:MM:SS'`: default date and hour formats with Oracle. `'MM/DD/YYYY'`: default date format with IBM Informix. `'yyyy-mm-dd HH:mm:ss'`: default date and hour formats with Microsoft SQL Server and for most IBM DB2 servers. `'mm/dd/yyyy hh:m:s am/pm'`: default date and hour formats with Sybase. `'yyyy-mm-dd'`: default date format with a Sybase gateway. **Note** If you need to use time or timestamp variables with ODBC, you must replace the default date format value with `{\t 'hh:mm:ss'}` or `{\t\s 'yyyy-mm-dd hh:mm:ss'}` in the `odbc.sbo` file.
Default	See values above.

4.4.15.24 USER_INPUT_NUMERIC_SEPARATOR

```
<Parameter Name="USER_INPUT_NUMERIC_SEPARATOR">.</Parameter>
```

Description	Specifies the default decimal separator that is used in the generated SQL script.
Values	'.' (period)
Default	'.'

4.4.15.25 DELIMIT_IDENTIFIERS

```
<Parameter Name="DELIMIT_IDENTIFIERS">YES</Parameter>
```

Description	Specifies if database identifiers can be quoted. Identifiers are quoted using the delimiter specified in the `IDENTIFIER_DELIMITER` parameter.
Values	`YES`: identifiers can be quoted. `NO`: identifiers cannot be quoted.
Default	`YES`

Result	Table name="my_table"

4.4.15.26 EXT_JOIN_INVERT

```
<Parameter Name="EXT_JOIN_INVERT">YES</Parameter>
```

Description	Specifies how to display an outer join symbol in a join expression. This parameter is used with IBM DB2, IBM Informix, Oracle, and Teradata.
Values	`YES`: when you click an *Outer join* check box in the *Edit Join* dialog box of Universe Designer, the outer join symbol appears reversed in position in a join expression. `NO`: when you click an *Outer join* check box in the *Edit Join* dialog box of Universe Designer, the outer join symbol appears on the same side on which you created the outer join.
Default	`YES`

4.4.15.27 KEY_INFO_SUPPORTED

```
<Parameter Name="KEY_INFO_SUPPORTED">YES</Parameter>
```

Description	Specifies if you can retrieve primary and secondary key definitions from the database.
Values	`YES`: the database lets you retrieve primary and secondary key definitions from the database. This parameter enables Universe Designer to display the keys in the *Structure* window. `NO`: the database does not let you retrieve primary and secondary key definitions from the database.
Default	`YES`

4.4.15.28 ORDER_BY_STRINGS

```
<Parameter Name="ORDER_BY_STRINGS">YES</Parameter>
```

Description	Specifies if the database is capable of correctly processing an ORDER BY clause based on a string column. This parameter corresponds to the ORDERBYSTRINGS capability of SAP BusinessObjects Data Federator. If the database cannot do the processing, Data Federator Query Server performs the sort.
Values	`YES`: the database can perform the sort processing. `NO`: the database cannot perform the sort processing.

5 Creating a schema with tables and joins

5.1 Overview

This chapter describes how you can create a schema that contains all the SQL structures necessary to build the objects that Web Intelligence users use to build reports. These SQL structures include tables, columns, joins, and database functions. Building a correct schema is the basis for building a universe that meets all its end user reporting requirements.

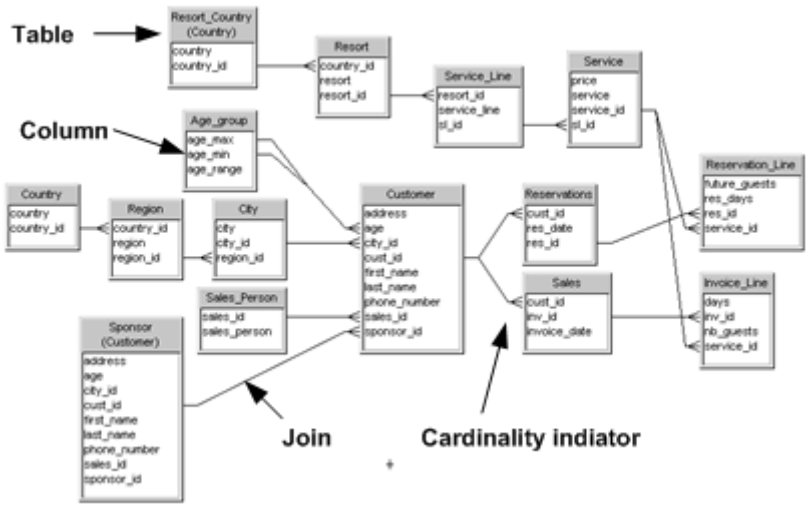
5.2 What is a schema?

A schema is a graphical representation of database structures. In the universe design tool, you create a schema for the part of the database that your universe represents.

The schema contains tables and joins. The tables contain columns that you eventually map to objects that end users use to create reports. The joins link the tables so that the correct data is returned for queries that are run on more than one table.

You design the schema in the *Structure* pane by selecting tables from the target database using the *Table Browser*. You create joins to link the tables. When you have designed the schema for your universe, you can verify the schema using an automatic integrity check.

A schema for the example Beach universe appears as follows:



5.2.1 Schema design is the basis for a successful universe

Good schema design is essential to good universe design. You populate the schema with tables based on the columns that correspond to the objects that end users need to create reports. These objects should be defined from a user needs analysis. You should be looking at the database for tables that allow you to create these necessary objects.

5.2.2 Schema design and the universe creation process

Creating a schema is the first phase of the implementation stage of the universe development cycle. The user analysis and planning phases can all be done without using the universe design tool; however, creating your schema is the first step using the tool to build your universe.

The following list indicates where the schema design phase appears in a typical universe development cycle (Implementation, step 1):

- Preparation

1. User needs analysis
2. Planning

- Implementation using the universe design tool

1. Design and test schema
2. Build and test universe objects
3. Deploy universe using repository

- Maintenance

1. Update and maintain universe based on changes in user requirements or data source

5.2.3 What are the stages of schema design?

This chapter covers the following stages of schema design:

- Inserting and organizing tables.
- Creating joins and setting cardinalities
- Resolving join problems such as loops, chasm traps, and fan traps.
- Testing the integrity of your schema.

5.3 Inserting tables

You start designing a schema by selecting tables from the target database and inserting symbols that represent the tables in the *Structure* pane. In the universe design tool, the table symbols are referred to simply as tables.

You use the *Table Browser* to select insert tables into your schema. The *Table Browser* is an independent window that shows a tree view of the tables available in the target database.

> **i** Note
>
> Before selecting tables, you can indicate strategies that you wish to use to help create your universe. For more information on this topic, see Selecting strategies [page 83].

5.3.1 Using the Table Browser

The *Table Browser* is an independent window that shows a tree view of the tables and columns in your target database. You use the *Table Browser* to view and select tables in your database that you want to insert into your schema. You expand the node next to a table name to display the columns for the table.

5.3.1.1 Activating the Table Browser

The *Table Browser* is not visible by default. You must activate the *Table Browser* when you want to add tables to the *Structure* pane. You can activate the *Table Browser* using any of the methods listed below.

To activate the *Table Browser*:

- Select *Insert* > *Tables*.
  Or
- Double-click an empty space in the *Structure* pane.
  Or
- Click the *Table Browser* button.
  The *Table Browser* window appears in the *Structure* pane.

5.3.1.2 Inserting Tables From the Table Browser

You can use any one of the following methods to insert one or multiple tables using the *Table Browser*:

Inserting a single table

To insert a single table:

- Click a table and click the *Insert* button.
  Or
- Right-click a table and select *Insert* from the contextual menu.
  Or
- Double-click a table.

Or

- Click a table and drag it into the *Structure* pane.
  The table appears in the *Structure* pane.

Inserting multiple tables

To insert multiple tables:

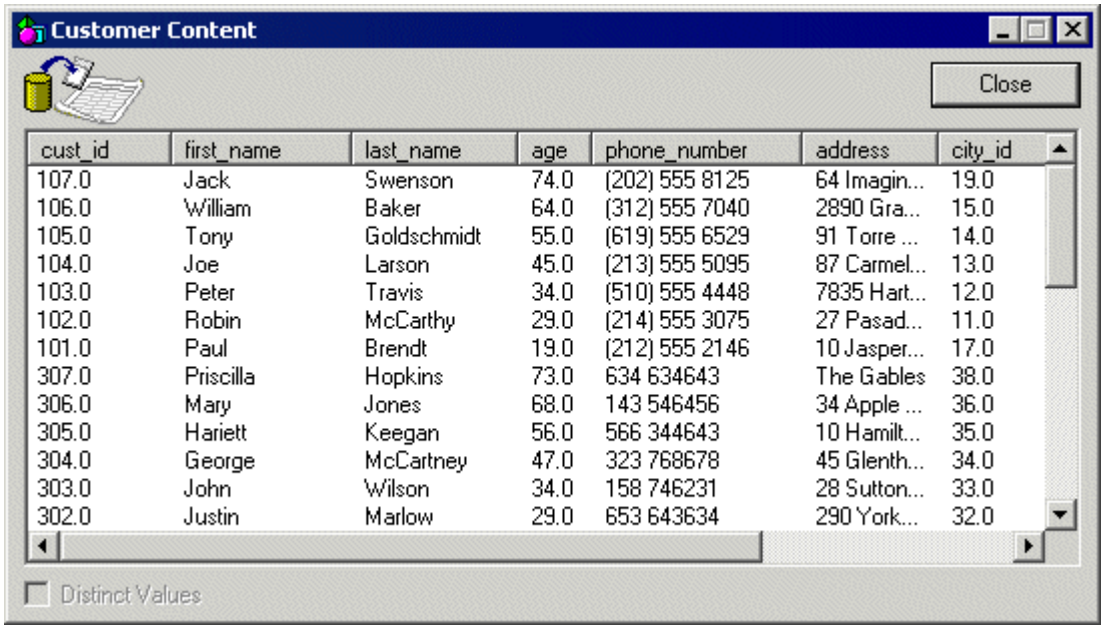
1. Hold down CTRL while you click individual tables.
   Or
2. Hold down SHIFT while you click the first table and last table in a continuous block of tables.
   Multiple tables are selected.
3. Click the *Insert* button.
   Or
   Drag the tables into the *Structure* pane.
   Or
   Right click the selected tables and select Insert form the contextual menu.
   Each table including all of its columns appears in the *Structure* pane. In the *Table Browser* any table that you insert in the universe is displayed with a check mark beside its name.

5.3.1.3 Viewing data from the Table Browser

You can use the *Table Browser* to view the data contained in a table, or in an individual column.

To view data from the *Table Browser* :

1. Right-click a table in the *Table Browser*
   Or
   Expand a table node in the *Table Browser* and right click a column for the table.
2. Select *View Table Values* from the contextual menu.
   Or
   Select *View Column Values* from the contextual menu.
   A box appears listing the data contained in the table or column.



→ Tip

If columns are to narrow to see complete row values, you can widen columns by pressing the key combination `CTRL-SHIFT` and `+`.

5.3.1.4 Optimizing Table Browser Performance

The time taken for a table to be inserted in the *Structure* pane from the *Table Browser* can vary depending on the following factors:

Table insertion slow because...	Optimize table insertion by...
There are a large number of tables in your database. The universe design tool queries the system catalog, so when the catalog is very large, retrieving tables can be slow.	Building a data warehouse using the tables that you want to insert in a separate database account. Create a connection to the new warehouse.
You are automatically inserting joins and checking cardinalities with the tables that you are inserting.	Inserting tables only. You do this as follows: 1. Select *Tools* > *Options*. The Options dialog box appears. 2. Click the database tab. The Database page appears. 3. Clear the following check boxes: *Extract Joins With Tables* and *Detect Cardinalities in Joins*. 4. Click *OK*.

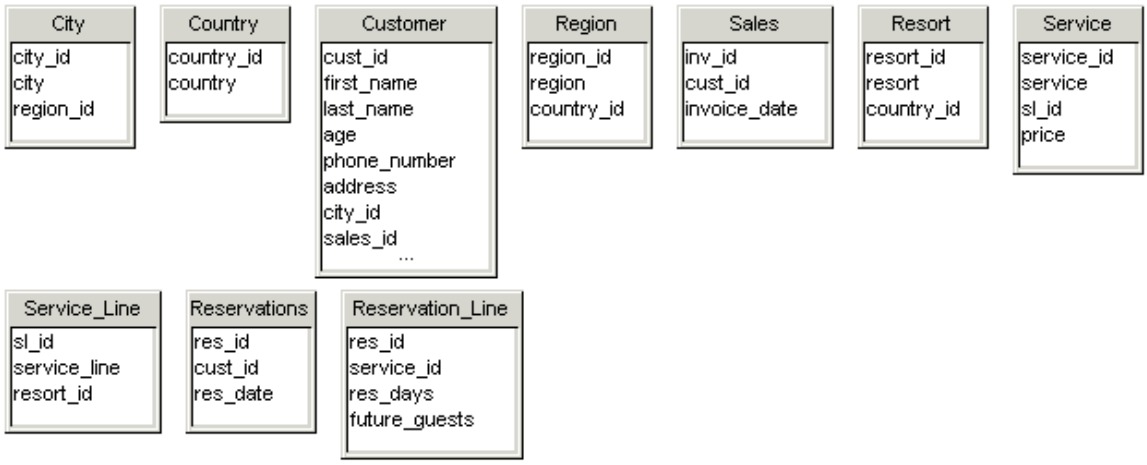
5.3.2 Arranging Tables in the Structure Pane

You can automatically arrange your tables in the *Structure* pane to tidy up your initial schema before you start manually rearranging the tables to create your joins.

5.3.2.1 Automatically arranging tables in the Structure pane

To automatically arrange tables:

- Select *View* > *Arrange Tables*.
  The tables are arranged in an orderly manner.



5.4 Using derived tables

Derived tables are tables that you define in the universe schema. You create objects on them as you do with any other table. A derived table is defined by an SQL query at the universe level that can be used as a logical table in the universe design tool.

Derived tables have the following advantages:

- Reduced amount of data returned to the document for analysis.
  You can include complex calculations and functions in a derived table. These operations are performed before the result set is returned to a document, which saves time and reduces the need for complex analysis of large amounts of data at the report level.
- Reduced maintenance of database summary tables.
  Derived tables can, in some cases, replace statistical tables that hold results for complex calculations that are incorporated into the universe using aggregate awareness. These aggregate tables are costly to

maintain and refresh frequently. Derived tables can return the same data and provide real time data analysis.

Derived tables are similar to database views, with the advantage that the SQL for a derived table can include prompts.

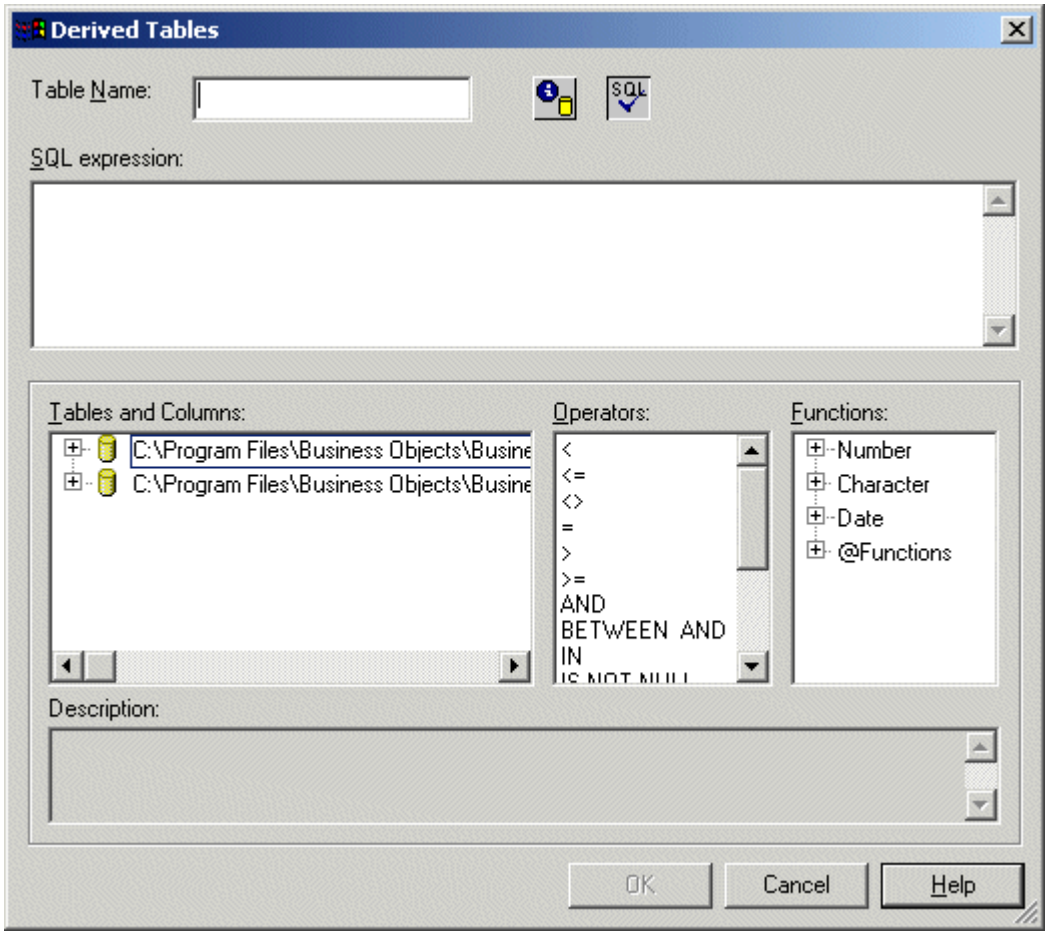
5.4.1 Adding, editing, and deleting derived tables

Derived tables appear in your universe design tool schema in exactly the same way as normal database tables, but the workflow for creating them is different. Adding, editing, and deleting derived tables is described in the following sections.

5.4.1.1 Adding a derived table

To add a derived table:

1. Click *Derived Tables* on the *Insert* menu.
   The *Derived Tables* dialog box appears.



2. Type the table name in the *Table Name* box.
3. Build the table SQL in the box beneath the *Table Name* box.
   You can type the SQL directly or use the Tables and Columns, Operators and Functions boxes to build it.
4. Click *OK*.
   The derived table appears in the schema with the physical database tables.
5. Build objects based on the derived table columns in exactly the same way you do with regular tables.

5.4.1.2 Editing a derived table

To edit a derived table:

1. Right-click the table in the universe design tool schema and select *Edit Derived Table* from the shortcut menu.
2. Edit the derived table, then click *OK*.

5.4.1.3 To delete a derived table

1. In the schema pane, select the derived table that you want to delete.
2. Press the Delete key.

5.4.1.4 Example: Creating a derived table

Example: Creating a derived table to return server information

In this example you want to create objects that allow the user to add information about the database server to their reports. You create two objects, servername and version, that return the values of the in-built variables @@SERVERNAME and @VERSION in a universe running on an SQL Server database.

Do the following:

1. Select *Derived Tables* on the *Insert* menu.
   The *Derived Tables* dialog box appears.
2. Type **`<serverinfo>`** in the *Table Name* box.
3. Type the SQL `Select @@SERVERNAME as servername, @@VERSION as version` in the SQL box.

   > **i** Note
   >
   > You must provide aliases in the SQL for all derived columns. The universe design tool uses these aliases to name the columns of the derived tables.

4. Click *OK*.
   The derived table serverinfo appears in the universe design tool schema.

5. Create a class called Server Info and add two dimension objects beneath the class, based on the servername and version columns of the serverinfo derived and columns of the table. Note that the serverinfo table appears in the list of tables like any ordinary database table, and its columns appear in the list of columns like ordinary table columns.

The user can now place the servername and version objects on a report.

Example: Showing the number of regions in each country

In this example you create a table that shows the number of regions in each country. The SQL is as follows:

```
select country,
count (r.region_id) as number_of_regions
from country c,
region r
where r.country_id = c.country_id
group by country
```

It is important in this case to alias the column that contains the calculation. The universe design tool uses these aliases as the column names in the derived table. In this case the table has two columns: country and number_of_regions.

5.5 Nested derived tables

A nested derived table (also known as a 'derived table on a derived table') is a table that is derived from at least one existing derived table. The nested derived table can also reference the database tables.

Use the *Derived Tables* editor to enter your SQL expression and select derived tables (and physical tables in the database, if required) to create your nested derived table. The SQL expression for the derived table is inserted into the SQL expression for the nested derived table when the report generates.

5.5.1 Using the Derived Tables editor

You use the *Derived Tables* editor to define derived tables or nested derived tables. You enter the `SQL` expressions and double-click on the objects (tables, derived tables, columns, functions) in the editor to create the `SQL` expression for your derived table or nested derived table.

Use the `@DerivedTable` function to reference a derived table in your nested derived table.

- The function `@DerivedTable(`Derived_table_name`)` is included in the functions catalog in the Derived Tables editor.

- A center pane in the bottom part of the Derived Tables editor displays existing derived tables and nested derived tables. This pane is only visible when derived tables exist in the universe.

The following checks are performed for both derived tables and nested derived tables when you click *Check Integrity*:

- Detects impacts on derived tables when a referenced derived table is removed.
- Checks for circular references.
- Checks for `@DerivedTable()` within Object definitions (`SELECT` and `WHERE`), which are not allowed.

5.5.2 To create a nested derived table

You create a nested derived table in the same way that you create a derived table. You can add and rename a nested derived table in the same way you add and rename a derived table.

To create a nested derived table:

1. Open your universe (*.unv) in the samples directory (`Business Objects\BusinessObjects Enterprise 12\Samples\en\UniverseSamples`).
2. Right-click in the Universe Structure pane and select *Derived Table* in the context menu.
   The *Derived Tables* editor opens and the center pane at the bottom of the *Derived Tables* editor lists the available derived tables.
3. Type the name your nested derived table.
4. Type the `SQL` expression. You can type the entire text or use the editor assistants.
5. Double-click on the objects (tables, derived tables, columns, functions).
6. Use the `@DerivedTable` function with the syntax: `@DerivedTable`(Derived_table_name) to choose a derived table.
7. Click *Check Syntax* to check the syntax of your derived table and correct any potential errors, then validate your nested derived table.
   The nested derived table is added to the universe.
8. Click *OK* to validate your nested derived table.
   The nested derived table appears in the *Structure* pane. Derived tables and nested derived tables are lighter in color than tables that represent actual database tables.

> **i** Note
>
> To display the table values, right-click the different tables.

You have created your nested derived table in your universe.

5.5.3 Renaming nested derived tables

When you rename a Derived Table, the new name is propagated and updated through all the other Derived Tables that reference it.

5.6 Using tables that have input columns

When a table containing an input column is inserted in the universe, the Web Intelligence or Query as a Web Service user will be required to choose values or enter values in order for the input columns to be calculated. The input columns are bound to values. The data for the input columns is not necessarily available in the original database, the data can be the following:

- Values hard-coded when you create the universe
- Values provided by the end user (after a prompt), or selected from a list
- Values provided via a join with another table

Tables with input columns are only supported when the Database connection is the Business Objects Data Federator server.

When analyzing joins that are eligible to solve the input column, note that:

- Only simple joins will be considered in the resolution algorithm
- No complex joins such as joins with multiple left columns or multiple right columns are allowed
- Only joins with equal or IN (INLIST) operands will be considered. Operators such as Between cannot be used to solve the input column

Tables with input columns have an arrow at the side of the input column in the *Structure* pane, and in the *Table Browser* pane, the input columns are identified by a specific icon.

When you insert a table that has an input column, use the *Input Columns* editor to enter your settings.

> **i Note**
>
> It is mandatory to assign default values for the input columns of a table when you add it to a universe.

This feature can be used with the following products and components:

- Web Intelligence
- Query as a Web Service

Related Information

To define a hard-coded list of values [page 133]
To define a list of values for the user to enter or select [page 134]

5.6.1 To define a hard-coded list of values

The database contains at least one table that has one or more input columns.

A hard-coded list of values is used as the input for the table to determine the value(s) for the input column(s). The end user does not enter any values. Follow the steps below to define the list of values.

1. Select the table from the database and add it to the *Structure* pane of the universe design tool.

The *Input Columns* editor appears.

2. In the *Input Columns* editor, click a parameter.
3. In the *Value* field, type a value or list of values. Type each value in double quotes and separate the values with a semicolon (;).
   The values appear in the *Value* column.
4. In the *Next execution* list, ensure *Use this value* is selected.
   Use this value appears in the *Next execution* column.
5. Click *OK*.

The table appears in the *Structure* pane of the universe design tool. The input column is identified by an arrow.

5.6.2 To define a list of values for the user to enter or select

The database contains at least one table that has one or more input columns.

The user can enter a value or select a value from a list of values that will be used by the table to determine the value(s) for the input column(s). Follow the steps below to define the values for the input column table in your schema.

1. Select the table from the database and add it to the *Structure* pane of the universe design tool.
2. In the *Input Columns* editor, click a parameter.
3. In the *Next execution* list, click *Prompt me for a value*.

   When a Web Intelligence or Query as a Web Service query executes, the user is prompted to select a value from the associated list of values.
4. In the *Prompt Label Edition* field, edit the default prompt that appears for the end user.
5. Click *Browse universe objects* to select a list of values from the universe.

   If you want to remove an object from the list of values that you have added to your settings, in the *Selected Object* pane, click the object and click *Erase*.
6. Click *OK*.

The table appears in the *Structure* pane of the universe design tool. The input column is identified by an arrow. In the *Table Browser*, the input column is identified by a specific icon.

5.7 Defining joins

Once you have inserted more than one table in the schema, you need to create joins between related tables. Joins are as important as the tables in a schema, as they allow you to combine data from multiple tables in a meaningful way.

5.7.1 What is a join?

A join is a condition that links the data in separate but related tables. The tables usually have a parent-child relationship. If a query does not contain a join, the database returns a result set that contains all possible

combinations of the rows in the query tables. Such a result set is known as a Cartesian product and is rarely useful.

For example, the Cartesian product of a query referencing two tables with 100 and 50 rows respectively has 5000 rows. In large databases or queries involving many tables, Cartesian products quickly become unmanageable. In the universe design tool, joins are represented as lines linking tables in a schema.

5.7.2 Why use joins in a schema?

You use joins to ensure that queries returning data from multiple tables do not return incorrect results. A join between two tables defines how data is returned when both tables are included in a query.

Each table in a schema contains data in one or more columns that correspond to user requirements. In a production universe, Web Intelligence users may want to run queries that combine a number of different objects (each inferring a column) returning data from any combination of tables.

Linking all tables in the schema with joins ensures that you restrict the number of ways that data from columns in different tables can be combined in a query. Joins limit column combinations between tables to matching or common columns. This prevents result data being returned that contains information from columns that have no sense being matched.

> **i** Note
>
> You should always create joins in the Structure pane. Joins that are not created from the Structure pane, for example a join manually defined in the Where clause for an object, are created at run time, so are not considered by the universe design tool for integrity checks and context detection. The information for these processes is required at design time. Contexts and universe integrity are covered later in this chapter.

5.7.3 What SQL does a join Infer?

By default the universe design tool specifies a join implicitly in a WHERE clause through a reference to the matching or common columns of the tables.

Normally there is one WHERE clause for each pair of tables being joined. So, if four tables are being combined, three WHERE conditions are necessary.

The result of a query run including two tables linked by a join is a single table with columns from all the combined tables. Each row in this table contains data from the rows in the different input tables with matching values for the common columns.

5.7.3.1 ANSI 92 support

If the target RDBMS supports ANSI 92, then you can set a universe parameter (*File* > *Parameters* > *Parameter*) ANSI92 to Yes to activate ANSI 92 support for joins created in your schema. When a universe supports the ANSI 92 standard for joins, newly created joins are specified in the FROM clause. You can also

select the objects that are inferred by columns to be included in the FROM clause. ANSI 92 support is described in the section ANSI 92 support for joins in a universe [page 147].

5.7.4 What tables do not have to be joined?

You should join all tables in the schema that are inferred in the SQL generated by objects in Web Intelligence queries run against the universe. The only exceptions to these are the following types of tables:

- Base tables from the schema that have been aliased for each use. These are the original tables for which you have created aliases either for renaming, or join problem resolution reasons. These base tables are typically not used in any object definition.
- Tables that are the target of aggregate awareness syntax (although this has to be taken on a case-by-case basis). For example the aggregate tables in the sample efashion universe (their names begin with "Agg_") are not joined to any table in the schema:

5.7.5 Joining primary and foreign keys

You normally create a join between the primary key in one table and the foreign key of another table. You can also create a join between two primary keys. It is very unusual for at least one side of a join to not include the primary key of the table.

You need to understand how each key is constructed in your database. Multi column keys can affect how you set cardinalities for joins, and this can affect how you set up contexts in your schema.

Detecting and Using contexts is described in Detecting and Solving Join Problems [page 184]

5.7.5.1 Displaying keys

You can display primary and foreign keys in all tables in the Structure pane. The key columns appear underlined in each table that contains keys. When you select the option to display keys, you must refresh the structure before keys appear underlined.

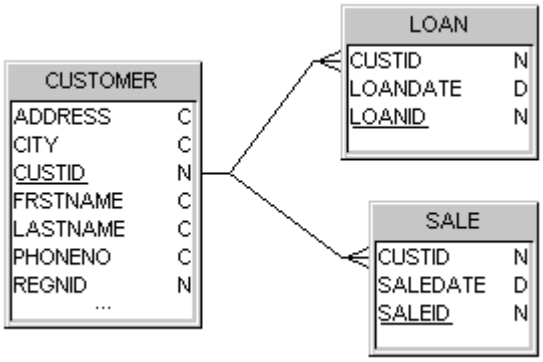
The ability to display key columns as underlined depends on primary keys being defined in the target database.

> **i Note**
>
> When you display underlined key columns, the information is stored in the .UNV file. This information is lost when you export a universe to the Central Management Server (CMS) repository. You have to re-display keys for a universe, each time it is imported.

To display keys:

1. Select Tools > Options.
   The Options dialog box opens to the General page.
2. Click the Graphics tab.
   The Graphics page appears.

3. Select the Underline Keys check box in the Columns group box.
4. Click OK.
   You need to refresh the structure before key columns appear underlined.
5. Select View > Refresh Structure.
   The database structure is refreshed. The key columns in your schema are underlined as shown below:



5.7.6 Understanding the cardinality of a join

Cardinalities further describe a join between 2 tables by stating how many rows in one table will match rows in another. This is very important for detecting join problems and creating contexts to correct the limitations of a target RDBMS structure.

You should set cardinalities for each join in the schema. The universe design tool can automatically detect and set cardinalities, but you should always manually check the cardinalities, taking into account the nature of the keys that are joined.

Setting and using cardinalities is described in the section Using cardinalities [page 167].

5.7.7 Creating joins

You have several approaches to creating joins with the universe design tool:

- Tracing joins manually in the schema.
- Defining join properties directly.
- Selecting automatically detected joins.
- Automatically creating joins on table insertion.

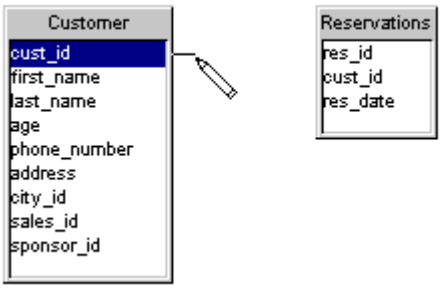
Each of these approaches is described in detail below.

5.7.7.1 Tracing joins manually in the schema

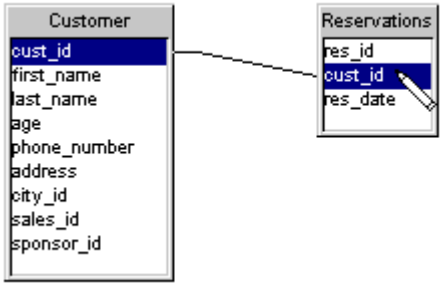
You can graphically create individual joins between tables by using the mouse to trace a line from a column in one table to a matching column in another table.

To create a join by tracing manually:

1. Position the pointer over a column that you want to be one end of a join.
   The pointer appears as a hand symbol.
2. Click and hold down the left mouse button.
   The column is highlighted.
3. Drag the mouse to the column in another table that you want to be the other end of the join.
   As you drag, the pointer is transformed into a pencil symbol.



4. Position the pencil symbol over the target column.
   The target column is highlighted.



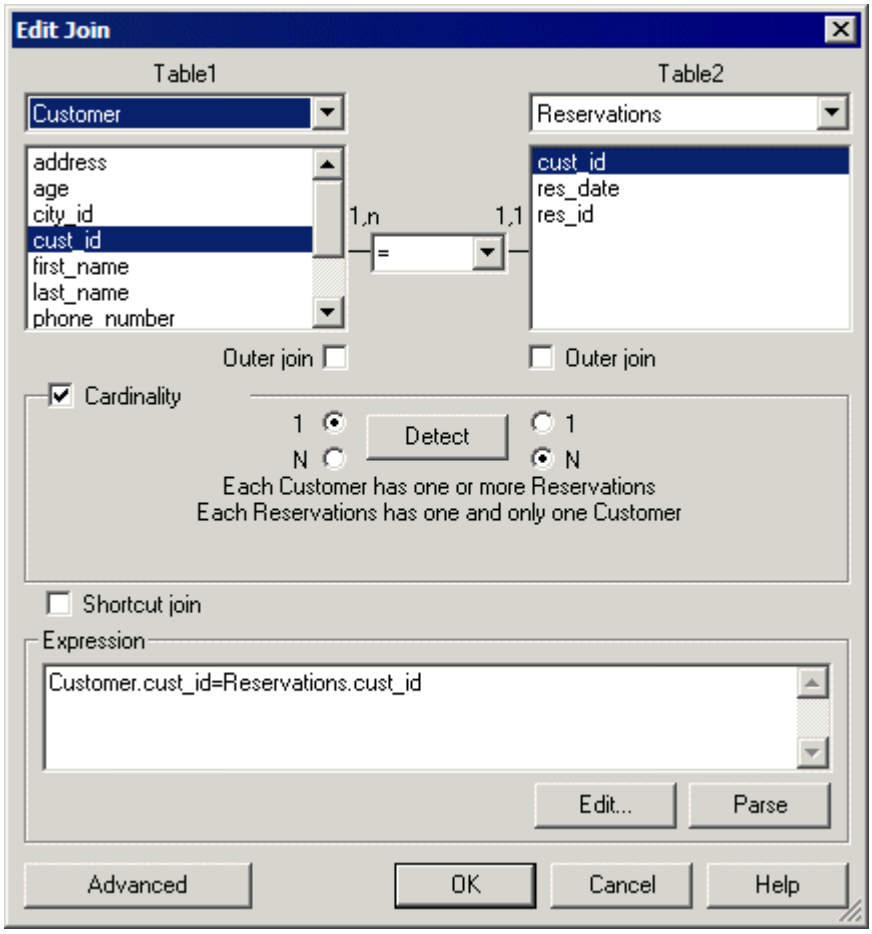
5. Release the mouse button.
   The join between the two tables is created.
6. Double click the new join.
   The Edit Join dialog box appears. It lists join properties. The properties that you can set for a join, including cardinality and join type, are described in the section Join properties [page 142].
7. Enter and select properties for the join.
8. Click *OK*.

5.7.7.2 Defining join properties directly

You create a join by directly defining join properties in the Edit Join dialog box.

To create a join directly:

1. Select Insert > Join.
   Or
   Click the Insert Join button.
   The Edit Join dialog box appears.



2. Select a table from the Table1 drop-down list.
   The columns for the selected table appear in the list box under the table name.
3. Click the name of the column that you want to be at one end of the new join.
4. Select a table from the Table2 drop-down list box.
   The columns for the selected table appear in the list box under the table name.
5. Click the name of the column that you want to be at the other end of the new join.
   The properties that you can set for a join, including the join operator, cardinality, and join type are described in the section Join properties [page 142]
6. Enter and select properties for the join.
7. Click *OK*.
   The new join appears in the schema linking the two tables defined in the Edit Join dialog box.

5.7.7.3 Selecting automatically detected joins

You can use the universe design tool feature Detect Joins to automatically detect selected joins in the schema. The tool identifies column names across tables in the target database and proposes candidate joins for the tables in your schema. You can then select which, or accept all, proposed joins you want to be created.

How are joins automatically detected?

The joins are detected based on the Joins strategy that appears in the Strategies page of the Parameters dialog box (File > Parameters > Strategies tab).

A strategy is a script file that automatically extracts structural information from the database. There are a number of inbuilt strategies that are shipped with the universe design tool. These are listed in drop-down list boxes on the Strategies page of the Parameters dialog box.

The default automatic join detection strategy detects joins based on matching column names, excluding key information. You can select which join strategy you want to apply when you use automatic join detection.

> **i** Note
>
> Refer toSelecting strategies [page 83] for more information on using strategies.

Using automatic join detection appropriately

Detecting joins automatically is useful to help you quickly create joins in your schema. However, you need to be aware of the limitations of automatic join detection when designing your schema.

Join strategies used to detect candidate joins match column names from the database. There may be instances in the target database when primary, foreign keys, and other join columns do not have the same name across different tables. The universe design tool will not pick up these columns. You should always verify manually each join that you accept to be created that has been automatically detected. You should be aware that there may be other joins necessary that have not been detected.

To create a join using automatic detection:

1. Verify that the join strategy that you want to use to detect joins is selected in the Joins drop down list box on the Parameters dialog box. You can verify this as follows:
   - Select File > Parameters and click the Strategies tab.
   - Select the strategy that you want to use to detect joins from the Joins drop-down list box and click OK.
2. Select multiple tables in the Structure pane.
   You can select multiple tables by pressing SHIFT while clicking each table, or you can select all tables in a zone by clicking in an empty space, and dragging the cursor to define a rectangular zone that includes any number of tables.
3. Select Tools > Automated Detection >Detect Joins.
   Or
   Click the Detect Joins button.

The Candidate Joins dialog box appears. It lists candidate or proposed joins for the selected tables. The candidate joins also appear as blue lines between selected tables in the Structure pane.



4. Click Insert to create all candidate joins.
5. Or
   Select one or more joins and click Insert.
   You can select one or more joins by holding down CTRL and clicking individual tables, or holding down SHIFT and clicking the first and last join in a continuous block.
   The joins are inserted in you schema.
6. Click Close.

5.7.7.4 Inserting joins automatically with associated tables

You can choose to insert joins automatically in the schema at the same time as the tables that use the joins are inserted into the structure pane. Automatic join creation is determined by two processes:

- The active join strategy determines the column information used to detect the join.
- The default creation option Extract Joins With Tables must be selected to allow the automatic creation of joins with their associated tables. This option is on the Database page of the Options dialog box.

Limitations when inserting joins automatically

Inserting joins automatically into your schema with associated tables is a quick way to get joins into your schema, but it can lead to serious design faults with your schema. The joins are inserted based on the database structure, so columns common to more than one table that have been renamed in the database will not be picked up.

You should not use this technique to create joins in a production universe. Instead, use it for demonstration purposes, or as a quick way to build a universe, in which you will then carefully validate each join after insertion.

To create a join automatically with an associated table:

1. Verify that the join strategy that you want to use to detect joins is selected on the Strategies page of the Parameters dialog box.
2. Select Tools > Options.
   The Options dialog box appears.
3. Click the Database tab.
   The Database page appears.
4. Select the Extract Joins With Tables check box.
5. Click OK.
   Now when you insert a table that has columns referencing other columns in tables that have already been inserted into the Structure pane, the references between tables are automatically inserted as joins between appropriate tables.

5.7.8 Join properties

You define join properties in the Edit Join dialog box. You can define the following properties for a join:

Property	Description
Table1	Table at the left end of the join. Columns are listed for the table selected in the drop-down list box.
Table2	Table at the right side of the join. Columns are listed for the table selected in the drop-down list box.
Operator	Operator that defines how the tables are joined. The operators available to a join are described in the section Join Operators [page 143].
Outer Join	When selected, determines which table contains unmatched data in an outer join relationship. Outer joins are described fully in the section Creating a theta join [page 157].
Cardinality	When selected, allows you to define the cardinality for the join. Defining and using cardinalities is described in the section Using cardinalities [page 167].
Shortcut Join	Defines the join as a shortcut join. Shortcut joins are described in the section Restrictions for the use of outer joins [page 162].
Expression	WHERE clause that is used to restrict the data that is returned when the two joined tables are included in a query.

Property	Description
Advanced	Available when ANSI 92 support is activated for the universe. When clicked, opens a second join properties box that lists the objects built on columns for the two tables in the join. You can select the objects to be included in the FROM clause. See the section ANSI 92 support for joins in a universe [page 147] for information on activating ANSI 92 support for join syntax.

5.7.8.1 Join Operators

You can select an operator for a join from the drop-down list box between the Table1 and Table2 boxes. The operator allows you to define the restriction that the join uses to match data between the joined columns.

You can select the following operators for a join:

Operator	Description
=	is equal to
!=	is not equal to
>	is greater than
<	is less than
>=	is greater than or equal to
<=	is less than or equal to
Between	is between (theta joins)
Complex	complex relationship

5.7.8.2 Edit and Parse

The Edit Join dialog box also has two features available that allow you to edit and verify the join syntax:

Edit

The Edit button opens an SQL editor. You can use this graphic editor to modify the syntax for tables, columns, operators, and functions used in the join. For more information on using this editor, refer to the section Using the Join SQL Editor [page 145].

Parse

The Parse button starts a parsing function that verifies the SQL syntax of the join expression. If the parse is successful, you receive a result is OK message. If the universe design tool encounters an error, you receive an error message indicating the source of the problem.

5.7.9 Editing a join

You can use any of the following methods to edit a join:

- Modify join properties from the Edit Join dialog box.
- Modify join SQL syntax directly using the Join SQL Editor.
- Modify join SQL syntax directly using the formula bar.

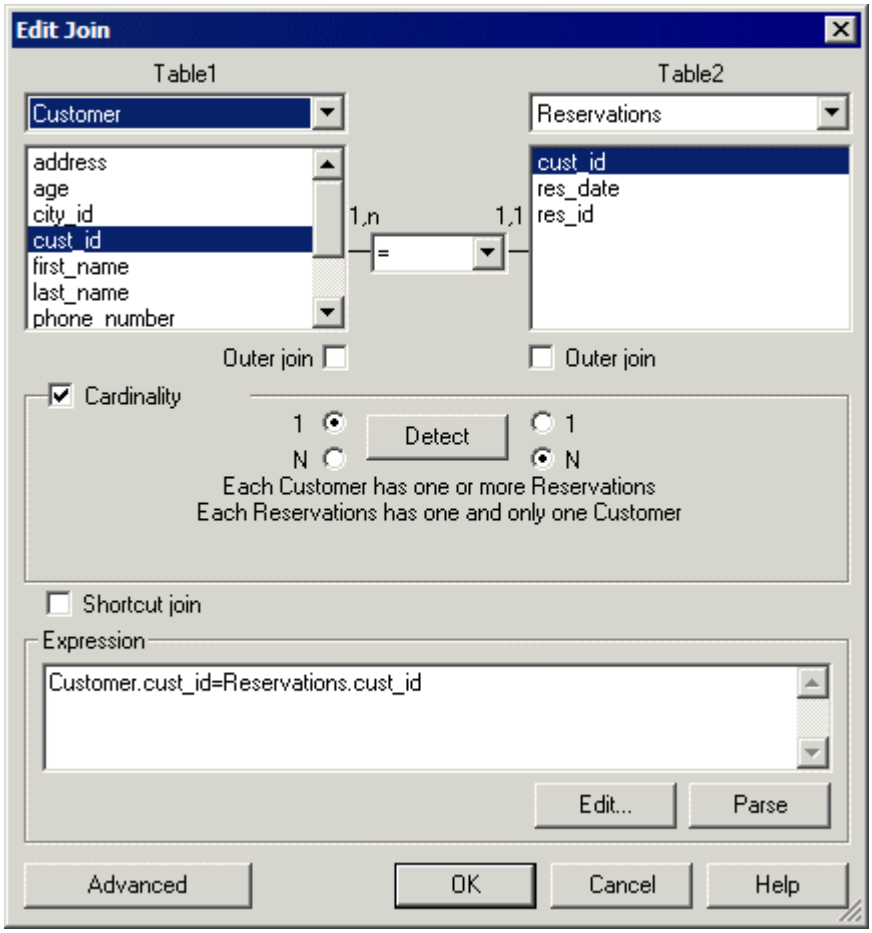
Each of these methods is discussed in this section.

5.7.9.1 Using the Edit Join dialog box

You can use the Edit Join dialog box to define and edit join properties. You can also access the Join SQL Editor to edit join syntax directly from this dialog box. Join properties are described in the section Join properties [page 142].

To edit a join using the Edit Join dialog box:

1. Double click a join in the Structure pane.
   Or
   Click a join and select Edit > Join.
   The Edit Join dialog box appears.



2. Select an operator from the drop-down list box between the tables.
3. Select other properties as required.
4. If you are defining a join with ANSI 92 syntax, then click the Advanced button.
5. Click OK.

> → Tip
>
> You can edit the SQL directly for the join by clicking the Edit button and using the Join SQL editor. See Using the Join SQL Editor [page 145] for more information.

5.7.9.2 Using the Join SQL Editor

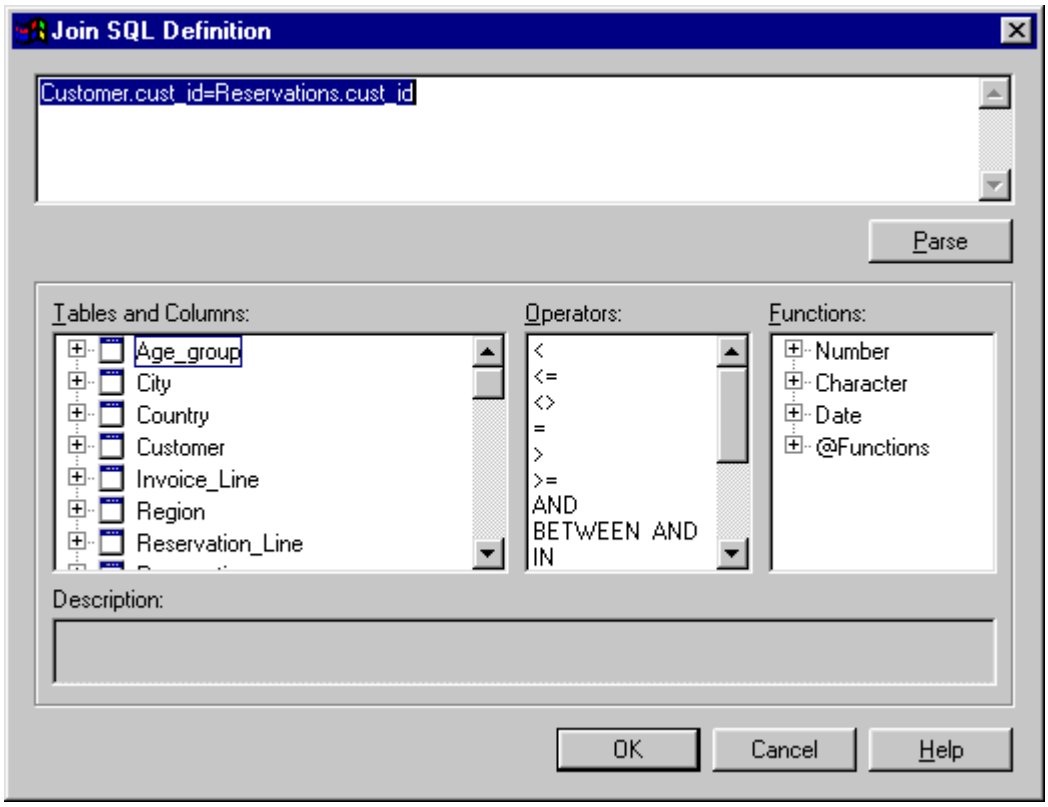
You can use a graphical editor to directly modify the SQL expression for a join. You access this editor from the Edit Joins dialog box.

To modify a join using the Join SQL Editor:

1. Double click a join in the Structure pane.
   Or
   Click a join and select Edit > Join.

The Edit Join dialog box appears.

2. Click the Edit button.
   The Join SQL Definition box appears. The SQL expression for the join appears in the text box.



3. Click the join expression in the edit box at the place where you want to add or modify the SQL syntax.
   You can use the editing features to modify or add SQL syntax as follows:

You want to...	Then do the following...
Change a column at either join end	○ Expand a table node in the Tables and Columns box. ○ Double click a column name.
Change an operator used by the join	Double click an operator in the Operators box.
Use a function in the join	○ Expand a function family node. ○ Double click a function.

   The column, operator, or function appears in the join definition.

4. Click OK.

> **i Note**
>
> @Select and @Where functions cannot be used for row access restrictions.

Related Information

To create a restriction [page 473]

5.7.9.3 Using the Formula bar

The *Formula* bar is a text box above the *Universe* window that shows the formula or expression of any selected join in the *Structure* pane, or selected object in the *Universe* pane. You can use three editing buttons placed to the left of the Formula bar:

Edit button	Description
	Cancel last modification that has not been validated. If you make several changes to a join expression without validating the changes, clicking *Cancel* returns the expression to its original state. If you want to undo any individual modifications, you should use the *Edit* > *Undo* option, or click *Undo*.
	Validate expression. This applies any changes to the join expression. You can undo changes after validation use the *Edit* > *Undo* option, or click *Undo*.
	Open Edit Join dialog box for selected join.

To display the Formula bar:

- Select *View* > *Formula Bar*.
  The *Formula Bar* appears above the *Universe* window.

To modify a join using the *Formula Bar*:

1. Click a join that you want to edit.
   The formula for the join appears in the *Formula Bar*.
2. Click the join expression in the *Formula Bar* at the place you want to modify the syntax.
3. Modify the expression as required.
4. Click *Validate* to apply the changes.
5. Press the `Return` key to quit the *Formula Bar* .
   Or
   Click anywhere outside of the *Formula Bar*.

5.7.10 ANSI 92 support for joins in a universe

The universe design tool supports ANSI 92 syntax for joins. ANSI 92 is not supported by default. You must activate support by setting the SQL universe parameter ANSI92 to YES. This parameter is listed on the

Parameter page of the universe parameters dialog box (File > Parameters > Parameter). Once activated, you can choose to use ANSI 92 syntax for joins in the universe.

> **i Note**
>
> The ANSI 92 setting is also declared in the .prm files. If the .prm setting is 'usual', then the universe design tool setting takes preference. If the .prm setting is 'ANSI92', then the universe design tool-level settings will be overridden. Refer to the *Data Access Guide* for more details about .prm files and the ANSI 92 setting. The behavior can depend on your database version. Refer to your database technical details for more information.

Ensure that you verify that the target RDBMS supports ANSI 92 before using the syntax in joins.

Activating ANSI 92 support in the universe and defining a join using ANSI 92 syntax are described below.

5.7.10.1 Example: comparing default join syntax and ANSI 92 syntax

Join syntax for two joins is shown below. The first shows the default behavior where the join is defined in the WHERE clause, the second shows the same join in the FROM clause using the ANSI 92 standard.

Default join syntax

```
SELECT
  Resort.resort,
  'FY'+Format(Sales.invoice_date,'YYYY'),
  sum(Invoice_Line.days * Invoice_Line.nb_guests * Service.price)
FROM
  Resort,
  Sales,
  Invoice_Line,
  Service,
  Service_Line
WHERE
  ( Sales.inv_id=Invoice_Line.inv_id  )
  AND  ( Invoice_Line.service_id=Service.service_id  )
  AND  ( Resort.resort_id=Service_Line.resort_id  )
  AND  ( Service.sl_id=Service_Line.sl_id  )
GROUP BY
  Resort.resort,
  'FY'+Format(Sales.invoice_date,'YYYY')
```

Same join using the ANSI 92 standard

```
SELECT
  Resort.resort,
  'FY'+Format(Sales.invoice_date,'YYYY'),
  sum(Invoice_Line.days * Invoice_Line.nb_guests * Service.price)
```

```
FROM
  Resort INNER JOIN Service_Line ON (Resort.resort_id=Service_Line.resort_id)
   INNER JOIN Service ON (Service.sl_id=Service_Line.sl_id)
   INNER JOIN Invoice_Line ON (Invoice_Line.service_id=Service.service_id)
   INNER JOIN Sales ON (Sales.inv_id=Invoice_Line.inv_id)

GROUP BY
  Resort.resort,
  'FY'+Format(Sales.invoice_date,'YYYY')
```

5.7.10.2 Activating ANSI 92 support in a universe

To activate ANSI 92 support for joins:

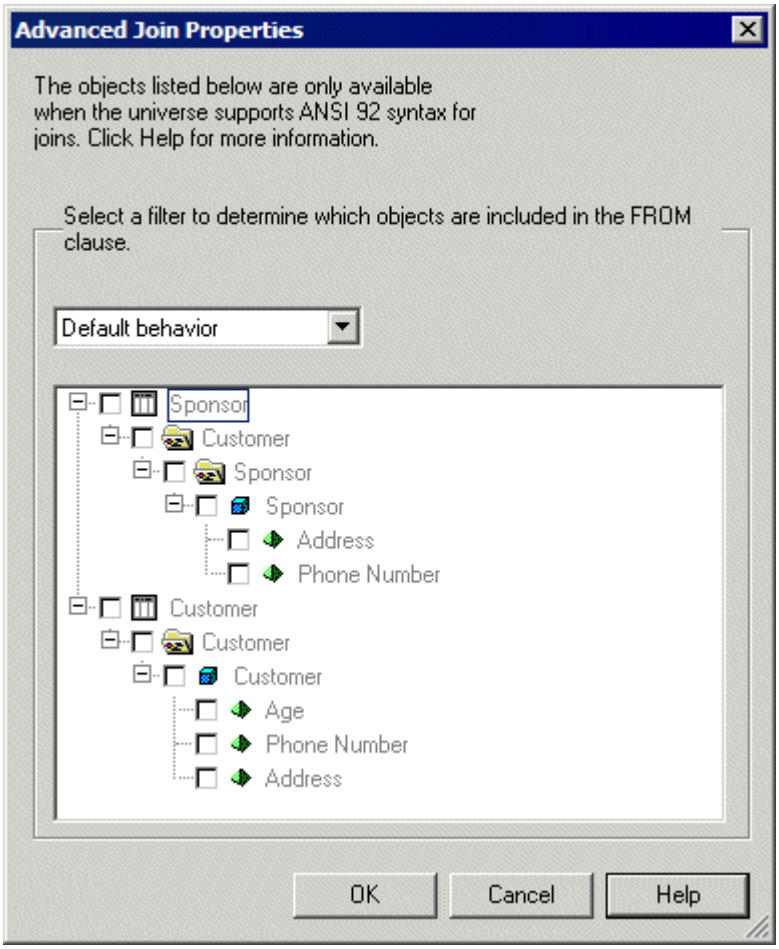
1. Select File > Parameters.
   The Universe Parameters dialog box appears.
2. Click the Parameter tab.
   The Parameters page appears. It lists certain SQL generation parameters that you can set at the universe level to optimize SQL generation for the current universe. These are parameters that were included in the PRM file for the target RDBMS in previous versions of Business Objects products. Certain RDBMS specific parameters are still contained in the PRM files, but many standard SQL parameters are now listed in the Parameter page. See the chapter Setting SQL generation parameters [page 90] for a complete list of the available parameters.
3. Click the ANSI92 parameter in the list.
4. Type YES in the value box.
5. Click Replace.
6. Click OK.
   The ANSI 92 standard can now be applied to join definitions for the current universe. When you click the Advanced button on the Edit Join dialog box, the Advanced Join box appears. You can define a filter to determine which dimensions you want to include in the FROM clause for a join.

5.7.10.3 Defining a join with ANSI 92 syntax

You can use ANSI 92 syntax to define a join from the Edit Join properties dialog box. You can do this by using an advanced editing box that allows you to select objects to be included in a join definition.

To define a join using ANSI 92 syntax:

1. Activate ANSI 92 support for the universe. See the section Activating ANSI 92 support in a universe [page 149] for information.
2. Double click a join in the schema.
   The Edit Join box for the join appears.
3. Click the Advanced button.
   The Advanced Joins Properties dialog box appears.



4. Select one of the following FROM clause filters from the drop-down list.

FROM option	Description
Default behavior	Default syntax for joins is applied. Joins are defined in the WHERE clause.
All objects in FROM	All objects defined on columns in the tables on the right and left side of the join are included in the FROM clause.
No objects in FROM	No objects are included in the FROM clause.
Selected objects in FROM	Only objects selected in the Advanced Join Properties tree view of the join tables are included in the FROM clause.

5. Select objects to be included in the FROM clause if you selected the Selected objects in FROM filter.
6. Click OK.
7. Enter any other join parameters in the Edit Join box.
8. Click OK.

5.7.11 Deleting joins

To delete a join:

1. Click a join.
   The join is selected
2. Do any of the following:
   - Press the backspace key on your keyboard
   - Press the Delete button on your keyboard
   - Right click the join and select Clear from the contextual menu.
     A confirmation box appears asking to you to confirm the join deletion.
3. Click Yes.
   The join is deleted.

> **i** Note
>
> Ensure that you are aware of all the consequences in both the schema and universe when you delete a join. Verify that deleting the join does not affect a context. If you try to delete a join, the universe design tool warns you if the join is used in one or more contexts. You need to manually verify which context, and access the effect on the universe if the context is affected by the join deletion.

5.8 Defining specific types of joins

You can define the following types of joins in the universe design tool:

Join type	Description
Equi-Joins (includes complex equi-joins)	Link tables based on the equality between the values in the column of one table and the values in the column of another. Because the same column is present in both tables, the join synchronizes the two tables. You can also create complex equi-joins, where one join links multiple columns between two tables.
Theta Joins (conditional joins)	Link tables based on a relationship other than equality between two columns.
Outer Joins	Link two tables, one of which has rows that do not match those in the common column of the other table.
Shortcut Joins	Join providing an alternative path between two tables, bypassing intermediate tables, leading to the same result, regardless of direction. Optimizes query time by cutting long join paths as short as possible.

Join type	Description
Self restricting joins	Single table join used to set a restriction on the table.

Each join type is described fully in its respective section in this chapter. You use the same method to create each type of join; however, you must define different properties for each join in the Edit Join box at join creation.

5.8.1 Creating Equi-joins

An equi-join links two tables on common values in a column in table 1 with a column in table 2. The restriction conforms to the following syntax:

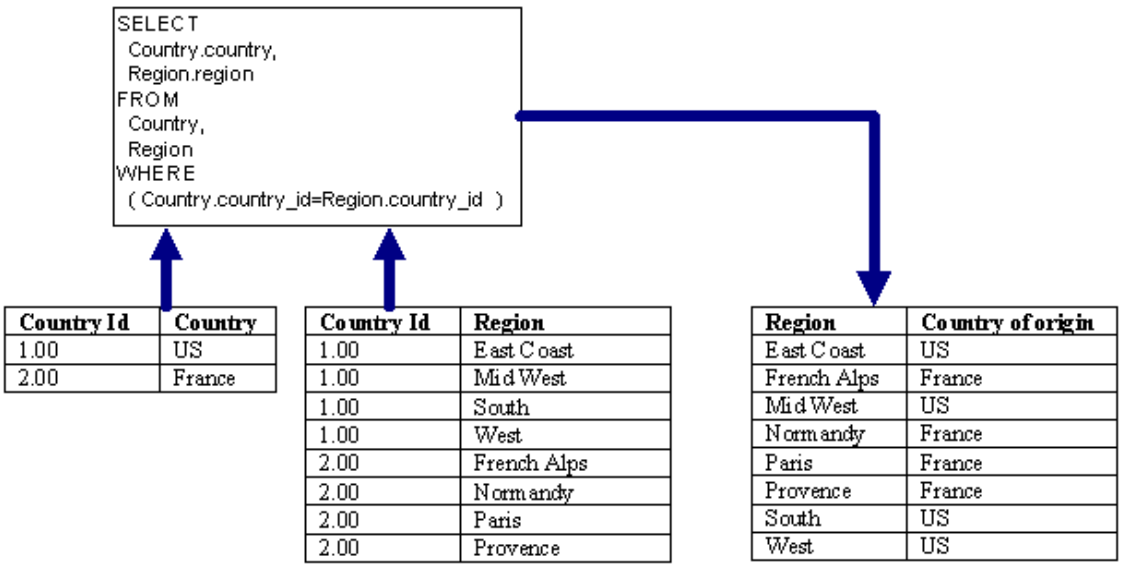
Table1.column_a = Table2.column_a

In a normalized database the columns used in an equi-join are usually the primary key from one table and the foreign key in the other. For information on keys, see the section What tables do not have to be joined? [page 136].

When you create a new join, it is an equi-join by default. Most joins in your schema should be equi-joins.

Example: Equi-join restricts data

When a Select statement is run in the example below, the Select and From clauses create a Cartesian product. However, before any data is returned, the Where clause applies a restriction so that only rows where there is a match between the Country ID column in both the tables are returned.



5.8.1.1 Creating a new equi-join

To create a new equi-join:

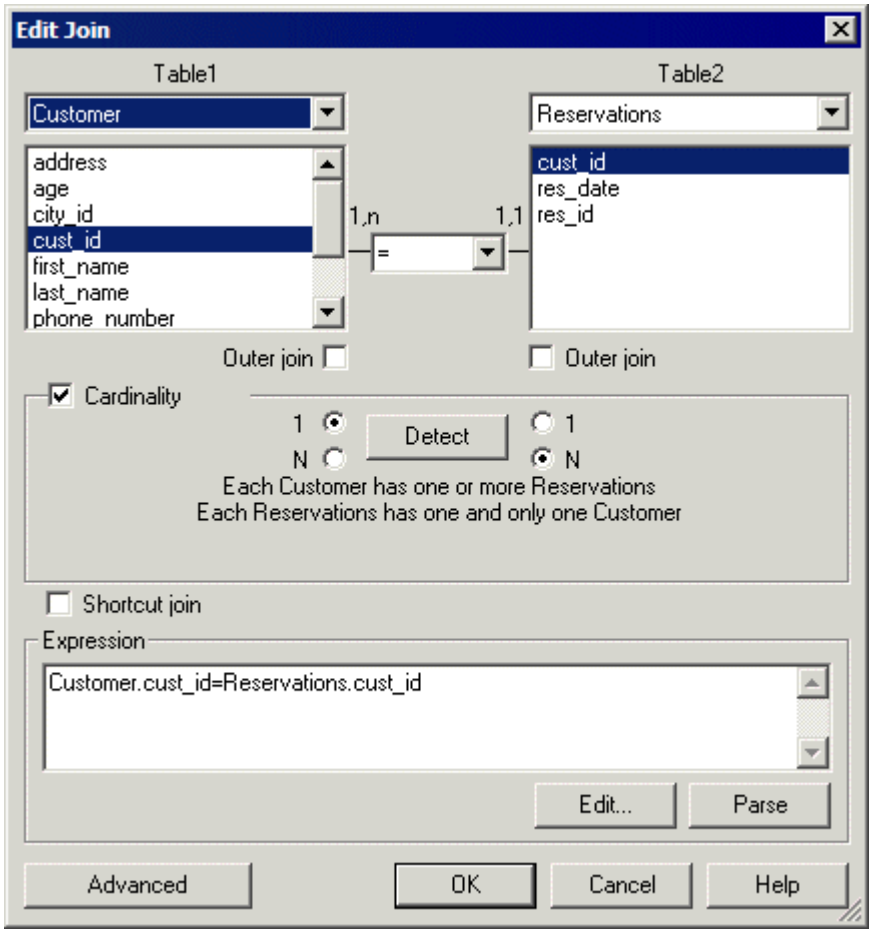
- Create a join between two tables.
  The default new join is an equi-join.

> → Tip
>
> The different methods you can use to create joins are described in the section Displaying keys [page 136].

5.8.1.2 Creating an equi-join from an existing join

To create an equi-join from an existing join:

1. Double click an existing join.
   The Edit Join box appears.
2. Select a column in the Table1 list box.
3. Select the matching column in the Table2 list box
4. Select = from the Operator drop-down list box.
   The Edit Join box below shows an equi-join between the tables Customer and Reservations.



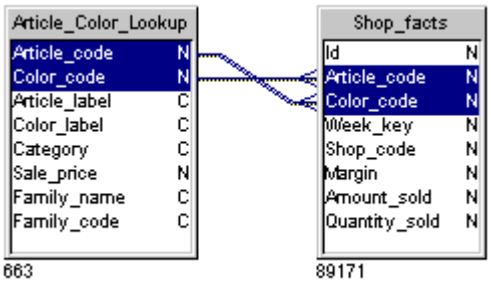
> **i** Note
>
> Common columns do not always have the same name. You need to verify primary and foreign key column names in the database. Different tables may use the same key columns, but have them renamed for each table depending on the table role in the database.

5. Click the Parse button to check the join syntax.
   If you receive an error message, check to see that the column is common to both tables.
6. Click OK.

5.8.1.3 Creating complex equi-joins

You can also create a complex equi-join. This is a single join that links multiple columns between two tables. You can create complex equi-joins by using the Complex operator for a join in the Edit Properties sheet for a join.

The sample eFashion universe contains a complex join shown below.

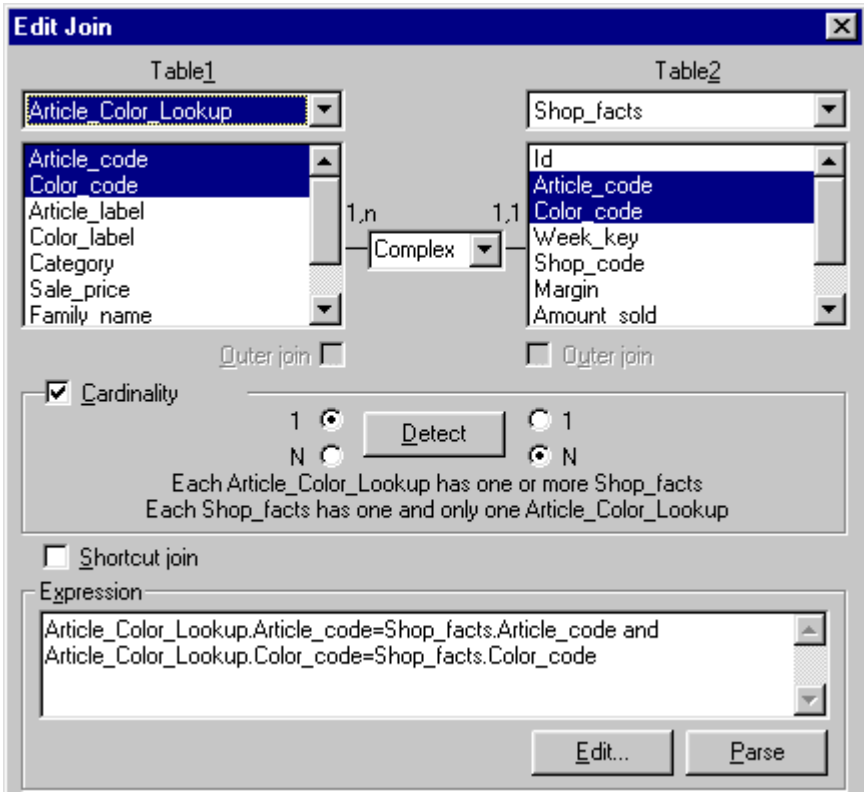


Using a complex equi-join instead of multiple single equi-joins between joined columns has the following advantages:

- Only one cardinality to detect. This can save time when detecting cardinalities, and also keeps the schema uncluttered and easier to read.
- You can view the SQL for all the joins between two tables in the Expression text box in the Edit Properties box for the join. When you use multiple single equi-joins between two tables, you have a one expression for each join.

To create a complex equi-join:

1. Double click an existing join.
   The Edit Join box appears.
2. Select multiple columns in the Table1 list box.
3. Select the matching columns in the Table2 list box
4. Select "Complex" from the Operator drop-down list box.
   The Edit Join box below shows a complex equi-join between the tables Article_Color_Lookup and Shop_facts.



5. Click the Parse button to check the join syntax.
   If you receive an error message, check to see that the column is common to both tables.
6. Click OK.

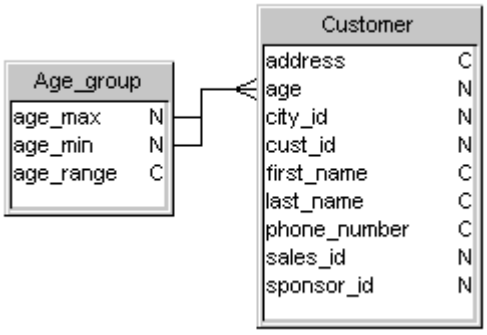
5.8.2 Theta joins

A theta join is a join that links tables based on a relationship other than equality between two columns. A theta join could use any operator other than the "equal" operator.

The following example and procedure show you how to create a theta join that uses the "Between" operator.

Example: Theta join

The Age_Group table below contains age range information that can be used to analyze data on the age of customers.

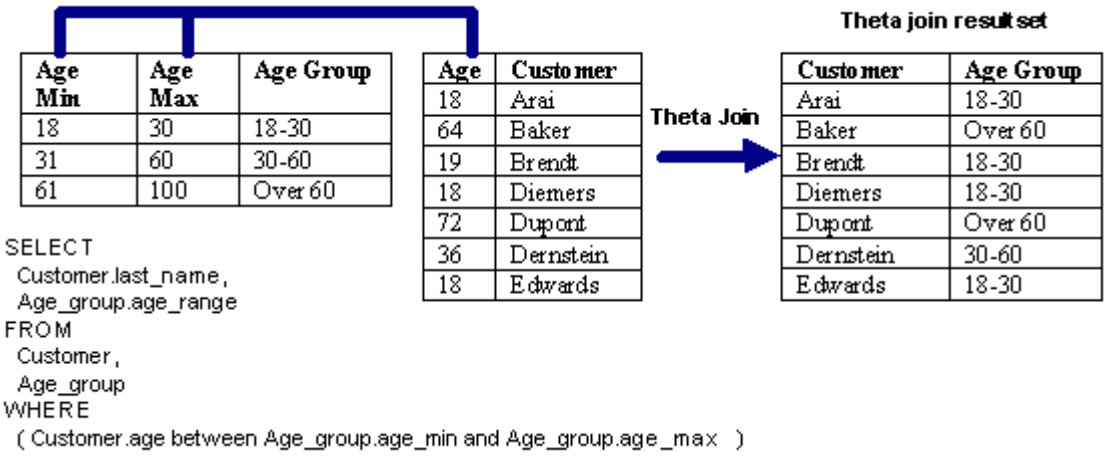


You need to include this table in the universe, but there is no common column between the Customer table and the Age_Group table, so you cannot use an equi-join.

You create a theta join using the operator "Between" for maximum age range and minimum age ranges. By using a theta join, you infer that a join exists where the value in a row of the Age column in the Customer table is between the values in a row for the Age_Min and Age_Max columns of the Age_Group table. The join is defined by the following expression:

Customer.age between Age_group.age_min and Age_group.age_max

The diagram below shows the joins between Age max, Age min, and Age, and the result set that is returned when the theta join is used in a query run on both Age_Group and Customer tables.

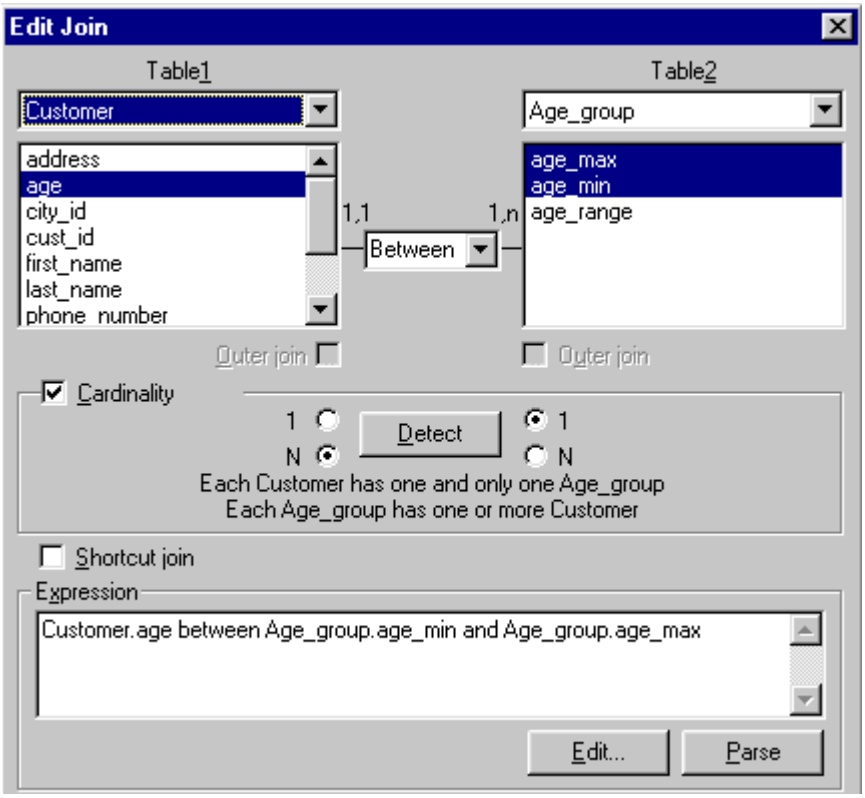


5.8.2.1 Creating a theta join

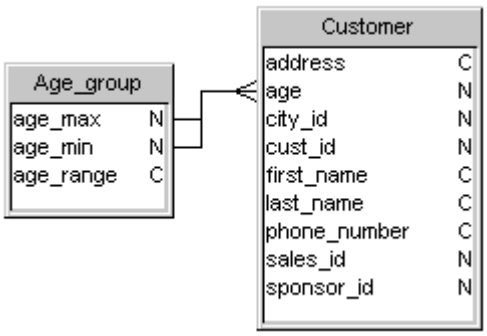
To create a theta join using range columns:

1. Create a join between two tables.
   An equi-join is created by default.

2. Double click the join.
   The Edit Join dialog box appears.
3. Click a column in the Table1 column list box.
4. Press and hold down the CTRL key and click two columns from the Table2 column list box.
   The example below shows the two columns age_min and age_max selected. The Between operator automatically appears in the operator drop-down list.



5. Click the Parse button to test for the validity of the join.
   If you receive an error message, check to see that you have correctly selected the columns.
6. Click OK.
   The join is created in the Structure pane.



5.8.3 Outer joins

An outer join is a join that links two tables, one of which has rows that do not match those in the common column of the other table.

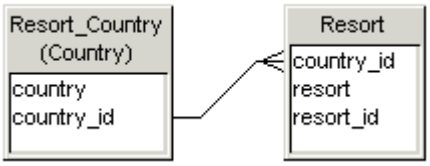
You define an outer join by specifying which table is the outer table in the original equi-join. The outer table contains the column for which you want to return all values, even if they are unmatched. You specify the outer table from the Edit Join dialog box for the selected join.

5.8.3.1 Full outer joins

By default you can create either a left outer, or a right outer join depending on which side of the join the outer table is designated. You can also create a full outer join by activating ANSI 92 support for joins in the universe. This is achieved by setting a universe SQL parameter ANSI 92 to YES (File > Parameters > Parameter). This allows the universe to support ANSI 92 syntax for joins, and you can select the tables on either side of a join to be outer tables. Refer to the section Defining a full outer join [page 162] for information on creating full outer joins.

Example: Outer join

The tables Resort_Country and Resort below are linked by an equi-join.



Each resort belongs to a country, but each country may not have a resort. If you use an equi-join, the result set of a query would only show information on the countries that have a resort; Australia, France, and the US.

Country	Resort
Australia	Australian Reef
France	French Riviera
US	Bahamas Beach
US	Hawaiian Club
US	Royal Caribbean

However, you may wish to show all countries irrespective of an equivalent value in the foreign key of the Resort table. To achieve this you define an outer join so that all counties are returned, despite having no match in the Resort column.

The syntax (Microsoft Access) for the outer join is as follows:

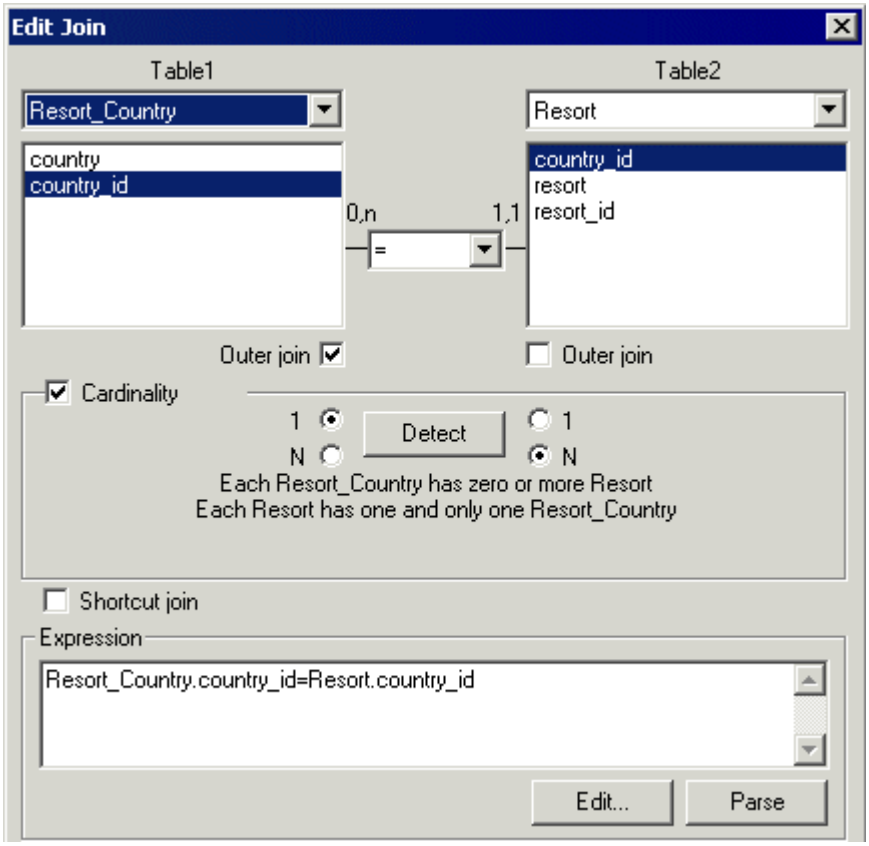
```
SELECT
Resort_Country.country,
Resort.resort
FROM
Country Resort_Country,
Resort,
{ oj Resort_Country LEFT OUTER JOIN Resort ON
Resort_Country.country_id=Resort.country_id }
```

> **i** Note
>
> The example above uses Microsoft Access, so any one-to-many joins following the table Resort, would also have to have to use outer joins. If not, then a NULL returned by the original outer join, will not be taken into account if there is no matching NULL returned by following joins. The treatment of outer joins is RDBMS specific, so refer to your RDBMS documentation for information. See also the section Restrictions for the use of outer joins [page 162] for more information on restrictions using outer joins.

5.8.3.2 Creating an outer join

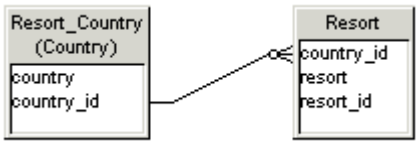
To create an outer join:

1. Double click an existing equi-join.
   The Edit Join dialog box appears.
2. Select the Outer Join check box for the table that returns all values in a query.
   In the example below, you want to return all values for Resort_Country.



3. Click the Parse button to validate the join syntax.
   If you receive an error message, check to see that you selected the columns correctly.
4. Click OK.
   The universe design tool displays the join in the Structure pane. If the outer join is created using *Arity* graphic option, a small circle is indicated on the opposite side of the join to the table that returns unmatched values.

> **i Note**
>
> If you create outer join using a graphic option other than *Arity*, the small circle does not appear.



5.8.3.3 Defining a full outer join

You can define an outer join using the ANSI 92 standard for defining outer joins. This allows you to specify a full outer join. To use the ANSI 92 standard for outer joins, you must set the ANSI 92 parameter to YES. This parameter is available on the Parameter page (File > Parameters > Parameter).

> **i** Note
>
> For information on setting this parameter and other SQL generation parameters for the universe, refer to the section Setting SQL generation parameters [page 90].

When the ANSI 92 parameter has been set to YES, you can select the tables on both sides of the join to be outer tables. Before setting this parameter, you must ensure that your target RDBMS supports the ANSI 92 syntax for outer joins.

You define a full outer join in two phases:

- Activate ANSI 92 support for outer joins for the universe. See the section Activating ANSI 92 support in a universe [page 149] for information.
- Use the Edit join dialog box to define the full outer join.

To define a full outer join:

1. Activate ANSI 92 support for the universe.
2. Double click a join in the schema.
   The Edit Join dialog box appears.
3. Select the Outer Join check box for both tables included in the join.
4. Click OK.
   The universe design tool displays the join in the Structure pane. The full outer join is indicated by two circles on the join link between two tables.

5.8.3.4 Restrictions for the use of outer joins

Using outer joins can be very useful, but you should be aware of the following performance and implementation issues:

- Performance can be slower. More rows are returned and some databases will not use indexes when outer joins are involved, so large amounts of data could slow query performance.
- Database limitations on the use of outer joins. Not all databases allow control over outer joins in the WHERE clause. This is necessary when using a self restricting join. For example, a self restricting join 'TYPE_CODE=10', could return all rows where TYPE=10 or Type is NULL, as TYPE=10 will never be true when the type code is NULL, whereas NULL values are generated by the outer join.
- You should verify how your target RDBMS processes outer joins to avoid incomplete query paths after the original outer join. For example, in the Microsoft Access sample Club.mdb database, all one-to-many joins following the outer join in the join path must also be defined as outer joins. If not, the original outer join will be ignored by the resulting query. In the example below, the join between Resort and Service_Line ignores the NULL values returned by the outer join between Resort_Country and Resort. When you run a query with the three tables, a database error is returned advising the user to create a separate query that performs the first join, and then include that query in the SQL statement. This type of error could be

confusing to many users, so it is preferable in such cases to either not use outer joins, or to complete the path with outer joins.



5.8.4 Shortcut joins

A shortcut join is a join that provides an alternative path between two tables. Shortcut joins improve the performance of a query by not taking into account intermediate tables, and so shortening a normally longer join path.

A common use of shortcut joins is to link a shared lookup table to another table further along a join path. The join path comprises several different tables in the same context.

In such a case, the shortcut join is only effective when the value being looked up has been denormalized to lower levels in a hierarchy of tables, so the same value exists at all the levels being joined.

The shortcut join will be ignored if it is not “short cutting” any join path for a given context. The SQL generated for the related Web Intelligence query will not take into account the ineffective shortcut join.

> **i** Note
>
> The universe design tool does not consider shortcut joins during automatic loop and context detection. However, if you set the cardinality for a shortcut join you avoid receiving the message 'Not all cardinalities are set' when detecting contexts.

5.8.4.1 Creating a shortcut join

To create a shortcut join:

1. Identify the two tables in a join path that can be linked directly.
2. Create a join between the two tables.
3. Double click the new join.
   The Edit Join dialog box appears.
4. Select the Shortcut join check box.
5. Select or type other join properties as required.
6. Click OK.
   The shortcut join appears joining the two tables. A shortcut join is shown as dotted line in the Structure pane.

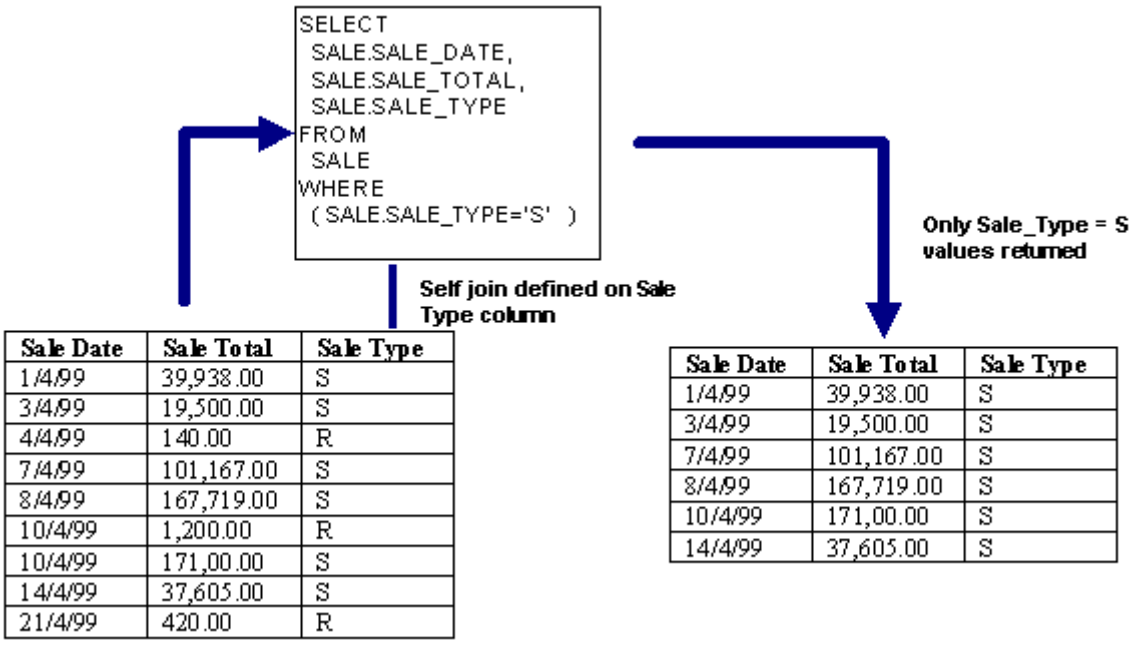
> **i** Note
>
> You should set the cardinality of a shortcut join to the same cardinality as the join path it replaces.

5.8.5 Self restricting joins

A self restricting join is not really a join at all, but a self restriction on a single table. You can use a self restricting join to restrict the results returned by a table values using a fixed value.

Example: Self restricting join

The Sales table shown below contains rows of data for cars both sold and rented. The Sale_Type column is used as a flag to indicate the type of transaction (S = car sale, R = car rental). The self restricting join restricts the data returned from Sales to Sale_Type = S. This ensures that any object based on the Sales table, or joins passing through that table, would produce query results covering only car sales.



Sale Date	Sale Total	Sale Type
1/4/99	39,938.00	S
3/4/99	19,500.00	S
4/4/99	140.00	R
7/4/99	101,167.00	S
8/4/99	167,719.00	S
10/4/99	1,200.00	R
10/4/99	171,00.00	S
14/4/99	37,605.00	S
21/4/99	420.00	R

Sale Date	Sale Total	Sale Type
1/4/99	39,938.00	S
3/4/99	19,500.00	S
7/4/99	101,167.00	S
8/4/99	167,719.00	S
10/4/99	171,00.00	S
14/4/99	37,605.00	S

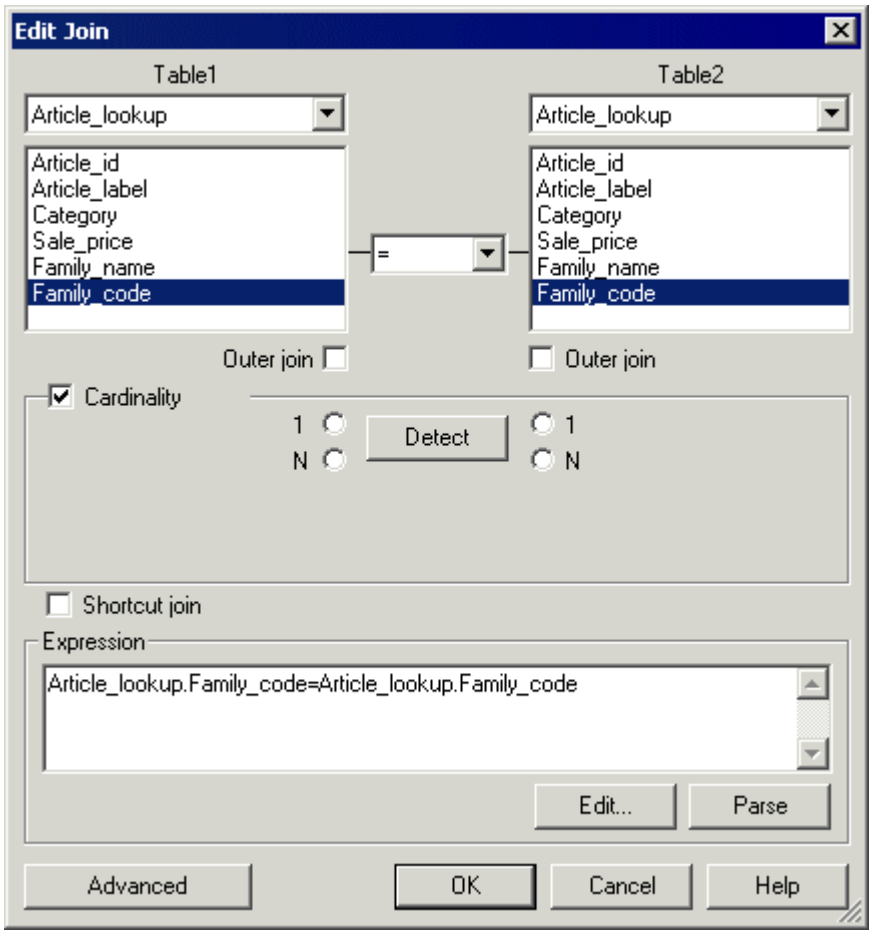
Without the self restricting join, the results set of the query would produce rows where the Sale_Type column is equal to either 'S' or 'R'.

> → Tip
>
> Setting the cardinality for a self restricting join helps to prevent receiving the message 'Not all cardinalities are set' when detecting contexts. You should set cardinality as one-to-one consistently, although the actual setting is not important, as long as it is set.

5.8.5.1 Creating a self restricting join

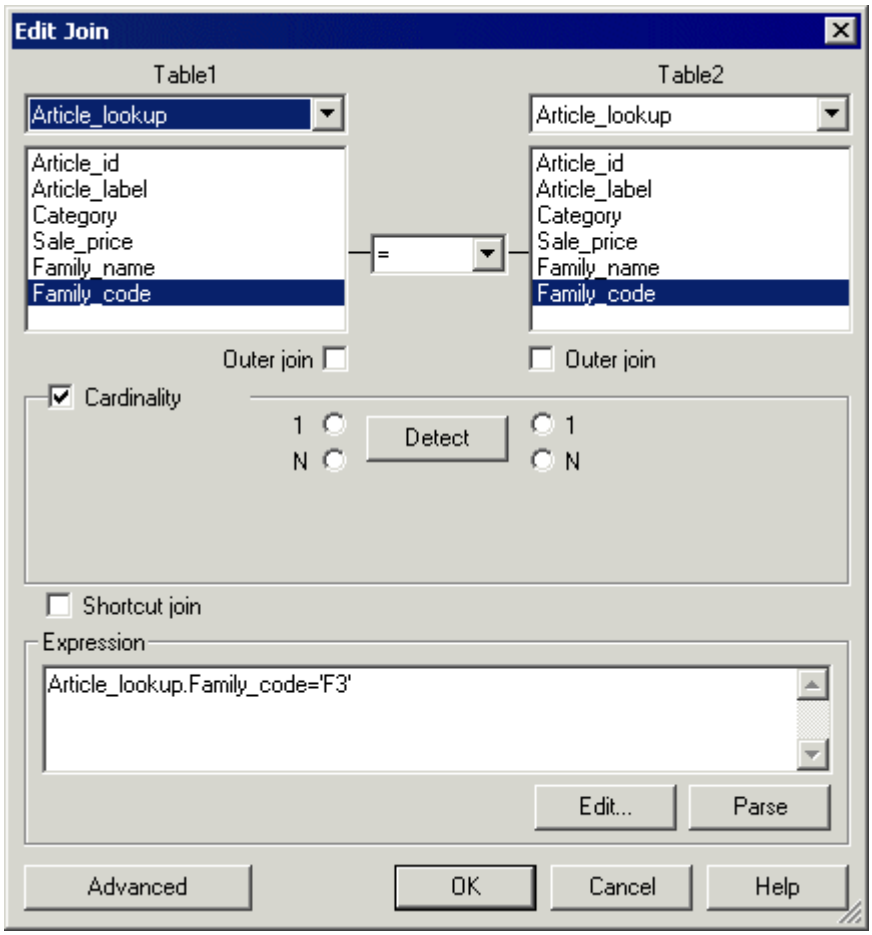
To create a self restricting join:

1. Select Insert > Join.
   The Edit Join dialog box appears.
2. Select the table that you want to set the self restricting join against from the Table1 drop- down list box.
   The columns for the selected table appear in the table column list.
3. Click the column that you want to use to define the restriction from the column drop-down list box.
4. Select the same table that you selected from the Table1 drop-down list box.
5. Click the same column that you selected in the Table1 column list box.
   The expression for the join appears in the Expression text box.

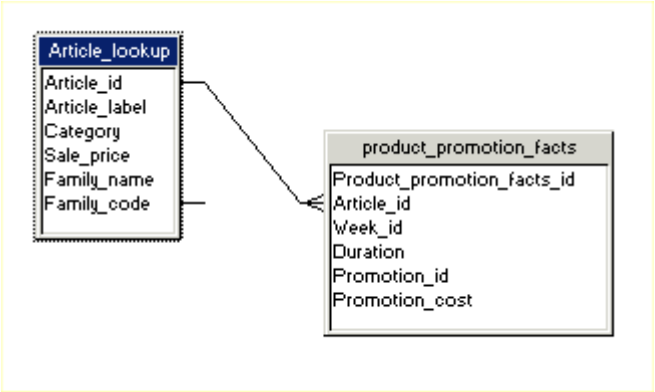


6. Replace the operand value in the join expression with the restriction value that you want to set on the join column.

For example, if you want to restrict the returned values from the Family_code column to 'F3', you replace Article_lookup.Family_code after the = sign with 'F3' as shown below:



7. Click the Parse button to verify the syntax.
8. Click OK.
   The self restricting join appears as a short line displayed against the column on which the self restricting join is defined.



5.9 Using cardinalities

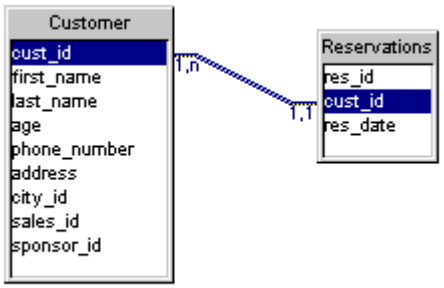
Cardinality is a property of a join that describes how many rows in one table match rows in another table.

Cardinality is expressed as the minimum and maximum number of rows in a column at one end of a join, that have matching rows in the column at the other end of the join.

The minimum and the maximum number of row matches can be equal to 0, 1, or N. A join represents a bidirectional relationship, so it must always have two cardinalities, one for each end of the join.

Example: Cardinality of a join

The two tables Customer and Reservations are linked by a join.



The cardinalities in the above join can be expressed as follows:

Description	Notation
For each customer, there can be one or more reservations	(1,N)
For each reservation, there can be one and only one customer	(1,1)

5.9.1 How are cardinalities used in the universe design tool?

The cardinality of a join does not have a role in the SQL generated when you run a query. However, the universe design tool uses cardinalities to determine contexts and valid query paths.

A context is a collection of joins which provide a valid query path. You use contexts to resolve join problems that can return too many or too few rows because of the way that tables are linked in the target database. Contexts are described in Detecting and Solving Join Problems [page 184].

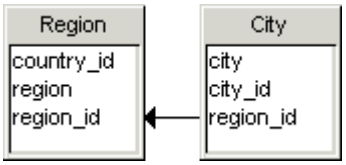
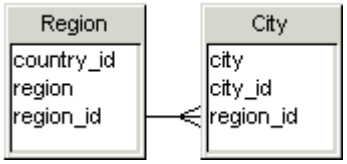
Contexts affect the SQL generated for a query as they either direct the end user to take a particular join path, or solve a join path problem:

You need to verify that cardinalities are correctly set for all joins in your schema to ensure that you have the correct contexts, and that you have valid join paths.

Setting cardinalities can also help you understand how tables are related in the database, and to graphically identify potential join path problems in your schema.

5.9.1.1 Displaying cardinalities

You can display cardinalities in the Structure pane using the following symbols:

Cardinality symbol	Example	Description
Arrow		Arrow indicates the "one" direction of the join. If cardinality is 1,1 then an arrow head is shown at each join end.
Arity		Crow's foot indicates the "many" end of the join. If cardinality is 1,1, then a straight line is shown.
1,N		Cardinality is shown as a ratio at each end of the join.

To display cardinalities:

1. Select Tools > Options.
   The Options dialog box opens to the General page.
2. Click the Graphics tab.
   The Graphics page appears.
3. Click the Arrow, Arity, or 1,n radio button.
4. Click OK.

5.9.2 Setting cardinalities manually

You can manually set cardinalities for joins by defining cardinality for a join in the *Edit Join* box for a join.

Why set cardinalities manually?

When you set cardinalities manually, you must consider each individual join. This helps you to become aware of potential join path problems in your schema. You may not find these problems if you only select automatically detected cardinalities; for example, isolated one-to-one joins at the end of a join path, or excessive primary keys where not all columns are required to ensure uniqueness.

Understanding keys

You determine cardinalities for most join cases by evaluating the primary and foreign keys in each table. Primary and foreign keys are described as follows:

Key	Description
Primary	Single or combination of columns in a table whose values identify each row in the table. The primary key guarantees row uniqueness in a table. Each table has only one primary key.
Foreign	Column or combination of columns whose values are required to match a primary or another unique key in another table. Foreign keys implement constraints such as 'you cannot create a sale for a customer if that customer hasn't yet been created'. Each table can have multiple foreign keys.

What are the criteria for setting cardinalities?

You evaluate the relationship between primary and foreign keys to determine the cardinality for a join as follows:

If join links...	Cardinality is likely to be...
Complete primary key of Table 1 with complete primary key of Table 2. For example:	One-to-one (1,1). Only one row from each table will be returned for each primary key value.

If join links...	Cardinality is likely to be...
Complete primary key of one Table 1 with corresponding foreign key of Table 2. For example:	One-to-many (1,N). Foreign key values of a table are not guaranteed to be unique and so can return many matching values for a single value of the primary key on the original table.

If join links...	Cardinality is likely to be...
Complete primary key of Table 1 with part of primary key of Table 2. For example:	One-to-many (1,N). The incomplete primary key match can return many matching values for a single value of the primary key on the original table.

5.9.2.1 Setting cardinalities manually

You can manually set cardinalities for joins by defining cardinality for a join in the *Edit Join* box for a join.

Why set cardinalities manually?

When you set cardinalities manually, you must consider each individual join. This helps you to become aware of potential join path problems in your schema. You may not find these problems if you only select automatically detected cardinalities; for example, isolated one-to-one joins at the end of a join path, or excessive primary keys where not all columns are required to ensure uniqueness.

Understanding keys

You determine cardinalities for most join cases by evaluating the primary and foreign keys in each table. Primary and foreign keys are described as follows:

Key	Description
Primary	Single or combination of columns in a table whose values identify each row in the table. The primary key guarantees row uniqueness in a table. Each table has only one primary key.
Foreign	Column or combination of columns whose values are required to match a primary or another unique key in another table. Foreign keys implement constraints such as 'you cannot create a sale for a customer if that customer hasn't yet been created'. Each table can have multiple foreign keys.

What are the criteria for setting cardinalities?

You evaluate the relationship between primary and foreign keys to determine the cardinality for a join as follows:

If join links...	Cardinality is likely to be...
Complete primary key of Table 1 with complete primary key of Table 2. For example:	One-to-one (1,1). Only one row from each table will be returned for each primary key value.

If join links...	Cardinality is likely to be...
Complete primary key of one Table 1 with corresponding foreign key of Table 2. For example:	One-to-many (1,N). Foreign key values of a table are not guaranteed to be unique and so can return many matching values for a single value of the primary key on the original table.

If join links...	Cardinality is likely to be...
Complete primary key of Table 1 with part of primary key of Table 2. For example:	One-to-many (1,N). The incomplete primary key match can return many matching values for a single value of the primary key on the original table.

5.9.2.2 Detecting cardinalities automatically

You can use the universe design tool feature Detect Cardinalities to automatically detect cardinalities for the following situations:

- Selected joins
- All joins
- At join creation
- From the Edit Join box

When using automatic cardinality detection, cardinalities are implemented automatically on detection.

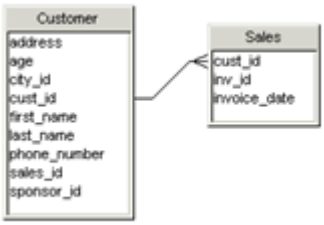
> **i Note**
>
> You should use automatic cardinality detection appropriately. It can be very useful to quickly get all the cardinalities detected in the schema, however, there are a number of structural problems inherent in many relational databases which can lead to incorrect cardinality detection. These include incomplete primary joins, and over engineered primary keys. These are discussed in the section Optimizing automatic cardinality detection [page 175].

Detecting cardinalities automatically for selected joins

To automatically detect cardinalities for a selected join:

- Click a join and select Tools > Detect Cardinalities.
- Right click a join and select Detect Cardinalities from the contextual menu.

The cardinality is displayed with the crow's foot at the many end.



If you select Tools > Detect Cardinalities directly without selecting a join, you receive a message indicating that no join is selected, and asking if you want to detect cardinalities for all joins.

Detecting cardinalities automatically for all joins

To automatically detect cardinalities for all joins:

1. Select Tools > Automated Detection >Detect Cardinalities.
   Or

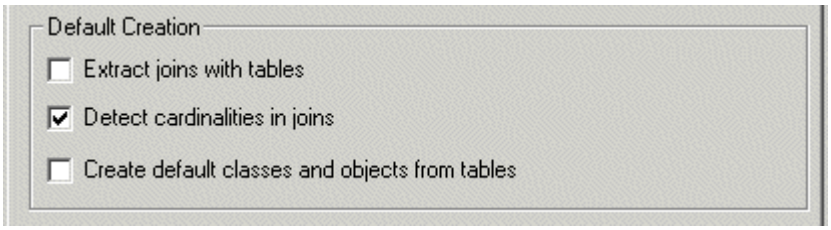
   
   Detect cardinalities

   Click the Detect Cardinalities button.
   A message box appears asking if you want to detect cardinalities for all joins.
2. Click Yes.
   All joins in the Structure pane are shown with cardinalities.

Automatically detecting cardinalities on join creation

To automatically detect cardinalities on join creation:

1. Select Tools > Options.
   The Options dialog box opens to the General page.
2. Click the Database tab.
   The Database page appears.
3. Select the Detect Cardinalities in Joins check box.

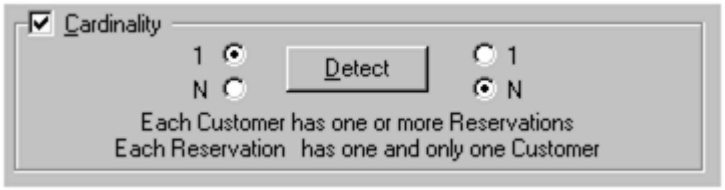


4. Click OK.
5. When you create a new join, the cardinality is automatically detected and displayed on the join.

Automatically detecting cardinality from the Edit Join box

To automatically detect cardinality from the Edit Join box:

1. Double click a join.
   The Edit Join dialog box appears.
2. Select the Cardinality check box.
3. Click the Detect button.
   The cardinality radio buttons are automatically selected for the detected cardinality. The two cardinalities are also expressed in sentence form.



4. Click OK.

5.9.2.3 Optimizing automatic cardinality detection

You can improve the response time of cardinality detection by modifying a parameter in the PRM file of the target RDBMS. This directs the detection algorithm to read two instead of three SQL statements, improving the performance of the algorithm.

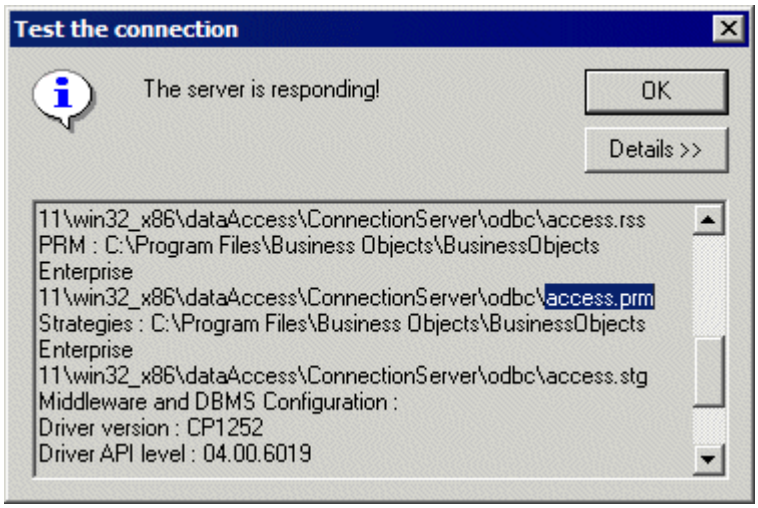
The PRM file is a text file that lists parameters used to configure universe creation and SQL query generation in Web Intelligence. There is a PRM file for each supported RDBMS.

PRM files are located in the database folders under <INSTALLDIR>\win32_x86\dataAccess\ConnectionServer\

Verifying which PRM file is used by a connection

To verify which PRM file is used by a universe connection:

1. Select File > Parameters.
   The Parameters dialog box appears.
2. Click the Test button.
   The Test Connection message box appears.
3. Click the Details button.
   The details of your connection appear in a drop down message box.
4. Scroll down the message box to the line that starts with PRM.
   This line indicates the file path and name of the PRM file currently used by the active universe.



5. Click OK.
   You return to the Parameters dialog box.
6. Click Cancel.

Optimizing cardinality detection using the PRM file

To optimize cardinality detection using the PRM file:

1. Open the PRM file for your target database in a text editor.
   The PRM files are stored in the Data Access folder in the Business Objects path.
2. Set the LIGHT_DETECT_CARDINALITY parameter to YES.
3. Save and close the PRM file.
   The next time you open the universe, automatic cardinality detection is optimized.

5.9.2.4 Optimizing automatic cardinality detection

You can improve the response time of cardinality detection by modifying a parameter in the PRM file of the target RDBMS. This directs the detection algorithm to read two instead of three SQL statements, improving the performance of the algorithm.

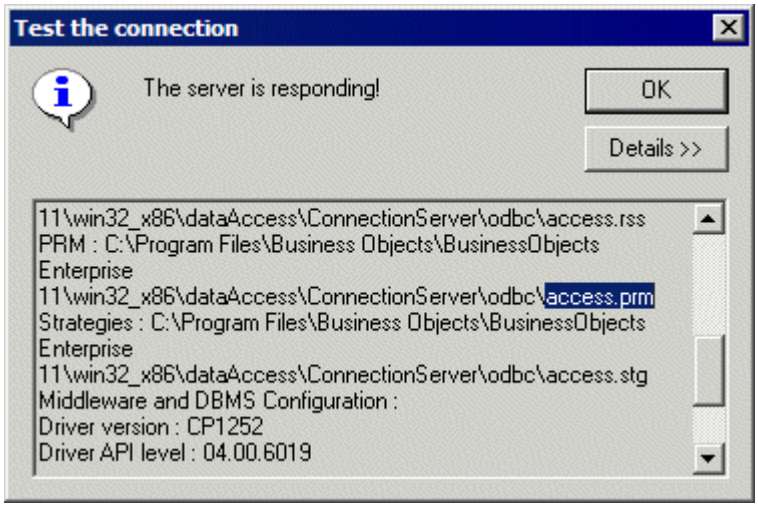
The PRM file is a text file that lists parameters used to configure universe creation and SQL query generation in Web Intelligence. There is a PRM file for each supported RDBMS.

PRM files are located in the database folders under <INSTALLDIR>\win32_x86\dataAccess\ConnectionServer\

Verifying which PRM file is used by a connection

To verify which PRM file is used by a universe connection:

1. Select File > Parameters.
   The Parameters dialog box appears.
2. Click the Test button.
   The Test Connection message box appears.
3. Click the Details button.
   The details of your connection appear in a drop down message box.
4. Scroll down the message box to the line that starts with PRM.
   This line indicates the file path and name of the PRM file currently used by the active universe.



5. Click OK.
   You return to the Parameters dialog box.
6. Click Cancel.

Optimizing cardinality detection using the PRM file

To optimize cardinality detection using the PRM file:

1. Open the PRM file for your target database in a text editor.
   The PRM files are stored in the Data Access folder in the Business Objects path.
2. Set the LIGHT_DETECT_CARDINALITY parameter to YES.
3. Save and close the PRM file.
   The next time you open the universe, automatic cardinality detection is optimized.

5.10 Checking the universe

As you design your universe, you should test its integrity periodically. You can verify universe integrity as follows:

Check universe	Description
Automatically	You can set the universe design tool options to check the SQL syntax of universe structures at creation, universe export, or when a universe is opened.
Manually	You run Check Integrity to check selected universe structures.

5.10.1 Checking Universe Integrity Automatically

You can set the following integrity check options in the universe design tool to parse SQL structures at creation, universe export, and universe opening:

Automatic check option	Description
Automatic parse upon definition	The universe design tool automatically checks the SQL definition of all objects, conditions, and joins at creation. It is applied when you click OK to validate structure creation.
Send check integrity	The universe design tool displays a warning each time you attempt to export an unchecked universe.
Check universe integrity at opening	All universes are checked automatically when opened.

5.10.1.1 Setting automatic universe check options

To set automatic universe check options:

1. Select Tools > Options.
   The Options dialog box opens to the General page.
2. Select or clear check boxes for appropriate universe automatic check options in the Integrity group box.



3. Click OK.

5.10.1.2 Checking universe integrity manually

You can use Check Integrity to test to verify if the design of your active universe is accurate and up-to-date.

Check Integrity detects the following:

- Errors in the objects, joins, conditions, and cardinalities of your universe.
- Loops in join paths.
- Any necessary contexts.
- Changes to the target database.

Before examining the elements of the universe against those of the database, the function checks whether the connection to the database is valid. If the connection is not valid, the function stops and returns an error message.

5.10.1.3 Types of errors detected by Check Integrity

Check Integrity can detect:

- Invalid syntax in the SQL definition of an object, condition, or join.
- Loops
- Isolated tables
- Isolated joins
- Loops within contexts
- Missing or incorrect cardinalities

How does Check Integrity determine changes in a connected database?

The Check Integrity function sends a request to the database for a list of tables. It then compares this list with the tables in the universe. It carries out the same action for columns.

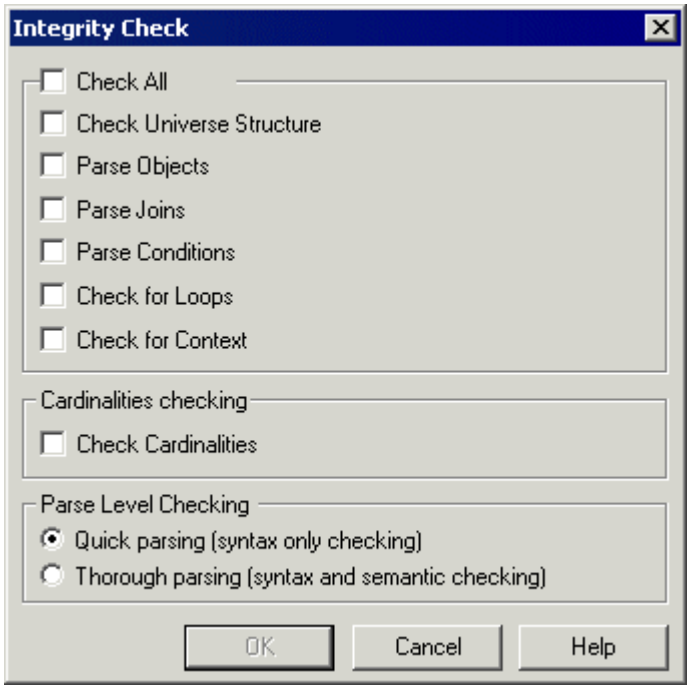
In the Structure pane, Check Integrity marks any tables or columns not matching those in the list as not available. These are tables or columns that may have been deleted or renamed in the database. See the section Verifying universe integrity with Check Integrity [page 178].

> **i** Note
>
> The option Check Cardinalities can be slow to run with large amounts of data. If there is ambiguous or missing data, results can also be inaccurate. If your database is large, and may have incomplete data entries, then you should not select the option Check Cardinalities. If you do use this option, then you can optimize the cardinality detection by modifying the PRM file. For more information, refer to the section Optimizing automatic cardinality detection [page 175].

5.10.1.4 Verifying universe integrity with Check Integrity

To verify universe integrity:

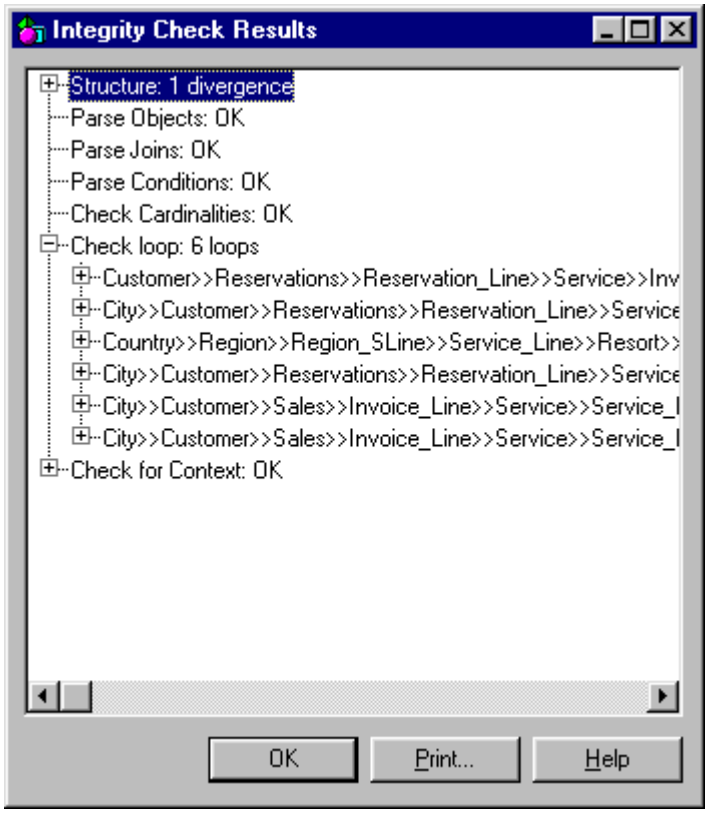
1. Select Tools > Check Integrity.
   Or
   Click the Check Integrity button.
2. The Integrity Check dialog box appears.



3. Select check boxes for components to be verified.

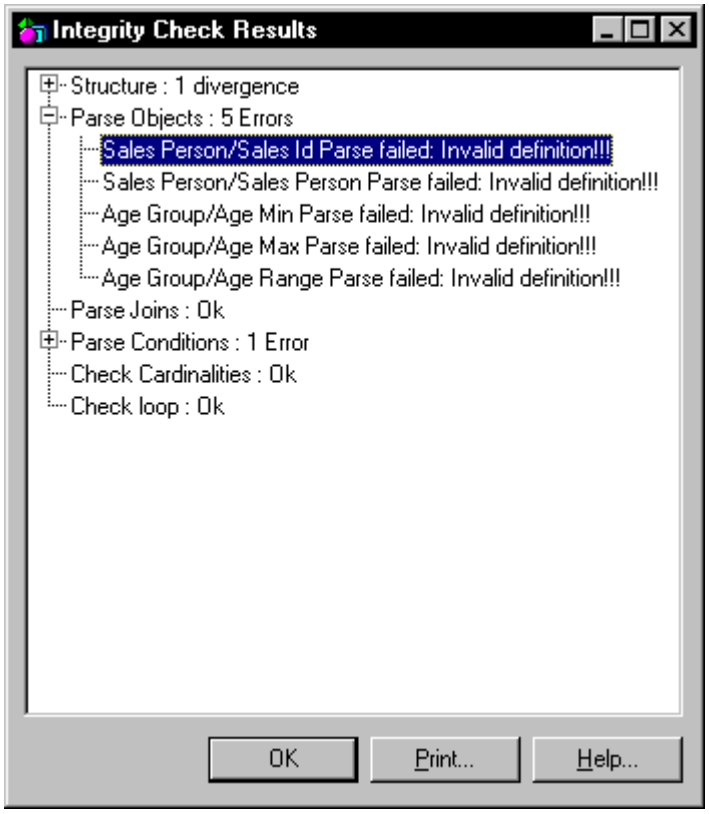
> **i Note**
>
> You can select Check Cardinalities independently of the Check All option. This allows you to verify the universe structure without checking cardinalities which may take a long time depending on the database.

4. Clear check boxes for components not to be verified.
5. Select the Quick Parsing check box to verify only the syntax of components.
   Or
   Select Thorough Parsing check box to verify both the syntax and semantics of components.
6. Click OK.
   A message box displays the universe check progress.



If Check Integrity encounters no errors, it displays "OK" beside each error type.

7. Click the plus sign (+) beside the error type to view the list of components in which the error occurred.



You can double click an item in the list to highlight the corresponding components in the Structure pane.

8. Click the Print button to print the window contents.
9. Click OK.

> **i** Note
>
> Before selecting the Check for Loops check box, ensure that the cardinalities of joins have already been detected. Otherwise, the function erroneously identifies loops in the joins.

5.10.1.5 Refreshing the Universe Structure

If Check Integrity indicates that the database of your universe connection has been modified, you can use Refresh Structure to update the contents of the Structure pane.

Refresh Structure can modify the universe structure to comply with changes in the database as follows:

If	Then the tool does the following
Columns were added to tables	Adds the columns to the corresponding tables in the universe.

If	Then the tool does the following
Columns were removed from tables	Displays a warning message indicating the columns and associated joins you should delete.
Tables were removed from the database	Displays a warning message indicating the tables and associated joins you should delete.
Tables were renamed in the database	Displays a message that says it no longer recognizes the corresponding tables in the universe. You should rename these tables to match those in the database. If the names still do not match, the universe design tool returns a message stating that the renamed tables do not exist in the database.
No changes were made to the database	Displays a message informing you that no update is needed.

To refresh the universe structure:

- Select View > Refresh Structure.
- A message box appears informing you of a change in the database, or that no update is needed if no changes have been made.

6 Resolving join problems in a schema

6.1 Overview

This chapter describes the types of problems that can arise as you create joins between the tables in your schema. It explains how you can detect and resolve these join problems to ensure that the join paths taken by queries run on the universe return correct results. You must resolve the join problems before building the universe.

6.2 What is a join path problem?

A join path is a series of joins that a query can use to access data in the tables linked by the joins.

Join path problems can arise from the limited way that lookup and fact tables are related in a relational database. The three major join path problems that you encounter when designing a schema are the following:

- loops
- chasm traps
- fan traps

You can solve all these problems by creating aliases (a copy of a base table), contexts (a defined join path), and using features available in the universe design tool to separate queries on measures or contexts.

This section briefly defines lookup and fact tables, and describes the types of join path problems that you can encounter using these tables. It explains how you can use aliases, contexts, and other universe design tool features to resolve join path problems in your universe schema.

In the universe design tool, you typically create joins between lookup tables and fact tables.

6.2.1 What is a Lookup Table

A lookup (or dimension) table contains information associated with a particular entity or subject. For example, a lookup table can hold geographical information on customers such as their names, telephone numbers as well as the cities and countries in which they reside.

In the universe design tool, dimension and detail objects are typically derived from lookup tables.

6.2.2 What is a Fact Table

A fact table contains statistical information about transactions. For example, it may contain figures such as Sales Revenue or Profit.

In a universe, most but not all, measures are defined from fact tables.

6.2.3 What Types of Join Paths Return Incorrect Results?

Queries can return incorrect results due to the limitations in the way that joins are performed in relational databases. Depending on how the lookup and fact tables in your table schema are related, join paths can produce instances where a query returns too few, or too many rows.

The following types of join paths can produce incorrect results:

Type of Join Path	Returns	Description
Loop	Too few rows	Joins form multiple paths between lookup tables.
Converging many to one joins	Too many rows	Many to one joins from two fact tables converge on a single lookup table. This type of join convergence can lead to a join path problem called a chasm trap.
Serial many to one joins	Too many rows	A one to many join links a table which is in turn linked by a one to many join. This type of fanning out of one to many joins can lead to a join path problem called a fan trap.

6.2.4 Detecting and Solving Join Problems

The universe design tool provides a number of methods for detecting and solving join problems. Each of these methods is fully described in its corresponding section.

You can use the following methods to detect and solve join path problems:

Join Problem	Detected by	Solved by
Loop	• Detect Aliases • Detect Contexts • Detect Loops • Check Integrity • Visual analysis of schema	Creating aliases and contexts to break loops.
Chasm trap (converging many to one joins)	Visual analysis of table schema.	• Creating a context. • Using the feature Multiple SQL statements for each measure. • Creating multiple universes (Web Intelligence only).
Fan trap (serial many to one joins)	Visual analysis of table schema.	• Creating an alias, creating a context using the alias, then building affected measure objects on the alias. • Using Multiple SQL Statements for Each Measure.

Most join path problems can be solved by creating an alias or implementing a context. You can use the automatic loop detection tools in the universe design tool to identify loops in the schema, and automatic context detection to identify where Chasm traps occur. However, to resolve fan traps, you have to be able to visually analyze the schema and create aliases and if necessary contexts manually.

6.3 Defining aliases

Aliases are references to existing tables in a schema. An Alias is a table that is an exact duplicate of the original table (base table), with a different name. The data in the table is exactly the same as the original table, but the different name "tricks" the SQL of a query to accept that you are using two different tables.

The Beach universe schema contains two alias tables; Resort_Country (the alias for the Country table) and Sponsor (the alias for the Customer table). For each alias table, the name of the original table is shown in brackets.

6.3.1 How are Aliases Used in a Schema?

You use aliases for two main reasons:

- To use the table more than once in a query. This is the main reason for using aliases, and includes using aliases to solve loops and fan traps. The example Beach universe contains 2 aliases; Resort_Country for Country, and Sponsor for Customer.
- To abbreviate the table name to save typing when writing freehand SQL.

→ Tip

Another possible use of aliases is to create an alias for each table as it is inserted into the schema. You then build the schema using the alias tables, not the original base tables. You place the base tables together away from the main universe structure. This allows you to give meaningful names to tables, and prevents the need to rebuild major sections of a universe structure should a base table need to be aliased at a later stage.

6.3.1.1 Using aliases to solve loops

The most common use of aliases in universe development is to solve potential loops in the use of common tables. A loop is a set of joins that defines a closed path through a set of tables in a schema. Loops occur when joins form multiple paths between lookup tables

You use an alias to break a loop by providing alternative table for an original lookup table that is being used for multiple query paths. This use of aliases is discussed in the section Resolving loops [page 199].

6.3.1.2 Using aliases to solve fan traps

Aliases are also used to solve potential fan traps. These can occur in a serial one-to-many join path that can return inflated results when aggregates are summed at the "many" end of the joins. This use of aliases is discussed in the section Resolving chasm traps [page 225].

6.3.2 Creating Aliases

You can create aliases manually, or let the universe design tool automatically detect potential aliases that will solve a join path loop.

You need to create an alias manually to solve a fan trap. You also create aliases manually if you are creating a schema using only aliases and not the base tables.

The automatic detection and creation of aliases to solve loops is described in the section Detecting and creating an alias [page 210].

6.3.2.1 Creating an alias manually

To create an alias manually:

1. Click the table that you want to use to create an alias.
2. Select Insert > Alias
   Or
   Click the Insert Alias button.

The Creating an Alias box appears. It prompts you to enter a name for the new alias.



3. Enter a new name for the aliased table, or keep the one proposed.

> **i** Note
>
> The name that you give to an alias should be relevant to the role of the alias to distinguish it from the base table. For example, Resort country is an alias for Country. Resort Country is used for queries returning data for resort countries, the base table Country is used in queries returning data for customer countries.

4. Click OK.
   The aliased table appears in the Structure pane.
5. Create any joins necessary between the alias and other tables in the schema.

> **→** Tip
>
> To avoid confusing base tables with aliases, you can display the alias with the name of the base table it represents in the table title as follows: Select Tools > Options > Graphics, and then select the Aliased Name check box.

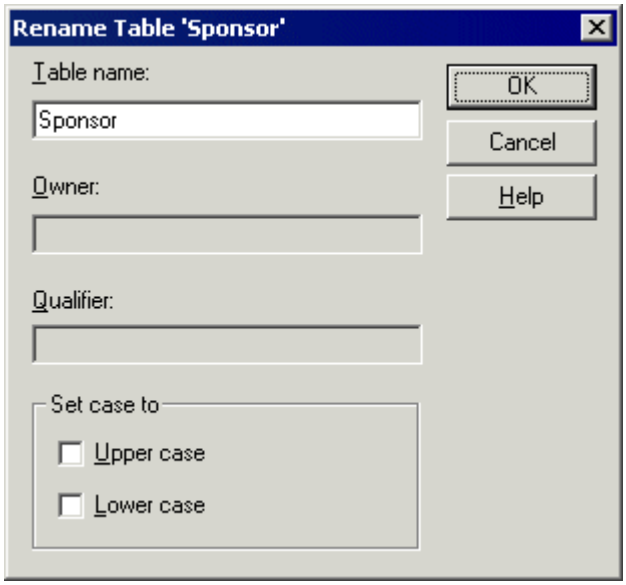
6.3.2.2 Renaming an alias

You can rename an alias at any time. Alias and table naming conventions are RDBMS dependent. You can rename an alias directly by renaming the table, or from a list of aliases in the universe.

Renaming an alias directly

To rename an alias directly:

1. Click a table and select Edit > Rename Table.
   Or
   Right click a table and select Rename table from the contextual menu.
   The Rename Table dialog box appears.



2. Type a new name in the Table Name box.
   The availability of the Owner and Qualification fields is database specific. If they are active, then you can modify these as necessary.
3. Select the Upper case check box if you want the alias name to be shown as all uppercase.
   Or
   Select the Lower case check box if you want the alias name to be shown as all lowercase.
4. Click OK.

Renaming an alias from a list

To rename an alias from a list:

1. Select Tools > List of Aliases.
2. The List of Aliases appears. It lists all the aliases in the active universe.
3. Click an alias name in the list.
4. Type a new name for the selected alias in the New Name text box.
5. Click Apply.
6. Click OK.

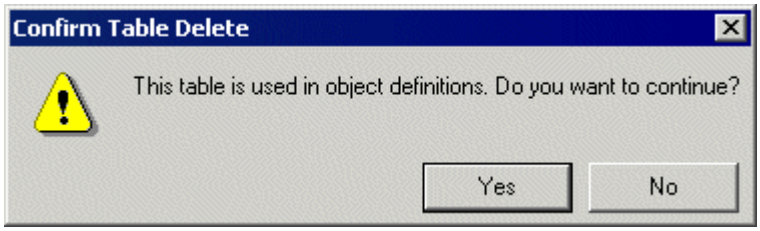
6.3.2.3 Deleting an alias

You delete an alias in the same way that you delete a table. If you have defined objects using the alias, you must modify these objects before you delete the alias, so that they use another table, or delete the objects if they are no longer necessary.

If you do not modify or remove the objects using a deleted alias, queries using those objects will generate errors in Web Intelligence.

To delete an alias:

1. Click an alias and select Edit > Clear.
   Or
   Right click an alias and select Clear from the contextual menu.
   Or
   Click an alias and press the DELETE key.
   If any objects use the alias, the following message appears:



   If no objects use the alias, you do not receive a confirmation box. The alias is deleted immediately.
2. Click Yes.
   The alias is deleted from the Structure pane.

6.4 Defining contexts

Contexts are a collection of joins which provide a valid query path for Web Intelligence to generate SQL.

6.4.1 How are contexts used in a Schema?

You can use contexts in a universe schema for the following purposes:

- Solving loops.
- Solving chasm traps.
- Assisting in some solutions for fan traps.
- Assisting in detecting incompatibility for objects using aggregate awareness.

6.4.1.1 Using contexts to solve loops

The most common use of contexts is to separate two query paths, so that one query returns data for one fact table, and the other query returns data for another fact table. You use contexts to direct join paths in a schema which contains multiple fact tables. Aliases are not appropriate in such schema. This use of contexts is covered in the section Resolving loops [page 199].

6.4.1.2 Using contexts to solve chasm and fan traps

Contexts are also used to solve potential chasm traps. These can occur when two many-to-one join paths converge on a single table. Multiple rows can be returned for a single dimension causing inflated results. Contexts can split out the query so that the correct number of rows are returned for the dimension. Contexts can also be used with aliases to solve fan traps. These uses of contexts are discussed in the section Resolving chasm traps [page 225].

6.4.1.3 Using contexts to determine AggregateAwareness incompatibility

You can use contexts to exclude objects that are not compatible with an object using the @AggregateAware function in its definition, from being used in a query with the aggregate aware object. .

6.4.2 Creating a context

You can let the universe design tool automatically detect contexts, or you can create contexts manually.

If you are using a context to resolve a loop or a chasm trap, you should always let the universe design tool detect the contexts. However, for solving a fan trap (another join path problem), you may have to manually build a context.

The automatic detection of contexts for loop resolution is described in the section Resolving loops [page 199].

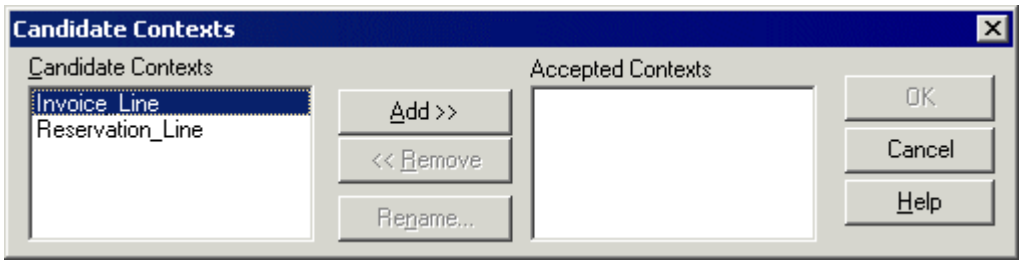
> **i Note**
>
> When you create one or more contexts, all joins must be included in one or multiple contexts. If a table is linked by a join that is not included in a context, the join will not be considered when a query is run.

The following procedures describe how you can create a context automatically and manually.

6.4.2.1 Creating a context automatically

To create a context automatically

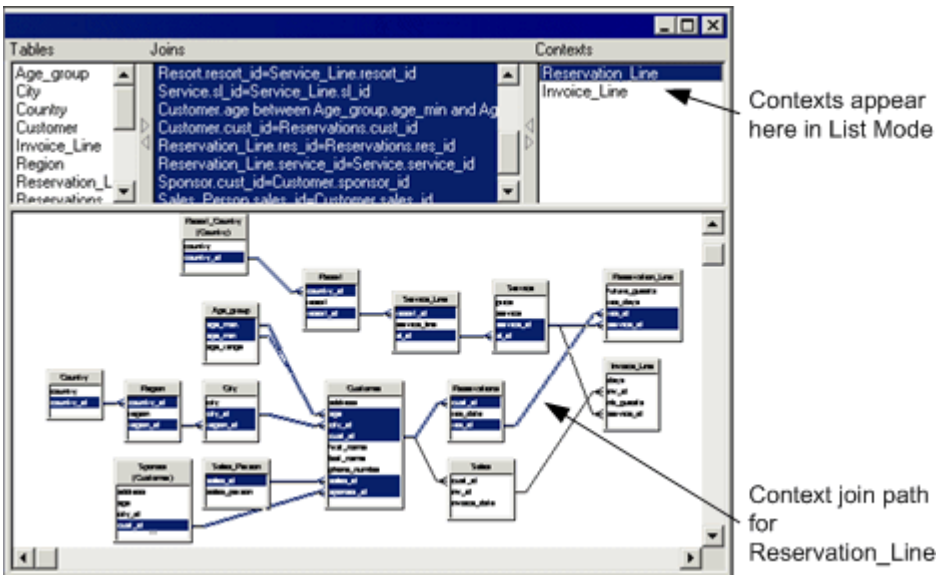
1. Select Tools > Automated Detection >Detect Contexts.
   The Candidate Contexts box appears. It proposes candidate contexts for your schema. These candidate contexts may be necessary to solve either loops or a chasm trap, as chasm traps exist at the branch where two contexts meet.



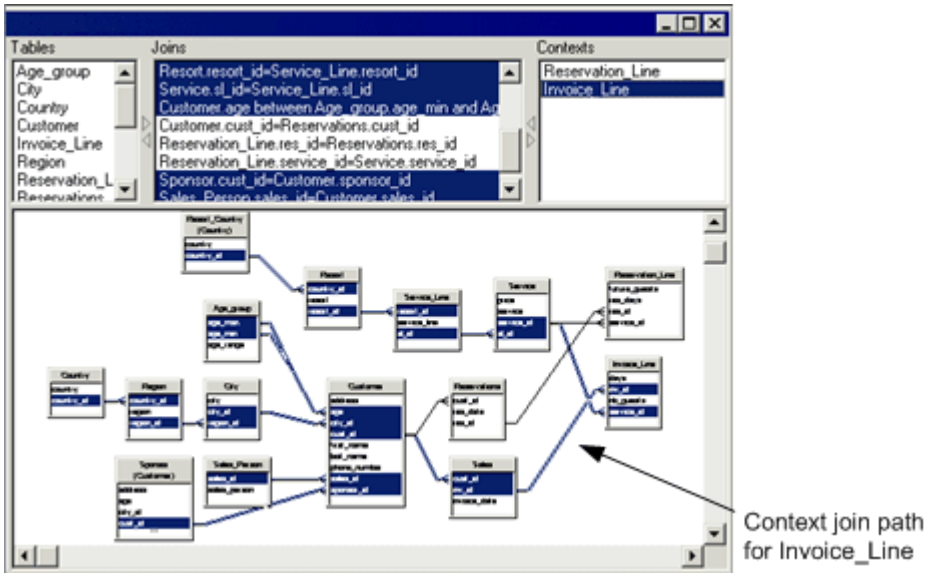
2. Click a context in the Candidate Contexts list and click the Add button.
3. Repeat step 2 for each candidate context in the list.

> **i Note**
>
> Once you have added the candidate context to the Accepted Contexts list, you can rename a context as follows: Click a context and click the Rename button. An edit box appears. Type the new name and click OK.

4. Click OK.
   The contexts are listed in the Contexts pane when List mode (View > List Mode) is active. The context for invoice Line is shown below.



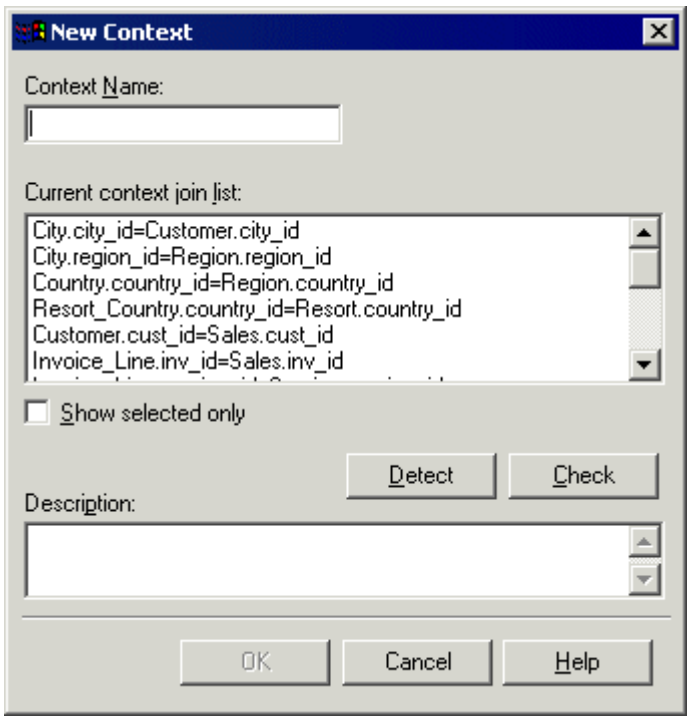
5. The context for Invoice_Line is shown below.



6.4.2.2 Creating a context manually

To create a context manually:

1. Select Insert > Context.
   Or
   Click the Insert Context button.
   The New Context box appears.



2. Type a name for the context in the Context Name text box.
3. Select all the joins defining the context in the Current Context Joins list.
   You have the following options when creating the context:
4. Click the Detect button to show the joins making up a suggested context with context name.
5. Select the Show Selected Only check box to see only selected joins.
6. Click the Check button.
   The universe design tool checks the selected joins for any loops.
7. Type a description of the data the context returns. This is the help text that a Web Intelligence user sees when they run a query that takes the context path. This text should be useful to the end user.
8. Click OK.
   The context is created.

6.4.3 Editing a context

You can use a context editor to modify the following properties of a context:

- Name
- Joins included in the context
- Description

You can also check the context for any unresolved loops.

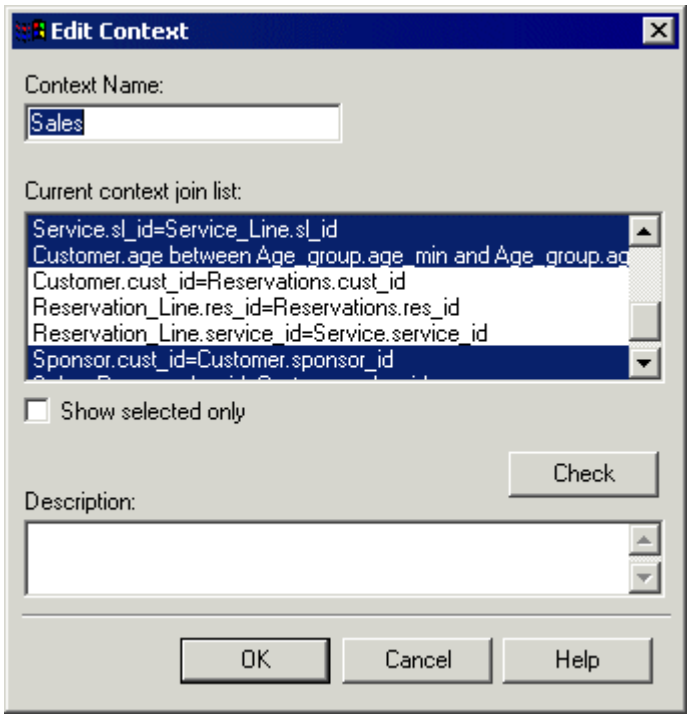
6.4.3.1 Editing context properties

To edit context properties:

1. Select View > List Mode.
   The List pane appears above the Structure pane. It contains list boxes for all the tables, joins, and contexts in the Structure pane.



2. Double click a context name in the Contexts list pane.
   The Edit Context box appears.



3. Type a new name in the Context Name box if you want to change the context name.
4. Click a highlighted join to remove it from the context.
   Or
   Click a join that is not highlighted to add it to the context.
5. Type a description for the context.
6. Click OK.
   The modifications appear in the context.

6.4.4 Deleting a context

You can delete a context at any time from the Context list in the List pane. If you are adding or deleting a table or join within a context, you should delete the context before making the modification to the table or join.

Once the modification is complete, you can either manually recreate the context if it is being used to solve a chasm trap, or use Detect Contexts to automatically detect a new context if it is being used to resolve a loop. Refer to the sectionDetecting and creating a context [page 211] for information on detecting contexts.

6.4.4.1 Deleting a context from the Context list

To delete a context from the context list:

1. Ensure that List mode is active (Select View > List Mode).
2. Right click a context name in the Contexts list box and select Clear from the contextual menu.
   Or
   Click a context name in the Context list box and select Edit > Clear.
   The context is removed from the list.

6.4.5 Updating contexts

Contexts are not updated automatically when the universe structure is changed. If you add or remove any tables to the structure, or if you add or remove any joins, you must update all the contexts.

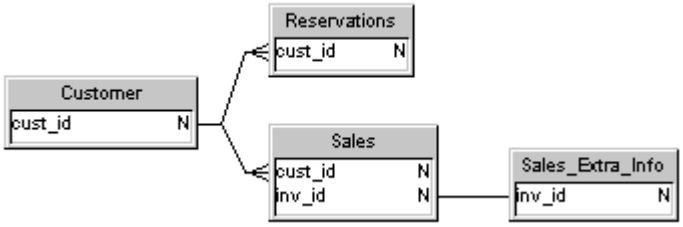
If you have made only a simple change to the structure, you can update the joins that are included in each context manually using either the Edit Context box or the List pane. However, if you have made significant changes to the universe structure, you should delete the current contexts and re-create them.

6.4.6 Join Paths that Prevent Context Detection

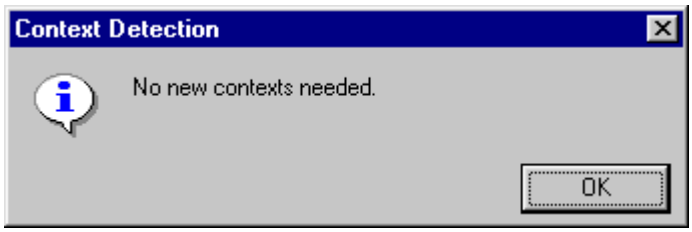
A one-to one-cardinality positioned at the end of a join path can prevent Context Detection in the universe design tool from detecting a context. You resolve this problem by changing the cardinality of the table at the end of the join path to one-to-many.

Example: One-to-one cardinality preventing context detection

The schema below shows a table Sales_Extra_Info that contains particular information about each sale. It is joined by a one-to-one join to the Sales table.



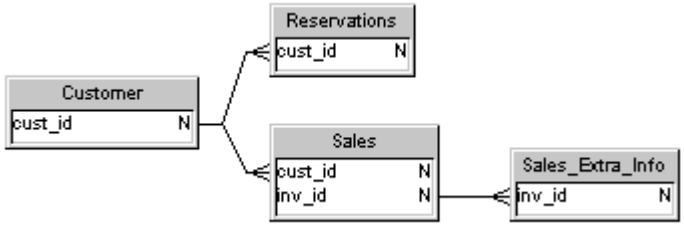
When you visually examine the join paths, there are clearly two contexts in this schema; a reservations context, and a sales context. However, when you automatically detect contexts on this type of join path (Tools > Automated Detection >Detect Contexts), you receive the following message:



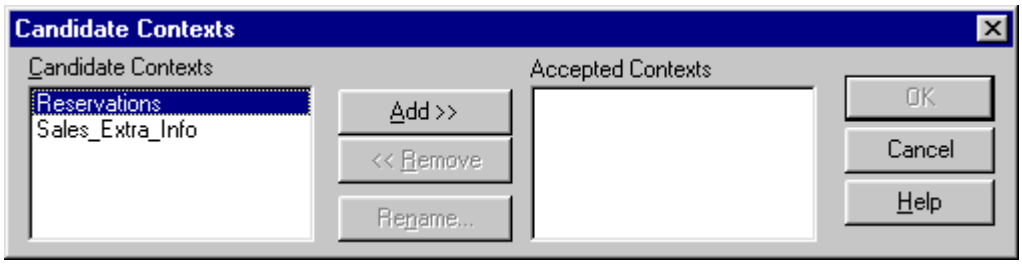
The universe design tool has not considered the one-to-one join at the end of the join path in the context detection, so does not consider that there are two contexts.

6.4.6.1 Changing cardinality to allow the context detection

You solve this problem by setting the cardinality of the join linking Sale_Extra_Info to Sales to one-to-many. It can also be many-to-one, the important factor is not to have the one-to-one join at the end of the join path. The schema below now has a one-to-many join at the end of the join path.



When you run Detect Contexts, the two contexts are detected as shown below:



6.4.7 How do Contexts Affect Queries?

Depending on how you allow Web Intelligence users to use the objects defined on schema structures, contexts can lead to three types of queries being run:

- Ambiguous queries
- Inferred queries
- Incompatible queries

You can run these types of queries in Web Intelligence to test the SQL generated by the contexts. If any of these query types produces an error, or returns incorrect data, you need to analyze the concerned join paths.

6.4.7.1 Ambiguous queries

An end user is prompted to choose between one query path or another. This occurs when a query includes objects that when used together do not give enough information to determine one context or the other.

When a query is ambiguous, Web Intelligence displays a dialog box that prompts the user to select one of two contexts. When the user selects a context, the corresponding tables and joins are inserted into the SQL query.

Example: Running an ambiguous query

A Web Intelligence user runs the following query:

Give me the services used by each age group of visitors for each resort:

Service Age group Resort

When the query is run, a dialog box appears asking the user to choose a context, in this case either the Reservations or Sales context:

The user must choose if they want information for services reserved by age group, or services paid by age group. If they select the Reservations context, the following SQL is generated:

```
SELECT Service.service, Age_group.age_range, Resort.resort FROM Service,
Age_group, Resort, Customer, Reservations, Reservation_Line, Service_Line WHERE
( Resort.resort_id=Service_Line.resort_id  ) AND
( Service.sl_id=Service_Line.sl_id  ) AND  ( Customer.age between
Age_group.age_min and Age_group.age_max  ) AND
( Customer.cust_id=Reservations.cust_id  ) AND
( Reservation_Line.res_id=Reservations.res_id  ) AND
( Reservation_Line.service_id=Service.service_id  )
```

The joins referenced by the other context (Sales) do not appear in the SQL.

6.4.7.2 Inferred queries

A Web Intelligence query is run without prompting an end user to choose a context. The query contains enough information for the correct context to be inferred. For example, a user runs the following query:

Give me the number of future guests by age group for each available service:



When the query is run, the data is returned without prompting the user to select a context. The `Future Guests` object is a sum on the Reservation_Line table, which is part of the Reservations context. Web Intelligence infers that the Reservation context is the one to use for the query.

6.4.7.3 Incompatible queries

Objects from two different contexts are combined in a query. The two Select statements are synchronized to display returned data in separate tables.

Example: Running an incompatible query

A Web Intelligence user runs the following query:

Give me the total number of guests company wide by age group and the months that reservations were made.



When the query is run, no prompt appears as Web Intelligence infers the use of both the Sales and Reservations contexts. The Select statements for both contexts are synchronized as follows:




```
SELECT
 Age_group.age_range,
 sum(Invoice_Line.nb_guests)
FROM
 Age_group,
 Invoice_Line,
 Service_Line,
 Sales,
 Customer,
 Service
WHERE
 ( Customer.cust_id=Sales.cust_id )
 AND ( Invoice_Line.inv_id=Sales.inv_id )
 AND ( Invoice_Line.service_id=Service.service_id )
 AND ( Service.sl_id=Service_Line.sl_id )
 AND ( Customer.age between Age_group.age_min and Age_group.age_max )
 AND ( Service_Line.service_line = 'Accommodation' )
GROUP BY
 Age_group.age_range
```

The query is split into two parts:

- Age Group and Number of Guests
- Reservation Month

When retrieving the results of the two queries, Web Intelligence combines the results (using `Age Group`). It then displays the results in two tables in the same report as follows.

18-30

Number of guests
451.00

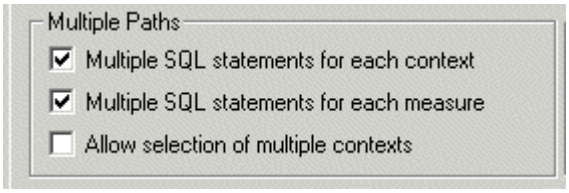
Reservation Month
Apr
Aug
Feb
Jan
Jun
May
Nov
Oct
Sep

To allow incompatible queries to be run in Web Intelligence, you must select the Multiple SQL statements in the universe design tool for each context option. This is described in the following section.

6.4.7.4 Selecting Multiple SQL statements for each context

To select Multiple SQL statements for each context:

1. Select File > Parameters.
   The Universe Parameters dialog box appears.
2. Click the SQL tab.
   The SQL page appears.
3. Select the Multiple SQL statements for each context check box.



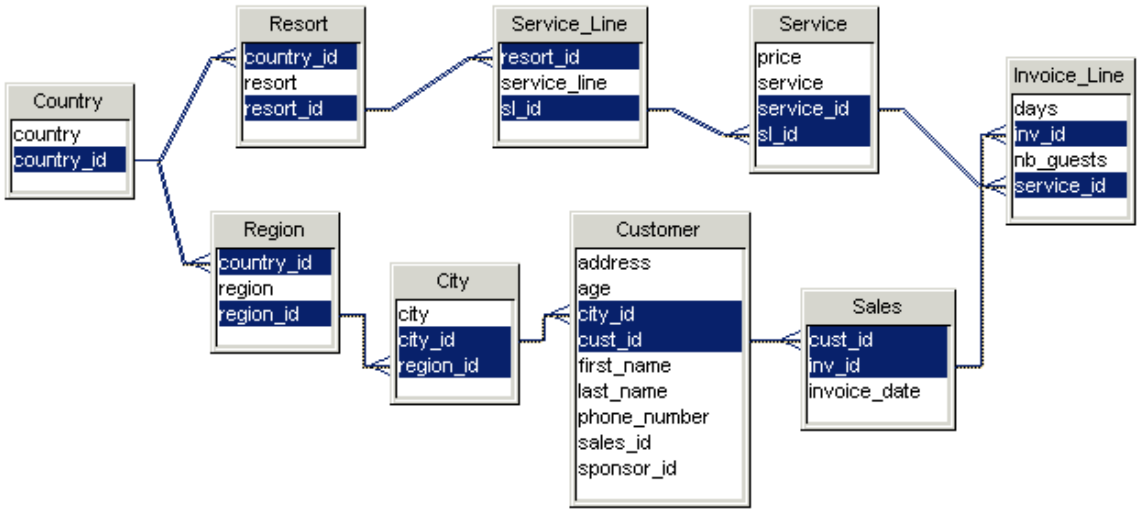
4. Click OK.

6.5 Resolving loops

In a relational database schema, a common type of join path that returns too few rows is called a loop.

6.5.1 What is a Loop?

A loop is a set of joins that defines a closed path through a set of tables in a schema. Loops occur when joins form multiple paths between lookup tables. An example of a loop is shown below.



The schema contains two linked sets of information:

For each...	the following information is linked
Resort	Available service lines, services for each service line, invoice information for each service, and the country where the resort is situated.
Customer	The city, region, and country where the customer lives, the sales for the customer, and the invoice information for each sale.

These two sets of information are linked in a common join path forming a loop. The lookup table Country can be the country where a resort is situated, or the country in which a customer lives.

6.5.1.1 Why loop in a universe schema and not in the database

In a database, multiple paths between tables may be valid and implemented to meet specific user requirements. When each path is included individually in a query it returns a distinct set of results.

However, the schema that you design in the universe design tool often needs to allow queries that include more than one path, which a relational database may not be designed to handle, so the information returned can be incorrect.

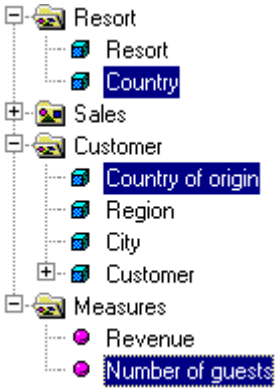
The rows that are returned are an intersection of the results for each path, so fewer rows are returned than expected. It is also often difficult to determine the problem when you examine the results.

6.5.2 How does a loop affect queries?

If you created a universe based on the above structure, any query run against the tables in the loop would return only results where the country values for resorts and the country values for customer origin are equivalent. This double restriction on the shared lookup Country table returns fewer rows than expected.

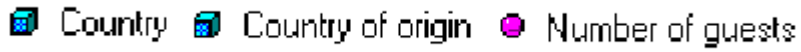
Example: Loop returns incorrect results

You create the following objects using the schema that contains the above loop:



You run the following query in Web Intelligence:

For each resort country, give me the number of guests from each country that stay at each resort.



You would expect the following type of result:

France

Country of origin	Number of guests
Germany	141.00
Japan	154.00
US	151.00

US

Country of origin	Number of guests
Germany	329.00
Japan	345.00
US	431.00

For the resorts in France and the US, you have the number of German, Japanese, and US visitors staying in resorts in those countries.

However, when you run the query using the universe containing the loop, you receive the following results:

Country	Country of origin	Number of guests
US	US	431.00

This suggests that only visitors from the US stayed in resorts in the US. No other visitors came from any other country.

6.5.2.1 What is the loop doing to the query?

The joins in the Structure are used to create the Where clause in the inferred SQL of a query. The purpose of the joins is to restrict the data that is returned by the query. In a loop, the joins apply more restrictions than you anticipate, and the data returned is incorrect.

The Where clause created by the loop is shown below:

```
 WHERE  ( Country.country_id=Resort.country_id  ) AND
( Resort.resort_id=Service_Line.resort_id  ) AND
( Service_Line.sl_id=Service.sl_id  ) AND
( Service.service_id=Invoice_Line.service_id  ) AND
( Sales.inv_id=Invoice_Line.inv_id  ) AND  ( Customer.cust_id=Sales.cust_id  )
AND  ( City.city_id=Customer.city_id  ) AND
( Region.region_id=City.region_id  ) AND
( Country.country_id=Region.country_id  ) AND  ( Service_Line.service_line =
'Accommodation'  )
```

The following two joins are both applying a restriction to the Country table:

- Country.country_id=Resort.country_id
- Country.country_id=Region.country_id

Country is serving two purposes:

- Lookup for the resort country.
- Lookup for the customer country of origin.

This creates a restriction so that data is returned only when the resort country is the same as the customer country. The resulting report shows only the number of visitors from the US who visited resorts in the US.

Depending on the nature of the loop, you can resolve the loop in the universe design tool using either an alias to break the join path, or a context to separate the two join paths so that a query can only take one path or the other.

6.5.2.2 How does an alias break a loop?

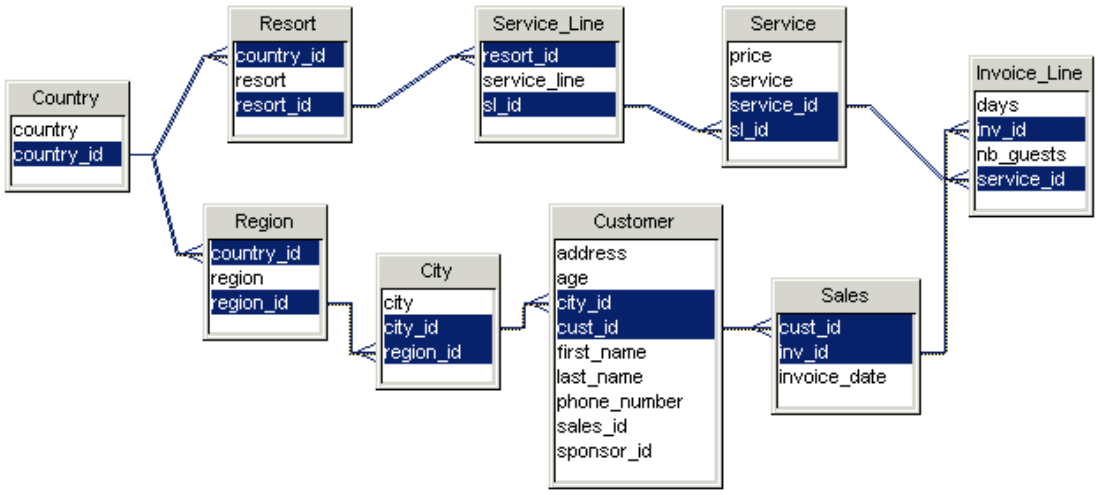
An alias breaks a loop by using the same table twice in the same query for a different purpose. The alias is identical to the base table with a different name. The data in the alias is exactly the same as the original table, but the different name "tricks" SQL into accepting that you are using two different tables.

> **i** Note
>
> You can resolve the loop satisfactorily by creating only one alias table in the example we have been using. The Region join uses the original Country table, while the Showroom join uses the alias table. However, you could create a separate alias table for each join in the original table. In some relational database systems, this is necessary.

Example: Breaking a loop with an alias

The schema below is the same schema that contained the loop in the previous section. It shows a join path in which the Country lookup table receives only the "one" ends of two joins, so it can be used for the following two purposes in the join path:

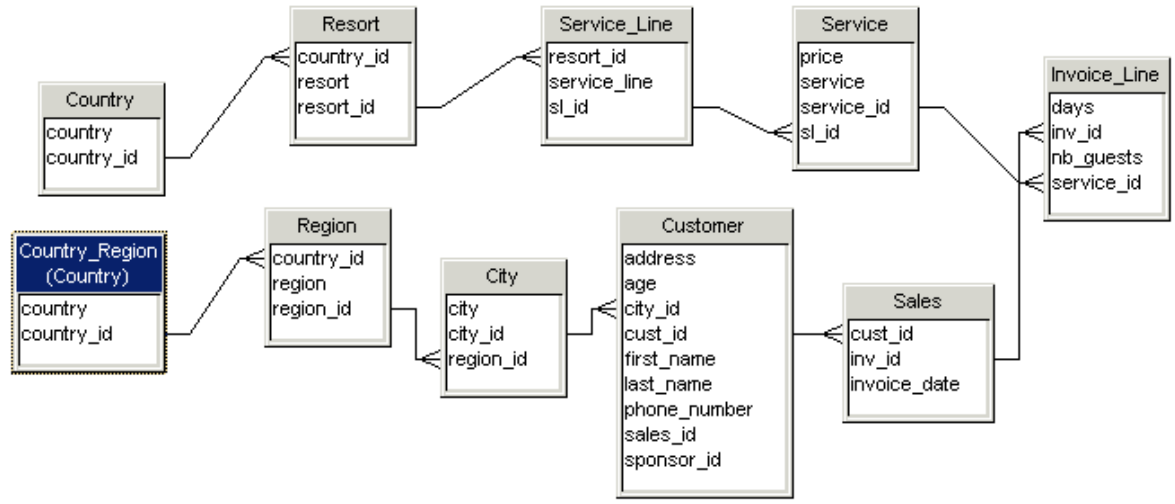
- Countries for resorts
- Countries for customers



You create an alias for Country and rename it Country_Region. The two "one" ended joins are now separated as follows:

- Country keeps a join to the Resort table.
- Country_Region is joined to the Region table.

The schema now appears as shown below:



When you run the same query that produced too few rows in the previous example:

For each resort country, give me the number of guests from each country that stay at each resort.

Country | Country of origin | Number of guests

The Where clause for this query is now:

```
WHERE ( City.city_id=Customer.city_id  ) AND
( City.region_id=Region.region_id  ) AND
( Country.country_id=Region.country_id  ) AND
( Resort_Country.country_id=Resort.country_id  ) AND
( Customer.cust_id=Sales.cust_id  ) AND  ( Invoice_Line.inv_id=Sales.inv_id  )
AND  ( Invoice_Line.service_id=Service.service_id  ) AND
( Resort.resort_id=Service_Line.resort_id  ) AND
( Service.sl_id=Service_Line.sl_id  ) AND  ( Service_Line.service_line =
'Accommodation'  )
```

There is now one join applying a restriction on the Country table and another join applying a restriction on the Resort_Country table. The loop has been broken.

When the query is run, the following table is returned:

Country	Country of origin	Number of guests
France	Germany	141.00
France	Japan	154.00
France	US	151.00
US	Germany	329.00
US	Japan	345.00
US	US	431.00

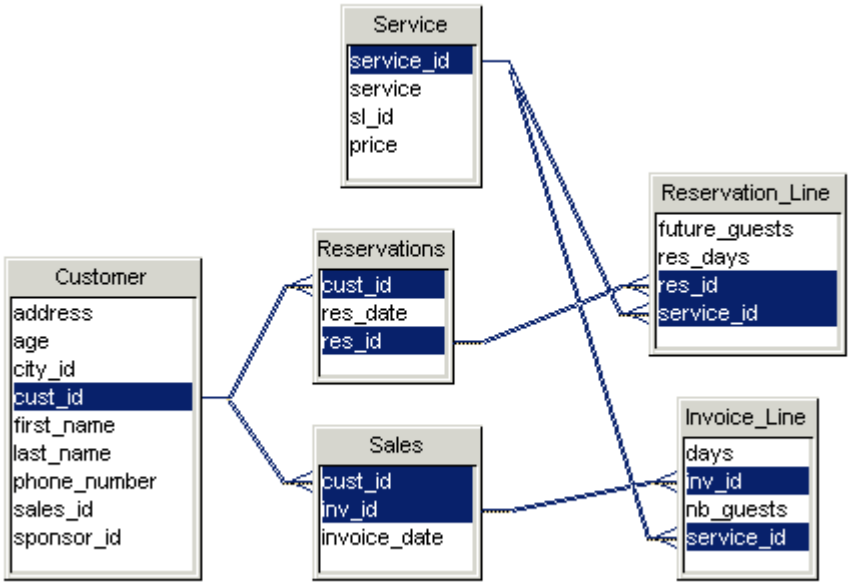
6.5.2.3 How does a context resolve a loop?

A context resolves a loop by defining a set of joins that specify one specific path through tables in a loop. It ensures that joins are not included from different paths within the same SQL query.

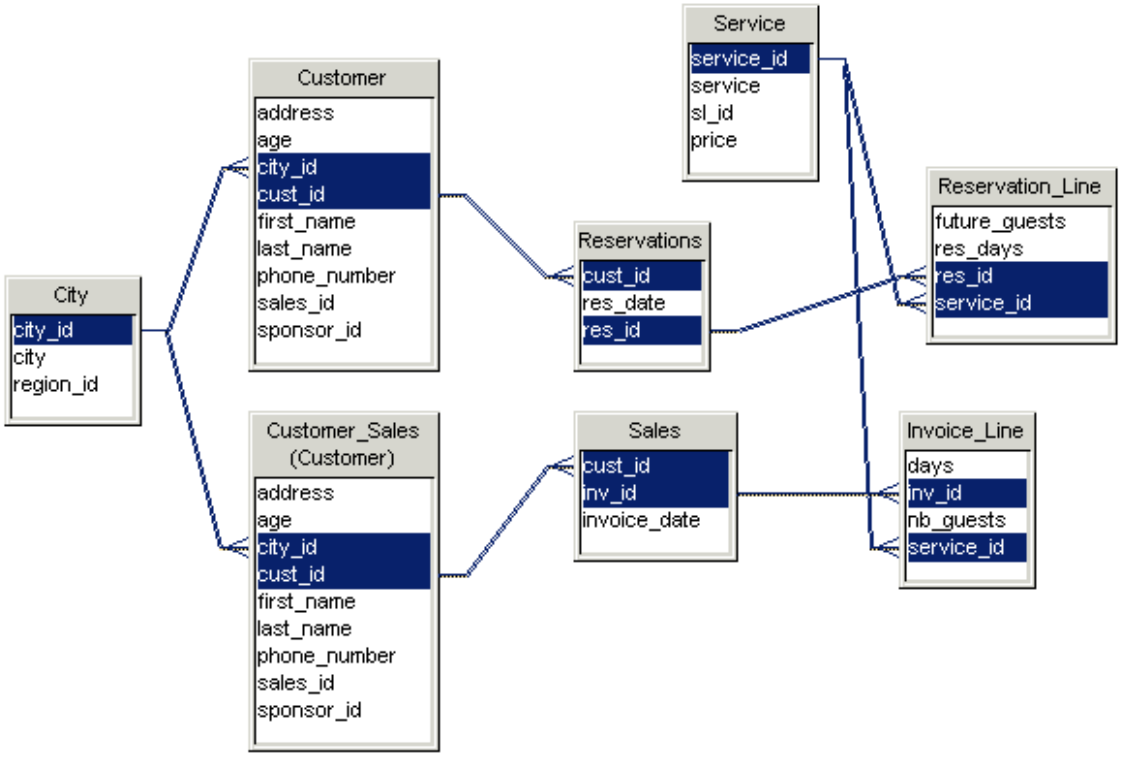
You often use contexts in schema that contain multiple fact tables ("multiple stars") that share lookup tables.

Example: Resolving a loop with a context

The schema below contains statistical information about sales and reservations. The statistics relating to each type of transaction are stored in the fact tables Sales and Reservations. The schema contains a loop as a join path can follow the sales path or the reservations path to get service information.



If you created an alias for the Customer so that you had a Customer to Reservation join and a Customer_Sales to Sales join, you break the loop, but if you want to add a City table to the schema, you end up with a loop again as shown below:



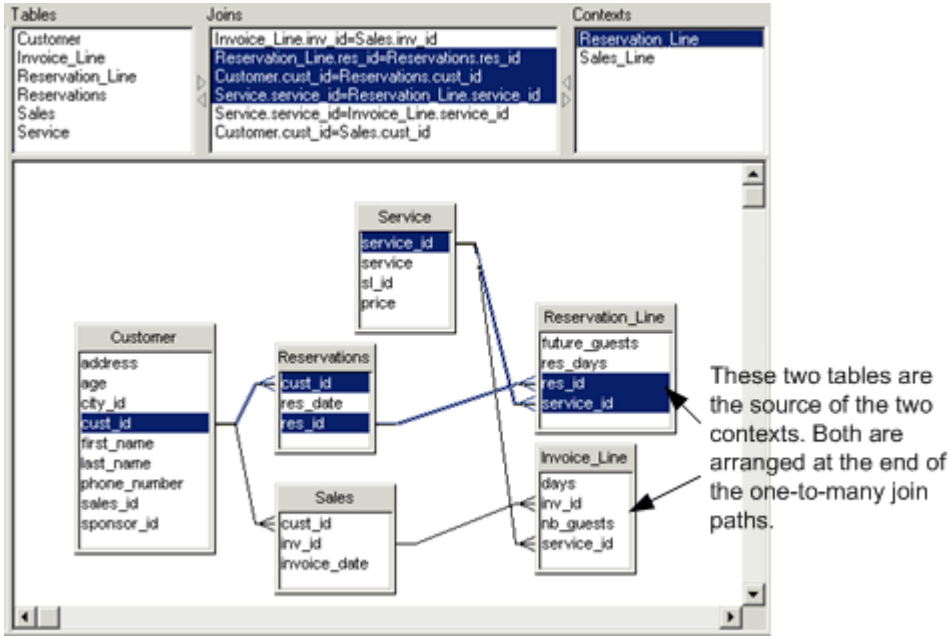
You must continue creating aliases for each new table you add to the schema. This is difficult to maintain, and also ends up proliferating the number of similar objects using each table in the universe.

The only way to resolve this loop is to leave the loop in place, and create a context that specifies one or the other path around the schema. This ensures that queries answer questions for one transaction or the other, such as: Is the customer information needed from the perspective of sales or reservations?

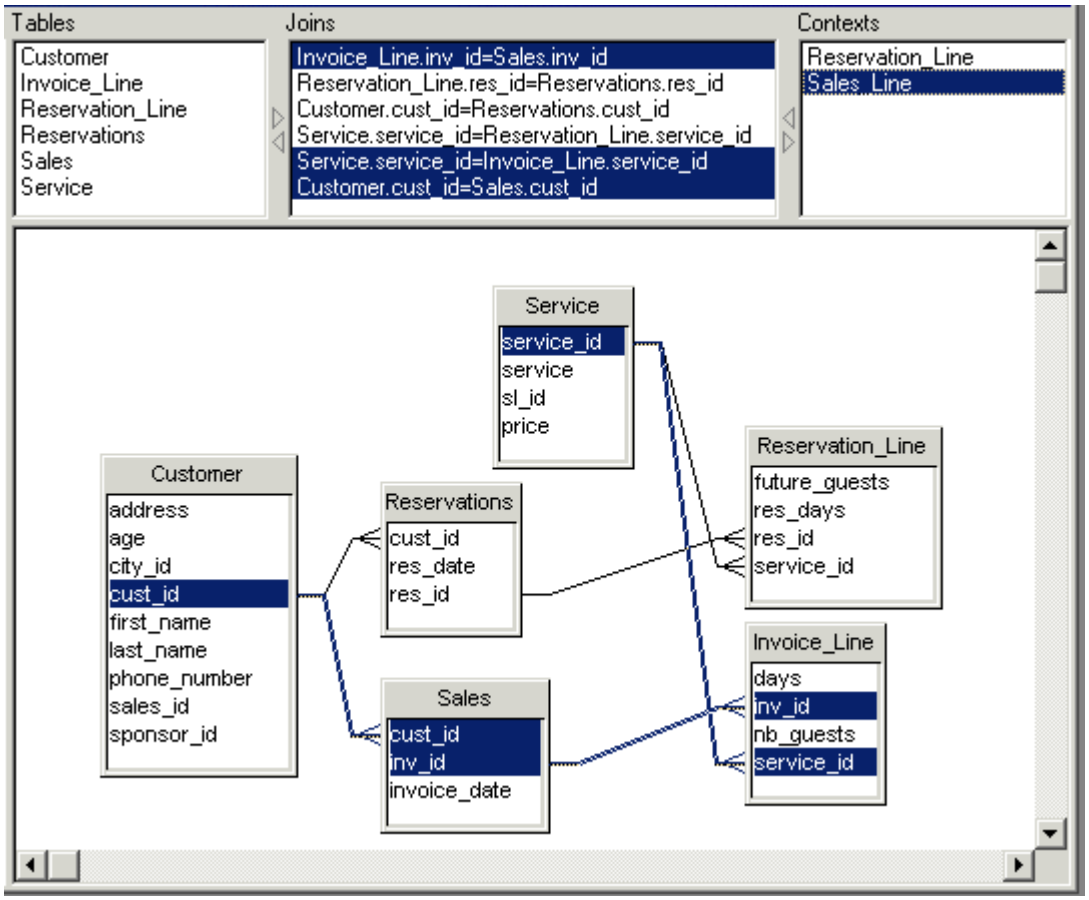
In the example, you can follow two different paths from the Customer table to the Service table:

For this path...	The tool detects these contexts...
Reservations and Reservation_Line	Reservation_Line
Sales and Invoice_Line	Sales_Line

The Reservation_Line context appears below:



The Sales_Line context appears below:



You then create different sets of objects from the tables in the different contexts. Users can then run either Reservation queries or Sales queries, depending on the objects they select.

6.5.3 Visually identifying loops

You can use the following guidelines to help you analyze your schema to determine whether an alias or context is appropriate for resolving loops. These can be useful to understand your schema, but you should use Detect Aliases and Detect Contexts to formally identify and resolve loops. See the section Detecting and creating an alias [page 210] and Detecting and creating a context [page 211] for more information.

If loop contains...	then loop can be resolved by...
Only one lookup table	Alias
A look up table that receives only "one" ends of joins	Alias
Two or more fact tables	Context

6.5.4 Automatically Identifying and Resolving Loops

You can use the universe design tool to automatically detect loops and propose candidate aliases and contexts that you can insert in your schema to resolve the loops.

6.5.4.1 Cardinalities must be set before detecting loops

Before using the automatic loop detection and resolution features, all cardinalities must be set for all joins in the schema.

It is good design practise to either define cardinalities manually, or manually validate each cardinality that the universe design tool proposes when using the automatic routine.

You can set cardinalities in two ways:

- Manually. Refer to the section Using cardinalities [page 167] for more information.
- Use Detect Cardinalities. Refer to the section Using cardinalities [page 167] for more information.

6.5.5 Tool features to detect and resolve loops

You can use the following features in the universe design tool to identify and resolve loops:

Identify and resolve loop using...	Description
Detect Aliases	Detects tables that can be aliased to solve a loop in the structure and proposes a candidate alias for each table. You can insert and rename the alias directly from the box. You should run Detect Aliases before Detect Contexts to ensure that aliases that you create are included in any contexts that you implement. It does not detect the need for an alias to resolve a fan trap.
Detect Contexts	Detects contexts that can be used to solve a loop in the structure and proposes candidate contexts. You can implement and rename each context directly from the box. Run Detect Contexts after Detect Aliases to ensure that any contexts that you implement include any new aliases. It does not always detect the need for a context to resolve a chasm trap. If not, you need to identify the context manually.
Detect Loops	Detects and highlights loops in the structure It proposes to insert an alias or context to resolve each loop. You can implement the proposed alias or context directly from the Detect Loops box. Use Detect Loops to run a quick check on the schema, or to visualize the loop. Do not use it to identify and then resolve loops as you cannot edit or see the candidate alias before insertion.

6.5.5.1 General method for identifying and resolving loops

A general procedure for detecting and resolving loops is given below. The sections that describe the step in detail are also given.

1. Verify that all cardinalities are set.
   See the section Using cardinalities [page 167].
2. Run Detect Aliases to identify if your schema needs an alias to solve any loops.
   See the section Detecting and creating an alias [page 210] for more information.
3. Insert the candidate aliases proposed by Detect Aliases.
4. Run Detect Contexts to identify if your schema needs a context to solve a loop that could not be solved with an alias only.
   See the section Detecting and creating a context [page 211] for more information.
5. Implement the candidate contexts proposed by Detect Contexts.
6. Test the resolved loop by creating objects and running queries.
   See the chapter Creating universes [page 246] for information on creating objects and testing the universe structures.

> **Note**
>
> If you are resolving loops for a schema that already has objects defined on the tables, then you must redefine any objects that now use an alias and not the base table.

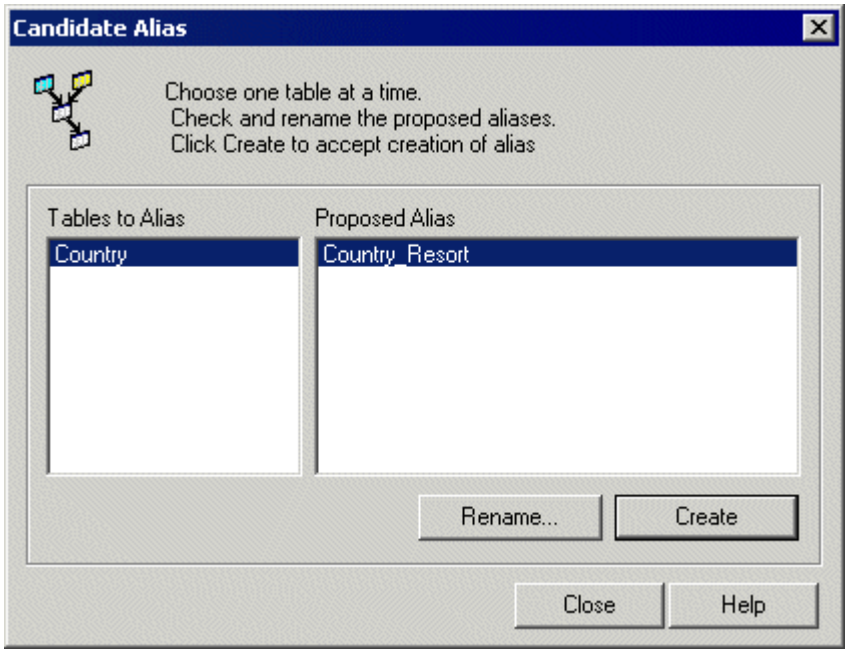
6.5.5.2 Detecting and creating an alias

You can use Detect Aliases, to automatically detect and indicate the tables causing loops in the active universe. Detect Aliases proposes candidate tables that you can edit, and insert in the schema.

> **Note**
>
> Before using Detect Aliases, verify that all the tables in schema are linked by joins, and that all cardinalities are set.

To detect and create an alias:

1. Select *Tools* > *Automated Detection* > *Deetct Aliases*.
   Or
   Click the *Detect Aliases* button.
   The *Candidate Alias* dialog box appears. The left pane lists the table or tables that need an alias. The right pane lists proposed aliases that can be inserted to break the loop.

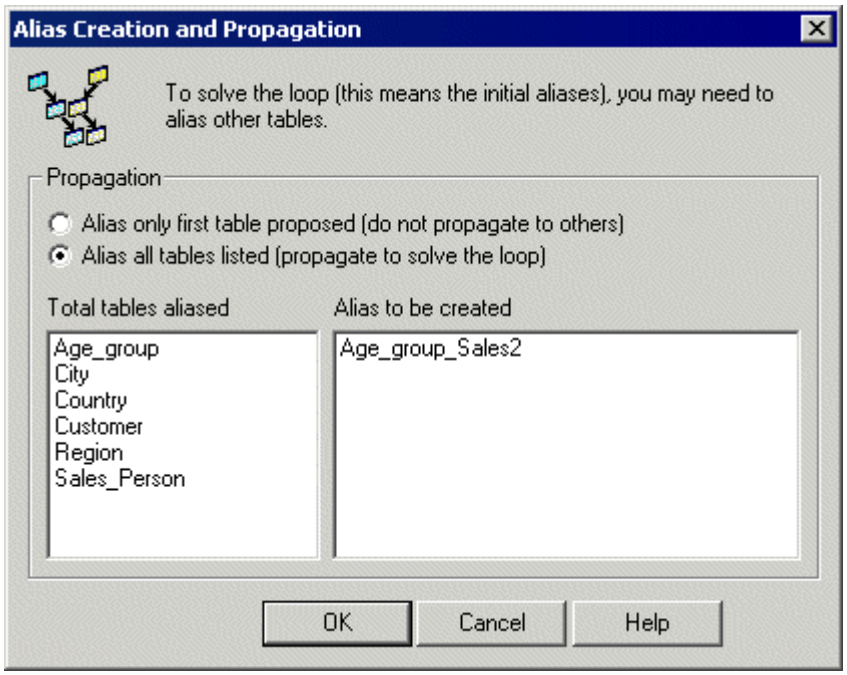


2. Select a table in the left pane.
   A suggested name for the candidate alias is listed in the right pane.
3. If you want to rename the proposed alias, click *Rename* and enter a new name in the *Rename* box.
4. Click *Create*.
   A message box prompts you to confirm the creation of the alias.

5. Click *OK*.
   The alias appear in the Structure pane.
6. Repeat steps 2 to 5 for any remaining tables.
7. Click *Close*.

6.5.5.3 Detecting and creating multiple aliases

Sometimes when you create an alias, you need to create additional aliases to accommodate new join paths. When using Detect Alias, if the universe design tool detects the need for further aliases, the following dialog box appears when you click the Create button.



In such a situation, two options are available to you:

- You can accept that only the first table proposed will be aliased.
- You can alias all the tables listed.

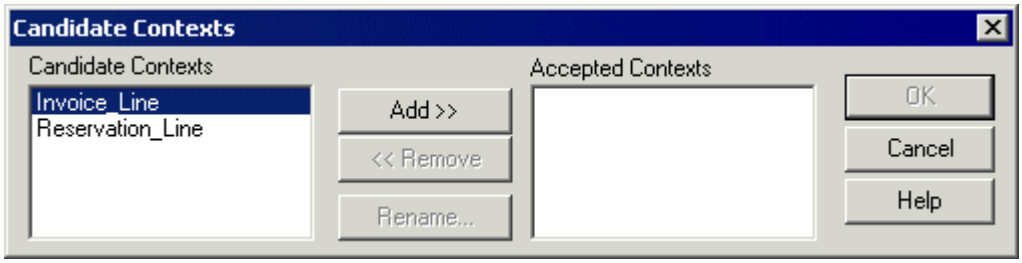
6.5.5.4 Detecting and creating a context

You can use Detect Contexts to automatically detect the need for a context. Detect Contexts also proposes a candidate context. You can edit the candidate context before it is implemented.

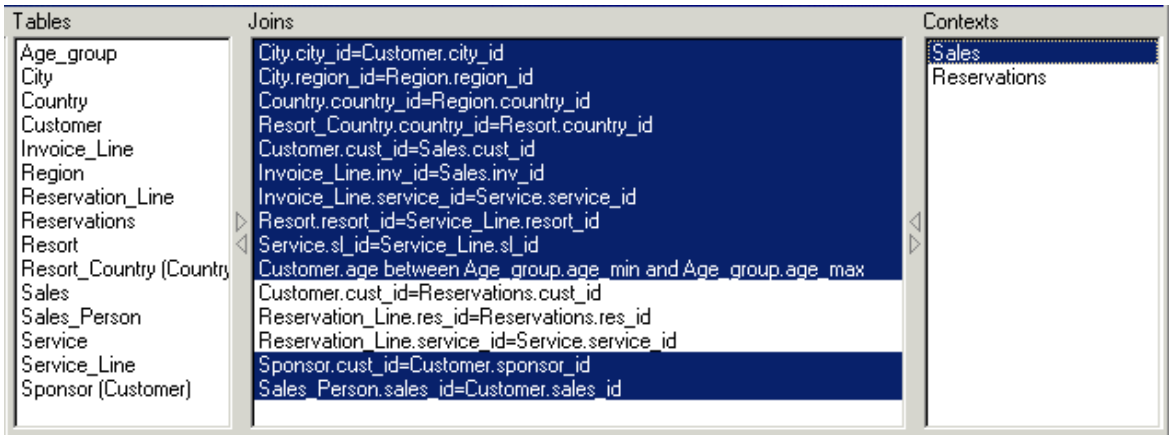
To detect and create a context:

1. Select Tools > Automated Detection > Detect Contexts.
   Or

Click the Detect Contexts button.
The Candidate Contexts dialog box appears. The proposed contexts appear in the left pane.



2. Click a context name.
   The tables included in the candidate context are highlighted in the schema.
3. Click the Add button.
   The context name appears in the Accepted Contexts pane. You can remove any context from the right pane by selecting it, and then clicking the Remove button.
4. Repeat steps 3 and 4, if applicable, to add the other contexts.
5. If you want to rename a context, select it from the right pane, and then click the Rename button.
   The Rename Context dialog box appears. Type a new name.
6. Click the OK button.
   The contexts are listed in the Contexts box in the Universe window.



> **i** Note
>
> If your universe contains a loop that could be ambiguous for a user, you should always give a name to the context resolving the loop that is easy for users to understand. It should be clear to the Web Intelligence user what information path is represented by a context.

6.5.5.5 Automatically detecting loops

You can detect loops in your universe using Detect Loops. This is a feature that automatically checks for loops in the schema, and proposes either an alias or context to solve the loop.

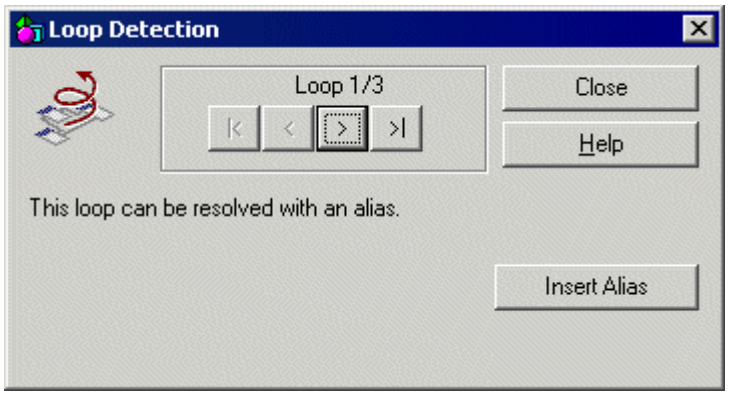
Detect Loops is useful to run quick checks for loops in the schema. It also proposes aliases and contexts to resolve detected loops; however, you have less control over the order that the alias and contexts are created than if you used Detect Aliases and Detect Contexts to resolve a loop.

The recommended process for resolving loops is described in the section General method for identifying and resolving loops [page 209].

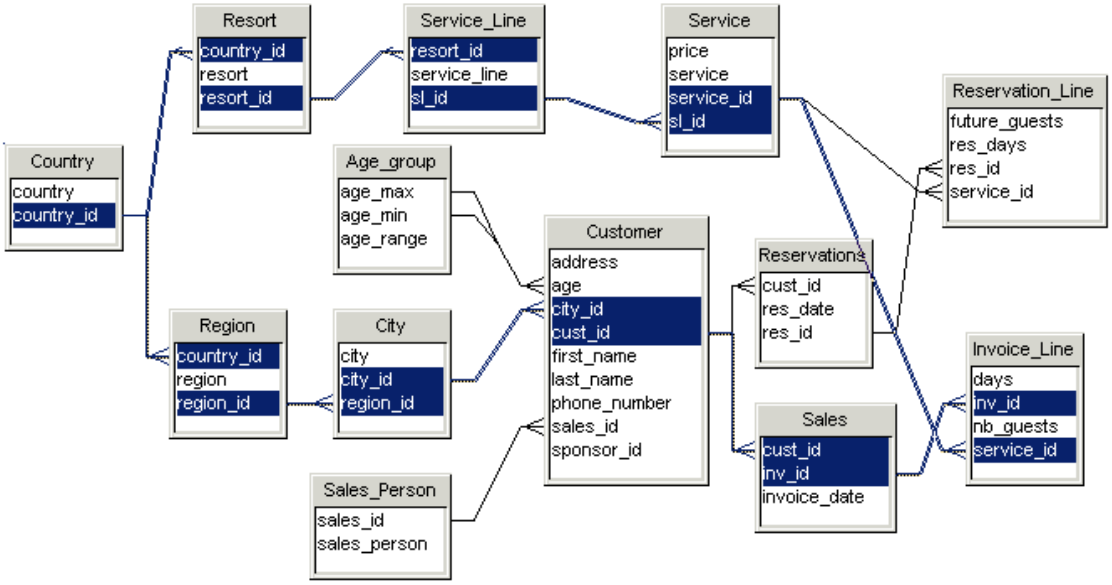
> **Note**
>
> You can also use Check Integrity to automatically check for errors in universe structures, including joins, cardinalities, and loops. Check Integrity proposes solutions to any errors it discovers. See the section Checking universe integrity manually [page 242] for more information.

To detect loops in a schema:

1. Verify that you have set cardinalities for all joins in the schema.
2. Select Tools > Automated Detection > Detect Loops.
   Or
   Click the Detect Loops button.
   The Loop Detection box appears. It indicates how many loops have been detected and proposes a possible solution.



   The detected join path that forms a loop is simultaneously highlighted in the Structure pane as follows:



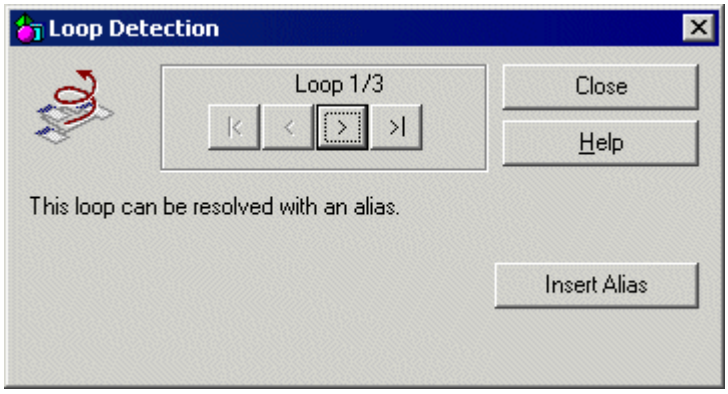
3. Click the forward button to display the next loop and proposed solution. For each loop that the universe design tool detects, the join path is highlighted in the structure pane.
4. Click Close.

6.5.5.6 Creating aliases and contexts automatically

The universe design tool proposes a candidate alias or a context to resolve a loop when you run Detect Loop. You can choose to insert the candidate alias or implement the candidate context directly from the Detect Loops box.

To create an alias using Detect Loop:

1. Select Tools > Automated Detection > Detect Loops.
   The Detect Loops box appears. It indicates one or more loops detected in the schema, and proposes a candidate alias or context for each loop.
2. Click the forward arrow button until the following message appears for a detected loop:
   This loop can be resolved with an alias.

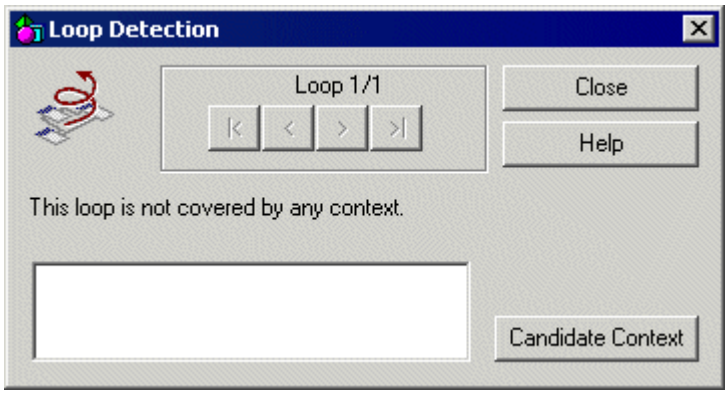


3. Click the Insert Alias button.
   An alias is automatically inserted in the Structure pane. It is joined to the table that table that is causing the loop in the schema.

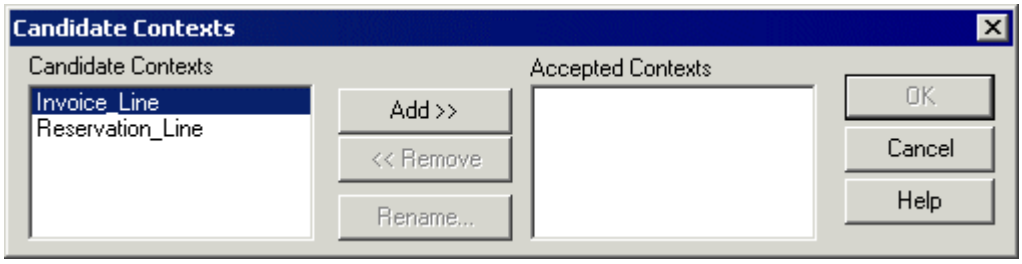
6.5.5.7 Creating a context using detect loop

To create a context using detect loop:

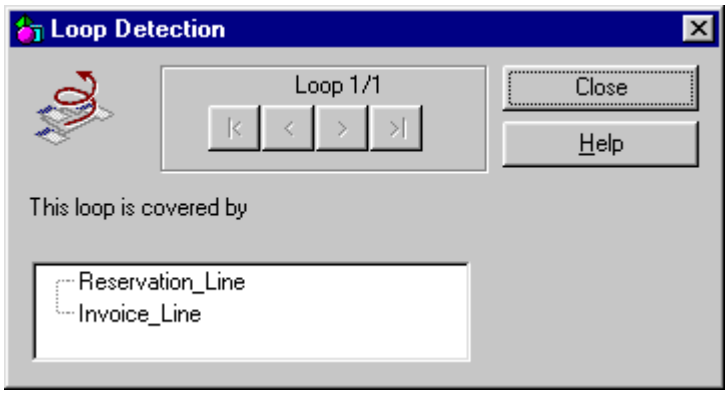
1. Select Tools > Automated Detection > Detect Loops.
   The Detect Loops box appears. It indicates one or more loops detected in the schema, and proposes a candidate alias or context for each loop.
2. Click the forward arrow button until the following message appears for a detected loop:
   This loop is not covered by any context.



3. Click the Candidate context button.
   The Candidate Contexts dialog box appears.



4. Click a context name.
   The tables included in the candidate context are highlighted in the schema.
5. Click the Add button.
   The context name appears in the Accepted Contexts pane. You can remove any context from the right pane by selecting it, and then clicking the Remove button.
6. Repeat steps 3 and 4, if applicable, to add the other contexts.
7. Click OK.
   A context confirmation box appears.



8. Click Close.
   The contexts are listed in the Contexts box in the Universe window.

6.5.6 Examples of resolving loops

The following are worked examples showing you how to do the following:

- Creating an alias to break a loop caused by shared lookup tables
- Creating an alias to break a loop caused by shared lookup tables
- Determining when an alias is not appropriate to break a loop
- Creating a context to resolve a loop
- Using an alias and context together to resolve a loop

These schemas are not based on the Beach universe. They use a schema based on a Shipping company and show another perspective of certain loop resolution examples already shown in this chapter with the Beach universe.

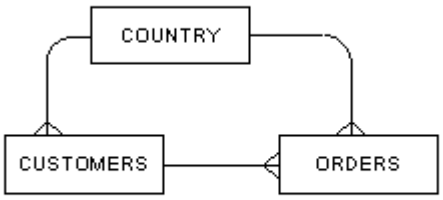
6.5.6.1 Creating an alias to break a loop caused by shared lookup tables

A sales database holds information about products sold to customers on a worldwide basis. These customers can:

- Reside anywhere in the world
- Order products from the company
- Request that these products be shipped to a destination in any country

For example, a customer residing in the UK can order a vehicle and then ask for it to be shipped to Brazil.

The schema for this type of database is as follows:



You can interpret this schema as follows:

- Each customer comes from one country.
- Each customer can place one or more orders for a product.
- The company ships each product ordered to a destination country, which may not necessarily be the same as the customer's country of residence.

The tables and their columns are shown below:

country_id	country
1	USA
2	UK
3	France
4	Germany
5	Spain

cust_id	last_name	loc_country
100	COLTRANE	1
101	MULLIGAN	1
102	WALDRON	3
103	HANCOCK	4
104	DAVIS	2
105	BARBIERI	5
106	STREATS	5

order_id	cust_id	order_date	ship_country
12345	100	1/1/95	2
12346	101	1/6/95	1
12347	101	2/6/95	3
12348	102	8/4/95	5
12349	103	10/3/95	4
12350	104	15/8/95	2
12351	105	6/2/95	5
12352	106	7/3/95	4

You run a query to obtain the following information:

- Names of customers
- Customer's country of residence
- Dates of each order
- Destination country of the shipment

The SQL to extract this data is as follows:

```
SELECT    CUSTOMERS.LAST_NAME,   COUNTRY.COUNTRY,   ORDERS.ORDER_ID,
ORDERS.ORDER_DATE,   COUNTRY.COUNTRY FROM   CUSTOMERS,   ORDERS,   COUNTRY WHERE
  (CUSTOMERS.CUST_ID=ORDERS.CUST_ID) AND
  (ORDERS.SHIP_COUNTRY=COUNTRY.COUNTRY_ID) AND
  (CUSTOMER.LOC_COUNTRY=COUNTRY.COUNTRY_ID)
```

When executed, this SQL returns incomplete results; only those customers who requested a shipment to their country of residence are returned. The customers who chose another country for shipment are not returned.

The returned rows are an intersection of both the customer's country of residence and the destination country of the shipment. Instead of generating the full results shown below

last _name	country	order_id	order_date	country
COLTRANE	USA	12345	1/1/95	UK
MULLIGAN	USA	12346	1/6/95	USA
MULLIGAN	USA	12347	2/6/95	France
WALDRON	France	12348	8/4/95	Spain
HANCOCK	Germany	12349	10/3/95	Germany
DAVIS	UK	12350	15/8/95	UK
BARBIERI	Spain	12351	6/2/95	Spain
STREATS	Spain	12352	7/3/95	Germany

The SQL returns only these results:

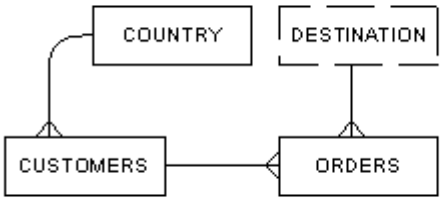
last _name	country	order_id	order_date	country
MULLIGAN	USA	12346	1/6/95	USA
HANCOCK	Germany	12349	10/3/95	Germany
DAVIS	UK	12350	15/8/95	UK
BARBIERI	Spain	12351	6/2/95	Spain

You can break the loop by inserting an alias. The first step in creating an alias is to identify the lookup table having more than one purpose in the database structure. This is described in the following section.

6.5.6.2 Identifying multi-purpose lookup tables

The COUNTRY table is used to look up both the customer's country of residence and the shipment destination. This type of table is called a shared lookup table.

You create an alias in the schema called DESTINATION.



The three original joins still exist but the loop has been broken by the DESTINATION alias so there is no longer a closed join path.

6.5.6.3 Referencing the shared lookup table and alias in the FROM clause

You now need to reference the table name twice in the From clause, the first time with its ordinary name and the second time with an alias; so the original name is suffixed with an alternative name.

The resulting SQL is as follows:

```
SELECT   CUSTOMER.NAME,   COUNTRY.NAME,   ORDERS.ORDER_DATE   DESTINATION.NAME
FROM   CUSTOMER,   ORDERS,   COUNTRY,   COUNTRY DESTINATION WHERE
(CUSTOMER.CUST_ID=ORDERS.CUST_ID) AND   (ORDERS.SHIP_DEST_ID=
DESTINATION.COUNTRY_ID) AND   (CUSTOMER.CUST_LOC_ID=COUNTRY.COUNTRY_ID)
```

6.5.6.4 Creating an alias to break a loop caused by shared lookup tables

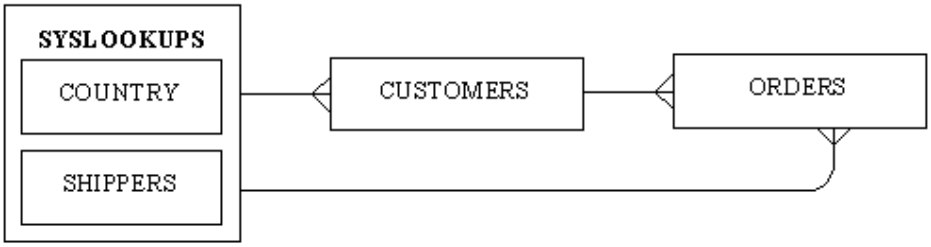
A sales database holds information about customers living in different countries. These customers can place orders for goods that can be delivered by a number of couriers or shipping companies.

In this database, the names of the countries and shippers have been normalized into lookup tables. Normalization is a process that refines the relationships of tables by removing redundancies.

For structural reasons, rather than two lookup tables, only one lookup table (SYSLOOKUPS) was created with a code, description and type field. The type field indicates the particular type of information the record holds; for example, country or shipper.

Referred to as a "flexible lookup," this type of table often appears in schemas automatically generated by CASE tools.

The schema and table layout are shown below:



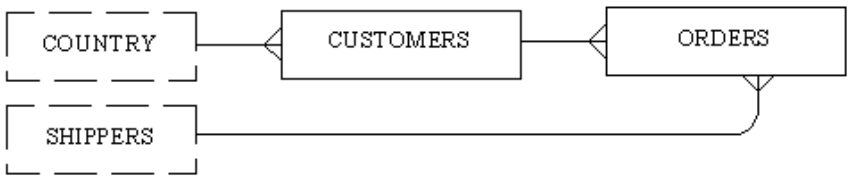
cust_id	last_name	loc_country
100	COLTRANE	1
101	MULLIGAN	1
102	WALDRON	3
103	HANCOCK	4
104	DAVIS	2
105	BARBIERI	5
106	STREATS	5

order_id	cust_id	order_date	ship_id
12345	100	1/1/95	2
12346	101	1/6/95	1
12347	101	2/6/95	3
12348	102	8/4/95	5
12349	103	10/3/95	4
12350	104	15/8/95	2
12351	105	6/2/95	5
12352	106	7/3/95	4

type	code	description
CTRY	1	USA
CTRY	2	UK
CTRY	3	France
CTRY	4	Germany
CTRY	5	Spain
SHIP	1	Man With A Van
SHIP	2	'Cut You Up' Couriers
SHIP	3	Parcel Fun
SHIP	4	Boggit & Leggit Couriers
SHIP	5	Deliveries 'R Us
SHIP	6	Sky Nut

The SYSLOOKUPS table serves more than one purpose so you have to create as many aliases as the table has domains (distinct values for the type field). Based on the two purposes that are represented in the SYSLOOKUPS table, you can create two aliases, COUNTRY and SHIPPERS.

The resulting schema is shown below:



In the universe design tool, you create the object `Customer's Country` defined as `COUNTRY.DESCRIPTION` and the object `Shipper` defined as `SHIPPERS.DESCRIPTION`.

The corresponding joins would be:

CUSTOMERS.LOC_COUNTRY=COUNTRY.CODE

ORDERS.SHIP_ID=SHIPPERS.CODE

Using self restricting joins to restrict results

Once you have defined the objects, you now need to restrict each alias so that it returns only its own domain information and not that of the others. For more information on creating self restricting joins, see the section Self restricting joins [page 164].

For example, if you wanted to know the names of the shippers who dispatched two orders to customer 101, you would expect two rows to be returned.

However, the following SQL

```
SELECT   ORDERS.ORDER_ID,    ORDERS.CUST_ID,   ORDERS.ORDER_DATE,
SHIPPERS.DESCRIPTION SHIPPER FROM   ORDERS,   SYSLOOKUPS SHIPPERS WHERE
(ORDERS.SHIP_ID=SHIPPERS.CODE)
```

would produce the results below:

order_id	cust_id	order_date	shipper
12346	101	1/6/95	Man With A Van
12346	101	1/6/95	USA
12347	101	2/6/95	Parcel Fun
12347	101	2/6/95	France

The query has returned the names of countries and shippers. Both "Man With a Van" and "USA" share code 1 while "France" and "Parcel Fun" share code 3.

You can correct the error as follows:

- Apply a new self restricting join to the SHIPPERS alias. In the Edit Join dialog box, you set both Table1 and Table2 to SHIPPERS and enter the SQL expression `SHIPPERS.TYPE='SHIP'`.
- Apply a new self restricting join to the COUNTRY alias. In the Edit Join dialog box, you set both Table1 and Table2 to COUNTRY and enter the SQL expression `COUNTRY.TYPE='CTRY'`.

Problems using restrictions

When you add the restriction to either the object's Where clause or to the existing join between the alias and the CUSTOMERS/ORDERS table, this can produce the following problems:

- When you add the restriction to the Where clause of an object, you must also add the same restriction to every object built from the alias. If you are creating a number of objects on an alias that has many columns, you could have problems maintaining the universe.
- The restriction to the join between the alias and another table only takes effect when the join is invoked. If you run a simple query containing only the `Shipper` object, every row in the SHIPPERS alias (including the unwanted Country rows) is returned as there is no reason to include the ORDERS table. As the join is not seen as necessary, the restriction is not applied.

Summary

In this example, we considered a schema with a shared lookup table. The actions carried out can be summarized as follows:

1. Create a COUNTRY and SHIPPERS alias for the shared lookup table.
2. Create self restricting joins for the aliases as restrictions.

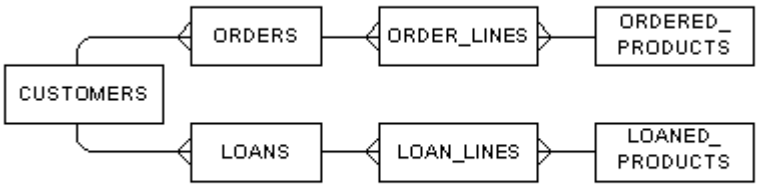
The aliases in this example resolve a loop by using one combined lookup table as two different lookup tables. These aliases also required the setting of restrictions (self-joins), so in some structures aliases may lead to the need for additional adjustments or restrictions.

6.5.6.5 Determining when an alias is not appropriate to break a loop

Creating an alias to resolve the loop described above is not the optimal solution. In this case, the use of contexts is a better solution. The following example describes why aliases are not appropriate, and why contexts are a better solution in this case.

If you try to identify the lookup table used for more than one purpose, it is not clear if it is the PRODUCTS table, or the CUSTOMERS table.

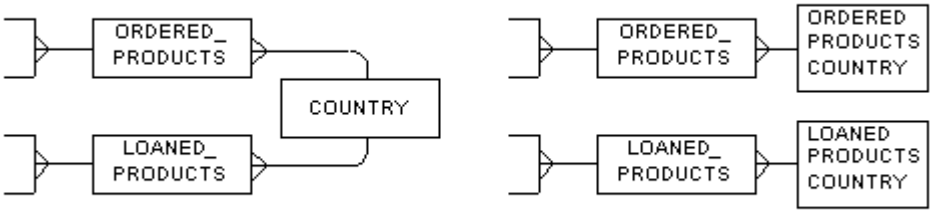
If you decide to create two aliases for the PRODUCTS table as shown below:



The two aliases are ORDERED_PRODUCTS and LOANED_PRODUCTS. This could be confusing for users as they are more likely to understand products, and not ordered products or loaned products.

If you also decide to add a COUNTRY table to indicate that the products are manufactured in several different countries you would have to join it directly to the PRODUCTS table.

The resulting schema would be as follows:



In the schema above, it was necessary to create two new aliases, ORDERED_PRODUCTS_COUNTRY and LOANED_PRODUCTS_COUNTRY. The use of aliases is obviously an unsatisfactory and complicated solution for this particular schema.

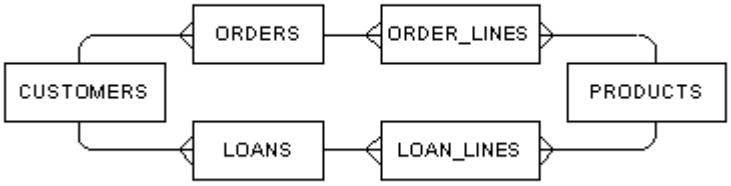
In this case, you should create contexts.

6.5.6.6 Creating a context to resolve a loop

A database contains information about customers who can either buy or rent products. In this database, there are two different ways to present the relationship between the customers and the products:

- By products that have been ordered by (or sold to) customers.
- By products that have been rented to customers.

This database has the following type of schema:



If we wanted to run a query that returns only a list of customer names and a list of products, we could use the ORDER and ORDER_LINES table. The result would be a list of products ordered by each customer.

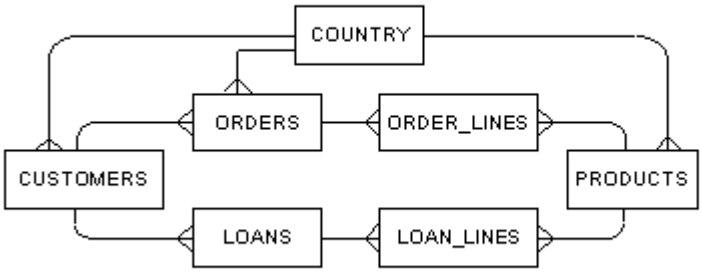
By using the LOANS and LOAN_LINES tables, we would obtain a list of products rented by each customer.

This schema contains a loop that causes any query involving all six joins simultaneously to result in a list made up of both products sold and rented to customers. If a product has been sold, but never rented to a customer or vice-versa, it would not appear in the list of results.

Using an alias and context together to resolve a loop

You can use contexts and aliases to resolve loops in a universe. The following example shows how to use both aliases and contexts together in a loop resolution.

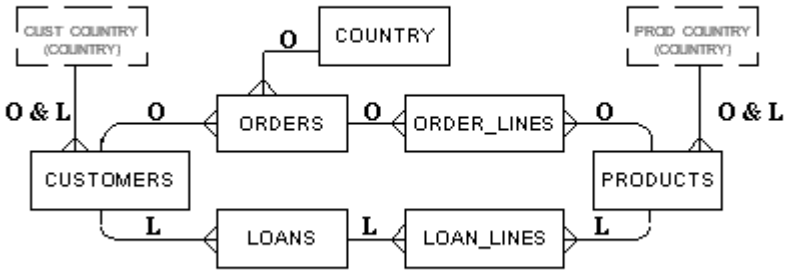
A universe has the following schema:



You can use aliases and contexts to resolve the loops as follows:

- Create two aliases for the COUNTRY table: CUST_COUNTRY and PROD_COUNTRY
- Define two contexts to resolve the CUSTOMERS to PRODUCTS loops (Orders and Loans)
- Ensure that the two joins between CUSTOMERS and CUST_COUNTRY and PRODUCTS and PROD_COUNTRY appear in both contexts.

The resulting schema appears as follows:



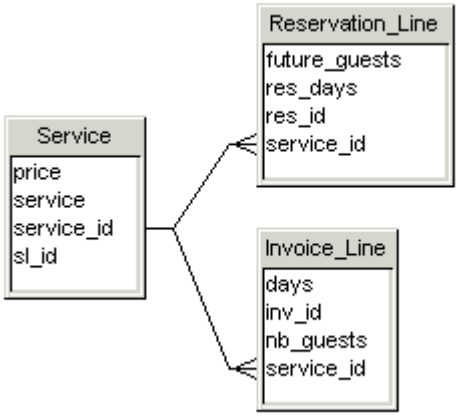
6.6 Resolving chasm traps

A chasm trap is a common problem in relational database schemas in which a join path returns more data than expected.

6.6.1 What is a chasm trap?

A chasm trap is a type of join path between three tables when two "many-to-one" joins converge on a single table, and there is no context in place that separates the converging join paths.

The example below shows a part of the Beach universe schema. The three tables have been separated from the rest of the schema to illustrate the chasm trap. It uses the same Club connection for data. The Service table receives the one ends of two one-to-many joins.



You will get incorrect results only when all the following conditions exist:

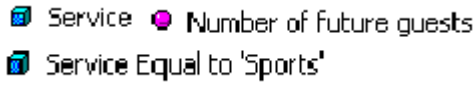
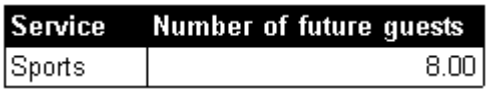
- A "many to one to many relationship" exists among three tables in the universe structure.
- The query includes objects based on two tables both at the "many" end of their respective joins.
- There are multiple rows returned for a single dimension.

The following is an example that shows how queries that are run when all the above conditions exist return a Cartesian product.

Example: A chasm trap inflates results without warning

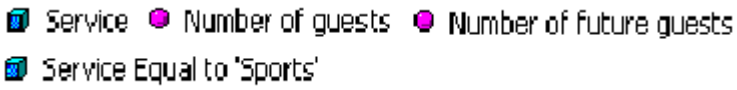
Using the schema above, a Web Intelligence user runs the following separate queries:

Query	Returned results
Service, Number of guests Service Equal to 'Sports'	Service: Sports Number of guests: 145.00

Query	Returned results
	

Service	Number of future guests
Sports	8.00

The user now runs a query that includes both paid guests and future guests:



The following results are returned:

Service	Number of guests	Number of future guests
Sports	188.00	96.00

The number of guests that have used, and future guests who have reserved to use the Sports service has increased considerably. A Cartesian product has been returned and the results are incorrect. This can be a serious problem if undetected. The above example could lead a manager at Island Resorts to think that sporting activities at the resorts are a more attractive service to guests, than the actual figures would indicate.

6.6.2 How does a chasm trap inflate results?

The chasm trap causes a query to return every possible combination of rows for one measure with every possible combination of rows for the other measure. In the example above, the following has occurred:

- Number of guests transactions *Number of future guest transactions
- Number of future guest transactions*Number of guests transactions

The following example examines in detail how a chasm trap returns a Cartesian product:

Example: Examining the Cartesian product of a chasm trap

We need to examine the rows that are returned by the queries to make the aggregated figures. In our example, we can do this by adding the dimensions Days Billed and Days Reserved to the queries to return individual transaction details.

The Number of Guests report appears as follows:

Service	Days billed	Number of guests
Sports	3.00	4.00
Sports	4.00	133.00
Sports	6.00	8.00

The Number of Future Guests report appears as follows:

Service	Days reserved	Number of future guests
Sports	1.00	7.00
Sports	2.00	1.00

The two reports show the following number of transactions:

- Number of Guests = 3 transactions
- Number of Future Guests = 2 transactions

When the two dimensions are both added to the query, the following results are returned:

Service	Days billed	Number of guests	Days reserved	Number of future guests
Sports	3.00	4.00	1.00	3.00
Sports	3.00	4.00	2.00	1.00
Sports	4.00	129.00	1.00	75.00
Sports	4.00	35.00	2.00	9.00
Sports	6.00	8.00	1.00	6.00
Sports	6.00	8.00	2.00	2.00
	Sum:	**188.00**	**Sum:**	**96.00**

The query returns every possible combination of Number of Guests rows with every possible combination of Number of Future Guests rows: the Number of Guests transactions each appears twice, and the Number of Future Guests transactions each appears three times.

When a sum is made on the returned data, the summed results are incorrect.

Unlike loops, chasm traps are not detected automatically by the universe design tool, however, you can use Detect Contexts (Tools > Detect Contexts) to automatically detect and propose candidate contexts in your schema.

Detect Contexts examines the many to one joins in the schema. It picks up the table that receives converging many to one joins and proposes contexts to separate the queries run on the table. This is the most effective way to ensure that your schema does not have a chasm trap.

You can also detect chasm traps graphically by analyzing the one-to-many join paths in your schema.

If you do not run Detect Contexts, nor spot the chasm trap in the schema, the only way to see the problem is to look at the detail rows. Otherwise there is nothing to alert you to the situation.

6.6.3 Detecting a chasm trap

You can find chasm traps by using Detect Contexts to detect and propose candidate contexts, and then examining the table where any two contexts diverge. This point where two contexts intersect is the source of a chasm trap.

If you have two fact tables with many to one joins converging to a single lookup table, then you have a potential chasm trap.

> → Tip
>
> For information on organizing the table schema to detect join problems, refer to Detecting join problems graphically [page 237].

6.6.4 Resolving a chasm trap

To resolve a chasm trap you need to make two separate queries and then combine the results. Depending on the type of objects defined for the fact tables, and the type of end user environment, you can use the following methods to resolve a chasm trap:

- Create a context for each fact table. This solution works in all cases.
- Modify the SQL parameters for the universe so you can generate separate SQL queries for each measure. This solution only works for measure objects. It does not generate separate queries for dimension or detail objects.

Each of these methods is described in the following sections.

6.6.4.1 Using contexts to resolve chasm traps

You can define a context for each table at the "many" end of the joins. In our example you could define a context from SERVICE to RESERVATION_LINE and from SERVICE to INVOICE_LINE.

When you run a query which includes objects from both contexts, this creates two Select statements that are synchronized to produce two separate tables in Web Intelligence, avoiding the creation of a Cartesian product.

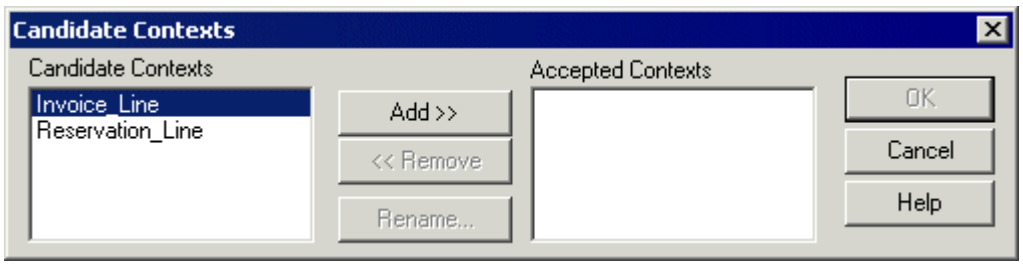
6.6.4.2 When do you use contexts?

Creating contexts will always solve a chasm trap in a universe. When you have dimension objects in one or both fact tables, you should always use a context.

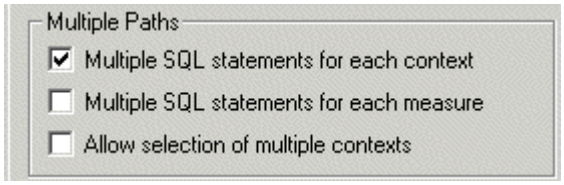
6.6.4.3 Using contexts to solve a chasm trap

To use contexts to resolve a chasm trap:

1. Identify the potential chasm trap by analyzing the "one-to-many-to-one" join path relations in the schema.
2. Select Tools > Detect Contexts.
   The Candidate Contexts box appears.



3. Select a proposed context in the Candidate Contexts list box and click the Add button to add it to the Accept Contexts list box.
4. Repeat for other listed contexts.
   The new contexts are listed in the Contexts pane of the List View bar.
5. Select File > Parameters.
   The Universe Parameters dialog box appears.
6. Click the SQL tab.
   The SQL page appears.
7. Select the Multiple SQL statements for each Context check box.



8. Click OK.
   When you run queries on the tables in the chasm trap, the query is separated for measures and dimensions defined on the affected tables.

6.6.4.4 Using Multiple SQL Statements for Each Measure

If you have only measure objects defined for both fact tables, then you can use the Universe Parameters option Multiple SQL statements for each measure. This forces the generation of separate SQL queries for each measure that appears in the Query pane.

This solution does not work for dimension and detail objects.

The following table describes when you can use Multiple SQL Statements for Each Measure and when you should avoid its use:

You...	In these situations...
Use Multiple SQL Statements for Each Measure	In universes that contain only measure objects defined for both fact tables. The advantage of using multiple SQL statements is that you can avoid using contexts that you need to maintain later.
Do not use Multiple SQL Statements for Each Measure	When you have dimension or detail objects defined for one or both of the fact tables. If a dimension or detail object is included in a query based on a universe using this solution, a Cartesian product will be returned. As this solution can slow query response time and produce incorrect results, than you should consider creating contexts to resolve the chasm trap.

To activate Multiple SQL Statements for Each Measure:

1. Select File > Parameters from the menu bar.
   The Universe Parameters dialog box appears.
2. Click the SQL tab.
3. Select the Multiple SQL Statements for Each Measure check box in the Multiple Paths group box.
4. Click OK.

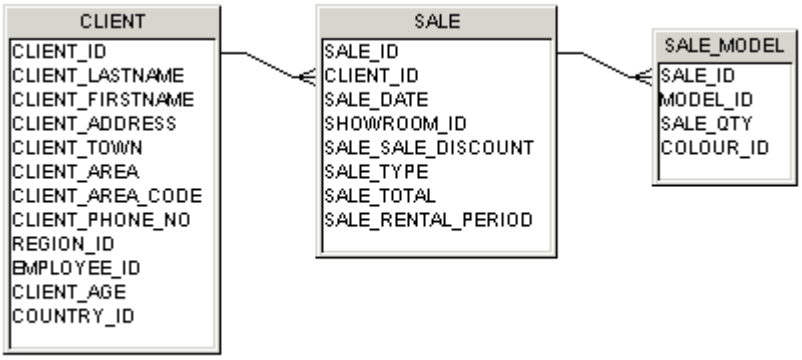
6.7 Resolving Fan Traps

A fan trap is a less common problem than chasm traps in a relational database schema. It has the same effect of returning more data than expected.

6.7.1 What is a Fan Trap?

A fan trap is a type of join path between three tables when a "one-to-many" join links a table which is in turn linked by another "one-to-many" join. The fanning out effect of "one-to-many" joins can cause incorrect results to be returned when a query includes objects based on both tables.

A simple example of a fan trap is shown below:



When you run a query that asks for the total number of car models sold by each model line, for a particular customer, an incorrect result is returned as you are performing an aggregate function on the table at the "one" end of the join, while still joining to the "many" end.

Example: A fan trap inflates results without warning

Using the schema above, the Web Intelligence user runs the following query:

ClientName SaleValue SaleQTY

ClientName Equal to 'WendyCraig'

The following results are returned:

ClientName	SaleQTY	SaleValue
WendyCraig	2.00	57,092.00

This result is correct. However, the end user adds the dimension Model ID to the query as follows:

ClientName SaleValue Model Id SaleQTY

ClientName Equal to 'WendyCraig'

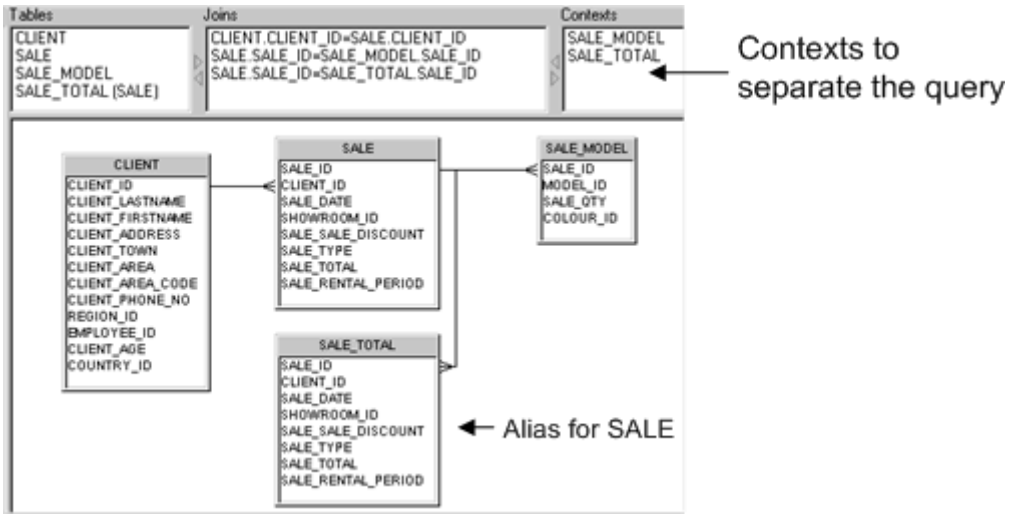
The following report is created with the returned results:

WendyCraig

Model Id	SaleValue	SaleQTY
1,034.00	57,092.00	1.00
1,081.00	57,092.00	1.00
Sum:	**114,184.00**	**2.00**

The Sale Value aggregate appears twice. Once for each instance of Model_ID. When these results are aggregated in a report, the sum is incorrect. The fan trap has returned a Cartesian product. Wendy bought two cars for a total of $57,092.00, and not 114,184.00 as summed in the report. The inclusion of Model_ID in the query, caused the SaleValue to be aggregated for as many rows as Model_ID.

The fan trap using dimension objects in the query is solved by using an alias and contexts. The following schema is the solution to the fan trap schema:



The original query which returned the Cartesian product for Wendy Craig, now returns the following table when run with the above solution:

WendyCraig

Sale Qty	Model Id
1.00	1,034.00
1.00	1,081.00

Sale Total
57,092.00

6.7.2 How do you detect a fan trap?

You cannot automatically detect fan traps. You need to visually examine the direction of the cardinalities displayed in the table schema.

If you have two tables that are referenced by measure objects and are joined in a series of many to one joins, then you may have a potential fan trap.

For a description to organize the table schema to detect join problems, see the section Detecting join problems graphically [page 237].

6.7.3 How do you resolve a fan trap

There are two ways to solve a fan trap problem.

- Create an alias for the table containing the initial aggregation, then use Detect Contexts (Tools > Detect Contexts) to detect and propose a context for the alias table and a context for the original table. This is the most effective way to solve the fan trap problem.
- Altering the SQL parameters for the universe. This only works for measure objects.

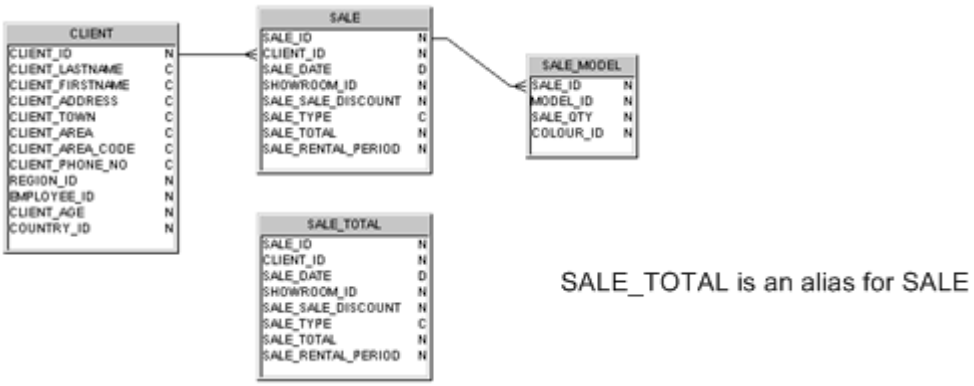
Both of these methods are described below.

6.7.3.1 Using aliases and contexts to resolve fan traps

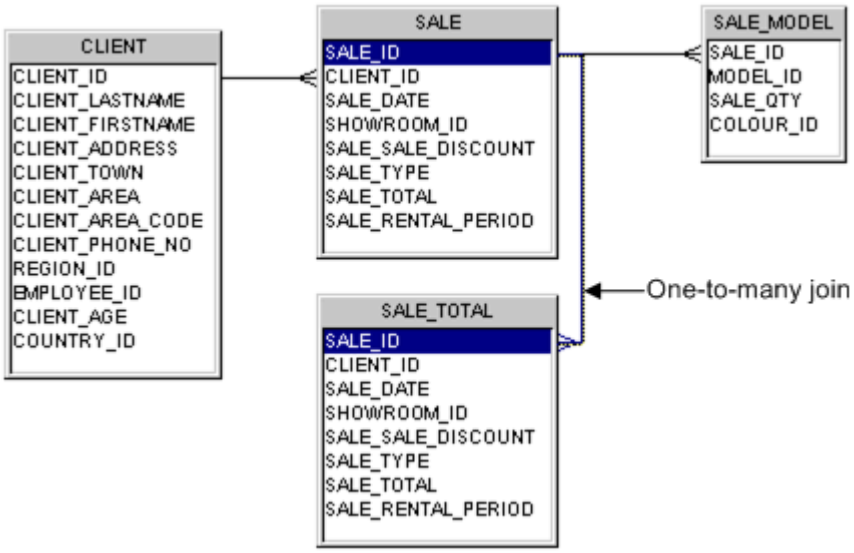
You create an alias table for the table producing the aggregation and then detect and implement contexts to separate the query. You can do this as follows:

To use aliases and contexts to resolve a fan trap:

1. Identify the potential fan trap by analyzing the "one-to-many-to-one-to-many" join path relations in the schema.
2. Create an alias for the table that is producing the multiplied aggregation.
   For example, SaleValue in the previous example is an aggregate of the Sale_Total column in the Sales table. You create an alias called Sale_Total for Sale.



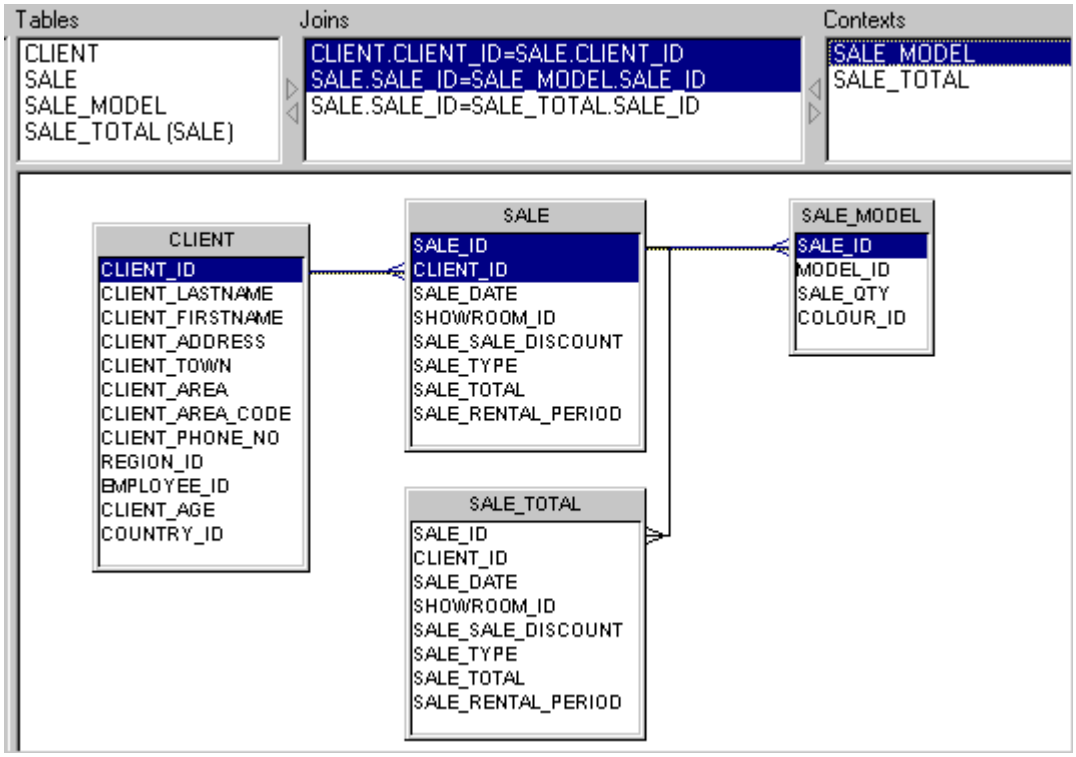
3. Create a join between the original table and the alias table.
   If you create a one-to-one join, the universe design tool does not detect the context, and you must build the context manually. In most cases you can use a one-to-many which allows automatic detection and implementation of contexts.
   For example you create a one-to-many join between Sale and Sale_Total.



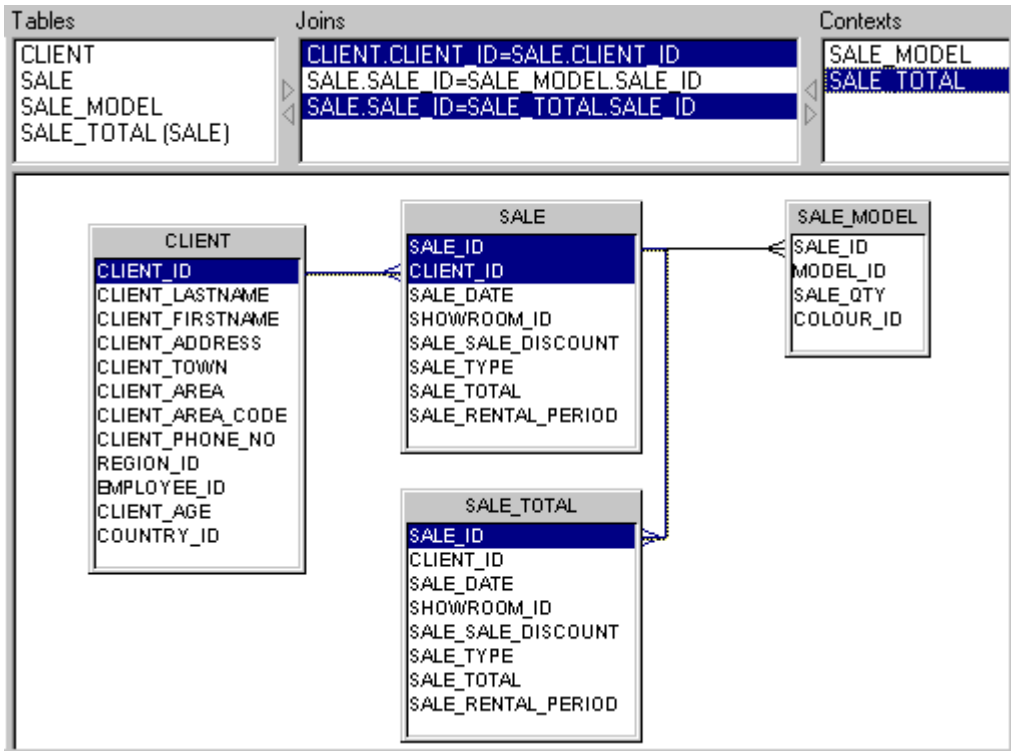
4. Build the object that is causing the aggregation on the alias tables.
   For example the original SaleValue object was defined as follows:
   sum(SALE.SALE_TOTAL). The new definition for SaleValue is:
   sum(Sale_Total.SALE_TOTAL).
5. Select Tools > Detect Contexts.
   The Candidate Contexts box appears. It proposes the candidate contexts for the join path for the base table and the new join path for the alias table.

   > **i Note**
   >
   > If you have used a one-to-one join between the alias and the base table, then you need to create the context manually.

6. Click a candidate context and click Add.
7. Repeat for the other candidate context.
8. Click OK.
   The contexts are created in the schema. You can view them in the Contexts pane when List Mode is active (View > List Mode). The context for the join path CLIENT>SALE>SALE_MODEL appears as follows:

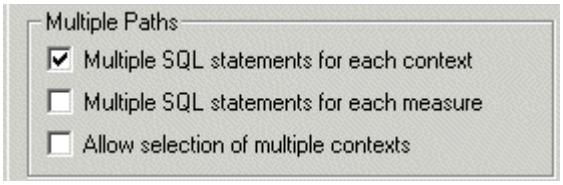


And a second context for the CLIENT>SALE>SALE_TOTAL join path:



9. Select File > Parameters.
   The Parameters dialog appears.

10. Click the SQL tab.SQL page.
    The SQL page appears.
11. Select the Multiple SQL Statements for Each Context check box.



12. Click OK.
13. Run queries to test the fan trap solution.

6.7.3.2 Using Multiple SQL Statements for Each Measure

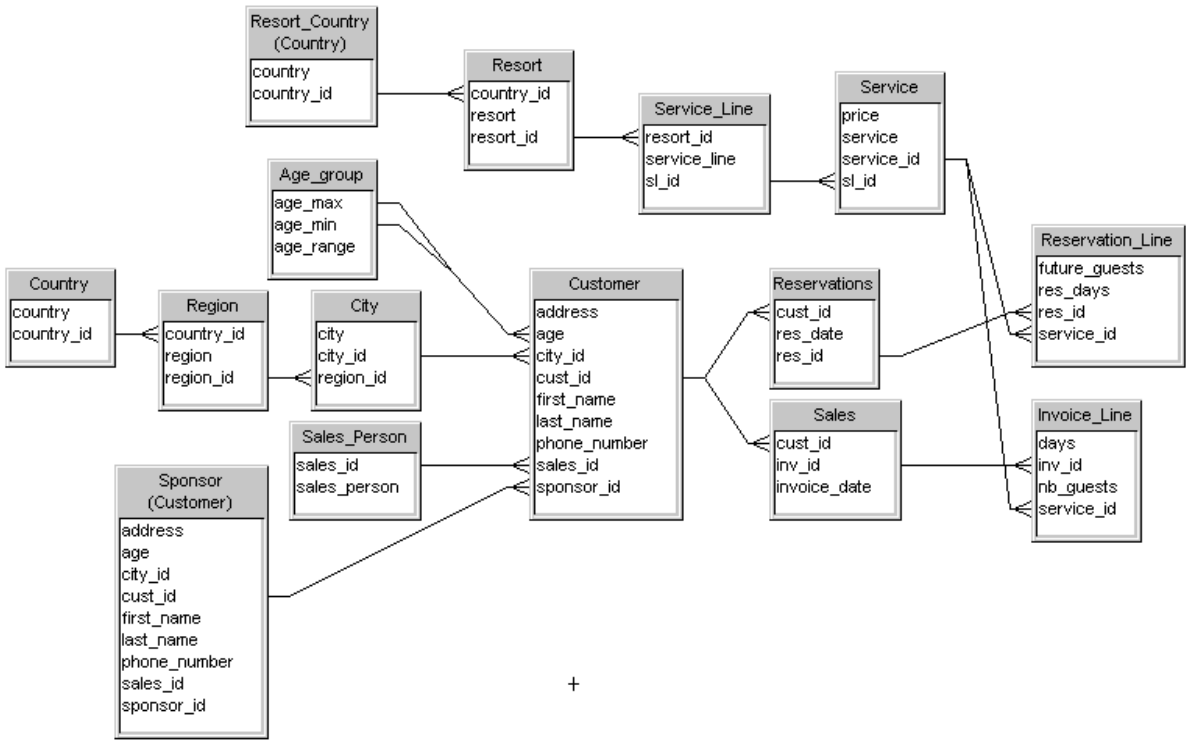
If you have only measure objects defined for both tables at the many end of the serial one-to-many joins, then you can use the Universe Parameters option Multiple SQL Statements for Each Measure. This forces the generation of separate SQL queries for each measure that appears in the Query pane.

You cannot use this method to generate multiple queries for dimensions. If an end user can include dimensions from any of the tables that reference the measure objects in the query, then you must use an alias and context to resolve the fan trap.

See the section Using Multiple SQL Statements for Each Measure [page 237] for more information and procedure to activate this option.

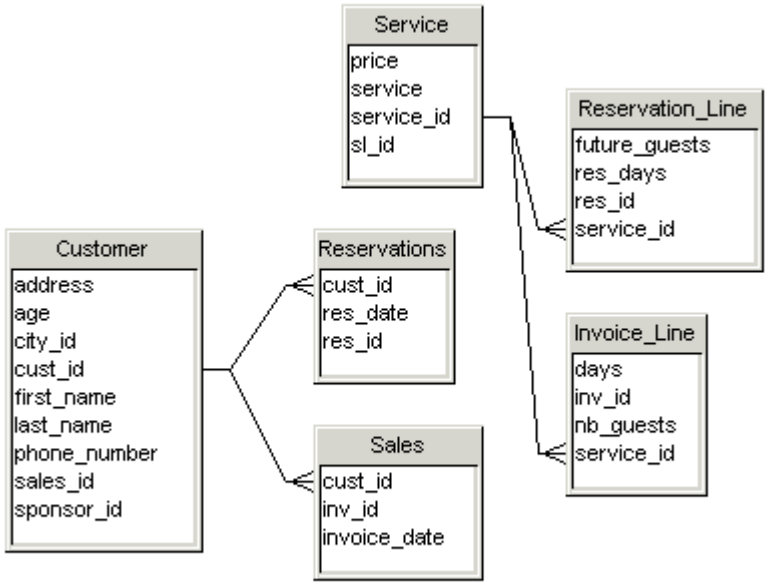
6.8 Detecting join problems graphically

You can visually detect potential chasm and fan traps in your table schema by arranging the tables in the Structure pane so that the "many" ends of the joins are to one side of the pane, and the "one" ends to the other. The example below shows the Beach universe schema arranged with a one to many flow from left to right.



6.8.1 Potential chasm trap

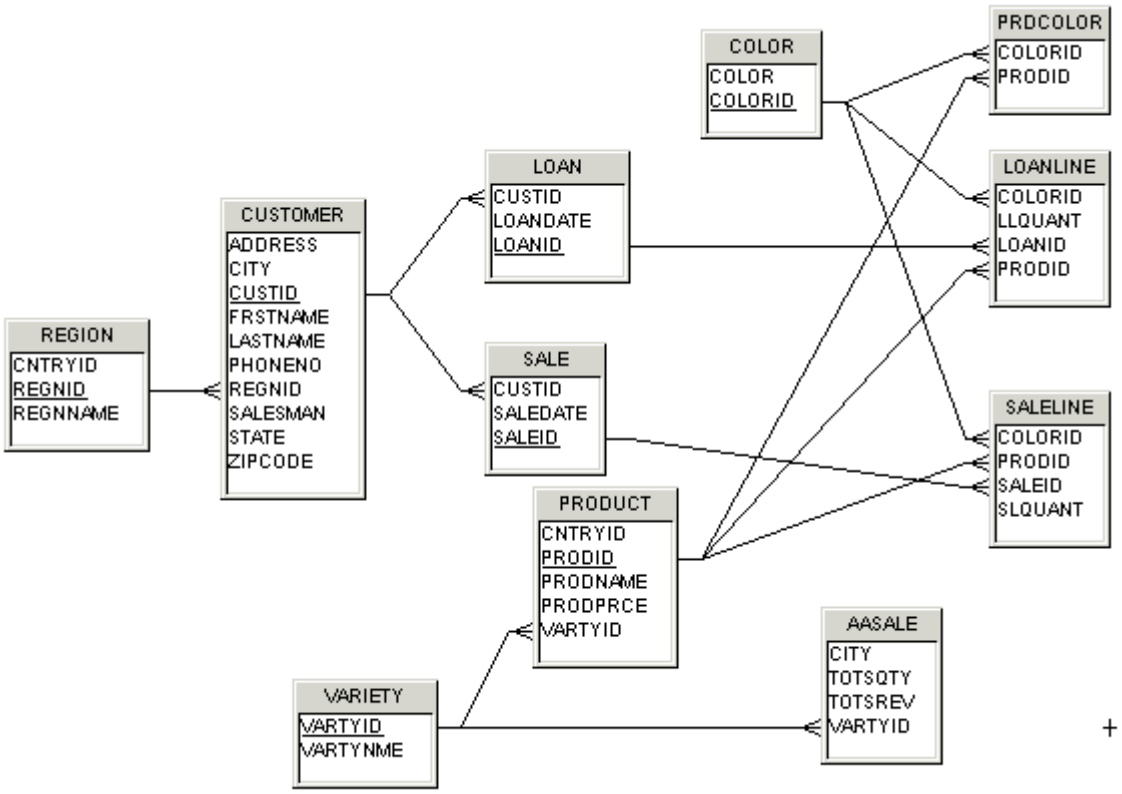
The potential chasm traps are shown below:



Both of these join paths have been separated using the contexts Sales and Reservations.

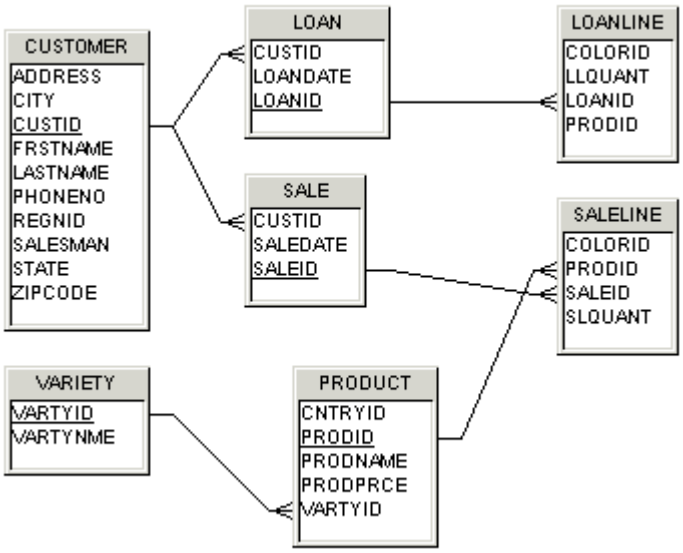
6.8.2 Potential fan trap

A universe schema for a car sales database is shown below:



The potential fan traps involve the following tables

- CUSTOMER, LOAN, and LOANLINE
- CUSTOMER, SALES, and SALELINE
- VARIETY, PRODUCT, and SALELINE



→ Tip

Once you have populated your schema with the necessary tables, don't start defining objects immediately. Allow some time to move tables around so that you have the all the one-to-many joins in the same direction. The universe design tool is a graphic tool, so use the visual capabilities of the product to help you design universes. An hour or so moving tables around could save you a lot of time later in the design process.

6.9 Checking the universe

As you design your universe, you should test its integrity periodically. You can verify universe integrity as follows:

Check universe	Description
Automatically	You can set universe design tool options to check the SQL syntax of universe structures at creation, universe export, or when a universe is opened.
Manually	You run Check Integrity to check selected universe structures.

6.9.1 Types of errors detected by Check Integrity

Check Integrity can detect:

- Invalid syntax in the SQL definition of an object, condition, or join.
- Loops
- Isolated tables
- Isolated joins
- Loops within contexts
- Missing or incorrect cardinalities

How does Check Integrity determine changes in a connected database?

The Check Integrity function sends a request to the database for a list of tables. It then compares this list with the tables in the universe. It carries out the same action for columns.

In the Structure pane, Check Integrity marks any tables or columns not matching those in the list as not available. These are tables or columns that may have been deleted or renamed in the database. See the section Verifying universe integrity with Check Integrity [page 178].

> **i** Note
>
> The option Check Cardinalities can be slow to run with large amounts of data. If there is ambiguous or missing data, results can also be inaccurate. If your database is large, and may have incomplete data entries, then you should not select the option Check Cardinalities. If you do use this option, then you can optimize the cardinality detection by modifying the PRM file. For more information, refer to the section Optimizing automatic cardinality detection [page 175].

6.9.1.1 Types of errors detected by Check Integrity

Check Integrity can detect:

- Invalid syntax in the SQL definition of an object, condition, or join.
- Loops
- Isolated tables
- Isolated joins
- Loops within contexts
- Missing or incorrect cardinalities

How does Check Integrity determine changes in a connected database?

The Check Integrity function sends a request to the database for a list of tables. It then compares this list with the tables in the universe. It carries out the same action for columns.

In the Structure pane, Check Integrity marks any tables or columns not matching those in the list as not available. These are tables or columns that may have been deleted or renamed in the database. See the section Verifying universe integrity with Check Integrity [page 178].

> **i Note**
>
> The option Check Cardinalities can be slow to run with large amounts of data. If there is ambiguous or missing data, results can also be inaccurate. If your database is large, and may have incomplete data entries, then you should not select the option Check Cardinalities. If you do use this option, then you can optimize the cardinality detection by modifying the PRM file. For more information, refer to the section Optimizing automatic cardinality detection [page 175].

6.9.2 Checking universe integrity manually

You can use Check Integrity to test to verify if the design of your active universe is accurate and up-to-date.

Check Integrity detects the following:

- Errors in the objects, joins, conditions, and cardinalities of your universe.
- Loops in join paths.
- Any necessary contexts.
- Changes to the target database.

Before examining the elements of the universe against those of the database, the function checks whether the connection to the database is valid. If the connection is not valid, the function stops and returns an error message.

6.9.2.1 Types of errors detected by Check Integrity

Check Integrity can detect:

- Invalid syntax in the SQL definition of an object, condition, or join.
- Loops
- Isolated tables
- Isolated joins
- Loops within contexts
- Missing or incorrect cardinalities

How does Check Integrity determine changes in a connected database?

The Check Integrity function sends a request to the database for a list of tables. It then compares this list with the tables in the universe. It carries out the same action for columns.

In the Structure pane, Check Integrity marks any tables or columns not matching those in the list as not available. These are tables or columns that may have been deleted or renamed in the database. See the section Verifying universe integrity with Check Integrity [page 178].

> **Note**
>
> The option Check Cardinalities can be slow to run with large amounts of data. If there is ambiguous or missing data, results can also be inaccurate. If your database is large, and may have incomplete data entries, then you should not select the option Check Cardinalities. If you do use this option, then you can optimize the cardinality detection by modifying the PRM file. For more information, refer to the section Optimizing automatic cardinality detection [page 175].

6.9.2.2 How does Check Integrity determine changes in a connected database?

The Check Integrity function sends a request to the database for a list of tables. It then compares this list with the tables in the universe. It carries out the same action for columns.

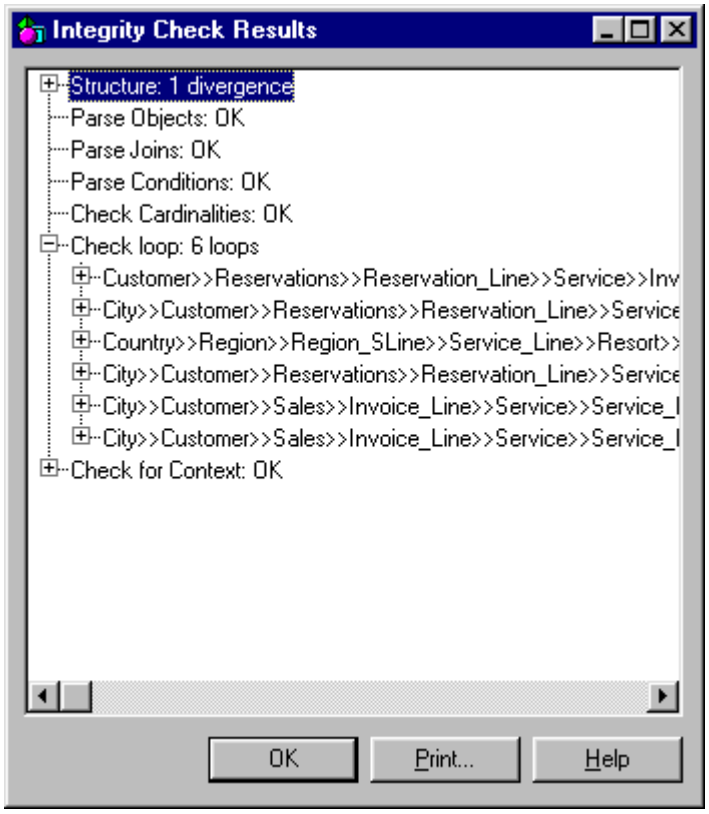
In the Structure pane, Check Integrity marks any tables or columns not matching those in the list as not available. These are tables or columns that may have been deleted or renamed in the database. See the section Refreshing the Universe Structure [page 245].

> **Note**
>
> The option Check Cardinalities can be slow to run with large amounts of data. If there is ambiguous or missing data, results can also be inaccurate. If your database is large, and may have incomplete data entries, then you should not select the option Check Cardinalities. If you do use this option, then you can optimize the cardinality detection by modifying the PRM file. For more information, refer to the section Optimizing automatic cardinality detection [page 175].

6.9.2.3 Verifying universe integrity with Check Integrity

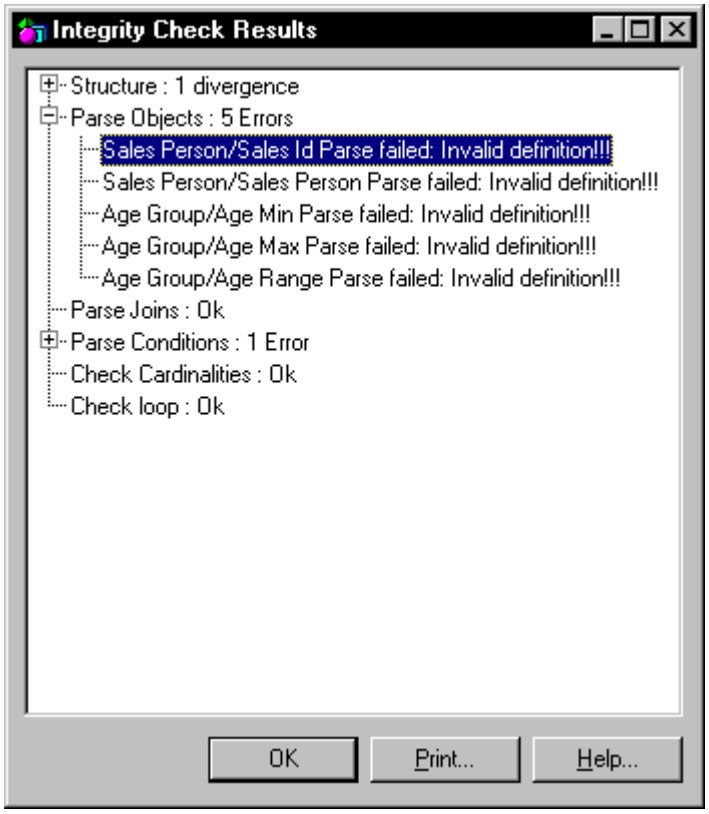
To verify universe integrity:

1. Select Tools > Check Integrity.
   Or
   Click the Check Integrity button.
   The Integrity Check dialog box appears.
2. Select check boxes for components to be verified.
3. Clear check boxes for components not to be verified.
4. Select the Quick Parsing check box to verify only the syntax of components.
   Or
   Select Thorough Parsing check box to verify both the syntax and semantics of components.
5. Click OK.
   A message box displays the universe check progress.



If Check Integrity encounters no errors, it displays "OK" beside each error type.

6. Click the plus sign (+) beside the error type to view the list of components in which the error occurred.



You can double click an item in the list to highlight the corresponding components in the Structure pane.

7. Click the Print button to print the window contents.
8. Click OK.

> **i** Note
>
> Before selecting the Check for Loops check box, ensure that the cardinalities of joins have already been detected. Otherwise, the function erroneously identifies loops in the joins.

6.9.3 Refreshing the Universe Structure

If Check Integrity indicates that the database of your universe connection has been modified, you can use Refresh Structure to update the contents of the Structure pane.

Refresh Structure can modify the universe structure to comply with changes in the database as follows:

If	Then the tool does the following
Columns were added to tables	Adds the columns to the corresponding tables in the universe.
Columns were removed from tables	Displays a warning message indicating the columns and associated joins you should delete.
Tables were removed from the database	Displays a warning message indicating the tables and associated joins you should delete.
Tables were renamed in the database	Displays a message that says it no longer recognizes the corresponding tables in the universe. You should rename these tables to match those in the database. If the names still do not match, the universe design tool returns a message stating that the renamed tables do not exist in the database.
No changes were made to the database	Displays a message informing you that no update is needed.

6.9.3.1 Refreshing a universe

To refresh the universe structure:

- Select View > Refresh Structure.
  A message box appears informing you of a change in the database, or that no update is needed if no changes have been made.

7 Creating universes

When you have created the schema and performed an integrity check and resolved loop problems, you are ready to create the universe that will be used by the reporting tools.

7.1 Overview

This chapter describes how you can create the classes and objects that are used by Web Intelligence users to run queries and create reports. it also covers optimizing object definitions to enhance end user reporting, and universe optimization.

The previous chapters have described how you plan a universe, create a table schema which contains the database structure of a universe: the tables, columns, and joins, and also how to resolve loops in join paths.

The schema that you have created is not visible by Web Intelligence users. Once this database structure is complete, you can now build the classes and objects that users see in the *Universe pane*, and will use to run queries on the databases structure to generate documents and reports.

7.2 Introduction to universe building

Building a universe is the object creation phase of the universe development cycle. The objects that you create must be based on a user needs study and use a sound schema design that has been tested for join path problems.

The following list indicates where the building (and test) phase appears in a typical universe development cycle (Implementation, step 2):

- Preparation

1. User needs analysis
2. Planning

- Implementation

1. Design and test schema
2. Build and test universe objects
3. Deploy universe using repository

- Maintenance

1. Update and maintain universe based on changes in user requirements or data source

7.2.1 What is an object?

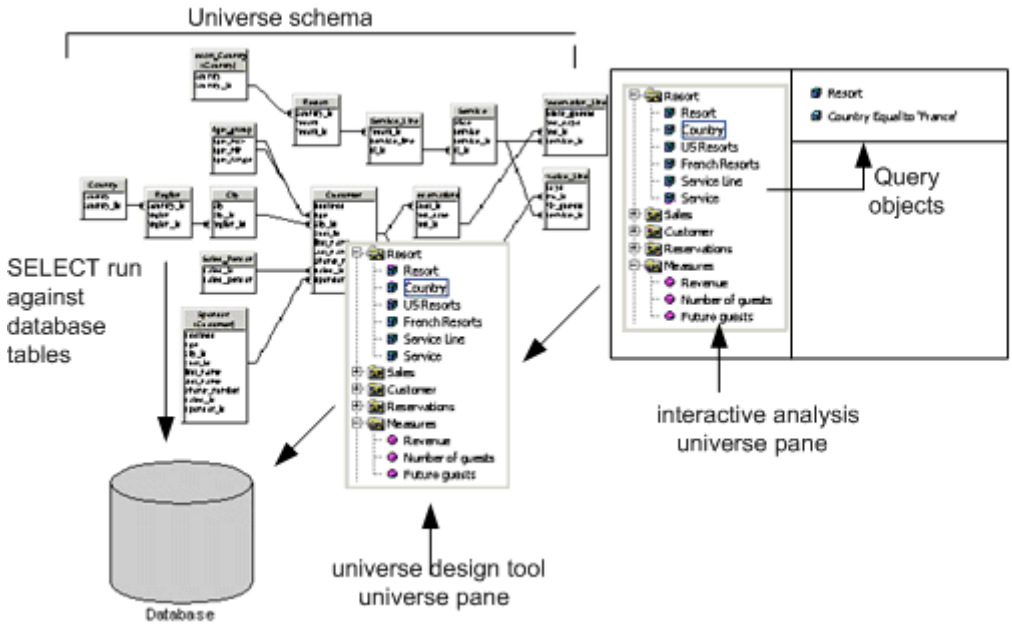
In Business Objects products an object is a named component in a universe that represents a column or function in a database.

Objects appear as icons in the *Universe pane*. Each object represents a meaningful entity, fact, or calculation used in an end users business environment. The objects that you create in the *Universe pane* in the universe design tool are the objects that end users see and use in the reporting tools. You can also create objects for use only in the universe design tool, which you can hide in the *Universe* pane seen by Web Intelligence users.

Web Intelligence users drag objects from the *Universe* pane across into the *Query* pane to run queries and create reports with the returned data.

Each object maps to a column or function in a target database, and when used in the *Query* pane, infers a SELECT statement. When multiple objects are combined, a SELECT statement is run on the database including the SQL inferred by each object and applying a default WHERE clause.

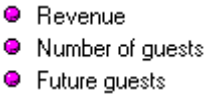
The diagram below shows objects in the Web Intelligence *Universe* pane and the same objects in the universe design tool *Universe* pane. Each object in the universe design tool *Universe* pane maps to a column in the universe schema, and infers a SELECT statement when used in a query.



As the universe designer, you use the universe design tool to create the objects that Web Intelligence users include in the *Query* pane to run their queries.

7.2.2 What types of objects are used in a universe?

You can qualify an object as being one of three types:

Object qualification	Examples	Description
Dimension		Focus of analysis in a query. A dimension maps to one or more columns or functions in the database that are key to a query.
Detail		Provides descriptive data about a dimension. A detail is always attached to a dimension. It maps to one or more columns or functions in the database that provide detailed information related to a dimension.
Measure		Contains aggregate functions that map to statistics in the database.

When you create an object, you assign it a qualification based on the role that you want that object to have in a query. This role determines the Select statement that the object infers when used in the *Query* pane.

7.2.3 Using classes and objects

You organize classes and objects together in the universe pane to correspond to the way that Web Intelligence users are accustomed to work with the information represented by the objects.

7.2.4 What is a class?

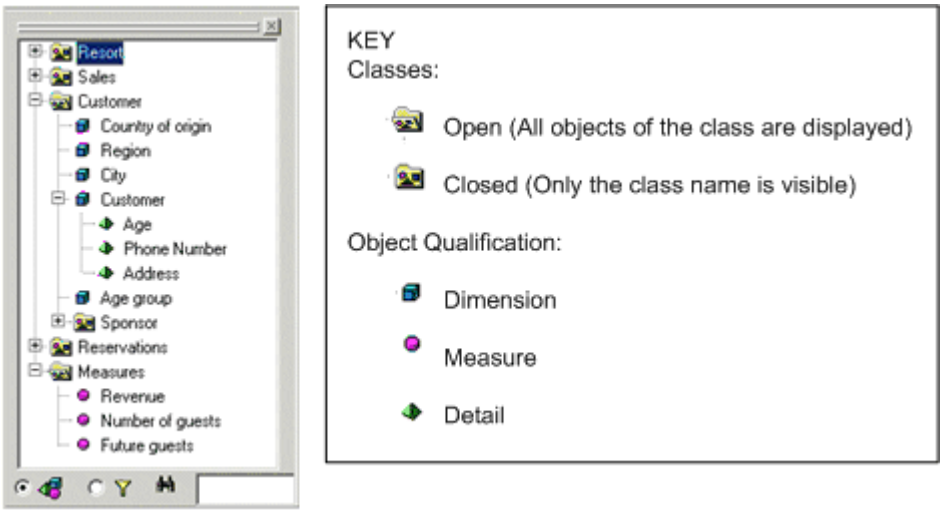
A class is a container of objects. A class is the equivalent of a folder in the Windows environment. You create classes to house objects that have a common purpose in the universe.

7.3 Using the Universe pane

You create the classes and objects in a universe using the *Universe pane*.

The *Universe pane* presents a hierarchical view of the classes and objects in the active universe. You use the *Universe pane* to view, create, edit, and organize classes and objects

The *Universe pane* is shown below. Class names appear beside a folder icon, and object names beside their qualification symbols.



7.3.1 Displaying classes and objects or conditions

You can use the two radio buttons at the bottom of the window to display classes and objects, or condition objects in the Universe Pane. Condition objects are predefined Where clauses that can be used within one or more Select statements.

You can display two views of the universe pane:

View	To display the view...	What it shows
Classes/Objects	Select left radio button	All classes and objects
Classes/Conditions	Select right radio button	All classes and conditions applied on objects contained within each class

Related Information

Defining restrictions for an object [page 277]

7.4 Basic operations on classes, objects, and conditions

You can perform the following operations in the *Universe* pane that are common to classes, objects and conditions:

7.4.1 Cut, copy, paste

You can cut, copy, and paste a selected component with the usual standard commands used in a Windows environment.

7.4.2 Moving classes, objects, or conditions

You can move a component to another position in the window by dragging and dropping it at the desired location.

7.4.3 Showing or hiding classes, objects and conditions

You can hide one or more components in the Universe Pane. These are hidden from Web Intelligence users, but remain visible in the universe design tool.

Hiding objects from end users can be useful for any of the following reasons:

- Components are from linked universes and are not needed in the active universe.
- Objects are used only to optimize SQL syntax and should be hidden from end users.
- You are in the process of developing a component that you do not want end users to view from the *Query* pane.
- You want to disable components temporarily without deleting them.

7.4.3.1 Hiding a class, object, or condition

To hide a class, object, or condition:

1. Click the component in the Universe pane.
2. Select *Edit* > *Hide Item(s)*.
   Or
   Click the *Show/Hide* button on the *Editing* toolbar.
   The component name is displayed in italics in the *Universe* pane

7.4.3.2 Showing a hidden class, object, or condition

The name of hidden components appears in italics.

To show a hidden class, object, or condition:

1. Click the hidden component in the *Universe* pane.
2. Select *Edit* > *Show Item(s)*.
   The name of the component is no longer in italics. It is now visible to end users.

7.5 Defining classes

A class is a container of one or more objects. Each object in a universe must be contained within a class. You use classes to group related objects. Classes make it easier for end users to find particular objects. You can create new classes and edit the properties of existing classes. Classes are represented as folders on a tree hierarchy in the Universe pane.

> → Tip
>
> A useful way to use classes is to group related dimension and detail objects into a class, and place measure objects in a separate class. The grouping of related objects can be further organized by using subclasses to break objects down into subsets. Subclasses are described in the section Using subclasses [page 253]

7.5.1 Creating a class

There are two ways to create a class in the Universe pane:

- Manually defining a class.
- Automatically by dragging a table from the table schema into the Universe pane.

Both methods are described as follows:

7.5.1.1 Creating a class manually

You can create classes manually within the Universe pane. If you have analyzed user needs and have listed and grouped the potential objects into classes, then creating classes manually from your list is the best way to ensure that your universe structure corresponds to the needs of end users.

To create a class in an empty Universe pane:

1. Select Insert > Class.
   Or
   Click the Insert Class button.
   A class properties box appears.

2. Type a name in the Class Name text box.
3. Type a description for the class in the Description text box.
4. Click OK.
   The new named class folder appears in the Universe pane.

> → Tip
>
> If you click Apply instead of OK, the name and description for a class are applied, but the properties box stays open. If you create another class, you can type properties for the new class in the same box. This allows you to create a series of classes using a single properties box. As you avoid a new properties box appearing with the creation of each class, you can save time and unnecessary clicking.

7.5.1.2 Creating a class in the universe pane with existing classes

To create a class with existing classes:

1. Click the class that you want to precede the new class in the tree view and select Insert > Class.
   Or
   Click the class that you want to precede the new class in the tree view and click the Insert Class button.
   A class properties box appears.
2. Type a name and description for the new class.
3. Click OK.
   The new named class folder appears in the Universe pane.

7.5.1.3 Creating a class automatically from the table schema

You can create classes automatically by selecting a table in the table schema and dragging it into the Universe pane. The table name is the class name by default. New objects are also automatically created under the class. Each new object corresponds to a column in the table.

You should edit the new class and object properties to ensure that they are appropriately named, and are relevant to end user needs. Editing object properties is described in the section Defining objects [page 254].

The Objects strategy selected on the Strategies page in the Universe Parameters dialog box determines how the classes and objects are created automatically (File>Parameters>Strategies tab). This strategy can be modified. You can also create strategies to customize the class and object creation process. See the section Using external strategies to customize universe creation [page 371], and the section Selecting strategies [page 83] for more information on strategies.

> ℹ Note
>
> When you create class and objects automatically, you are creating the universe components directly from the database structure. The class and objects that you create should be the result of a user needs analysis, and not be directed by the columns and tables available in the database. Designing the universe from user needs is described in the section Universe design methodology [page 21].

To create a class automatically from the table schema:

1. Select a table in the table schema.
2. Drag the table across to the Universe pane and drop the table at the desired position in the class hierarchy. A new class appears in the hierarchy. It contains an object for each column in the table dragged into the Universe pane. By default, the class name is the same as the table name, and each object name is the same as its corresponding column name.

7.5.2 Class properties

You can define the following properties for a class:

Property	Description
Name	Can contain special characters. Must be unique in universe. A class name is case sensitive. You can rename a class at any time.
Description	Comment that describes a class. This description can be viewed by users in the Query pane. Information in this field should be expressed in the business language of the user, and be relevant to their query needs. You create a line break by pressing CTRL + Return.

7.5.3 Modifying a class

You can modify the name and description of a class from the class properties dialog box at any time. You can access a class properties dialog box by any of the following methods:

- Double click a class folder.
- Right click a class folder, and select Edit > Class Properties.
- Click a class folder, and select Edit > Class Properties.

> **i** Note
>
> You can perform any of the above click operations on either the class folder or the class name to access the class properties dialog box.

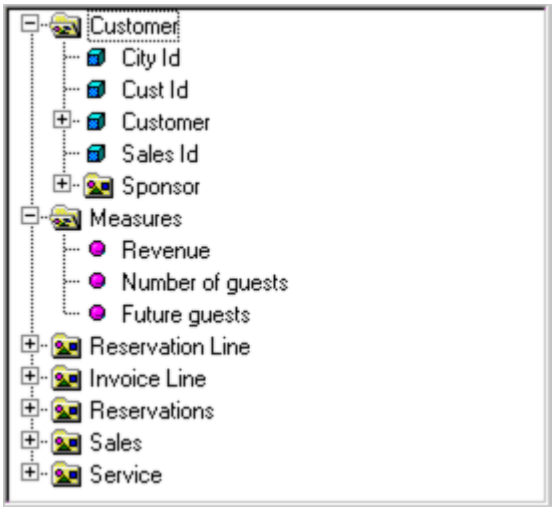
7.5.4 Using subclasses

A subclass is a class within a class. You can use subclasses to help organize groups of objects that are related. A subclass can itself contain other subclasses or objects.

7.5.4.1 Creating a subclass

To create a subclass:

- Click a class folder or a class name, then select Insert > Subclass.
- Right click a class folder or name, then select Insert Subclass from the contextual menu.
  The Universe pane below shows a subclass Sponsor listed under the class Customer.



7.6 Defining objects

An object is a universe component that maps to one or more columns in one or more tables in the universe database schema. An object can also map to a function defined on one or more columns.

Each object infers a Select statement for the column or function to which it maps. When an Web Intelligence user builds a query using one or more objects in the Query pane the content of the Select clause line in the Select statement is inferred using the column(s) or function represented by each object.

7.6.1 Creating an object

You create objects in the Universe pane. Web Intelligence users identify an object by its name and qualification. You can create objects manually in the Universe pane, or automatically by dragging the appropriate database structure from the Structure pane to the Universe pane.

7.6.1.1 Creating an object manually

You create an object manually by inserting an object in the Universe pane, and then defining the properties for the object. An object must belong to a class.

To create an object manually

1. Right click a class in the Universe pane and select Insert Object from the contextual menu.
   Or
   Click a class and click the Insert Object tool.
   An object is inserted under the selected class and the Edit Properties box for the object appears.
2. Type a name in the Name box.
   Ensure that object names are always expressed in the end user business vocabulary. This name may be different from the actual column names that the object is associated with in the database schema.
3. Click the Properties tab and select object properties.
4. Type a Select statement in the Select box, or click the Select button to use the SQL editor.
5. Click OK.

Related Information

Using the SQL Editor [page 269]
Object properties [page 256]

7.6.1.2 Creating an object automatically

You can create an object automatically by selecting a column in a table in the Structure pane and dragging it to the Universe pane. An object is created under the nearest class to the point where you drop the column. The default name for the object is the column name. All underscores are replaced with spaces. The default object datatype is derived from the column datatype. You can change this value by selecting a new datatype from the drop down list box in the Edit Properties sheet for the object.

You should edit the new object properties to ensure that it is appropriately named, and is relevant to end user needs. Editing object properties is described in the section Defining objects [page 254].

The Objects strategy selected on the Strategies page in the Universe Parameters dialog box determines how the classes and objects are created automatically (File>Parameters>Strategies tab). This strategy can be modified. You can also create strategies to customize the class and object creation process.

Refer to Using external strategies to customize universe creation [page 371], and Selecting strategies [page 83] for more information on using strategies.

> **i Note**
>
> When you create class and objects automatically, you are creating the universe components directly from the database structure. The classes and objects that you create should be the result of a user needs analysis, and not be directed by the columns and tables available in the database. Designing the universe from user needs is described in the section Universe design methodology [page 21].

To create an object automatically:

1. Click a table column in the Structure pane.
2. Drag the column across to the Universe pane and drop the table at the desired position in the class hierarchy. The column must be dropped under an existing class.
   A new object appears in the hierarchy. By default, the object name is the same as the column name.

You should ensure that object names are always expressed in the end user business vocabulary. This name may be different from the actual column names that the object is associated with in the database schema.

7.6.2 Object properties

You define the following object properties from the Edit Properties dialog box for a selected object:

Edit Properties page	Properties
Definition See Object definition [page 257] for full information on available object definition properties.	• Name • Datatype • Description • Select statement • Where clause You can access the SQL editor from this page to define SELECT and WHERE syntax.
Properties See Properties [page 260] for full information on available object properties.	• Object qualification • Associated list of values
Advanced See Advanced [page 261] for full information on available advanced object properties.	• Security • User rights on object • Date formats
Keys See Defining index awareness [page 262] for information on defining index awareness for an object.	• Key type • Select • Where • Enable
Source Information See Source Information [page 266] for information on using this tab.	• Technical information • Mapping • Lineage

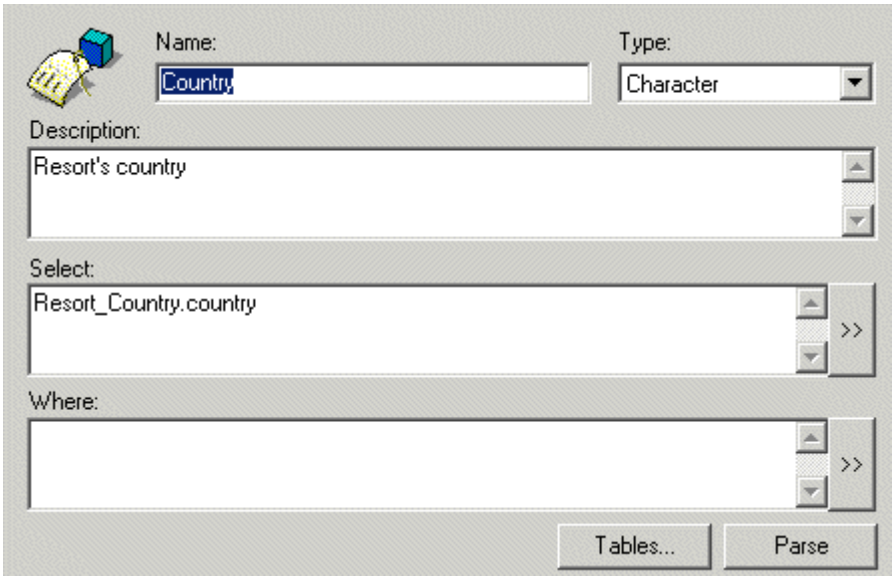
You can modify object properties at any time. Each object property listed above is fully described for each Edit Properties page in the section Modifying an object [page 257].

7.6.3 Modifying an object

You can define object properties at object creation, or modify them at any time. You define object properties from the Edit Properties dialog box for the object (right-click object > Object Properties). The properties you can define on each page of the Edit Properties dialog box are described as follows.

7.6.4 Object definition

The Definition page is shown below:



You can define the following properties from the Definition page of the Edit Properties dialog box.

Property	Description	Required/Optional
Name	Object name. It can consist of alphanumeric characters including special characters and spaces. Name is case-sensitive. Object names must be unique within a class. Objects in different classes can have the same name.	Required

Property	Description	Required/Optional
Type	Object datatype. It can be one of four types: • Character • Date • Long text • Number Blobs are not supported in the current version of the universe design tool .	Required
Description	Comments for object. This field can be viewed from the Query pane, so you can include information about the object that may be useful to an end user. Press Ctrl+Return to move the pointer to the next line.	Optional
Select	Select statement inferred by the object. You can use the SQL Editor to create the Select statement. See the section Properties [page 260].*	Required
Where	Where clause of the Select statement inferred by the object. The Where clause restricts the number of rows returned in a query. You can use the SQL Editor to create the Where clause.*	Optional

* You can insert or edit an @Prompt in the Select statement or Where clause. Right-click in the Select statement or Where clause, a shortcut menu offers either *New @Prompt* when there is no @Prompt in the statement, or *Edit @Prompt* when you click inside an existing @prompt. The *@Prompt* editor opens.

Tables button

When you click the Tables button a list of tables used in the schema appears. From this list you can select other columns in other tables to be included in the object definition. This allows an object to infer columns from several tables in a the Select statement. Refer to the section Applying a restriction by inferring multiple tables [page 287] for more information.

Parse button

When you click the Parse button, the Select statement for an object is parsed. If there are syntax errors detected, a message box appears describing the error.

Related Information

Using the SQL Editor [page 269]
Calculated measures in OLAP universes [page 420]
The @Prompt Editor [page 349]

7.6.4.1 Editing an object definition

To edit an object definition:

1. Double click an object.
   The Edit Properties dialog box opens to the Definition page.
2. Type or select object definitions and properties as required.
3. Click OK.

7.6.4.2 Defining an object as a dynamic hyperlink

You can define the text in a cell as a hyperlink. This method is useful for dynamic hyperlinks in a report, where the text in the cells of a column becomes a hyperlink to a specific resource (dependent on the result object).

Edit the object's select statement to include a hyperlink declaration, and select the object's format property Read As Hyperlink.

7.6.4.3 To define an object as a dynamic hyperlink

Using this approach, the object will create a dynamic hyperlink in the resulting report.

1. Right-click on the object and select *Object Properties*.
   The *Edit Object Properties* dialog displays.
2. Type the select statement and include the appropriate hyperlink.
3. Save the new properties.
4. Right-click on the object and select *Object Format*.
   The *Object Format* pane displays.
5. Select the option *Read As Hyperlink*.
6. Click *OK* to save the format settings.
7. Use your reporting tool to create the report and test the link.

When the object is used in a report, the resulting column will include hyperlinks.

Example: Using a hyperlink to calendar information

The following Select statement retrieves year calendar information from timeanddate.com depending on the year value in the column cell. Note that the declaration takes the four rightmost characters (the year) and removes the 'FY' (Financial Year') from the string, which is not recognized by the target URL.

```
'<a href=http://www.timeanddate.com/calendar/?year=>' +right(@Select(Reservations
\Reservation Year),4) +'</a>'
```

7.6.5 Properties

You can specify the following object qualifications and properties for a list of values from the Properties page of the Edit Properties dialog box:

Property	Description
Qualification	Defined role that object takes when used in the Query pane. You can qualify an object as being one of three types: • Dimension • Detail • Measure Refer to the section What types of objects are used in a universe? [page 247] for a more detailed description of object qualifications.
Associate a List of Values	When selected, associates a file containing data values with an object. Activated by default. Refer to the section Using lists of values [page 298] for more information.

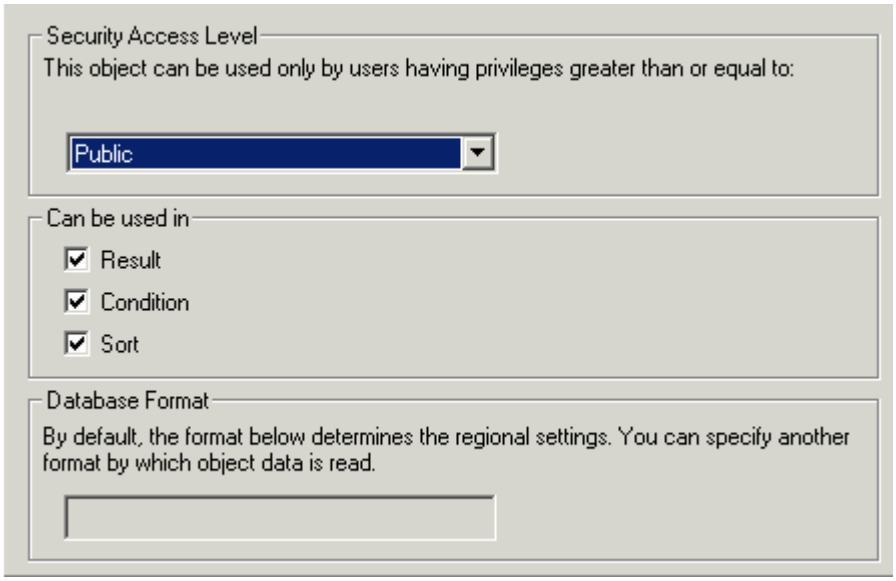
7.6.5.1 Specifying object qualification and list of values properties

To specify qualification and list of values properties for an object:

1. Double click an object.
   The Edit Properties box for the object appears.
2. Click the Properties tab.
   The Properties page appears.
3. Click a qualification radio button to determine whether the object is a dimension, detail, or measure.
   If you want to associate a list of returned values with the object, select the Associate a List of Values check box.
   For information on creating and using lists of values, see the section Using lists of values [page 298].
4. Click OK.

7.6.6 Advanced

The Advanced page is shown below.



You can define the following properties from the Advanced page of the Edit Properties dialog box:

Property	Description
Security Access Level	Defines the security access level of the object.You can select a security level which restricts use of the object to users with the appropriate security level. You can assign the following security access levels: • Public • Controlled • Restricted • Confidential • Private If you assign Public then all users can see and use the object. If you assign Restricted, then only users with the user profile of Restricted or higher can see and use the object.
Can be used in Result	When selected, the object can be used in a query.
Can be used in Condition	When selected, the object can be used to set in a condition.
Can be used in Sort	When selected, returned values can be sorted.

Property	Description
Database Format	Option only available for date objects. By default, the date format for the object is defined in the Regional Settings Properties dialog box of the MS-Windows Control Panel. You can modify this to use the target database format for storing dates. For example, the date format could be US format, or European format. For information on modifying this value, see the section Defining an object format [page 269].

7.6.6.1 Defining object security and user rights

To define security and user rights for an object:

1. Double click an object.
   The Edit Properties box for the object appears.
2. Click the Advanced tab.
   The Advanced page appears.
3. Select a security access level from the Security Access Level drop down list box.
4. Select one or more check boxes in the Can Be Used In group box.
5. Type a date format in the database Format text box, if you want to modify the default date format.
6. Click OK.

7.6.7 Defining index awareness

The Keys tab allows you to define index awareness for an object. Index awareness is the ability to take advantage of the indexes on key columns to speed data retrieval.

The objects that you create with the universe design tool are based on database columns that are meaningful to an end user. For example, a Customer object retrieves the field that contains the customer name. In this situation the customer table typically has a primary key (for example an integer) that is not meaningful to the end user, but which is very important for database performance. When you set up index awareness in the universe design tool you tell the tool which database columns are primary and foreign keys. This can have a dramatic effect on query performance in the following ways:

- The universe design tool can take advantage of the indexes on key columns to speed data retrieval.
- The universe design tool can generate SQL that filters in the most efficient way. This is particularly important in a star schema database. If you build a query that involves filtering on a value in a dimension table, the universe design tool can apply the filter directly on the fact table by using the dimension table foreign key. This eliminates unnecessary and costly joins to dimension tables.

The universe design tool does not ignore duplicates with index awareness. If two customers have the same name, the universe design tool will retrieve one only unless it is aware that each customer has a separate primary key.

Example: Finding customers in a list of cities

In this example you build a report on the Island Resorts Marketing Universe that returns revenue by customer for customers in Houston, Dallas, San Francisco, San Diego or Los Angeles. To do this you drag the Customer and Sales Revenue objects into the Result Objects pane in the Query pane, then drag the City object to the Conditions pane and restrict the city to the list above.

Without index awareness, the universe design tool generates the following SQL:

```
SELECT
  Customer.last_name,
  sum(Invoice_Line.days * Invoice_Line.nb_guests * Service.price)
FROM
  Customer,
  Invoice_Line,
  Service,
  City,
  Sales
WHERE
  ( City.city_id=Customer.city_id  )
  AND  ( Customer.cust_id=Sales.cust_id  )
  AND  ( Sales.inv_id=Invoice_Line.inv_id  )
  AND  ( Invoice_Line.service_id=Service.service_id  )
  AND  (
  City.city  IN  ('Houston', 'Dallas', 'San Francisco', 'Los Angeles', 'San
Diego')
  )
GROUP BY
  Customer.last_name
```

In this case the tool has created a join to the City table in order to restrict the cities retrieved.

With index awareness, you tell the universe design tool that `city_id` is the primary key of the City table and that it also appears in the Customer table as a foreign key. Using this information, the tool can restrict the cities without joining to the City table. The SQL is as follows:

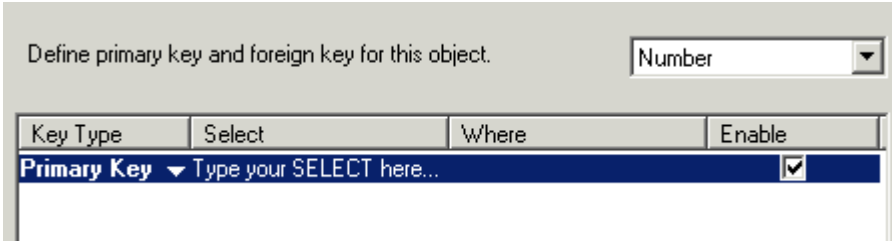
```
SELECT
  Customer.last_name,
  sum(Invoice_Line.days * Invoice_Line.nb_guests * Service.price)
FROM
  Customer,
  Invoice_Line,
  Service,
  Sales
WHERE
  ( Customer.cust_id=Sales.cust_id  )
  AND  ( Sales.inv_id=Invoice_Line.inv_id  )
  AND  ( Invoice_Line.service_id=Service.service_id  )
  AND  (
  Customer.city_id  IN  (10, 11, 12, 13, 14)
  )
GROUP BY
  Customer.last_name
```

In this case the tool is able to generate SQL that restricts the cities simply by filtering the values of the `city_id` foreign key.

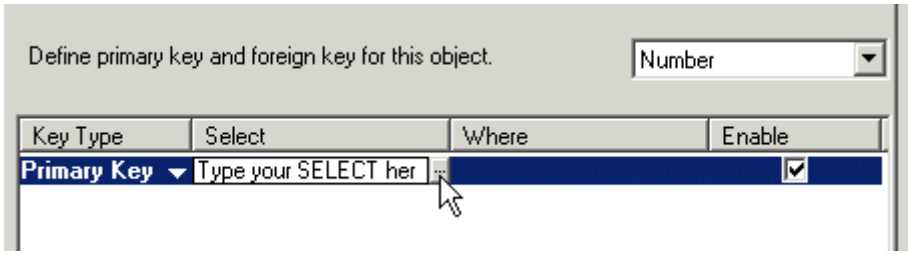
7.6.7.1 Setting up primary key index awareness

To set up primary key index awareness:

1. Right-click the object on which you want to set up index awareness and select *Object Properties* from the menu.
   The Edit Properties Of dialog box appears.
2. Click the *Keys* tab.
3. Click *Insert*.
   A Primary Key line is inserted as shown below in the Keys page.

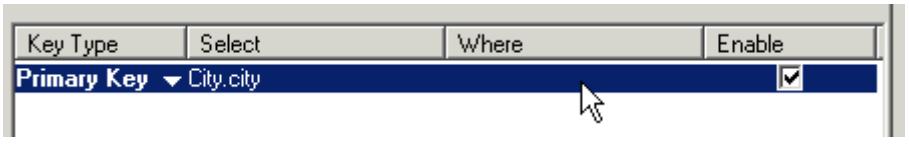


4. Do the following actions in to create key awareness for the primary key:
   - Select Primary in the Key Type list.
     Click the ... button in the Select field to open the SQL editing dialog box.



   The SQL Editor appears.
   - Use the SQL Editor to build the primary key SQL SELECT clause or type it directly. For example, for the City object above, the primary key SQL is `City.city_id`

   For more information on the SQL Editor, see Using the SQL Editor [page 269].
   - Select the primary key data type from the drop-down list of key types.
5. If you want to add a WHERE clause, do the following:
   - Click within the line, under the Where column as shown below:



   - Click the ... button in the Where field to open the SQL editing dialog box.

   The SQL Editor appears.
   - Use the SQL Editor to build the primary key SQL WHERE clause or type it directly. There is no Where clause in the example above.

- Select Number from the drop-down list of key types.

6. Select *Enabled*.
7. Click OK.

> **i Note**
>
> To add more than one column for primary key, you can concatenate multiple columns to define Primary key. These columns should belong to the same table and have same datatype.
>
> e.g in sample database "`club.mdb`" Resort table has a multi-column primary key based on `Country_id` and `Resort_id`.
>
> Therefore to define index awareness on <resort> object user can concatenate "`Country_id`" and "`Resort_id`" to define primary key:
>
> `Resort.country_id & Resort.resort_id`
>
> & is MS Access concatenation operator.

7.6.7.2 Setting up foreign key awareness

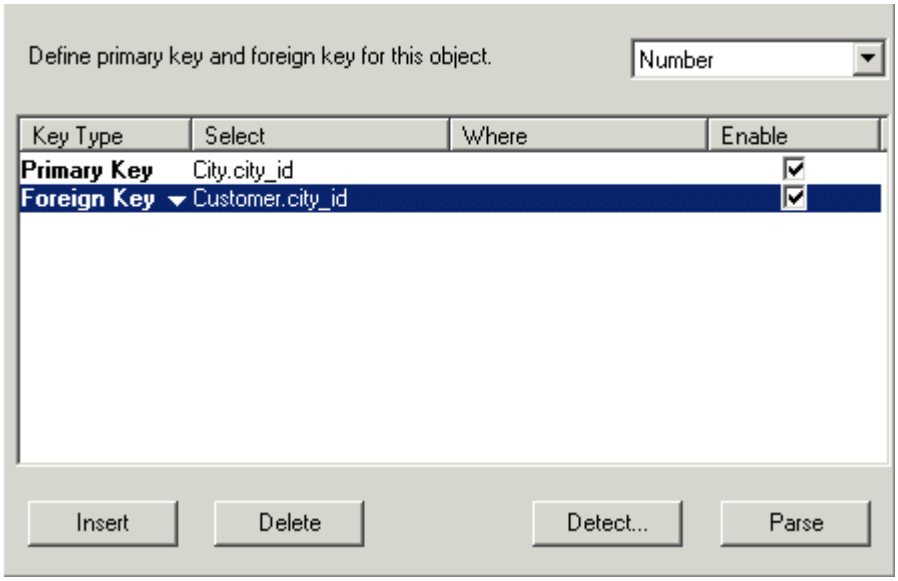
To set up foreign key awareness:

1. Right-click the object on which you want to set up index awareness
   Select Object Properties from the menu.
   The Edit Properties Of dialog box appears.
2. Click the *Keys* tab.
3. Click *Insert*.
   A key line is inserted in the Keys page.
4. Do the following to create key awareness for the foreign key:
   - Select Foreign Key in the Key Type list.
   - Click the ... button in the Select field to open the SQL editing dialog box.

   The SQL Editor appears.
   - Use the SQL Editor to build the foreign key SQL SELECT clause or type it directly.
   - Select the foreign key data type from the drop-down list of key types.
5. Repeat steps 3 and 4 for all columns that make up the foreign key.
6. If you want to add a WHERE clause, do the following:
   - Click in the highlighted line, under the Where column.
   - Click the ... button in the Where field to open the SQL edit dialog box.

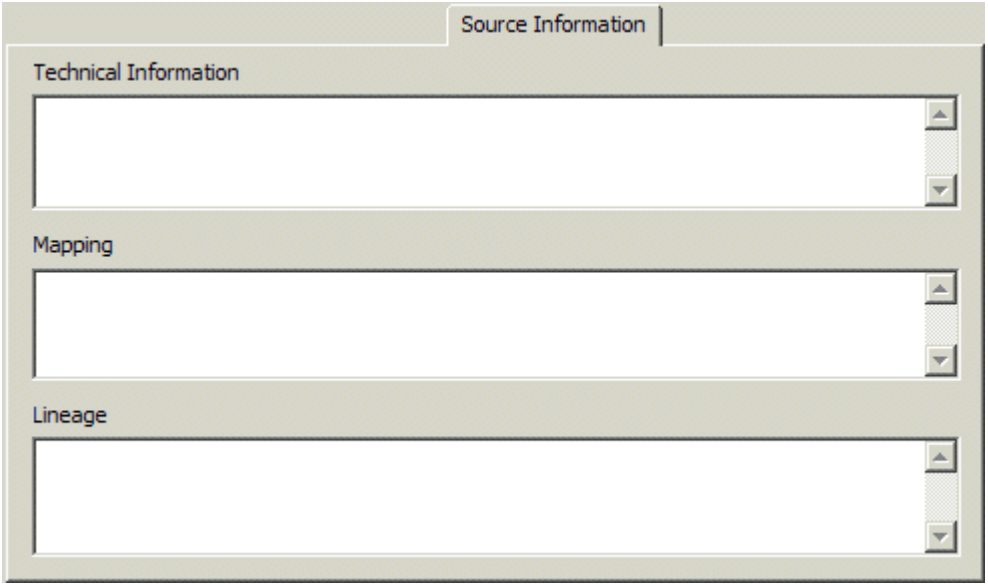
   The SQL Editor appears.
   - Use the SQL Editor to build the foreign key SQL WHERE clause, or type it directly.
   - Select Number from the drop-down list of key types.
7. Select *Enabled*.
8. Repeat the steps above for all columns in the foreign key.

For the example in the *Defining Index Awareness* section, the *Keys* tab should look like this:



7.6.8 Source Information

The Source Information page is used by universes generated from Data Integrator. The Sources Information tab is shown below:



For universes generated from Data Integrator, Technical descriptions and formulas used to calculate target tables from source tables are displayed in this tab. This information is available to Web Intelligence users.

You can specify the following types of information in the Source Information tab:

- Technical description: Technical descriptions that are available in universes generated from Data Integrator.
- Mapping information: The mapping applied within Data Integrator between the source tables and the target tables. The goal is not to provide the expression of the mapping, but to display it as a descriptive comment to inform the user of the source columns used in the object definition.
- Data Lineage information: List of source columns involved in a target column. This information facilitates the impact analysis through Data Integrator and Web Intelligence reports.

7.6.9 Using the SQL editor to define an object

You can use an SQL editor to help you define the Select statement or a Where clause for an object, and to insert MDX operators and functions for OLAP universe objects. The SQL Editor is a graphical editor that lists tables, columns, objects, operators, and functions in tree views. You can double click any listed structure to insert it into the Select or Where boxes.

You have the following editing options available in the SQL Editor:

Edit options	Description
Tables and columns	All tables and their respective columns that appear in the Structure pane. **i Note** This option is only available for relational universes and is not available for OLAP universes.
Classes and objects	All classes and their respective objects that appear in the Universe pane.
Operators	Operators available to combine SQL structures in a Select statement, or to set conditions in a Where clause.
Functions	• Database functions, for example number, character, and date functions. • @Functions specific to Business Objects products. Available functions are listed under the Functions entry in the parameters (.PRM) file for the target database. There is a .PRM file for each supported database. They are stored in the Data Access folder in the BusinessObjects path. You can add or modify the available functions by editing the .PRM file. Editing .PRM files is described in the Data Access Guide.
Show object SQL	When selected, the SQL syntax is displayed for the objects that appear in the Select, or Where boxes.
Parse	When clicked, parses the syntax. If the syntax is not valid, a message box appears describing the problem.
Description	Displays a description of a selected object or function.

Related Information

About MDX functions for cube queries [page 422]
Using the SQL Editor [page 269]

7.6.9.1 Setting aggregate projection for a measure

When you create a measure you must specify the way the aggregate function will be projected onto a report.

Returned values for a measure object are aggregated at two levels of the query process:

- Query level. Data is aggregated using the inferred SELECT statement.
- Microcube to block level. When data is projected from the microcube to the block in a report. This projection function of measures allows local aggregation in the microcube.

> **i Note**
>
> A microcube is a conceptual way to present the data returned by a query before it is projected onto a report. It represents the returned values held in memory by a Business Objects reporting product. The block level is the 2 dimensional report that a user creates with the returned data. A user can choose to use all, or only some of the data held in the microcube to create a report. A user can also do aggregate functions on the returned values in the microcube (local aggregation) to create new values on a report.

The two levels of aggregation fit into the query process as follows:

- User creates a query in Web Intelligence.
- Web Intelligence infers the SQL from the query and sends a SELECT statement to the target database.
- The data is returned to the microcube. This is the first aggregation level.
- The microcube projects the aggregated data onto the report. Data is split out in the Query pane requiring aggregation to lower levels. This is the second aggregation level.

When you initially make a query the result set of the Select statement is stored in the microcube, and all data then held in the microcube is projected into a block. As data is projected from the lowest level held in the microcube no projection aggregation is taking place.

However, when you use the Query pane to project only partial data from the microcube, aggregation is required to show measure values at a higher level.

For example, in the previous example, if you do not project the year data into the block, the three rows related to Year need to be reduced to one row to show the overall Sales Revenue for that resort, so a sum aggregation is used.

You set projection aggregation on the *Properties* page of the *Edit Properties* sheet for a measure (right-click Object > Object Properties > Properties).

Projection aggregation is different from SELECT aggregation.

7.6.9.2 Using the SQL Editor

You can use the SQL editor to insert SQL and MDX expressions in an objects definition. You can also right-click in the SQL statement to select *New @Prompt* to insert an @Prompt expression in the SQL or select *Edit @Prompt* to edit an existing @Prompt expression. This opens the @Prompt editor.

To use the SQL Editor:

1. Double click an object.
   The Edit Properties dialog box for the object appears.
2. Click the >> button next to the Select or Where box.
   The Edit Select Statement or Edit Where Clause dialog box appears.
3. Click in the Select statement or Where clause at the position where you want to add syntax for a structure. If the box is empty, click anywhere in the box. The cursor automatically appears at the top left corner of the box.
4. Expand table nodes to display columns.
5. Double click a column to insert the column definition in the Select statement or Where clause.

   > → Tip
   >
   > To select one or more values from a list of values for a selected column, right click the column and select List of Values.

6. Expand class nodes to display objects.
7. Double click an object to insert a @Select or @Where function in the Select statement or Where clause. These functions direct the current object to use the Select statement or Where clause of a selected object. For more information on using @Functions, see the section Using @Functions in the SQL of an object [page 344].
8. Double click an operator to insert the operator in the edit box.
9. Expand function nodes to display available functions.
10. Double click a function to insert the function in the edit box.
11. Click the Parse button to validate the syntax.
12. Click OK.

7.6.10 Defining an object format

You can define a format for the data values of a selected object. The format applies to the related data values displayed in the cells of Web Intelligence reports.

The tabs of the Object Format dialog box include settings for numbers, alignment, font, border, and shading.

For example, you can display an integer in a format such as $1,000 rather than the default 1,000.00. Or you can apply a color, such as red, to critical data values.

Number, Currency, Scientific and Percentage categories apply only to objects and variables with a numeric type, and the Date/Time category applies only to those with a date type.

Information about formats is exported and imported with the universe.

You can use the Remove Object Format command to remove any format you defined.

Related Information

About MDX functions for cube queries [page 422]

7.6.10.1 Modifying an object format

To modify an object format:

1. Right click an object
2. Select Object Format from the contextual menu.
   The Object Format sheet appears.
3. Click a format tab and select or type a format for the object.
4. Click OK.

7.6.10.2 Removing an object format

You can remove a format for an object at any time.

To remove an object format:

- Select an object and then select File > Remove Format.
  Or
- Right click an object and select Remove Format from the contextual menu.

7.6.11 Viewing the table used in an object definition

You can view the table in the Structure pane that is used in an object definition from the Universe pane. This can be useful to quickly identify a table used by an object when object names do not easily indicate a specific table.

7.6.11.1 Viewing the table used by an object

To view the table used by an object:

1. Right click an object in the Universe pane.
   A contextual menu appears.
2. Select View Associated table from the contextual menu.
   The associated table is highlighted in the Structure pane.

7.6.12 Defining a dimension

A dimension is an object that is a focus of analysis in a query. A dimension maps to one or more columns or functions in the database that are key to a query. For example Country, Sales Person, Products, or Sales Line.

Dimension is the default qualification at object creation. You can change the qualification to dimension at any time.

To define a dimension object:

1. Double click an object.
   The Edit Properties dialog box for the object appears.
2. Click the Properties tab.
   The Properties page appears.
3. Select the Dimension radio button in the Qualification group box.
4. Click OK.

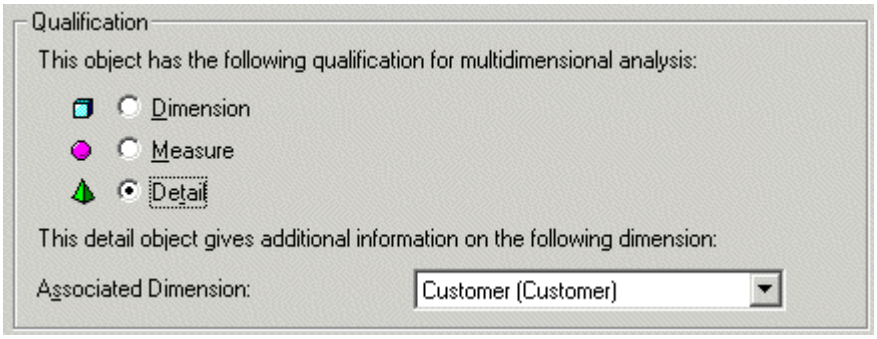
7.6.13 Defining a detail

A detail provides descriptive data about a dimension. A detail is always attached to a dimension. It maps to one or more columns or functions in the database that provide detailed information related to a dimension.

You define a detail object by selecting Detail as the qualification for an object, and specifying the dimension attached to the detail.

To define a detail object:

1. Double click an object.
   The Edit Properties dialog box for the object appears.
2. Click the Properties tab.
   The Properties page appears.
3. Select the Detail radio button in the Qualification group box.
   An Associated Dimension drop down list box appears listing all the dimension objects in the universe.
4. Select a dimension from the drop-down list box. The detail describes a quality or property of this dimension.



5. Click OK.

7.6.14 Defining a measure

You can define a measure object by selecting Measure as the qualification for an object. Measures are very flexible objects as they are dynamic. The returned values for a measure object vary depending on the dimension and detail objects used with it in the query. For example; a measure Sales Revenue returns different values when used with a Country object in one query, and then with Region and Country objects in a separate query.

As measure objects are more complex and powerful than dimensions and details, they are discussed in more depth in the following sections.

7.6.14.1 What type of information does a measure return

A measure object returns numeric information. You create a measure by using aggregate functions. The five most common aggregate functions are the following:

- Sum
- Count
- Average
- Minimum
- Maximum

7.6.14.2 How are measures different from dimensions and details

Measures differ from dimensions and details in the following ways:

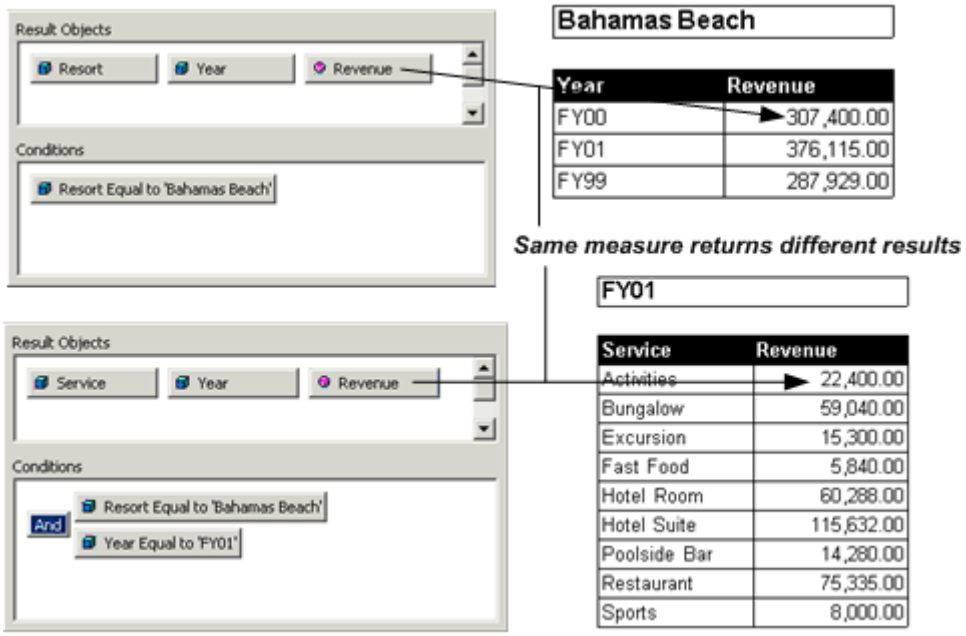
- Measures are dynamic
- Measures can project aggregates

Both these properties are described as follows:

7.6.14.3 How do measures behave dynamically?

Returned values for a measure object vary depending on the dimension and detail objects used with the measure object in a query.

The following example shows the same Revenue measure object used in two separate queries with different dimensions, resulting in the measure returning different values.



7.6.14.4 Measures infer a Group By clause

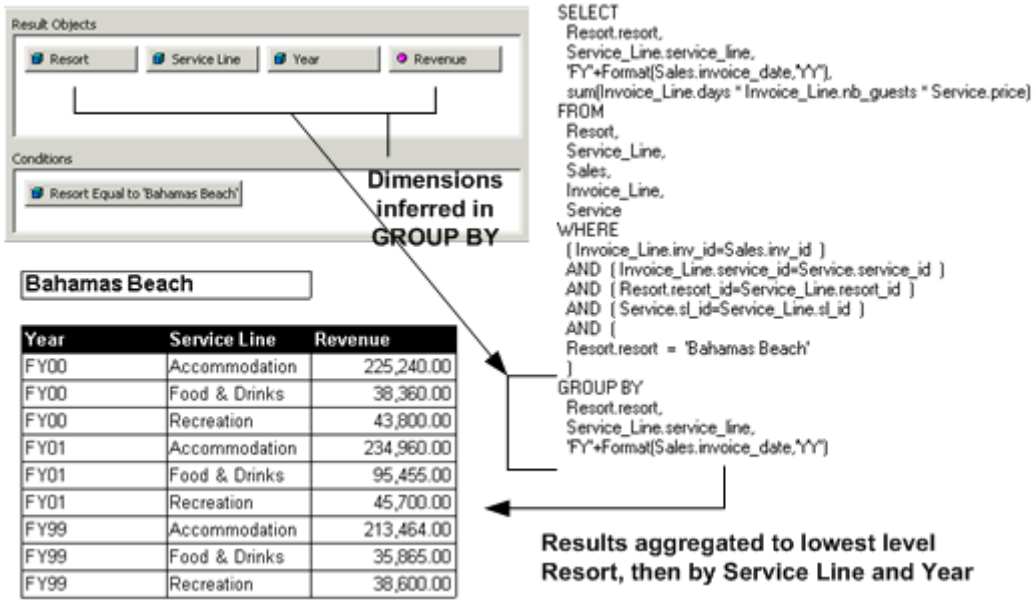
When you run a query that includes a measure object with other types of objects, a Group By clause is automatically inferred in the Select statement.

The inference of the Group By clause depends on the following SQL rule:

> If the Select clause line contains an aggregate, everything outside of that aggregate in the clause must also appear in the Group By clause.

Based on this rule, any dimension or detail used in the same query as a measure object will always be included in an automatically inferred Group By clause. To ensure that the query returns correct results, dimension and detail objects must NOT contain aggregates.

The following example shows that the Resort, Service Line, and Year dimension objects are all inferred in the Select clause and in the Group By clause.



> **i Note**
>
> If a query contains only measure objects, no Group By clause is inferred.

7.6.14.5 Setting aggregate projection for a measure

When you create a measure you must specify the way the aggregate function will be projected onto a report.

Returned values for a measure object are aggregated at two levels of the query process:

- Query level. Data is aggregated using the inferred SELECT statement.
- Microcube to block level. When data is projected from the microcube to the block in a report. This projection function of measures allows local aggregation in the microcube.

> **i Note**
>
> A microcube is a conceptual way to present the data returned by a query before it is projected onto a report. It represents the returned values held in memory by a Business Objects reporting product. The block level is the 2 dimensional report that a user creates with the returned data. A user can choose to use all, or only some of the data held in the microcube to create a report. A user can also do aggregate functions on the returned values in the microcube (local aggregation) to create new values on a report.

The two levels of aggregation fit into the query process as follows:

- User creates a query in Web Intelligence.
- Web Intelligence infers the SQL from the query and sends a SELECT statement to the target database.
- The data is returned to the microcube. This is the first aggregation level.
- The microcube projects the aggregated data onto the report. Data is split out in the Query pane requiring aggregation to lower levels. This is the second aggregation level.

When you initially make a query the result set of the Select statement is stored in the microcube, and all data then held in the microcube is projected into a block. As data is projected from the lowest level held in the microcube no projection aggregation is taking place.

However, when you use the Query pane to project only partial data from the microcube, aggregation is required to show measure values at a higher level.

For example, in the previous example, if you do not project the year data into the block, the three rows related to Year need to be reduced to one row to show the overall Sales Revenue for that resort, so a sum aggregation is used.

You set projection aggregation on the *Properties* page of the *Edit Properties* sheet for a measure (right-click Object > Object Properties > Properties).

Projection aggregation is different from SELECT aggregation.

7.6.14.5.1 Database delegated projection function

In a universe, any measure can hold a projection function (*Sum*, *Min*, *Max*, *Count*, and *Avg*). The projection function is used to aggregate the measure locally in Web Intelligence when the number of dimensions displayed in a report is smaller than the number of dimensions in the query result set.

Non-additive measures, such as ratio, average, and weight, can only be shown at the same aggregation level as the query result set. Therefore, non-additive measures generally have their projection function set to *None* in the universe.

The projection function *Database delegated* allows you to delegate the aggregation of a non-additive measure to the database server. These are called smart measures in Web Intelligence. A smart measure has its projection function set to *Database delegated* on the properties page of the object properties. For more information about how these and other functions are used in Web Intelligence , please refer to section *Calculating values with Smart Measures* in the document *Using Functions, Formulas and Calculations in Web Intelligence*.

> **i Note**
>
> For OLAP universes based on MSAS and Essbase data sources, all measures are created in the universe with the projection function set to *Database delegated* by default.

> **i Note**
>
> Please be aware of the following limitation when using a smart measure based on a measure that has aggregate aware set: It is strongly recommended to ensure that the aggregate tables used in the measure definition have consistent data (the aggregate values are accurate regarding detail values) otherwise the smart measures can produce inconsistent data. For example, if a year aggregate table and a day aggregate table are used for a smart measure, the year aggregate table is consistent with the day aggregate table for complete years, but for the current year, the year table can be empty while the day table is accurate on a daily basis. In this case, a report that uses smart measures based on the current year and daily tables can give incoherent results.

Example: Smart measure

In this example, a query contains two dimensions: Country and Region, and three measures: Order Amount, Delivered Amount, and % of Delivered Amount.

L01 Region	Amount Delivered	Order Quantity	% Delivered
Reg1	497,318,880	497,332,680	99.997
Reg2	199,463,776	199,466,536	99.998
Reg3	198,927,552	198,933,072	99.997
		Sum:	299.992

The sum for % Delivered is incorrect because it is a summation of the % Delivered column.

If this measure has a projection function set to *Database delegated* in the universe, when the user refreshes the report, Web Intelligence connects to the database to calculate the correct value.

L01 Region	Amount Delivered	Order Quantity	% Delivered
Reg1	497,318,880	497,332,680	99.997
Reg2	199,463,776	199,466,536	99.998
Reg3	198,927,552	198,933,072	99.997
		Sum:	299.992
		Total:	99.997

> **Note**
>
> The some functions such as the ratio function (Average) must be used with caution. When calculating an average from a column, the behavior of this function can be unexpected when it is not configured correctly.
>
> For example, the SQL function `sum(Shop_facts.Margin)/sum(Shop.facts.Quantity_sold)` can have unexpected results. When incorrectly configured, it will calculate the average for each cell and return the sum of those averages. To correct this behavior, the parametrization of the function must be performed as follows:
>
> 1. Go to the *Edit Properties* option for the function.
> 2. For the option *Choose how this measure will be projected when aggregated*, select the Function *Db delegated* from the Function dropdown list.
> 3. Save your changes.

Related Information

Setting aggregate projection for a measure [page 419]

7.6.14.6 Creating a measure

To create a measure:

1. Double click an object.
   The Edit Properties dialog box for the object appears.
2. Click the Properties tab.
   The Properties page appears.
3. Select the Measure radio button in the Qualification group box.
   A Function drop down list box appears listing aggregate functions.
4. Select a function.
5. Click OK.

Related Information

Setting aggregate projection for a measure [page 419]
Database delegated projection function [page 275]

7.6.15 Defining restrictions for an object

A restriction is a condition in SQL that sets criteria to limit the data returned by a query.

You define restrictions on objects to limit the data available to users. Your reasons for limiting user access to data should be based on the data requirements of the target user. A user may not need to have access to all the values returned by an object. You might also want to restrict user access to certain values for security reasons.

You can define two types of restrictions in a universe:

Restriction type	Description
Forced	Restriction defined in the Where clause for an object. It cannot be accessed by users and so cannot be overridden in Web Intelligence.
Optional	Restriction defined in special condition objects that users can choose to use or not use in a query. A condition object is a predefined Where clause that can be inserted into the Select statement inferred by objects in the Query pane.

> **i Note**
>
> In Web Intelligence, users can apply conditions in the Query pane. As the universe designer, you should avoid creating optional restrictions that are simple to apply at the user level. Users can create these conditions themselves when necessary.

7.6.15.1 Defining a Where clause for an object

You apply a further restriction on an object by adding a condition in the *Where* box from the *Definition* page of the *Edit Properties* dialog box for an object.

You can define the condition at object creation, or add it to the object definition at any time.

In a universe, the Where clause in an SQL statement can be used in two ways to restrict the number of rows that are returned by a query.

- A WHERE clause is automatically inferred in the SELECT statement for an object by joins linking tables in the schema. Joins are usually based on equality between tables. They prevent Cartesian products being created by restricting the data returned from joined tables.
- You add a condition in the WHERE clause for an object. This is an additional condition to the existing WHERE clause inferred by joins. You define a WHERE clause to further restrict the data that is returned in a query, for example when you want to limit users to queries on a sub-set of the data.

Example: Modifying the default (join only) Where clause for an object

The report below is an unrestricted block containing data for sales people from all countries:

Sales Person	Country of origin
Barrot	France
Carlin	France
Edwood	UK
Fischer	Germany
Galagers	US
Ishimoto	Japan
Nagata	Japan

The SQL for this query appears below. The Where clause contains only restrictions inferred by the joins between the tables Customer, City, Region, and Sales_Person.

```
SELECT
  Sales_Person.sales_person, Country.country
FROM
  Sales_Person,
  Country,
  Region,
  City,
  Customer
WHERE
  ( City.city_id=Customer.city_id  )
  AND  ( City.region_id=Region.region_id  )
  AND  ( Country.country_id=Region.country_id  )
  AND  ( Sales_Person.sales_id=Customer.sales_id  )
```

If you want to restrict users to see only returned values specific to France, you can add a condition to the Where clause of the Country object. The following report shows sales people for France only:

Sales Person	Country of origin
Barrot	France
Carlin	France

The SQL for the query is as follows:

```
SELECT
  Sales_Person.sales_person,
  Country.country
FROM
  Sales_Person,
  Country,
  Region,
  City,
  Customer
WHERE
  ( City.city_id=Customer.city_id  )
  AND  ( City.region_id=Region.region_id  )
  AND  ( Country.country_id=Region.country_id  )
  AND  ( Sales_Person.sales_id=Customer.sales_id  )
  AND  ( Country.country = 'France'  )
```

The WHERE clause has an additional line. This is the restriction that you have added to the WHERE clause of the Country object.

> **i** Note
>
> Apart from self restricting joins, you should not create a join in a WHERE clause. A join in a WHERE clause is not considered by Detect Contexts (automatic context detection) or aggregate aware incompatibility detection. You should ensure that all joins are visible in the *Structure* pane. This ensures that all joins are available to the universe design tool automatic detection tools.

7.6.15.2 Defining a Where clause

To define a Where clause:

1. Double click an object.
   The Edit Properties dialog box opens to the Definition page.
2. Type the syntax directly into the Where clause text box.
   Or
   Click the >> Button next to the Where box to open the Where clause editor.
3. Double click columns, objects, operators, or functions that appear in the SQL structures and features lists.

   > → Tip
   >
   > You can select values for a Where clause as follows: Right click a column in the Tables and Columns list. Select View Values. A list of all values for the column appear. You can select one or more values to insert in the Where clause, for example when using the In operator.
4. Click OK to close the editor.
   The Where clause for the Country object is shown below. It restricts the values for Country to France only.



5. Click OK.

7.6.15.3 Problems using Where clauses

Where clauses are a useful way to restrict data, but they must be used carefully in a universe to avoid the following problems:

Problem	Description	Solution
Proliferation of similar objects.	If you restrict data for an object by creating several objects, each inferring a Where clause for one part of the data, you can end up with multiple objects with similar names. For example, French clients, US clients, and Japanese clients. This can be confusing for users to see multiple objects that appear similar.	Create condition objects for each restriction.
Difficulty creating hierarchies.	If you have multiple objects inferring Where clauses on the same data, it will be difficult for users to construct a logical default hierarchy to use for drill down.	Create condition objects for each restriction.

Problem	Description	Solution
Confusion between object name and applied restriction.	Unless your objects are very precisely named, then a restriction may not be obvious to the user simply from the name of the object. A user can see the Where clause by viewing the SQL for a query, but not all users will view the SQL before running a query.	• Create condition objects for each restriction. • Name each object appropriately.
Conflict between Where clauses.	If two or more similarly restricted objects are included in the same query, the conflict between the Where clauses will result in no data being returned.	Create condition objects for each restriction, and ensure that users do a union or synchronization of the queries at the report level.

Creating condition objects will solve the multiple objects, hierarchy difficulties, and object name confusion problems.

The conflict between Where clauses can be solved by creating condition objects and ensuring that users know that they must join the queries using a UNION or SYNCHRONIZE operator at the report level.

Given the potential problems with Where clauses defined in an object definition, you should avoid using them, and where possible create condition objects which, when used correctly can avoid the problems with hard coded Where clauses.

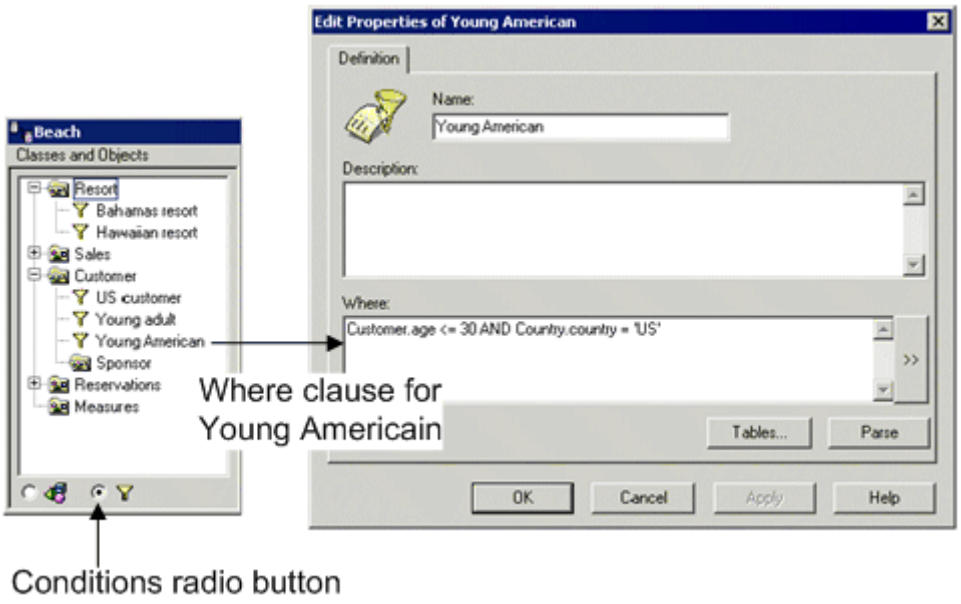
> **i** Note
>
> Apart from self restricting joins, you should not create a join in a condition object. A join in a condition object is the equivalent to creating a join in a reusable Where clause, and so is not considered by Detect Contexts (automatic context detection) or aggregate aware incompatibility detection. You should ensure that all joins are visible in the Structure pane. This ensures that all joins are available to the automatic detection tools.

7.6.16 Defining condition objects

A condition object is a predefined Where clause that can be inserted into the Select statement inferred by objects in the Query pane.

Condition objects are stored in the Conditions view of the Universe pane. You access the conditions view by clicking the Conditions radio button at the right bottom of the universe pane.

The condition objects for the Beach universe and the Where clause that the Young American condition infers are shown below.



7.6.16.1 Advantages and restrictions for using condition objects

Using condition objects has the following advantages:

- Useful for complex or frequently used conditions.
- Gives users the choice of applying the condition.
- No need for multiple objects.
- Condition objects do not change the view of the classes and objects in the Universe pane.

> **i** Note
>
> You may need to direct users to use the condition objects view of the Universe pane.

The only disadvantages for using condition objects is that you may want to force a condition on users to restrict their access to part of the data set. In this case you need to define a Where clause in the object definition.

7.6.16.2 Condition objects do not solve conflicting Where clauses

Using condition objects does not solve the problem of conflicting Where clauses returning an empty data set. If a user runs a query that includes two condition objects that access the same data, the two conditions are combined with the AND operator, so the two conditions are not met, and no data is returned. This problem can be solved at the report level by users creating two queries, one for each condition object and then combining the queries.

7.6.16.3 Mandatory filters

There are two types of mandatory filter:

- Universe: A universe mandatory filter has no dependency on the class to which it belongs. A universe mandatory filter is included in the query independently of the objects (dimensions, measures, and details) that are included in the query.
  Most SAP Business Warehouse (BW) variables are created as universe mandatory filters when generating OLAP universes on SAP BW.
- Class: Class mandatory filters appear only if an item of the class of the object is used in the query.
  A class mandatory filter is triggered when users:
  - Add an object (dimension, measure, or detail) to the *Result* pane of the *Query Panel* in Web Intelligence.
  - Add a universe pre-defined filter to the *Filter* pane of the *Query panel*, even if no object that belongs to the same class has been selected in the Result pane.
  - Create a filter with an object (dimension, measure, or detail) that belongs to a class with a mandatory filter.

A mandatory filter can have default values or be associated with a list of values.

A mandatory filter is hidden and cannot be selected in the *Query Panel* in Web Intelligence. In the universe design tool, when you set a filter as mandatory in the query, then it is hidden automatically and the *Show Item(s)* command is disabled. If you uncheck the mandatory option, the filter is no longer hidden. The *Hide Item(s)* command is enabled.

An end-user query can include more than one mandatory filter. By default, all mandatory filters are joined in the query with the AND operator.

All sub-classes inherit the mandatory filters from the parent class. Note, however:

- An object (dimension, measure, detail) that references another object with the @Select function does not inherit the class mandatory filter of the referenced object.
- A WHERE clause of an object that references another object where clause with the @Where function does not inherit the class mandatory filter of the referenced object.
- A pre-defined filter that references another pre-defined filter or an object where clause with the @Where function does not inherit the class mandatory filter of the referenced object.

Example: Mandatory filter in an OLAP universe

The following filter (shown in XML code) authenticates the code entered by a user in a prompt.

```
<FILTER KEY="[BCOMUSI]">
    <CONDITION OPERATORCONDITION="InList">
        <CONSTANT TECH_NAME=
        "@Prompt('CO_CODE Char User MultiSingle Man Def',
        'A','Company code\Lov[BCOMUSI]Base',
        multi,primary_key)"/>
    </CONDITION>
</FILTER>
```

Related Information

Mandatory filters examples [page 284]
Mandatory filters and list of values [page 284]

7.6.16.3.1 Mandatory filters examples

The following examples show how universe mandatory filters can be used:

To verify the Login entered by a user with a Login stored in a table:

```
1 = (Select 1 from Club.dbo.Login
where Login = @Variable('BOUSER')
AND Password = @Prompt('Password?','A',,mono,free) )
```

To limit the use of a universe to hours between 9 am and 6 pm:

```
1 = (select 1
where datepart(HH,getdate()) between 9 and 18)
```

The following is an example of a class mandatory filter:

Defined in a class containing Country/Region/City/Customer, limit the query to sales information for a particular period. Prompt the user for the period.

```
Club.dbo.Customer.cust_id in
(Select cust_id from Club.dbo.Sales
where @Select(Sales\Year) in
@Prompt('Sales Periods?','A',
'Sales\Year',multi,constrained))
```

Related Information

Mandatory filters [page 283]

7.6.16.4 Mandatory filters and list of values

Mandatory filters can be associated with a list of values. To associate a list of values, you must explicitly select the list of value options in the object properties page of the object to which the filter applies.

A universe mandatory filter can be associated with a cascading list of values.

A class mandatory filter can be associated with a cascading list of values if at least one object in the corresponding class is part of the cascading list of values. This is true even if the cascading list of values groups objects from different classes.

Recommendations

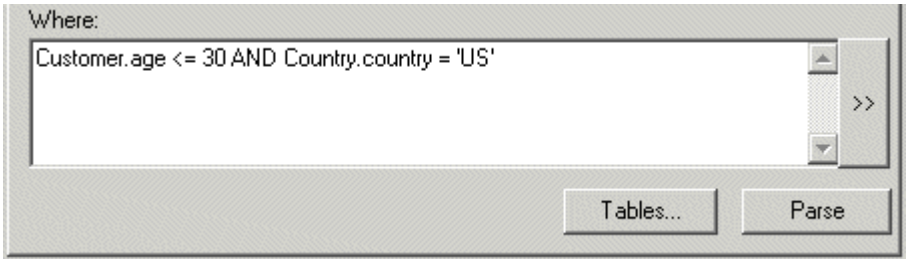
Generate mandatory filters only on the top level of a cascading list of values.

Do not associate a cascading list of values with a mandatory filter that contains a prompt. Web Intelligence does not support prompts in cascading lists of values.

7.6.16.5 Creating a condition object

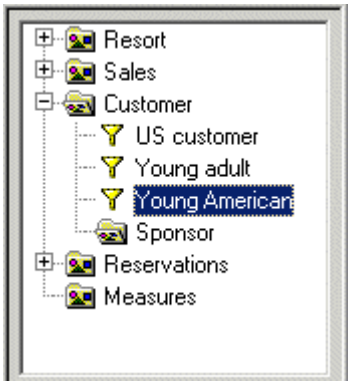
To create a condition object:

1. Click the *Conditions* radio button at the bottom right of the *Universe* pane.
   The *Conditions* view of the *Universe* pane appears. It contains a tree view of all the classes in the universe.
2. Right-click a class and select *Insert Condition* from the contextual menu.
   Or
   Click a class and click the *Insert Condition* button.
   An *Edit Properties* dialog box appears. A default name appears in the *Name* box. The *Where* box is empty.
3. Type a name for the condition.
4. Type the WHERE clause syntax directly into the *Where* clause box.
   Or
   Click the >> button next to the *Where* clause box to open the *Where* clause editor.
5. Double-click columns, objects, operators, or functions that appear in the *SQL structures* and *features* lists.
6. Click *OK* to close the editor.
   The definition for a condition called Young American is shown below. It restricts the returned values to American customers less than or equal to 30 years old.



7. Click *Parse* to verify the syntax.
8. To define the filter as a compulsory filter, select the *Compulsory filter* check box.
   By default, a compulsory filter applies to the class, and does not apply to a List of Values.
9. Select the radio button to define the compulsory filter as applying to the class or to the universe.
10. To apply the compulsory filter to a List of Values, select the *Apply on List of Values* check box.
11. Click *OK*.
    The new condition object appears in the *Conditions* view of the *Universe* pane.

> **i** Note
>
> To edit the condition object using the predefined filter editor, click >>.



Related Information

Mandatory filters [page 283]
Mandatory filters and list of values [page 284]
Predefined conditions in OLAP universes [page 423]
Optional prompts in OLAP universes [page 428]

7.6.16.6 Using condition objects in the same query

If you have two condition objects defined for the same object, and both are used in the same query, no data is returned, as the two WHERE clauses create a false condition. Where possible you should avoid hard coding WHERE clauses in the definition of an object, but also when you use condition objects, users need to be aware of the potential problems.

Users can solve the problem of returning an empty data set by joining two queries, one query for each condition object.

> **i** Note
>
> To avoid Web Intelligence users combining two condition objects in the same query, you can include in the description for a condition object 'X' that it should not be used with object 'Y'.

7.6.16.7 Why do multiple Where clauses return an empty data set?

When you add a Where clause to the definition of an object, the restriction is added to the restrictions set by the joins using the AND operator. If you combine two objects in a query, both applying a restriction on the same data set, then the two Where clauses are combined in successive AND clauses. The result of such a query is that no data will satisfy both conditions, and no data is returned.

For example, a user wants to know the services that are available at the Bahamas and Hawaiian Club hotel resorts. The following query is run using the condition objects for Bahamas resort and Hawaiian Resort:

The SQL for this query is as follows:

```
SELECT Service.service, Resort.resort FROM Service, Resort, Service_Line WHERE
( Resort.resort_id=Service_Line.resort_id  ) AND
( Service.sl_id=Service_Line.sl_id  )  AND  ( ( Resort.resort = 'Bahamas
Beach'  ) AND  ( Resort.resort = 'Hawaiian Club'  ))
```

The two Where clause restrictions are combined in AND clauses at the end of the Where clause.

When the query is run, the two restrictions on country cannot be met in the same query, so no data is returned. A message box appears informing you that there is no data to fetch.

Creating two queries to combine restrictions

Users can solve the problem of using two condition objects in the same query by running two queries, one for each Where clause, and using the UNION operator to combine the results.

7.6.17 Using self restricting joins to apply restrictions

You can use self restricting joins to restrict data to one or another column in a table, based on a flag which is used to switch between the two columns. A flag is a third column whose values determine which one of the two alternate columns is used in a query.

See the section Self restricting joins [page 164] for more information on creating and using self restricting joins.

7.6.18 Applying a restriction by inferring multiple tables

You can limit the data returned for an object to values from the table inferred by the object that also match values in another table.

For example, an object called Country of Origin infers the table Country. The object Country of Origin returns the following data:

Country of origin
Australia
France
Germany
Holland
Japan
UK
US

If you want to use the object Country origin under a class Sales_Person, so that it only returns the countries where sales people are based, you can rename the object to Sales people countries and restrict the table Country to return only values for countries of Sales people from the Sales_Person table.

The Sales people countries object has the following SQL:

```
SELECT  Country.country FROM  Country,  Sales_Person,  Customer,  City,  Region
WHERE  ( City.city_id=Customer.city_id  )  AND
( City.region_id=Region.region_id  )  AND
( Country.country_id=Region.country_id  )  AND
( Sales_Person.sales_id=Customer.sales_id  )
```

The Sales people countries object returns the following data:

Sales people countries
France
Germany
Japan
UK
US

You apply the restriction by specifying that when the Country object is used in a query, the Sales_Person table must also be inferred in the From clause of the Select statement.

Country under the Sales_Person class then only returns countries in which sales people are based. You apply the restriction by using the Tables button in the object definition sheet.

The Country table must be joined to the Sales_Person table by intermediary joins using only equi-joins.

> **i** Note
>
> If you make any changes to the SQL for an object that has a table restriction defined in its Select statement, then the universe design tool automatically redetermines which tables are needed by the object's Select statement and Where clause. You are not notified if the table restriction is over ridden in the tables inferred by the object.

7.6.18.1 Inferring multiple tables to apply a condition

To infer multiple tables that apply a condition for an object:

1. Double click an object.
   The Edit Properties dialog box for the object appears.
2. Click the Tables button.
   A list of tables in the universe appears.
3. Select one or more tables that you want the object to infer in addition to the current table. You can select multiple tables by holding down CTRL and clicking table names in the list. The tables Country and Sales_Person are selected below:



4. Click OK in each dialog box.
5. Run queries in Web Intelligence to test the table restriction.

7.6.18.2 When do you use each method to apply a restriction?

You can use the following guidelines to set restrictions in a universe:

- Avoid using Where clauses in object definitions. If you need to use a Where clause, you should be aware of the potential problems using multiple objects, and conflicting Where clauses.
- Use Condition Objects when you want to assist users by providing optional pre-defined Conditions, avoiding multiple objects and changes to the classes and objects view of the Universe pane.
- Use Self-Restricting Joins to apply restrictions to tables when you want the restriction to apply irrespective of where the table is used in the SQL. This method is ideal when a table uses a flag to switch between two or more domains.
- Use Additional Joins when a lookup table serves more than one purpose in the universe.

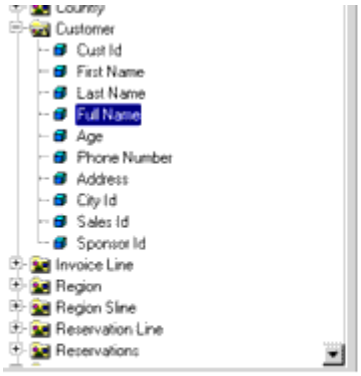
7.6.19 Concatenating objects

A concatenated object is a combination of two existing objects. For example, you create an object `Full Name`, which is a concatenation of the objects `Last Name` and `First Name` in the `Customer` class.

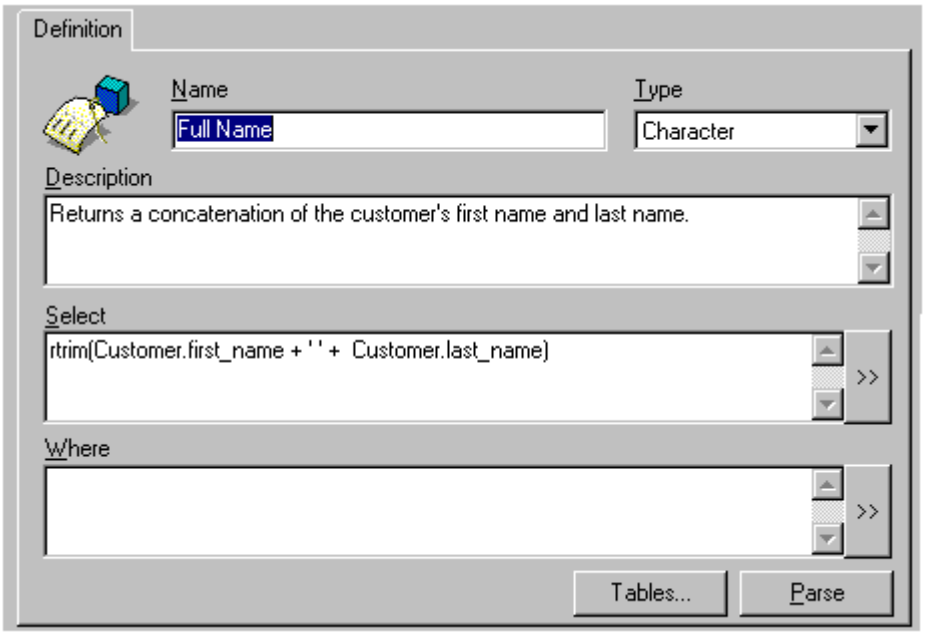
7.6.19.1 Creating a concatenated object

To create a concatenated object:

1. Create an object.
   For example, you create a new object Full Name in the Customer class. You should also type a description for the object such as "This object is a concatenation of the customer's first and last name."



2. Double click the object.
   The Edit Properties dialog box appears.
3. Type the syntax for the concatenated object in the Select box.
   For example you type the following syntax for the Full Name object (MS Access syntax):
   rtrim (Customer.first_name + ' ' + Customer.last_name)
   Where rtrim is a function that removes the blank space at the end of a character string, and the two quotes are used to insert a space between the first and last name.



> **i Note**
>
> You can also click the Edit button to open the SQL Editor. You can use the graphic tools in the editor to help you specify the SQL syntax for the object. For more information on this editor, refer to the Designing a Schema chapter.

4. Click OK in each of the dialog boxes.
   When you run a query on the Full Name object, the full names are returned, listed in aphabetical order of the first name.

7.7 Defining hierarchies

You create object hierarchies to allow users to perform multidimensional analysis.

7.7.1 What is multidimensional analysis?

Multidimensional analysis is the analysis of dimension objects organized in meaningful hierarchies.

Multidimensional analysis allows users to observe data from various viewpoints. This enables them to spot trends or exceptions in the data.

A hierarchy is an ordered series of related dimensions. An example of a hierarchy is `Geography,` which may group dimensions such as `Country`, `Region,` and `City`.

In Web Intelligence you can use drill up or down to perform multi dimensional analysis.

7.7.1.1 Drill

A user can use drill to navigate through hierarchical levels of detail. Users can "drill up" or "drill down" on a hierarchy.

For example, a manager wants to track reservation data over time. As the universe designer, you could set up a `Reservation` Time hierarchy to include the dimensions `Reservation Year`, `Reservation Quarter`, `Reservation Month,` and `Reservation Date`.

From a higher level of aggregation for example `Reservation Quarter`, the manager can drill down to a more detailed level such as `Reservation Month` or `Reservation Date`. He or she could also drill up from `Reservation Quarter` to `Reservation Year` to see a more summarized view of the data.

7.7.2 How to identify a hierarchy

Hierarchies can take different forms. Examples of classic hierarchies include:

- Geography: Continent Country Region City
- Products: Category Brand Product
- Time: Year Quarter Month Week Day

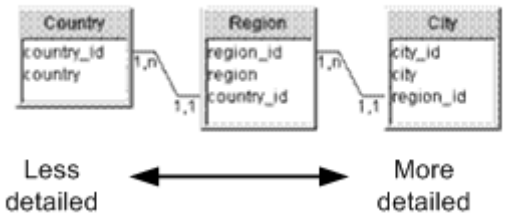
It is also possible for a hierarchy to be "mixed" such as the following:

Geography/Products: Continent Country Category Brand Product

The hierarchies implicit in the data are dependant on the nature of the data and the way it has been stored in the database. You may need to analyze the data very carefully in order to find the hierarchies in your specific system that are best suited to the analysis requirements of your user group.

While there are no precise rules for determining where the hierarchies in the data lie, the one-to-many (1-N) relationships inherent in the database structure can indicate the existence of hierarchies.

In the schema below, the one-to-many relationships between the tables imply a geographical hierarchy.



7.7.3 Setting up hierarchies

By default, the universe design tool provides a set of default hierarchies for multidimensional analysis. These are the classes and the objects arranged in the order that they appear in the Universe pane. When you create objects, you should organize them hierarchically, to ensure that default hierarchies have a sense to users.

You often need to create customized hierarchies that include objects from different classes. In these cases you need to create a new hierarchy.

You can view default, and create new hierarchies from the *Hierarchies Editor*. This is a graphic editor that allows you to manage the hierarchies in the universe.

> **i** Note
>
> When you define a custom hierarchy, the default hierarchies are no longer active, and are not available to the end-user. If you want to make them active you must explicitly select them in the *Hierarchies Editor* and add them to the custom hierarchy list.

7.7.3.1 Viewing hierarchies

You can view hierarchies as follows:

7.7.3.1.1 To view hierarchies in the universe

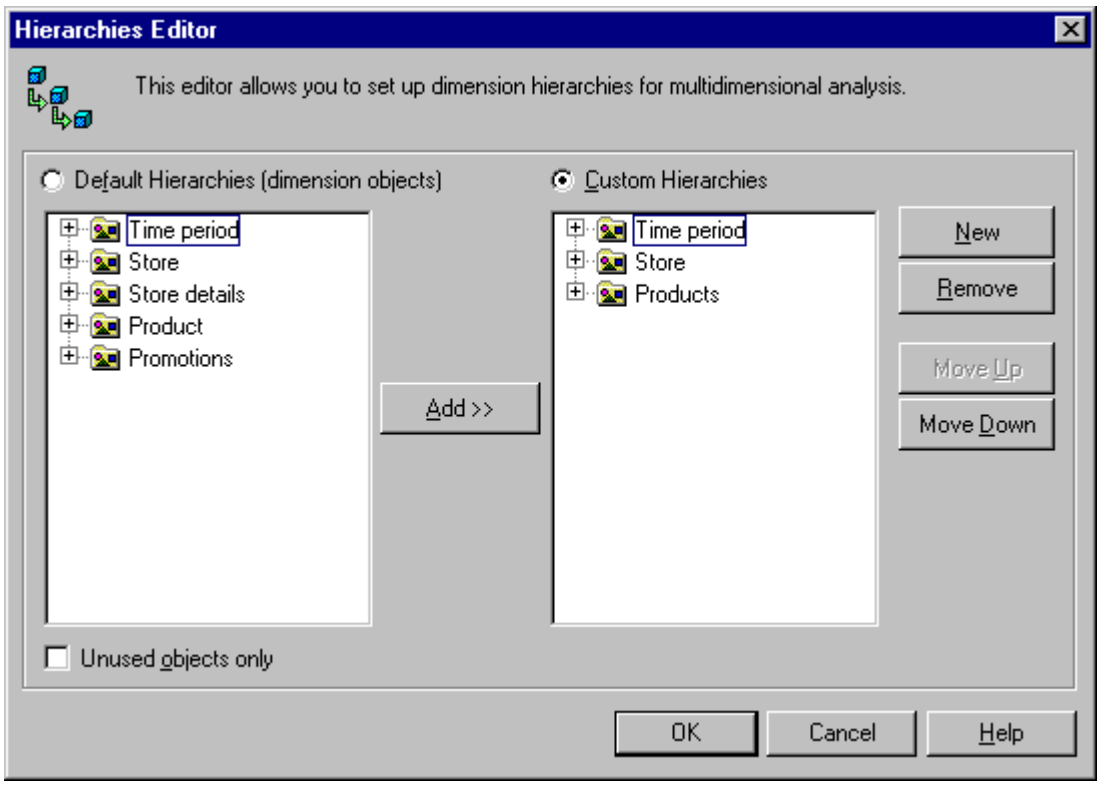
1. Select Tools > Hierarchies.

   Or

   Click the Hierarchies button.

   The Hierarchies editor appears. The universe design tool represents hierarchies with a folder symbol, and dimensions with a cube symbol.

   The left pane lists all the classes that contain dimension objects in the active universe. The right pane shows all the customized hierarchies that you create.



2. Click a hierarchy node (the + sign) to see the dimensions organized hierarchically.
3. Click Cancel.

7.7.3.2 Setting up the hierarchies

You create a new hierarchy by creating a new folder in the Custom Hierarchies pane, then adding the appropriate dimensions in a hierarchical order.

You can delete a hierarchy or a dimension in a hierarchy by selecting the hierarchy or dimension and clicking the Remove button.

7.7.3.2.1 To create a new hierarchy

1. From the Hierarchies editor, click the New button.

   Or

   From the Hierarchies editor, select a class in the left pane and drag it over to the right pane.

   A folder representing the hierarchy appears in the right pane.
2. Type a name for the hierarchy.
3. Press RETURN to apply the name.
4. Select the new hierarchy.

   The hierarchy is highlighted.
5. Expand a default hierarchy node in the left pane.

   This is the hierarchy that contains dimensions that you want to add to the new custom hierarchy.
6. Click a dimension. To select a series of dimensions, hold down CTRL and click each dimension.

   One or more dimensions are highlighted.
7. Click the Add button.

   One or more dimensions appear in the right pane, under the selected hierarchy.

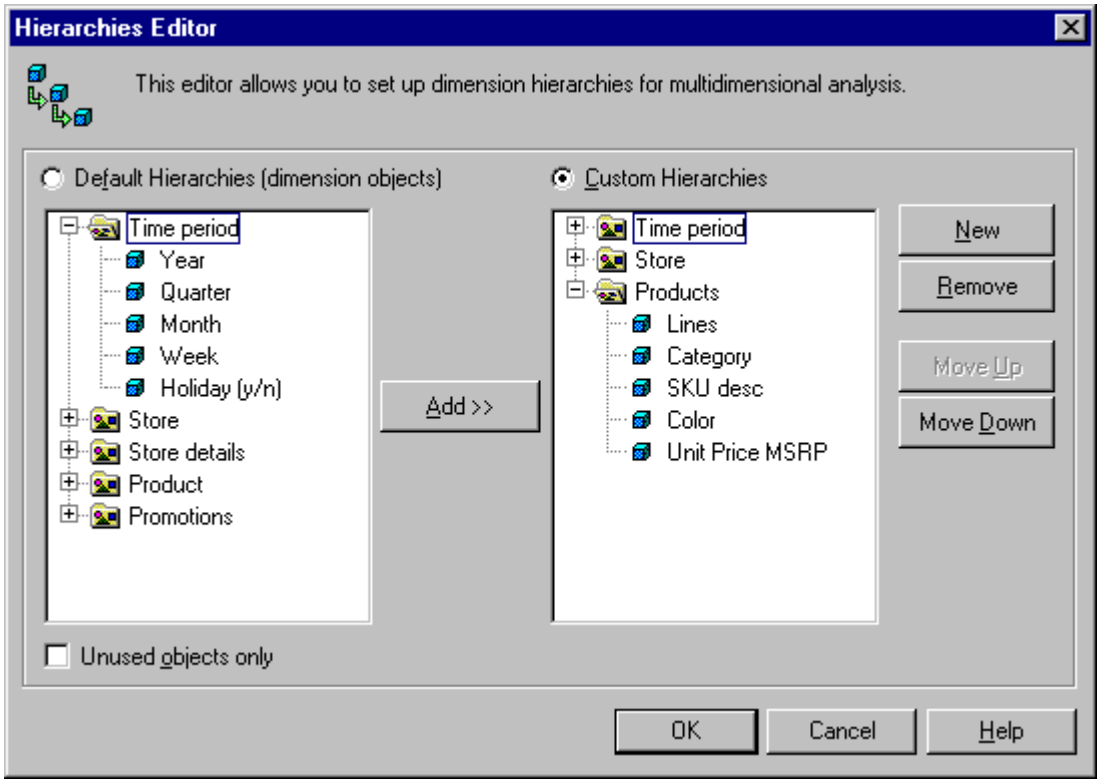
> **i** Note
>
> The Unused objects only check box is a useful way to view only the dimension objects that you have not yet selected for inclusion in a hierarchy.

7.7.3.3 Rearranging the order of dimensions and hierarchies

You can rearrange the order in which the dimension objects appear within a hierarchy. To move an object, click it, and then click the Move Up or Move Down button. You can also re-arrange the order of hierarchies in the same way.

You can also move a dimension object or a hierarchy by drag and drop.

Examples of hierarchies and dimension objects are shown below:



In the Hierarchies Editor above, three customized hierarchies have been set up: Time Period, Store and Products. The Products Hierarchy consists of the following dimensions: Lines, Category, SKU desc, Color and Unit Price MSRP.

7.8 Using cascading lists of values for hierarchies

You can associate a default or custom hierarchy with lists of values, called Cascading lists of values.

> **i** Note
>
> A list of values (LOV) is a list that contains the data values associated with an object. Lists of values are fully described in the section Using lists of values [page 298].

A cascading list of values is a sequence of lists of values associated with a hierarchy in a universe. Prompts are defined for each hierarchy level to return a list of values for the level.

When a report containing a hierarchy associated with a cascading list of values is refreshed, the hierarchy is displayed, and you are prompted to choose a level, and select one or more values from the list of values, before the query is run.

For example; Reservation quarter is associated with a Year hierarchy. When Reservation quarter month is used in a query, the Year hierarchy is displayed, and the user is prompted to select a year for the quarter before running the query.

7.8.1 Creating a cascading list of values

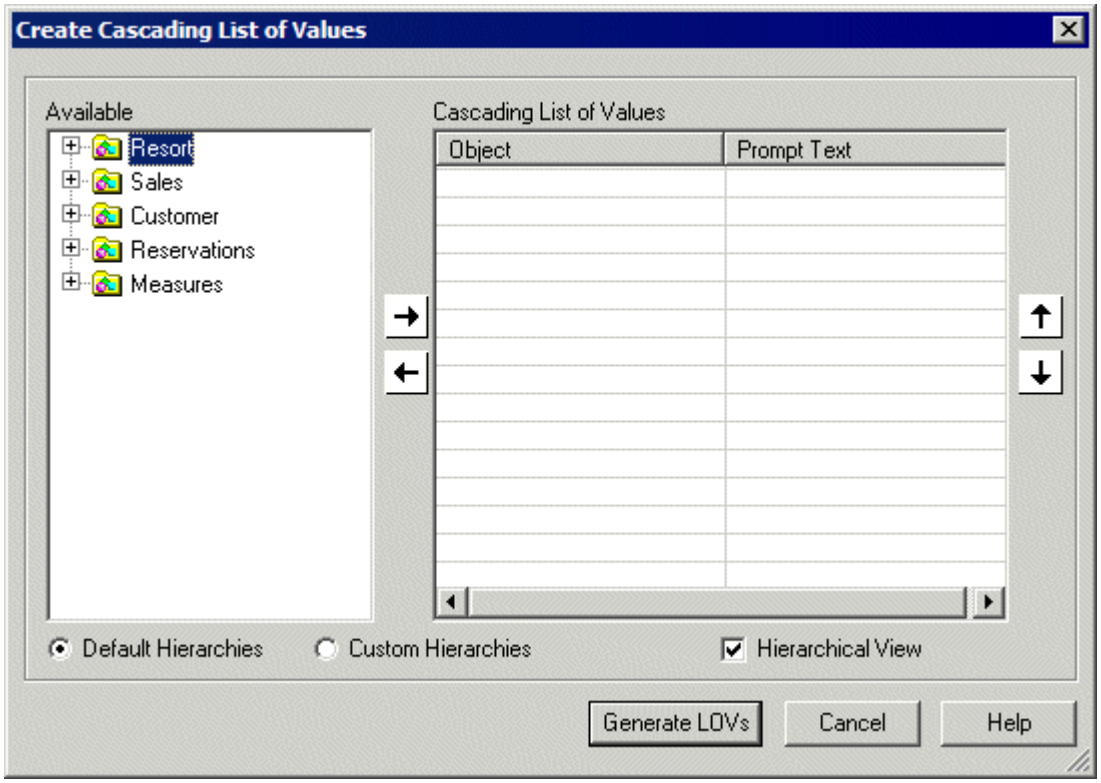
You can create a cascading list of values for a default hierarchy or a custom hierarchy. A .LOV file is created for each level. When a query is run, only the LOV for a prompted hierarchy level is returned.

> **i** Note
>
> This iterative use of lists of values for a hierarchy is different from creating a hierarchy for a list of values, where all the lists of values for all levels of hierarchy are returned in the microcube. When cascading lists of values are used, no LOV is returned in the microcube until the prompt for the level is filled, and only the LOV for the level is returned.

7.8.1.1 To create a cascading list of values

1. Select *Tools* > *Lists of Values* > *Create cascading lists of values*.

   The *Create Cascading List of Values* dialog box appears.



   You have the following options:

Cascading LOV option	Description
Default Hierarchies Custom Hierarchies	When one is selected, the corresponding default or custom hierarchies defined in the universe appear in the *Available* pane. See the section Setting up hierarchies [page 292] for more information on these hierarchy types.
Hierarchical View	When selected, the hierarchy is displayed in a tree view in the *Query Panel*. This facilitates navigation through a hierarchy. When a level is clicked, the list of values appears in a pane to the right of the *Query Panel*.
Object	The hierarchy level for a dimension.
Prompt text	The text that appears in the prompt for the level list of values.

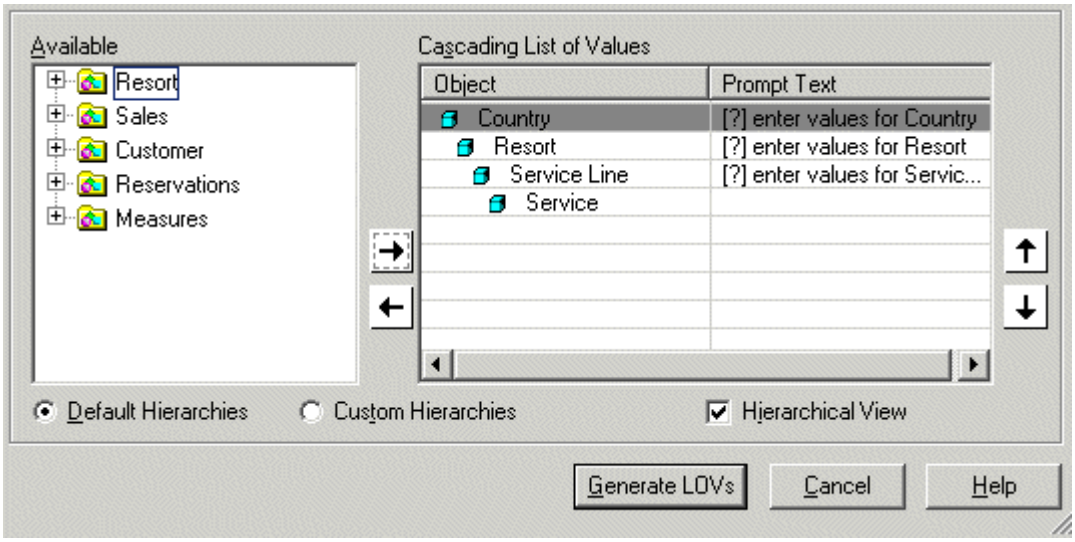
2. Click the *Default Hierarchies* or *Custom Hierarchies* radio button.

   The corresponding list of hierarchies available in the universe appears.
3. Click a class or expand a class and select one or more objects.
4. Click the right head arrow.

   All the objects in the class appear in the Object list.

   Or

   The selected objects appear in the *Object* list.



5. Type a prompt text for each object.
6. If you want to change the position of an object in the *Cascading List of Values* list, click the object and use the up and down arrows to move it up or down the list.

   If you want to remove an object, click the object and click the left arrow.

7. Select or clear the *Hierarchical View* check box.
8. Click *Generate LOVs*.

   The *Create Cascading List of Values* dialog box is removed. A LOV is created for each level of the cascading lists of values. Each .LOV file is saved in the universe sub folder on the file system, for example; `C:\Documents and Settings\<user>\Application Data\Business Objects\Business Objects 12.0\Universes\<CMS name>\beachXI3.0\`.

> **Note**
>
> For information on editing, exporting to the CMS, and creating lists of values for individual objects, see the section Using lists of values [page 298].

7.9 Using lists of values

A list of values is a list that contains the data values associated with an object. A list of values can contain data from two types of data source:

List of values data source	Description
Database file	When you create an object, the universe design tool automatically associates a list of values with the object. The list of values is not created until a user, or you the designer, choose to display a list of values for the object in the Query pane. A SELECT DISTINCT query is then run against the column or columns inferred by the object. The returned data is stored in a file with a.LOV extension in the universe sub folder created under the same folder that stores the universe file.The.LOV file is then used as the source for values for the list.
External file	Personal data, for example a text file, or an Excel file can be associated with a list of values. A list of values that is based on an external file is fixed. You cannot have a dynamic link with an external file. You must refresh the.LOV file if your external file has changed.

7.9.1 How is a list of values used?

In Web Intelligence , a user can create a query in the *Query Pane* using the operand *Show list of values* to apply to an object when applying a condition.

> **Note**
>
> A .LOV file is also created whenever any condition is applied to an object in the *Query Pane* that requires a restriction on the column values inferred by the object.

The list of values for an object appears showing values available for the object, allowing the user to choose the terms for the condition. The first time a list of values is used, it is saved as a .LOV file in the universe sub folder on the file system. This allows the SELECT DISTINCT query to be run only once for an object.

This folder also stores the .LOV files created in the universe design tool which are used to restrict the list of values returned for objects for which the designer wants to control access to the data.

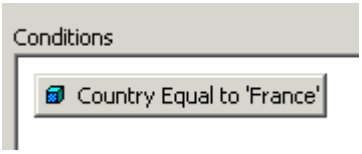
Example: Using a list of values for Country

An object called Country has the following Select clause definition: COUNTRY.COUNTRY_NAME. The default list of values associated with the object contains all the distinct country names in the COUNTRY_NAME column. This list is returned when the object Country is used in a condition in a query.

A user that wants to limit the values in a query to France only, can select France from the following list that shows all country values in the Country table for the condition:



When France is selected from the list, the condition appears as follows in the *Conditions* pane of the *Query Pane*:



The query only returns values for France.

7.9.2 Defining how a list of values is used with an object

When you create a dimension or detail object in the universe design tool, it is automatically assigned an associated list of values. This list does not physically exist when you create an object, but by default, the object has the ability to query the database to return a list of its values when used in the Query pane.

> **i** Note
>
> No default list of values is assigned to measure objects.

When a condition is first placed on an object in the Query pane that requires a list of values to be displayed in the universe design tool, a SELECT DISTINCT statement is run against the appropriate columns inferred by the object, and the list of values is returned.

A.LOV file is automatically created in the universe subfolder to hold the list values. The next time that the list of values is required for the object in the universe design tool, the values are returned from the.LOV file and not from the database.

7.9.2.1 The designer's role in controlling lists of values

As the universe designer, you can define how the data is presented in the list, and define restrictions on the amount and type of data returned to the list.

You can set the properties for an object to determine the following actions for a list of values:

- If a list of values is associated with an object.
- When the list is refreshed.
- Define a query that sets conditions on the SELECT DISTINCT query that an object uses to return a list of values. You save this query in the properties of an object.
- Display list values either as a simple list, or as an object hierarchy.
- If the list is based on column values, or values from an external file, for example an Excel spreadsheet.

You can also create a permanent list for values for an object and export this list to the repository. This.LOV file is then always used as the list of values for that object. It is not updated.

7.9.3 List of values properties and options

You can define the following object properties which allow you to control how a list of values for an object is used in Web Intelligence.

Property	Description
Associate a List of Values	• When selected, allows a list of values to be associated with the object. It is selected by default. • When cleared, no list of values is associated with the object. • Selected by default for dimensions and details. Not selected for measures.
List name	Name of the .LOV file that stores the returned list data. Limited to 8 characters.
Allow users to edit this List of Values	• When selected, users can edit the list of values file in Web Intelligence. • When cleared, the user cannot edit the list. **i Note** This does not apply to personal data files such as Excel spreadsheets. These are not exported to the repository. They remain on a local machine. A user can edit a local file, or change the target list of values for another local data file. The purpose of a list of values is usually to limit the set of available values to a user. If they can edit a list, you no longer have control over the values they choose. Normally, if you are not using a personal data file as a list of values source, you clear this option to ensure that users do not edit lists of values.
Automatic refresh before use (BusinessObjects only)	• When selected, the list data is refreshed each time the list of values for an object is displayed in the Query pane. This can have an effect on performance each time the .LOV is refreshed. This option does not apply to Web Intelligence reports. • When cleared, the list is refreshed only once at the start of a user logon session. If the list contains values that regularly change, then you can select this option, but you should take into account the effect on performance. If the list contents are stable, then you should clear this option.
Hierarchical Display	Select the Hierarchical Display property to display the cascading list of values as a hierarchy in Web Intelligence.
Export with universe	• When selected, the .LOV file associated with the object is exported with the universe to the repository. • You must create the list of values that is associated with the object for it to be exported. This list is saved as a .LOV file. • When cleared, a .LOV file for the object is not exported to the repository. Select this option if you customize this list regularly. This allows your modifications to be exported and imported with the universe.

<table>
<tr><th>Property</th><th>Description</th></tr>
<tr><td>Delegate search</td><td>Select the Delegate search property to allow Web Intelligence users to limit the number of values returned in the list of values. When the Delegate search property is selected, Web Intelligence presents an empty list of values box to the user at query run time. The user enters a value to define search criteria to filter the list of values.

Many data sources support wildcard characters to facilitate search on the database. The following wildcards are supported in Web Intelligence:

<table><tr><td>*</td><td>Matches any number of characters, even zero characters</td></tr><tr><td>?</td><td>Matches exactly one character</td></tr><tr><td>\</td><td>Escapes the next character allowing you to search for a wildcard character</td></tr></table>
The Delegate search option has the following restrictions:

Delegate Search is not supported for cascading list of values.
Delegate Search can be activated only for list of values on objects that are type character.
Delegate Search cannot be activated when custom SQL is entered for the list of values.
Delegate Search cannot be activated when the Hierarchical Display option for list of values is used.</td></tr>
</table>

You can edit, display, or assign the default name to a list of values by clicking the following buttons:

Option	Description
Restore Default	Restores default name assigned to the .LOV file at object creation.
Edit	Allows you to edit the values displayed in the list. You can use the editor to restrict the values displayed in the list when used in the Query pane.
Display	Displays the list of values for the object. When you want to create a permanent list to be exported with the universe to the repository, you must click Display to create the .LOV file. You can then edit the file.

7.9.3.1 Defining properties and options for a List of Values

To define properties and options for a list of values (.LOV) file:

1. Double click an object.
   The Edit Properties dialog box opens to the Definition page.
2. Click the Properties tab.
   The Properties page appears.

3. Select or clear check boxes in the list of values group box at the bottom of the page.
4. Type a name for the associated .LOV file in the List Name box.
5. Click the Edit button if you want to define restrictions on the list values
6. Use the Query pane to create a query on the list data.
7. Click the Display button to see the list of values.
   When you click this button, a SELECT DISTINCT query is run against the columns inferred by the object in the database. This is the same method used in the reporting products to create the .LOV file for the object.
8. Click OK.

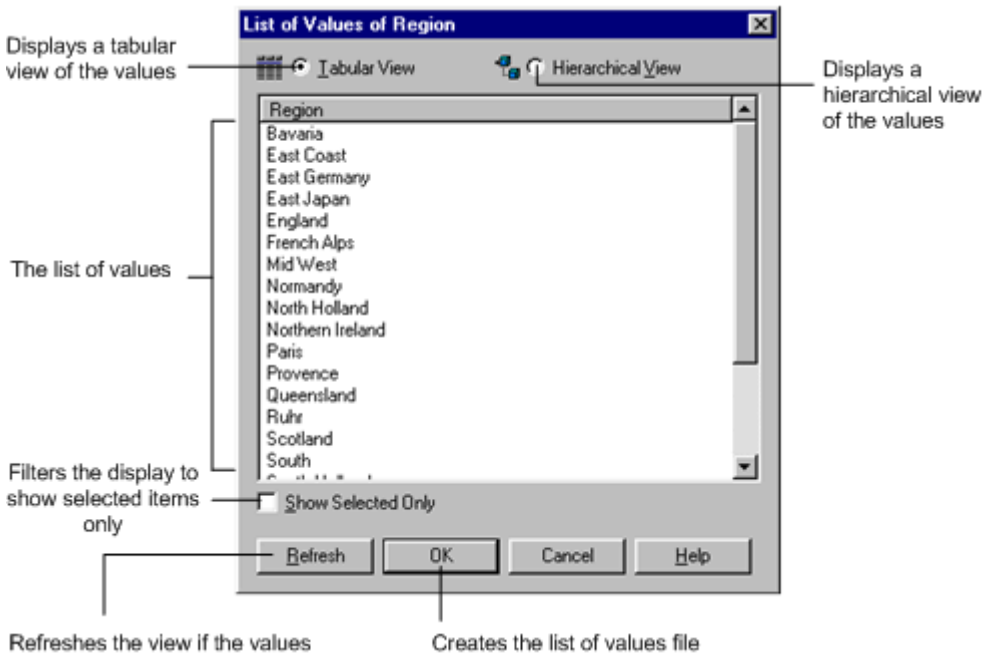
7.9.3.2 Viewing a list of values associated with an object

In the universe design tool, you can view the list of values associated with an object. When you view a list of values, a default .LOV file is automatically created in the User Docs directory to hold the returned data. By default, when you view a list of values you automatically create a .LOV file.

You can view a list of values in a list format, or as an object hierarchy.

To view a list of values:

1. Double click an object.
   The Edit Properties dialog box opens to the Definition page.
2. Click the Properties tab.
   The Properties page appears.
3. Click the Display button.
   The List of Values dialog box displays all the possible data values associated with the object.



4. Click Cancel.

7.9.3.3 Creating a list of values

You create a list of values as follows:

1. View the list of values for an object.
2. Click OK.
   The universe design tool stores list of values (.LOV) files in a universe subfolder in the same folder that contains the universe file. The name of the subfolder is the same as the universe that contains the object used to create the .LOV.

Once you have created the .LOV file, you can edit the list to restrict the data that is returned to the .LOV file, or modify how the data is presented in the list.

7.9.4 Editing a list of values

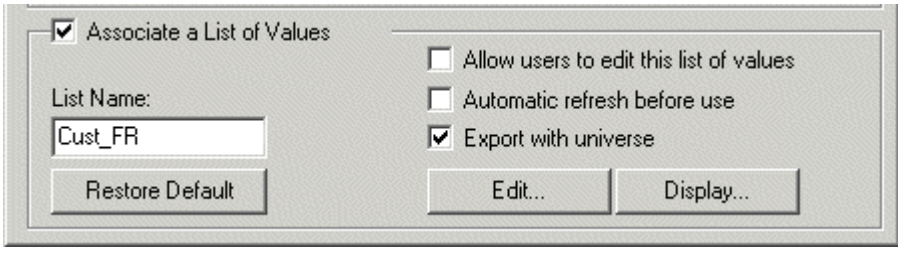
You can modify the contents of a list of values in two ways:

- Apply a condition to the SELECT DISTINCT query that generates the list. For example, you restrict the resorts in the list of values for the Resort object to those resorts that have more than a minimum number of reserved guests.
- Create a hierarchy to simplify for users the process of choosing a value from the list. This can be very useful if a list contains a lot of values.

7.9.4.1 Applying a condition to a list of values

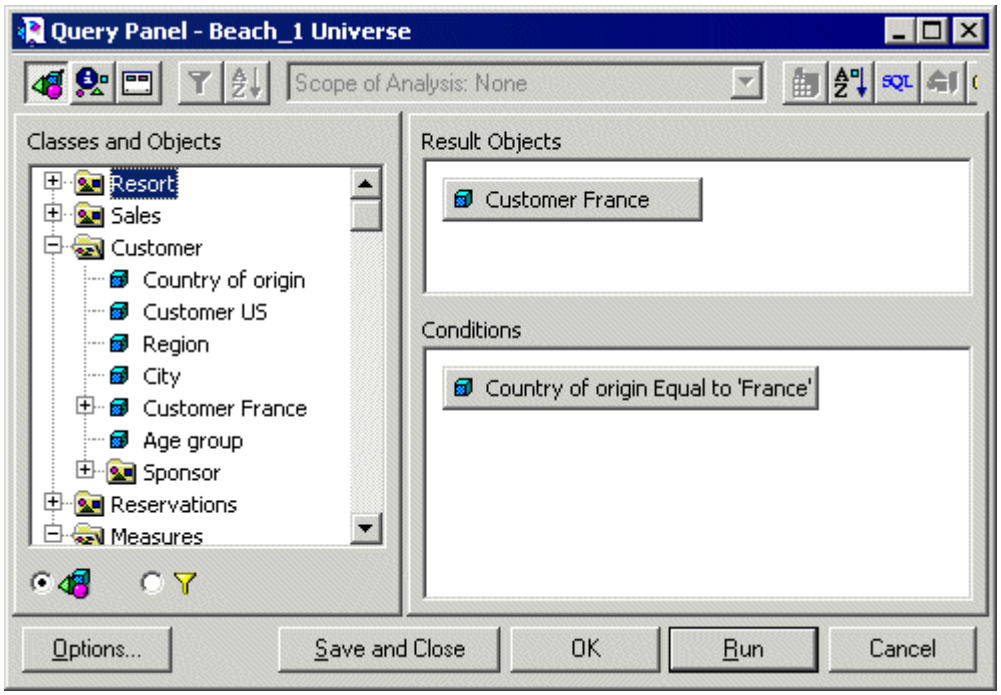
To apply a condition to a list of values:

1. Double click an object.
   The object Edit Properties sheet appears.
2. Click the Properties tab.
   The Properties page appears.
3. Select the Associate a List of Values check box.
4. If you want to rename the list, then type a name for the.LOV file in the List Name box.

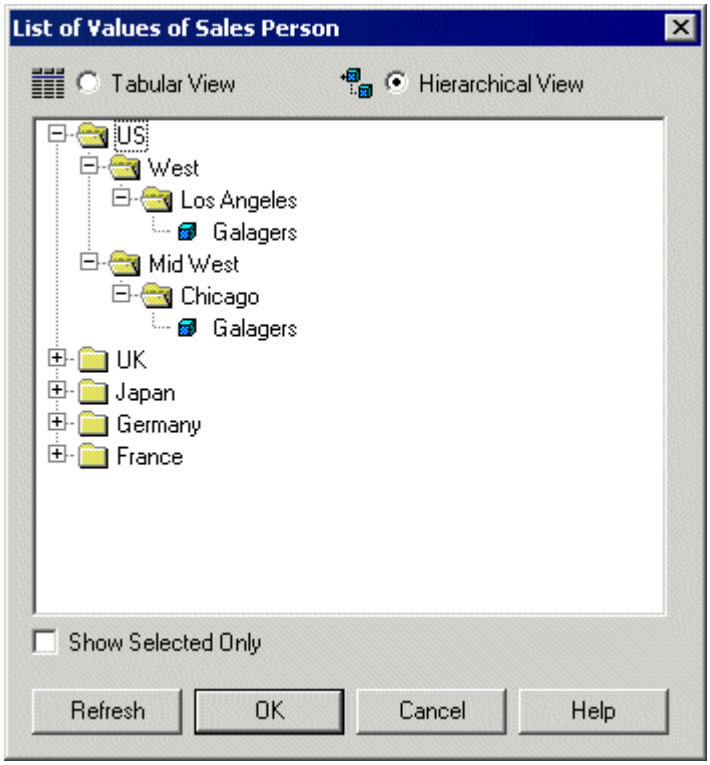


5. Click the Edit button.
   The Query pane appears. The active object is listed in the Result Objects pane.
6. Drag an object that you want to serve as a condition on the list of values for the active object over to the Conditions pane.

7. Double click an operator in the Operators pane.
8. Double click an operand in the Operand pane.
9. Select or type values as required.
   For example the following query returns customers only from France.



10. Click OK.
11. Click Display to view the restricted list of values.
    A blank list appears.
12. Click Refresh.
13. The values appear in the list.

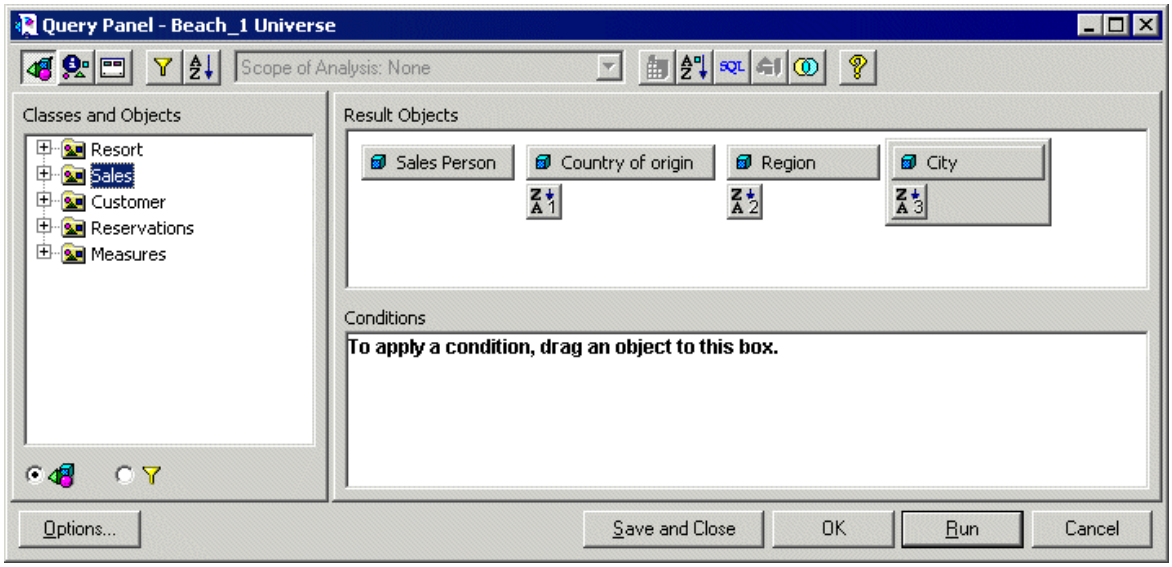


14. Click OK in each of the dialog boxes.

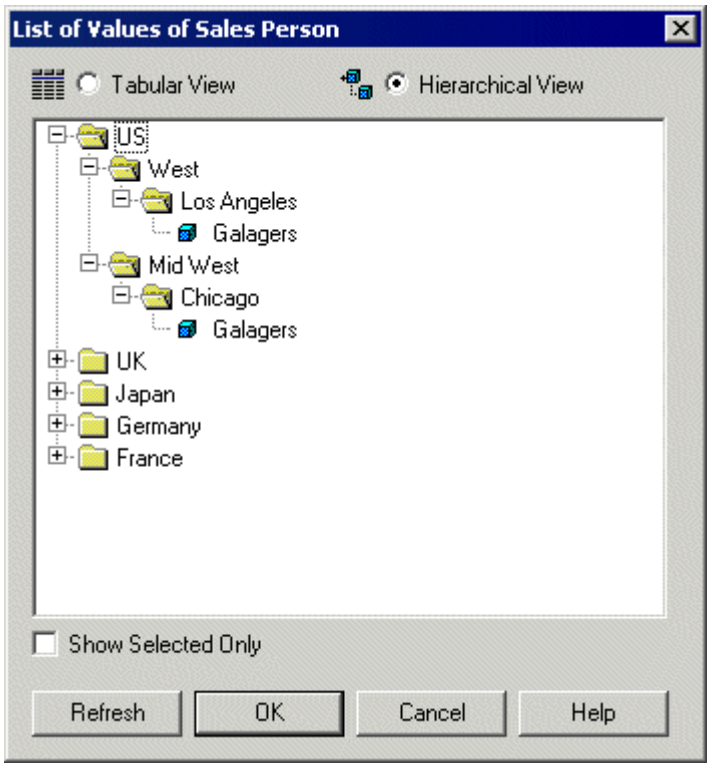
7.9.4.2 Creating a hierarchy for a list of values

To create a hierarchy for a list of values:

1. Double click an object.
   The object Edit Properties sheet appears.
2. Click the Properties tab.
   The Properties page appears.
3. Select the Associate a List of Values check box.
4. If you want to rename the list, then type a name for the.LOV file in the List Name box.
5. Click the Edit button.
   The Query pane appears. The active object is listed in the Result Objects pane.
6. Drag the objects that you want to place in the hierarchy into the Result Objects box to the right of the existing object, as shown below:



7. Click OK.
8. Click Display to view the restricted list of values.
   A blank list appears.
9. Click Refresh.
   The values appear in the list.



10. Click OK in each of the dialog boxes.

7.9.5 Exporting a list of values

You can export a list of values with the universe to the CMS. On the file system, the associated .LOV file is copied to a universe sub directory in the same folder that stores the universe file.

7.9.5.1 How is an exported .LOV used in Web Intelligence

When a user runs a query in Web Intelligence using an object that is associated with a .LOV file exported from the universe design tool, the list of values that is returned for the object is determined by one of the following:

- The data contained in the .LOV file.
- The SQL for the SELECT DISTINCT query defined in the .LOV file.

If you have created a condition in the universe design tool to restrict the data values returned for an object, the restricted list appears, and not the default list of all the data values. The list retains all conditions and formatting implemented in the universe design tool.

If you had not exported the .LOV file with the universe, then the object would simply return the default list with no conditions and formatting. A default .LOV file would then be created to hold the data.

7.9.5.2 Exporting a list with or without data

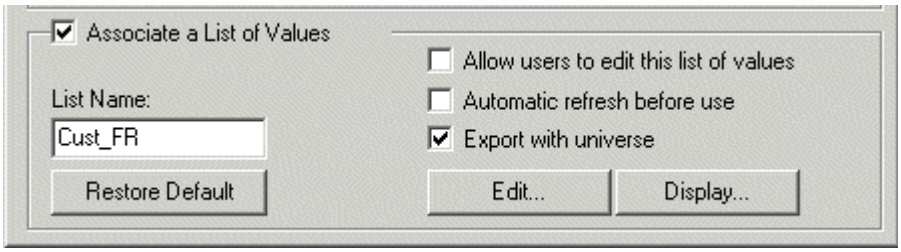
You can export a list of values to the Central Management Server (CMS) repository in two ways:

Export .LOV...	Description
With query definition only (no data)	The .LOV file is exported with the definition of the SELECT DISTINCT query to return values to the list. All conditions that you set for the .LOV in the universe design tool Query pane are retained. The .LOV file contains no data, and is populated the first time the object is used to return values in the Query pane. You should use this method for data that is updated regularly, or if the list of values can be very large.
With data	The .LOV file is exported or imported with all the data that is returned when you display or edit a list of values in the universe design tool. This can be useful if the data in the .LOV does not change. However, if the data is regularly updated, or if the list contains a lot of values, then you should not export the data with the .LOV as it can slow the export process.

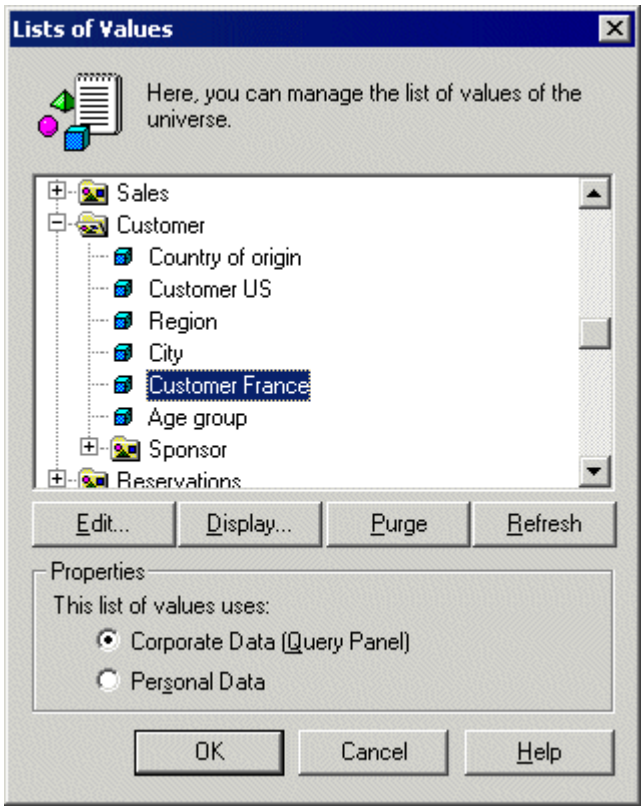
Exporting a list of values definition

To export a list of values definition (no data):

1. Create a list of values for an object.
2. Select the Export with Universe check box on the Properties page for the object.
   Below, a list of values Cust_FR is associated with the Customer to return only values for customers in France.



3. Select Tools > Lists of Values.
   The Lists of Values dialog box appears. It lists the classes and objects in the current universe and contains options to manage the list of values for each object.
4. Expand a class and select the object with an associated .LOV file that you want to export to the repository.



5. Click the Purge button.
   The data is deleted from the .LOV file for the object. The .LOV file now only contains the query definition for the list of values.

6. Click OK.
7. Select File > Export.
   The Export Universe box appears.
8. Select the universe filename from the list of universes.
9. Click OK.
   A message box appears telling you that the universe was successfully exported.

Exporting a list of values with data

To export a list of values with data:

1. Create a list of values for an object.
2. Select the Export with Universe check box on the Properties page for the object.
3. Click the Display button.
   The list of values appears.
4. If the list is empty, click the Refresh button to populate the list.
5. Click OK in each of the dialog boxes.
6. Select File > Export.
   The Export Universe box appears.
7. Select the universe filename from the list of universes.
8. Click OK.
   A message box appears telling you that the universe was successfully exported.

7.9.6 Refreshing values in a list of values

You can refresh the data in a list of values in the universe design tool using two methods:

- Display the list of values for an object, and click the Refresh button.
- Select Tools > Lists of Values to display the Lists of Values management box, select an object and click the Refresh button.

7.9.7 Using data from a personal data file

You can assign a list of values to an object that contains personal rather than corporate data retrieved from a database server.

Personal data is data stored in a flat file such as a text file or data from one of the following applications: Microsoft Excel, Lotus 1-2-3, or dBASE.

Using a personal data file as a list of values has the following advantages:

- Retrieving data from a personal data file can be quicker than accessing your corporate database.
- Users need these values which do not exist in the database.

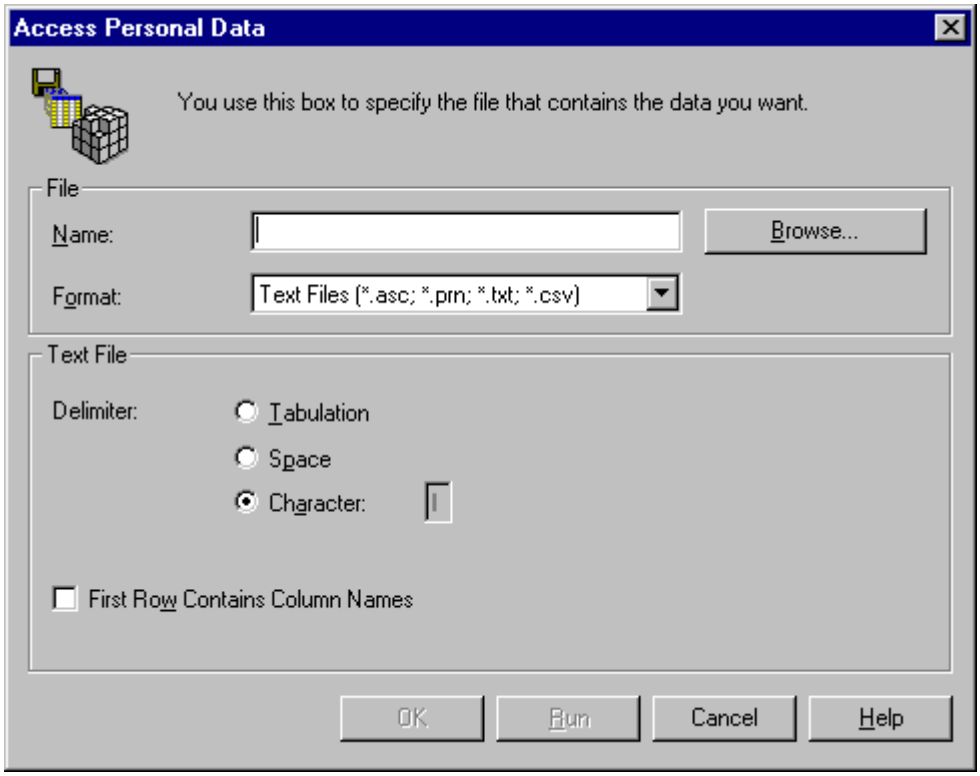
- You control the values that users see when they work with lists of values.

The disadvantage using a personal data file, is that the data is fixed. You must update the data manually if the values need to be changed.

7.9.7.1 Creating a list of values from a personal data file

To create a list of values from personal data file:

1. Select Tools > Lists of Values.
   The List of Values dialog box appears.
2. Expand a class and click an object.
3. Click the Personal Data radio button in the Properties group box.
   A message box tells you that you are about to change the list of values type from corporate to personal.
4. Click OK.
   The Access Personal Data dialog box appears. The available options depend on the file type you select.



5. Click the Browse button and select the file that you want to use as the list of values.
   Or
   Type the file name in the Name text box.
6. Select the file format from the Format list box.
7. You can select one of the following file formats:
   - Text Files (*.asc; *.prn; *.txt; *.csv)
   - Microsoft Excel Files

- dBASE
- Microsoft Excel 97.

> **i** Note
>
> If your file was created in Excel 97, you must use the Microsoft Excel 97 option, not the Microsoft Excel Files option.

8. Specify the remaining options, as necessary.
   In a text file, one line is equivalent to one row. For a text file, indicate the type of column delimiter: a tabulation, space, or character. If you select character as the type, enter the character in the text box.
9. Click OK.

7.9.8 Administering lists of values in the universe

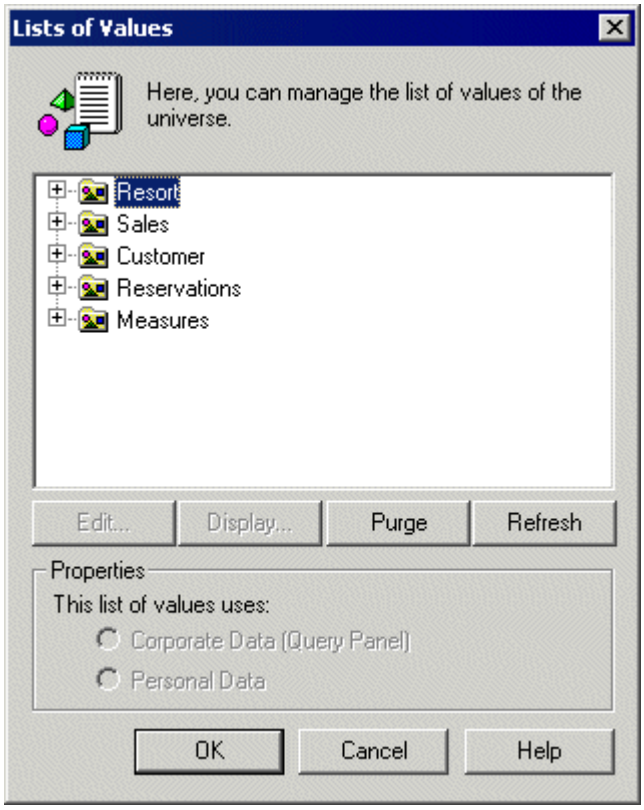
You can manage all the lists of values in the active universe from the Lists of Values dialog box (Tools > Lists of Values). All the classes and objects are presented in a tree view. You can select any object, and access its list of values. You can perform the following actions from the Lists of Values dialog box:

Option	Description
Edit	Displays the Query pane used to define a query for the selected object. You can define and edit existing queries for a list of values.
Display	Displays the current list of values for the selected object.
Purge	Clears the contents of the list of values currently assigned to the selected object.
Refresh	Refreshes the display of the list of values.

7.9.8.1 Accessing the Lists of Values administration tool

To access the Lists of Values administration tool:

1. Select Tools > Lists of Values > Edit a list of values.
   The Lists of Values dialog box appears.



2. Expand a class and select an object.
3. Click a button or select an option to perform an administrative task.
4. Click OK.

7.9.9 Optimizing and customizing LOV files

Some common methods used to optimize and customize LOVs are as follows:

Method	Description
Point LOV to a smaller table	By default LOV point to the same object as the object they are attached to. But if this object points to a large table (number of rows) then refreshing the LOV may be slow. If there is an alternative smaller or faster table that returns the same values, then the LOV should be edited to point to that alternative table.

Method	Description
Combining code and description	A typical customization of a .LOV is to combine a 'code' and 'description'. An object returns a 'sales type code' which may not have a meaningful value to some users. Editing the LOV to display the 'sales type description' will help them when viewing the LOV. The opposite can be done for the 'sales type description' object to display the code along with the description.

7.10 Linking universes

You can dynamically link one or more universes.

7.10.1 What are linked universes?

Linked universes are universes that share common components such as parameters, classes, objects, or joins.

When you link two universes, one universe has the role of a core universe, the other a derived universe. When changes are made in the core universe, they are automatically propagated to the derived universes.

> **i** Note
>
> For information on deploying linked universes, see the section Derived universes and lists of values [page 323]

7.10.1.1 What is a core universe?

The core universe is a universe to which other universes are linked. It contains components that are common to the other universes linking to it. These universes are called derived universes.The core universe represents a re-usable library of components.

A core universe can be a kernel or master universe depending on the way the core universe components are used in the derived universes. Kernal and master universes are described in the section Creating a link between two universes [page 318].

7.10.1.2 What is a derived universe?

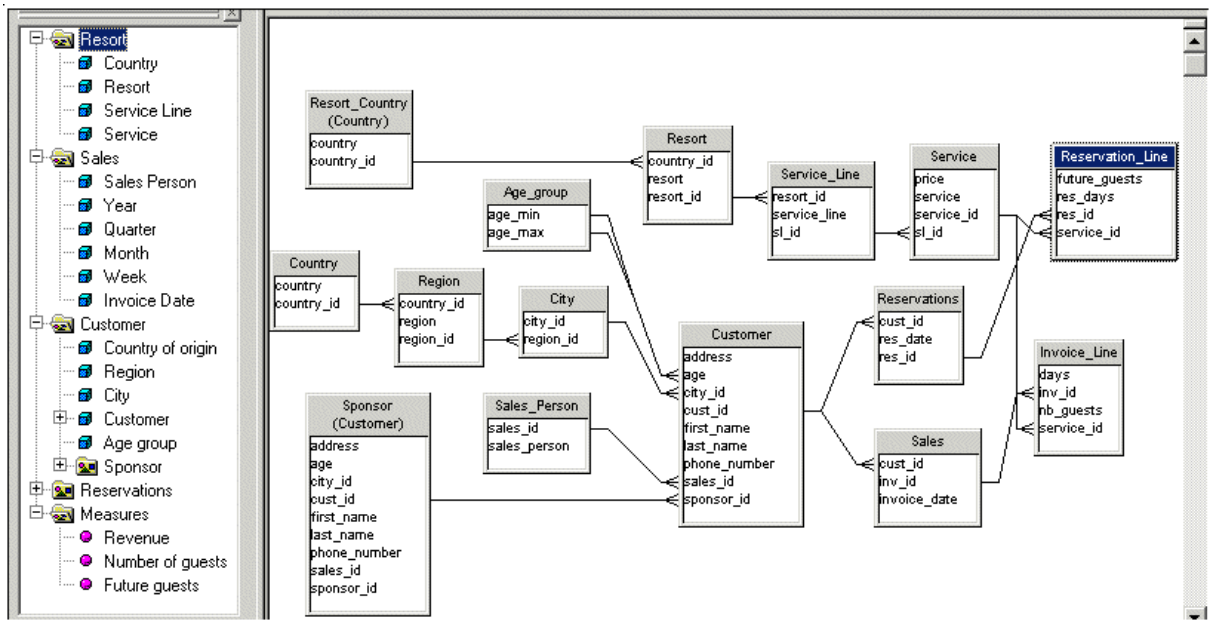
A derived universe is a universe that contains a link to a core universe. The link allows the derived universe to share common components of the core universe:

- If the linked core universe is a kernel universe, then components can be added to the derived universe.
- If the linked core universe is a master universe, then the derived universe contains all the core universe components. Classes and objects are not added to the derived universe. They can be hidden in the derived universe depending on the user needs of the target audience.

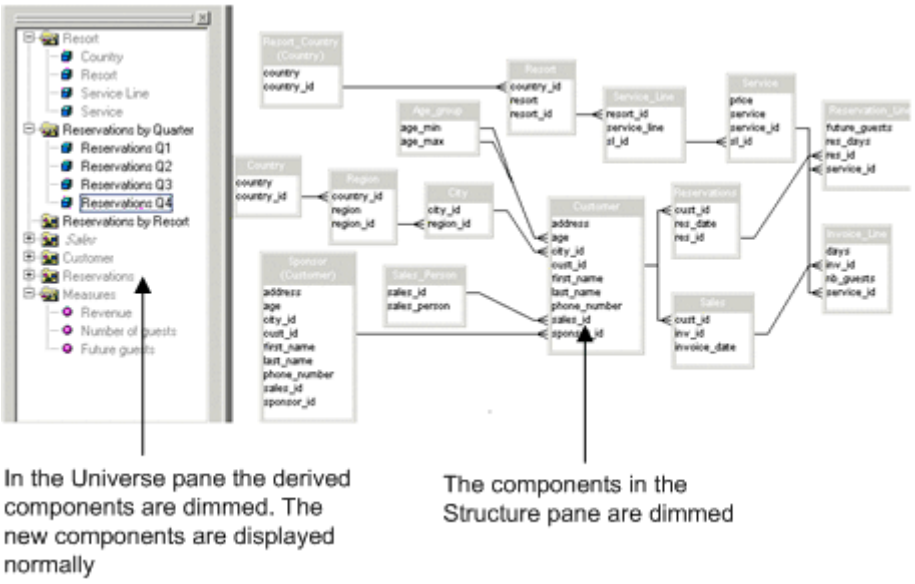
Example: Linked core and derived universes

The example shows two linked universes; one the core universe containing the common components, the other the derived universe that uses the core structures, but has also new classes and objects specific to itself.

Beach.unv is the core universe. It is used by the sales manager of Island Resorts to perform marketing analysis. This universe is one of the demo universes delivered with this release. The contents of the universe are shown below:



Using this core universe, the manager creates a derived universe, which focuses on reservations.



The components derived from the core universe are dimmed. The manager has created two new classes; Reservations by Quarter and Reservations by Resort. These classes and their objects are displayed normally. The manager has also chosen to hide the Sales class, which is not needed in the Reservations universe. Any changes to the core universe components are automatically propagated to the derived universe.

7.10.2 Different ways to link universes

You can use any the following approaches when linking universes:

- Kernel approach
- Master approach
- Component approach
  You can use any of the three approaches individually, or, combine one or more together.

7.10.2.1 Kernel approach

With the kernel approach, one universe contains the core components. These are the components common in all universes. The derived universes that you create from this kernel universe contain these core components as well as their own specific components.

Any changes you make to the kernel universe are automatically reflected in the core components of all the derived universes.

7.10.2.2 Master approach

The master approach is another way of organizing the common components of linked universes.

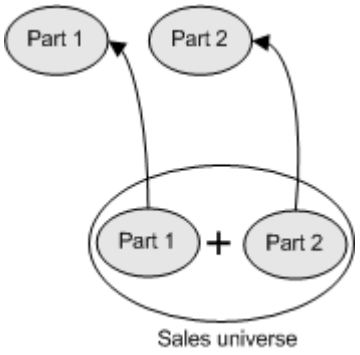
The master universe holds all possible components. In the universes derived from the master, certain components are hidden depending on their relevance to the target users of the derived universe.

The components visible in the derived universes are always a subset of the master universe. There are no new components added specific to the derived universe.The example below shows the universes Human Resources and Sales are derived from a master universe. They contain components from the master universe, some of which may be hidden.

Any changes you make to the master universe are automatically reflected in the core components of all the derived universes.

7.10.2.3 Component approach

The component approach involves merging two or more universes into one universe. The Sales universe below was created by merging two universes: Part 1 and Part 2.



7.10.3 Advantages of linking universes

You have the following advantages when linking universes:

- Reduce development and maintenance time. When you modify a component in the core universe, the universe design tool propagates the change to the same component in all the derived universes.
- You can centralize often used components in a core universe, and then include them in all new universes. You do not have to re-create common components each time you create a new universe.
- Facilitate specialization. Development can be split between database administrators who set up a basic core universe, and the more specialized designers who create more functional universes based on their specific field.

7.10.4 Requirements for linking universes

You can link the active universe to a core universe, only if the following requirements are met:

- The core universe and derived universe use the same data account, or database, and the same RDBMS. Using the same connection for both the core and the derived universe makes managing the universes easier, but this can be changed at any time.
- The core and derived universes must be in the same repository.
- The core universe was exported and re-imported at least once. The derived universe does not need to have been exported before creating a link.
- Exported derived universes are located in the same universe domain as the core universe.
- You are authorized to link the given universe.

7.10.5 Restrictions when linking universes

You need to be aware of the following restrictions when linking universes:

- You cannot link to a universe that uses stored procedures.
- You can use only one level of linking. You cannot create derived universes from a universe which is itself derived.
- All classes and objects are unique in both the core universe and the derived universes. If not conflicts will occur.
- The two universe structures must allow joins to be created between a table in one universe to a table in the other universe. If not, then Cartesian products can result when a query is run with objects from both structures.
- Only the table schema, classes and objects of the core universe are available in the derived universe. Contexts must be re-detected in the derived universe.
- Lists of values associated with a core universe are not saved when you export a derived universe with the core universe structures.

7.10.6 Creating a link between two universes

You can link an active universe to another universe. When you do so, the active universe becomes the derived universe, and the linked universe becomes the core universe. Components from the core universe are inherited by the derived universe.

To link a universe to a core universe, the core universe must have been exported to the repository.

> **i** Note
>
> When you link universes, you can relocate the core universe within the same repository with out breaking the link. This allows you to export the core universe to a different repository folder, while keeping the links with derived universes valid.

7.10.6.1 To create a link between a derived universe

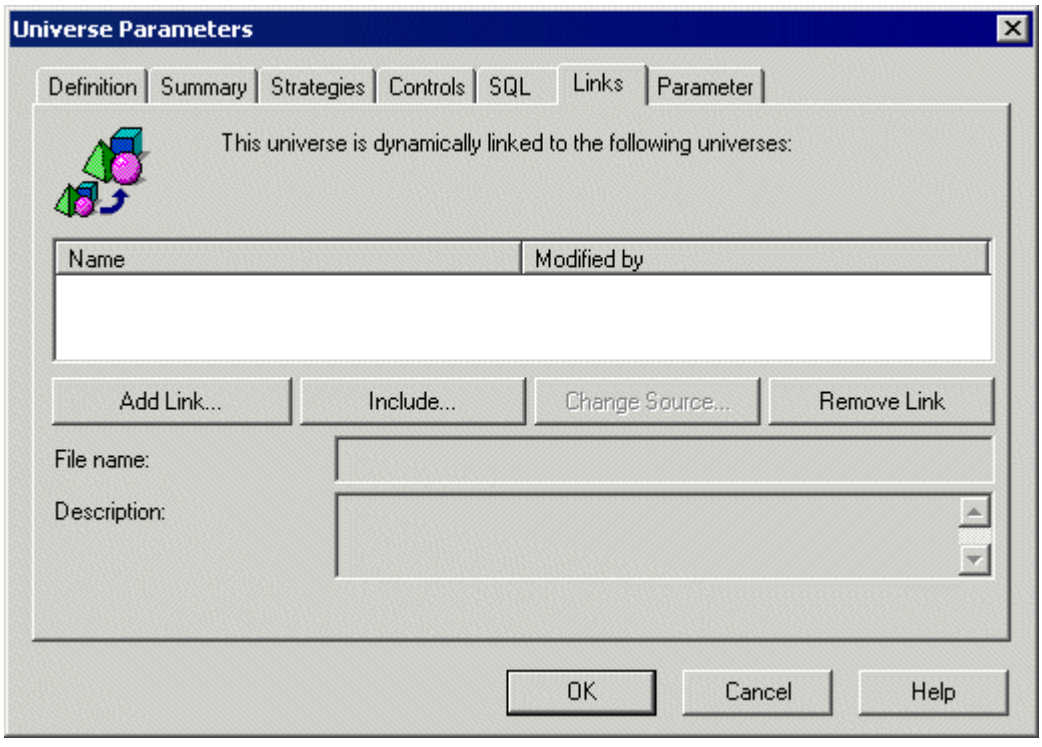
1. Ensure that the active universe is the one that you want to link to the core universe.

   For example, the universe below is a version of the Beach universe that contains only sales information for countries, but no resort data. You want to link this sales universe with a resort universe that contains resort data. The sales Beach universe below is the derived universe, and the Resort universe is the core universe.



2. Select Edit > Links.

   The Universe Parameters dialog box opens to the Links page:

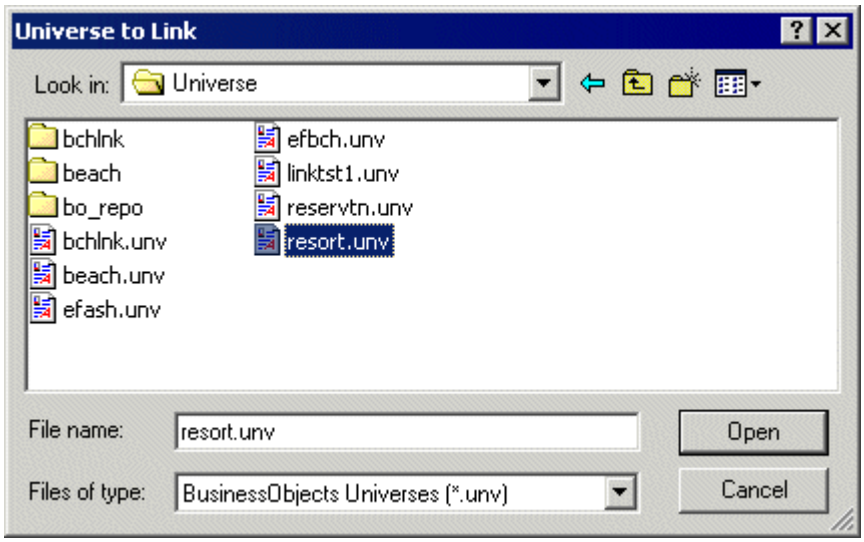


3. Click the Add Link button.

The Universe to Link dialog box appears. It lists universes in the available domains.

4. Browse to the universe that you want to link. This is the core universe that contains the components that you want to use in the active universe.

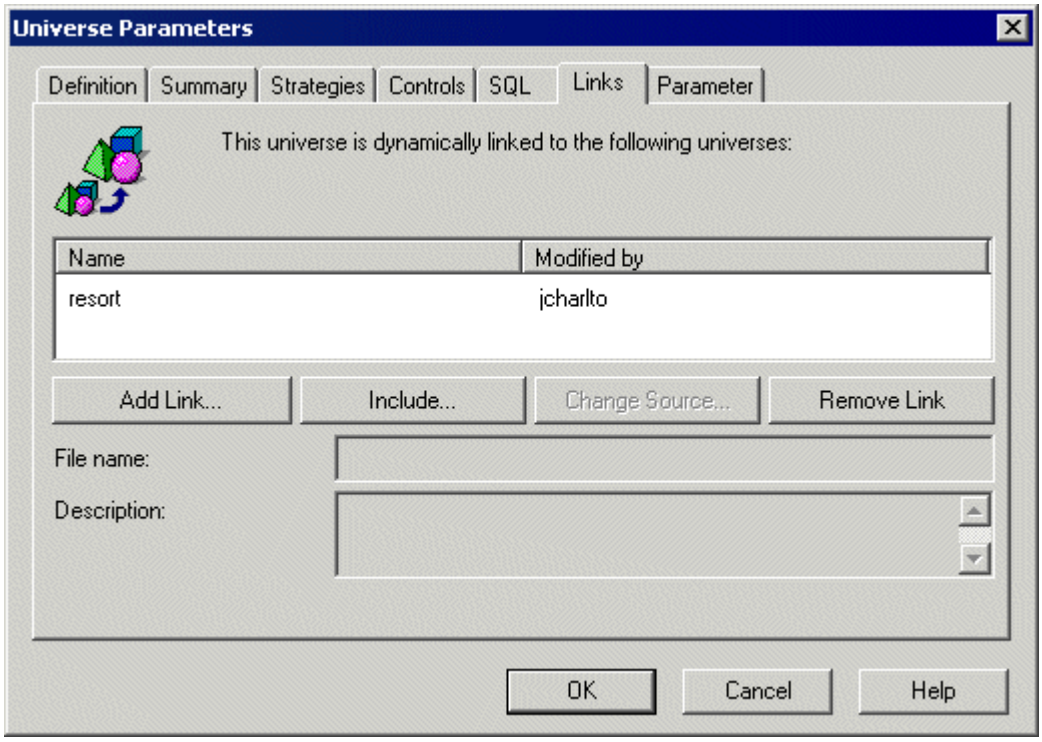
   In the example, you select the resort universe.



   If the universe you selected has never been exported, then you receive an error message. You must export the universe before it can be linked.

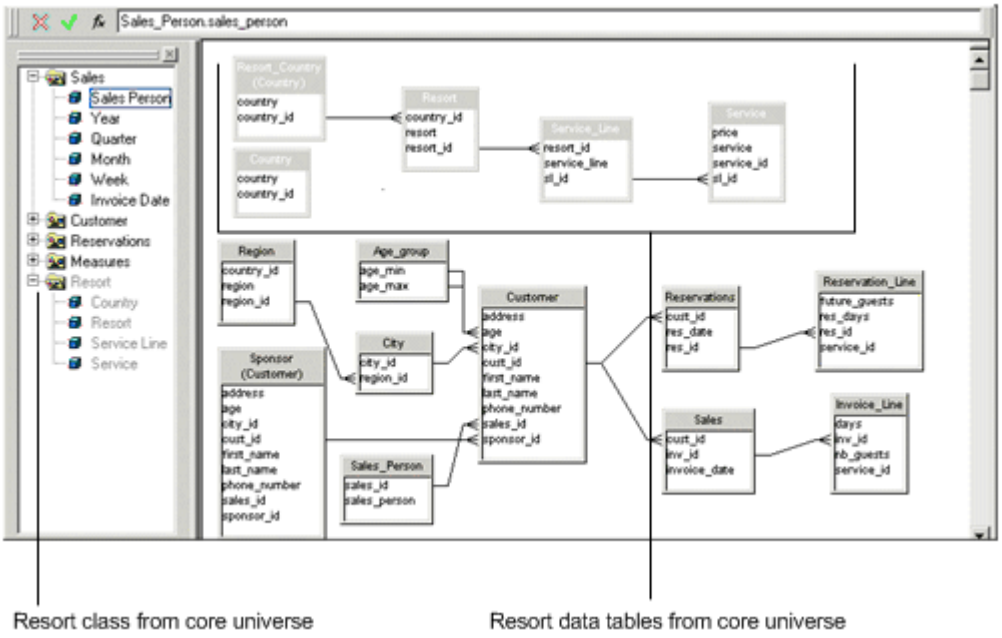
5. Click the Open button.

   The selected universe appears in the list.



6. Click OK.

   The link is created. The core components are displayed dimmed within the active universe.



7.10.7 Editing a derived universe

You complete the linking process by creating joins between the core tables and the derived universe tables. You must delete all current contexts and re-detect the contexts for the new structure.

> **i** Note
>
> You can not edit any structure, class, or object from the linked universe (core universe), within the derived universe.

7.10.7.1 Editing the derived universe

To edit the derived universe:

1. Create joins between the core and derived universe structures.
   Creating joins ensures that Cartesian products are not returned for objects included in a query from both structures.
2. Remove existing contexts.
3. Detect aliases.
4. Detect contexts.
5. Hide or Create new objects as required.

> **i** Note
>
> For information on hiding a component, refer to the section Showing or hiding classes, objects and conditions [page 250].

7.10.8 Removing a link

You can remove a link to a core universe only if the derived universe does not contain objects based on core components, or joins to core components.

7.10.8.1 Removing a link in the derived universe

To remove a link in the derived universe:

1. Open the derived universe.
2. Select Edit > Links.
   The Links page of the Universe Parameters dialog box appears.
3. Click the name of the core universe in the list.
4. Click the Remove Link button.
5. Click the OK.
   The components from the core universe are removed from the active universe.

7.10.9 Relocating the core universe

If the location of your core universe has changed, then you need to indicate the new location in order to maintain the link.

7.10.9.1 Updating a link to a relocated core universe

To update the link to a relocated core universe:

1. Open the derived universe.
2. Select Edit > Links.
3. Click the linked core universe in the list.
4. Click the Change Source button.
   The Universe to Link dialog box appears.
5. Browse to the new location of the core universe.
6. Click the Open button.
   The new core universe appears in the Links list.

7.10.10 Derived universes and lists of values

Lists of values associated with core objects are not saved with the derived universe, when it is exported to the repository.

One method you can use to save lists of values associated with the core objects is as follows:

1. Create new objects using the same definition as the objects containing lists of values that you want to export to the repository with the derived universe.
2. Assign the new objects the same lists of values as the core objects.
3. Hide these new objects.
   The hidden objects serve the function of holding the lists of values so that they can be exported and imported with the derived universe.

7.10.11 Presenting objects in the order of the core universe

By default, the order in which you arrange the objects of the derived universe is that which will be seen by users of the universe, even if the order later changes in the core universe. If you want your derived universe to present objects always in the order they are presented in the core universe, you must set a parameter accordingly in the *.PRM file of the database you are using.

The parameter setting is CORE_ORDER_PRIORITY = Y.

See the Data Access Guide (Help > Data Access guide) for details on how to set the parameters in the relevant *.PRM file.

7.11 Including one universe within another

You can copy the components of a core universe to a derived universe. The resulting components in the derived universe are independent of those in the core universe. These components are not linked to the core universe. Any changes made to the core universe are not inherited by the derived universe.

7.11.1 Copying a core universe into a derived universe

When you copy a core universe into a derived universe, the resulting components in the derived universe are independent of those in the core universe. These components are not linked to the core universe. Any changes made to the core universe are not inherited by the derived universe.

You copy a core universe into a derived universe for any of the following reasons:

- To copy the contents of a given universe into an active universe.
- To no longer keep the dynamic link between two universes.

> **i** Note
>
> If your two universes were linked before the operation, the procedure removes the dynamic link components in the active universe are no longer dynamically linked to the external universe.

7.11.1.1 Copying a core universe into derived universe

To copy a core universe into a derived universe:

1. Open a universe.
2. Select Edit > Links.
   The Links page of the Universe Parameters dialog box appears.
3. Click the Add Link button.
   The Universe to Link dialog box appears. It lists universes in the available domains.
4. Browse to and select the universe that you want to copy. This is the core universe that contains the components that you want to use in the active universe.
5. Click the Include button.
6. Click OK.
   The components from the core universe are displayed within the active universe.

7.12 Creating stored procedure universes

A stored procedure universe is a special universe that enables Web Intelligence users to access stored procedures residing in the database. This is the only way that Web Intelligence users can access stored

procedures. Web Intelligence users can use the stored procedures universes to create reports that are based on stored procedures in the database.

A stored procedure is a compiled SQL program, consisting of one or more SQL statements, which resides and runs on the target database.

Stored procedures offer the following benefits:

- They encapsulate code. The database operation appears once, in the stored procedure, and not multiple times throughout the application source. This improves debugging as well as maintainability.
- Changes to the database schema affect the source code in only one place, the stored procedure. Any schema changes become a database administration task rather than code revision.
- Since the stored procedures reside on the server, you can set tighter security restrictions. This saves more trusted permissions for the well-protected stored procedure themselves.
- Because stored procedures are compiled and stored outside the client application, they can use more sensitive variables within the SQL syntax, such as passwords or personal data.
- Using stored procedures reduces network traffic.

Note the following restrictions apply to stored procedures universes:

- No joins are permitted between objects in stored procedures universes.
- Filters cannot be used on the stored procedure universe.
- You cannot link a stored procedure universe to a standard universe.
- The Web Intelligence supervisor grants access to the database or account where stored procedures are located.
- Not all RDBMSs support stored procedures. Consult your database guide to see if yours does.
- COMPUTE, PRINT, OUTPUT or STATUS statements contained in stored procedures are not executed.

7.12.1 Stored procedures in Java bean universes

BusinessObjects XI Release 3.0 supports the creation of a universe based on Java beans. The universe based on Java beans returns a result set from which you build a universe entity-relation.

The universe based on Java beans uses the same workflow and offers the same benefits as a universe based on stored procedures. The limitations are also the same:

- No joins are permitted
- Filters cannot be used on the universe

For more information about how to access Java beans, you can consult the *Data Access Guide*.

7.12.2 Creating a universe based on stored procedures

Business Objects supports stored procedures:

- without parameters
- with parameters (`IN`)
- with multi-result sets

- with multi-statements (with SQL sentences different from `SELECT`)

To create the stored procedures with parameters, click on the Quick Design wizard and follow the steps described in *Creating a universe with stored procedures with parameters*.

Related Information

To use stored procedures [page 326]
Creating a universe based on stored procedures with input parameters [page 327]
Stored procedures with multi-result set [page 328]

7.12.2.1 Classes and objects in stored procedures

- The universe design tool generates one table per selected stored procedure (or many tables if multiple result sets) and one object per column returned by a stored procedure.
- The result set structure is determined when you describe the function.

7.12.2.2 To use stored procedures

You can create a universe based on one or several stored procedures without parameters. Use the Quick design Wizard, available from the toolbar.

1. Click the *Quick Design Wizard* toolbar button.
   The welcome pane appears.
2. Click the check box *Click here to choose stored procedures universe* at the bottom of the pane.
3. Click *Begin*.
   The *Define the Universe Parameter* panel appears.
4. Type in a universe name in the *Enter the universe name* field.
5. Choose the database connection from the dropdown list in the *Select the database connection* list.
6. Click *Next*.
   The *Create initial classes and objects* panel appears.
7. Click on a stored procedure.
8. Click *Add*.
   The stored procedure is created in the *Universe classes and objects* pane.
9. Click *Next*.
10. Click *Finish*. The Congratulations panel appears.

7.12.2.2.1 Stored procedure parameter in the universe

To improve performance, have several stored procedures based on the same universe for the same data source (via the Quick Design Wizard or Insert > Stored procedures > Update).

In Universe parameters, check the parameter `STORED_PROC_UNIVERSE` is set to `YES`. This indicates that the current universe is based on a stored procedure.

In order to avoid parsing errors on stored procedures columns, it is recommended that you alias result columns based on complex SQL, for example using the aggregate functions - sum, count. The creation of aliased objects cannot be constrained.

> **!** Restriction
>
> The stored procedures do not support `OUT` or dynamic result sets parameters.

7.12.2.3 Creating a universe based on stored procedures with input parameters

You already have objects declared if the procedure prompts the user to select a value from a list in the database.

You can create a universe based on stored procedures that requires one or more input parameters. Depending on the entered value, the procedure returns the fact data of the value from the appropriate fact table.

1. Click the *Quick Design Wizard* toolbar button.
   The welcome pane appears.
2. Click the check box *Click here to choose stored procedures universe* at the bottom of the pane.
3. Click *Begin*.
   The *Define the Universe Parameter* panel appears.
4. Type in a universe name in the *Enter the universe name* field.
5. Choose the database connection from the dropdown list in the *Select the database connection* list.
6. Click *Next*.
   The *Create initial classes and objects* panel appears.
7. Click on a stored procedure.
8. Click *Add*.
   When the stored procedure requires input parameters, the *Stored Procedure Editor* appears
9. Click on a paramater in the list of parameters.
10. Enter a value for the parameter in the *Value* field, or type in a prompt
11. Select *Use this value* or *Prompt me for a value*.
    When you enter a value, when the procedure is executed, it retrieves the columns and the result set structure, this value is passed on to the procedure.
12. If you have selected *Prompt me for a value* , type in the prompt.
    You can enter a prompt message or browse to select an existing object (for example, list of subscriber IDs from a table).

13. Click *OK*.
14. Click *Next*.
15. Click *Finish*. The Congratulations panel appears.

7.12.2.3.1 Value prompts

Use a prompt to define the value for the parameter when the stored procedure is executed.

By default, the stored procedure parameter name originates from the stored procedure structure and displays a prompt with the name of the stored procedure.

You can adjust the semantic and associate a list of values to this prompt to let you add more values from the list.

In front of each parameter, in the stored procedure parameter dialog box, there is a button that opens the advanced dialog box.

7.12.2.4 Using a list of values in a prompt for stored procedures

Syntax

When defining a stored procedure with a dynamic parameter, you may choose to select a universe object based on a standard table by associating a list of values (lov) to the prompt definition (the list of values must come from a standard table). This is a useful way to propose a universe object list of values to the user.

The list of values can only contain simple values. You cannot edit or create a custom list of values.

Note

When inserting classes or objects that are not part of the stored procedure definition, these classes or objects are hidden. You cannot change the status to display them.

7.12.2.5 Stored procedures with multi-result set

Example: A stored procedure that returns more than one result set. At design time, several tables are created in the universe structure based on the same stored procedure.

```
CREATE PROCEDURE qaputel.sp_getcustomer_2results
@location varchar(10)
AS
SELECT customer_key as KEYID, CUST_LNAME as Lname
FROM CUSTOMER
WHERE ADDRESS_LINE1 like @location
```

```
SELECT PREFIX as PREFIX, GENDER as GENDER, BIRTH_DT as BirthDATE
FROM CUSTOMER
```

The idea to handle multi result set (RS) is:

Stored procedure RS1: a, b + RS2: b, d, e

Table A1: A, B

Table A2: B, D, E

This sample syntax generates two tables based on the same ID. When editing Table A1 in the universe design tool module, you can also edit Table A2.

Columns are distributed according to the stored procedure result set structure. Two tables are generated based on the same stored procedure. The result set structure is the same in this example for the two flows. The universe design tool renames the table name coming from the second result set in the stored procedure. The universe design tool can elaborate the business elements.

The universe design tool generates one table in the universe for each result set and creates for each table some corresponding objects independent from each other. You can modify the universe as if it were a normal stored procedure.

7.13 Testing the universe

You can test the integrity of the objects and classes in your universe by running regular checks with Check Integrity (Tools > Check Integrity), and by testing objects in Web Intelligence. You can also view the SQL that objects generate in a query by using the Query Panel to create a query with universe objects and clicking the View SQL button.

7.13.1 Testing objects in the Query Panel

You can view the SQL for a query using the Query Panel as follows:

1. Select Tools > Query Panel.
   The Query Panel appears.
2. Drag objects to the Results pane on the right.
3. Click the SQL button.



4. The SQL for the query appears.
5. Click OK then Cancel to close the Query Panel.

7.13.2 Testing the integrity of the universe

As you create and modify classes and objects, you should use Check Integrity regularly to test the integrity of your universe regularly using Check Integrity. Refer to Checking Universe Integrity Automatically [page 177] for information on using Check Integrity.

7.13.3 Testing the universe with Web Intelligence

You can test objects by running test queries in Web Intelligence. When you test objects you can ask the following type of questions:

- Do the objects exist? If not, did you save the universe after it was last created?
- Is the SQL correct?
- Are the results of the query correct?

You must also test the joins, by evaluating if the returned results are correct, and by checking the schema components with Check Integrity.

8 Optimizing universes

Query time can often be shortened by optimizing a universe. There are several ways you can optimize a universe:

- Optimizing the Array Fetch parameter in the Universe Parameters.
- Allocating a weight to each table.
- Using shortcut joins.
- Creating and using aggregate tables in your database.

Each of these methods is described as follows:

8.1 Overview

You can optimize universes by using the following techniques:

- Using aggregate tables [page 331]
- Using @Functions in the SQL of an object [page 344]
- Using external strategies to customize universe creation [page 371]
- Using analytic functions [page 384]

8.2 Using aggregate tables

You can use features in the universe design tool to allow you to define the Select statement for an object to run a query against aggregate tables in the database instead of the base tables. You can set conditions so that a query will be run against aggregate tables when it optimizes the query, and if not, then the query will be run against the base tables. This ability of an object to use aggregate tables to optimize a query is called aggregate awareness.

This chapter describes how you can set up aggregate awareness in your universe.

8.2.1 What is aggregate awareness?

Aggregate awareness is a term that describes the ability of a universe to make use of aggregate tables in a database. These are tables that contain pre-calculated data. You can use a function called @Aggregate_Aware in the Select statement for an object that directs a query to be run against aggregate tables rather than a table containing non aggregated data.

Using aggregate tables speeds up the execution of queries, improving the performance of SQL transactions.

The reliability and usefulness of aggregate awareness in a universe depends on the accuracy of the aggregate tables. They must be refreshed at the same time as all fact tables.

A universe that has one or more objects with alternative definitions based on aggregate tables is said to be "aggregate aware". These definitions correspond to levels of aggregation. For example, an object called Profit can be aggregated by month, by quarter, or by year. These objects are called aggregate objects.

Queries built from a universe using aggregate objects return information aggregated to the appropriate level at optimal speed.

8.2.2 Applying aggregate awareness to Data Warehouses

Aggregate awareness is particularly useful when working with data warehouses. For example, consider a data warehouse organized into three dimensions: time, geography, and product.

At its lowest level, this data warehouse can store daily information about customers and products. There is one row for each customer's daily product purchases; this can be expressed as follows:

365 days x 100 cities x 10 products = 365,000 rows.

If you ask for information about yearly sales, the database engine must add up a large number of rows. However, the yearly sales of companies may actually involve fewer rows, as follows:

3 years x 3 countries x 3 companies = 27 rows

So, in this example, 27 rows from a table are sufficient to answer the question. Based on this information, it would be far more efficient to pre-summarize these rows into aggregate tables.

8.2.3 Setting up aggregate awareness

Setting up aggregate awareness in a universe is a four-part process. The main steps of the methodology are summarized below.

- Build the objects:

1. Identify all the possible definitions (table/column combinations) of the objects.
2. Arrange the objects by level of aggragation.
3. Build the objects using the @Aggregate_Awareness function.

- Specify the incompatible objects:

1. Build an objects/aggregate tables matrix.
2. For the first aggregate table, decide whether each object is either compatible (at the same level of aggregation or higher), or incompatible (at a lower level of aggregation).
3. Check only the boxes of the objects that are incompatible for that table.
4. Repeat steps 1 to 3 for the remaining aggregate tables.

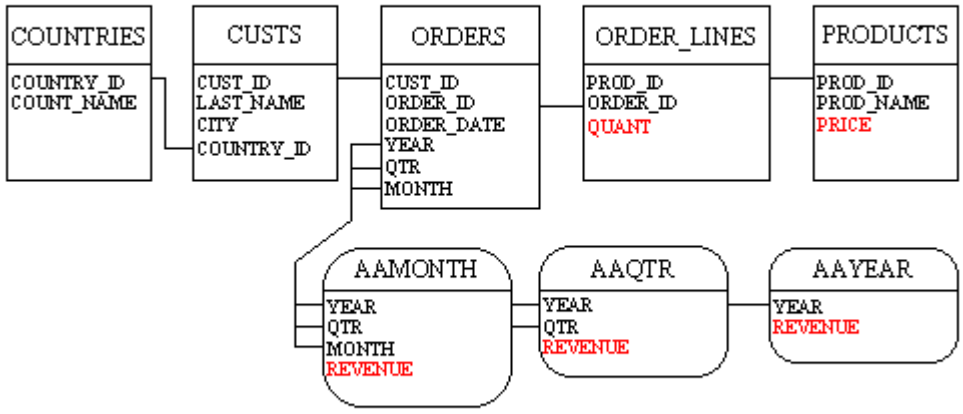
- Define any necessary contexts

Define one context per level of aggregation.

- Test the results

1. Run several queries.
2. Compare the results.

Each stage of the above process is described in detail in the following sections. The example schema shown below is used to illustrate each stage:



The schema contains three predefined aggregate tables: AAMONTH, AAQTR, and AAYEAR.

> **i** Note
>
> The example schema is not representative of a typical schema. Use it as a way to follow the steps to set up aggregate awareness. In a production schema, an aggregate table would generally combine several dimensions rather than a single dimension based on time. The time dimension (Year, Quarter, and Month) would also normally be defined from within a master table, not an aggregate table.

8.2.4 Building the objects

The first step in setting up aggregate awareness in a universe is to determine which objects are to be aggregate aware. You can use either measure objects or dimension objects.

An object Sales Revenue has the following definition based on the above schema:

PRODUCTS.PRICE*ORDER_LINES.QUANT

You want to redefine Sales_Revenue to use the aggregate tables where possible instead of performing a aggregation using the non aggregate tables.

Each of the stages that you complete to redefine Sales Revenue as aggregate aware, you also need complete for any other objects that you want to use aggregate tables in their definitions.

8.2.5 Identifying all combinations of aggregate objects

You need to identify all possible combinations of the objects in the various tables. The Sales Revenue object can be defined in the following ways:

- AAMONTH.REVENUE
- AAYEAR.REVENUE
- AAQTR.REVENUE
- PRODUCTS.PRICE*ORDER_LINES.QUANT

8.2.6 Arranging objects in aggregate order

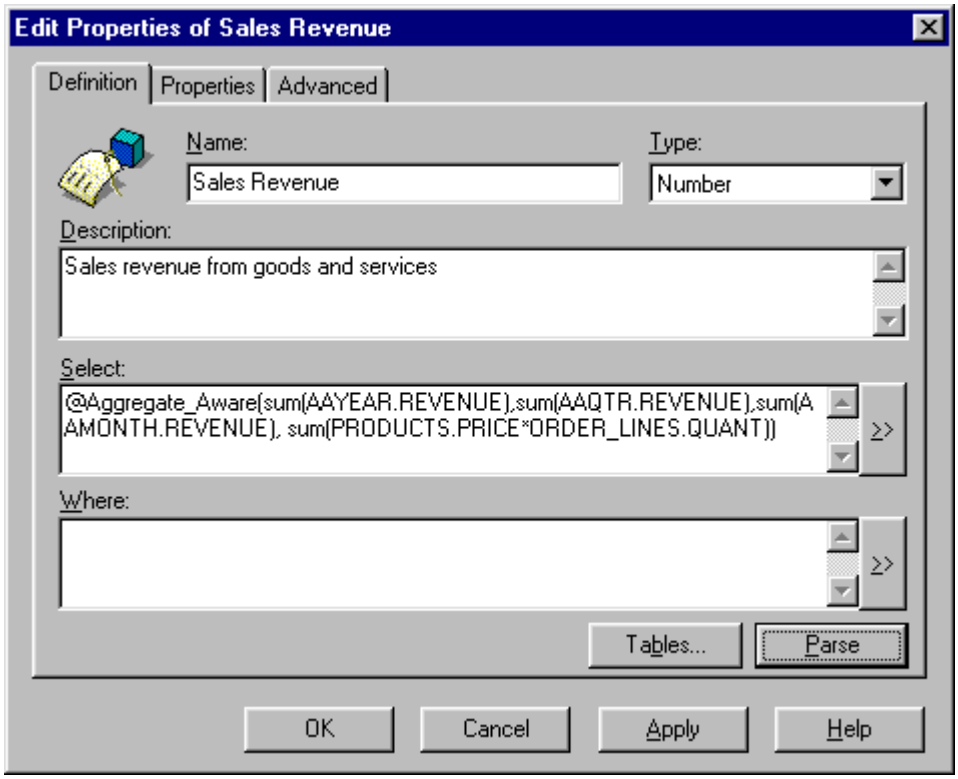
Once you have identified all combinations of the objects, you arrange them according to their level of aggregation as follows:

- AAYEAR.REVENUE is the highest level of aggregation.
- AAQTR.REVENUE is the next level.
- AAMONTH.REVENUE is the next level.
- PRODUCTS.PRICE*ODER_LINES.QUANT is the lowest level of aggregation.

8.2.7 Defining aggregate objects with the @Aggregate_Aware function

You then re-define the Select statement using the @Aggregate_Aware function for all aggregate aware objects. The @Aggregate_Aware function directs an object to query first of all the aggregate tables listed as its parameters. If the aggregate tables are not appropriate, then the query is run with the original aggregate based on the non-aggregated table. For more information about @Functions see the section Using @Functions in the SQL of an object [page 344].

The Select statement for Sales Revenue using the @Aggregate_Aware function appears below.



The syntax of the @Aggregate_Aware function is as follows:

```
@Aggregate_Aware(sum(agg_table_1), ... sum(agg_table_n))
```

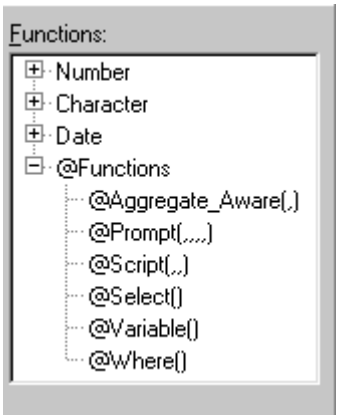
where agg_table_1 is the aggregate with the highest level of aggregation, and agg_table_n the aggregate with the lowest level.

You must enter the names of all aggregate tables as arguments. You place the names of tables from left to right in descending order of aggregation.

8.2.7.1 To define an object using @Aggregate_Aware

To re-define an object using @Aggregate_Aware:

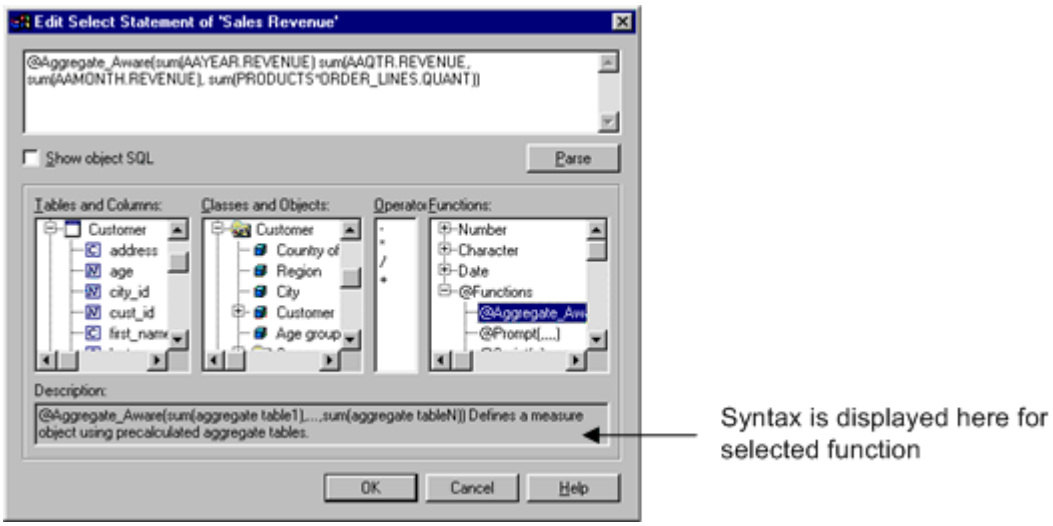
1. Double-click an object.
   The *Edit Properties* dialog box for the object appears.
2. Click the >> button next to the *Select* box.
   The *Edit Select Statement* dialog appears.
3. Click at the beginning of the SELECT statement.
   Or
   Click anywhere in the select box if the object does not yet have a SELECT statement.
   The cursor appears at the top left corner of the box.
4. Click the @Functions node in the *Functions* pane.
   The list of available @functions appears.



5. Double-click @Aggregate_Aware.
   The syntax for @Aggregate_Aware is inserted in the Select statement. A description of the syntax appears in the *Description* box at the bottom of the dialog box. You can use this to help you type the parameters for the @function.
6. Insert the aggregates within the brackets of the @AggregateAware function in order (highest to lowest level of aggregation data).
7. Separate each aggregate with a comma. For the example, the syntax for the Sales Revenue is:

   ```
   @Aggregate_Aware(sum (AAYEAR.REVENUE), sum(AAQTR.REVENUE), sum (AAMONTH.REVENUE), sum(PRODUCTS.PRICE*ORDER_LINES.QUANT))
   ```

8. Click *Parse* to verify the syntax.
   The *Edit Select* page of the *SQL editor* for Sales Revenue is shown below.



9. Click *OK* in each of the dialog boxes.
   In the example, you also re-define the dimension objects Year and Quarter with the @Aggregate_Aware function.

8.2.8 Specifying the incompatible objects

You must now specify the incompatible objects for each aggregate table in the universe. The set of incompatible objects you specify determines which aggregate tables are disregarded during the generation of SQL.

With respect to an aggregate table, an object is either compatible or incompatible. The rules for compatibility are as follows:

- When an object is at the same or higher level of aggregation as the table, it is compatible with the table.
- When an object is at a lower level of aggregation than the table (or if it is not at all related to the table), it is incompatible with the table.

8.2.8.1 Using a matrix to analyze the objects

You may find it useful to build a matrix in order to analyze the compatibility of objects and aggregate tables. In the first two columns of this matrix, you can list the names of classes and objects. Then you can create a column heading for each aggregate table in your universe. A blank matrix based on the schema of the example would look like this:

Class	Object	AAYEAR	AAQTR	AAMONTH
Customers	Customer Code (CUSTOMER.CUST_ID)			
	Customer Name (CUSTOMER.LAST_NAME)			
	Customer City (CUSTOMER.CITY)			
	Customer Nationality (COUNTRIES.COUNT_NAME)			
Products	Product Code (PRODUCT.PROD_ID)			
	Product Name (PRODUCT.PROD_NAME)			

Class	Object	AAYEAR	AAQTR	AAMONTH
Orders	Order Year (AA-YEAR.PROD_NAME)			
	Order Quarter (AAQTR.QTR)			
	Order Month (AAMONTH.MONTH)			
	Order Date (ORDERS.OR-DER_DATE)			
Sales Measure	Sales Revenue (@Aggre-gate_Aware(...))			

For each table, enter an X if the object is incompatible.

A completed matrix based on the example is given below.

Class	Object	AAYEAR	AAQTR	AAMONTH
Customers	Customer Code (CUSTOMER.CUST_ID)	X (n)	X (n)	X (n)
	Customer Name (CUS-TOMER.LAST_NAME)	X (n)	X (n)	X (n)
	Customer City (CUSTOMER.CITY)	X (n)	X (n)	X (n)
	Customer Nationality (COUN-TRIES.COUNT_NAME)	X (n)	X (n)	X (n)
Products	Product Code (PRODUCT.PROD_ID)	X (n)	X (n)	X (n)

Class	Object	AAYEAR	AAQTR	AAMONTH
	Product Name (PROD-UCT.PROD_NAME)	X (n)	X (n)	X (n)
Orders	Order Year (AA-YEAR.PROD_NAME)	- (s)	- (h)	- (h)
	Order Quarter (AAQTR.QTR)	X (l)	- (s)	- (h)
	Order Month (AAMONTH.MONTH)	X (l)	3 (l)	- (s)
	Order Date (ORDERS.OR-DER_DATE)	X (l)	X (l)	X (l)
Sales Measure	Sales Revenue (@Aggre-gate_Aware(...))	-	-	-

X (n): This object has nothing to do with the aggregation table. It is therefore compatible.

X (l): This object is at a lower level of aggregation than this aggregate table; it cannot be used to derive information. It is therefore incompatible.

- (s): This object is at the same level of aggregation as this aggregate table; it can be used to derive information. It is therefore compatible.

- (h): This object is at a higher level of aggregation as this aggregate table; it can be used to derive information. It is therefore compatible.

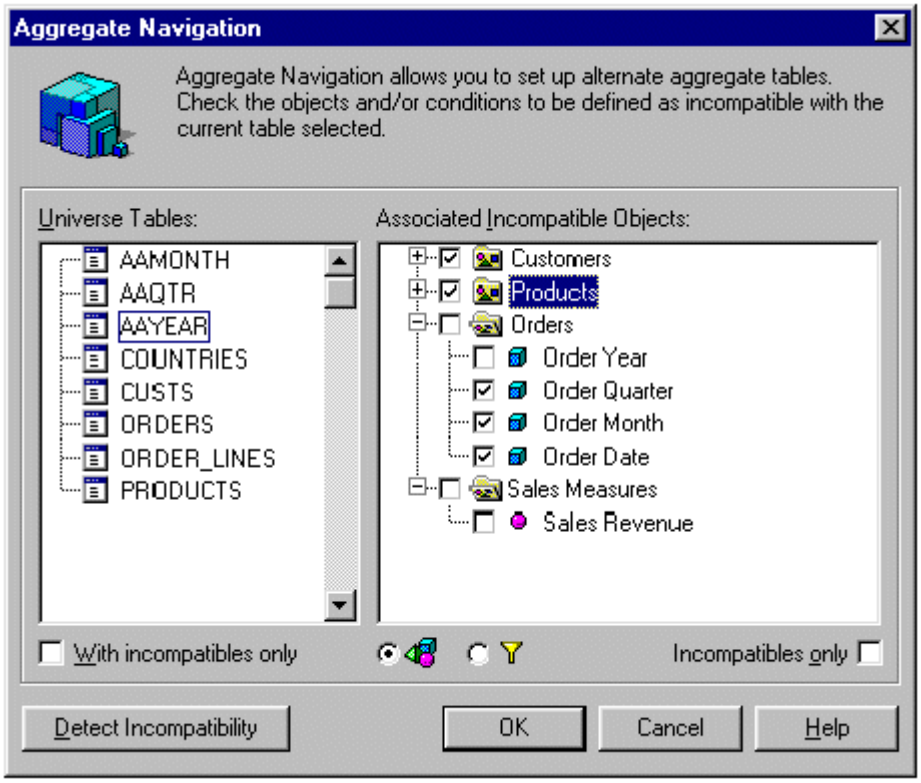
8.2.9 Specifying incompatible objects

You now specify the incompatible objects. You use the *Aggregate Navigation* dialog box (*Tools* > *Aggregate Navigation*) to specify the incompatible objects.

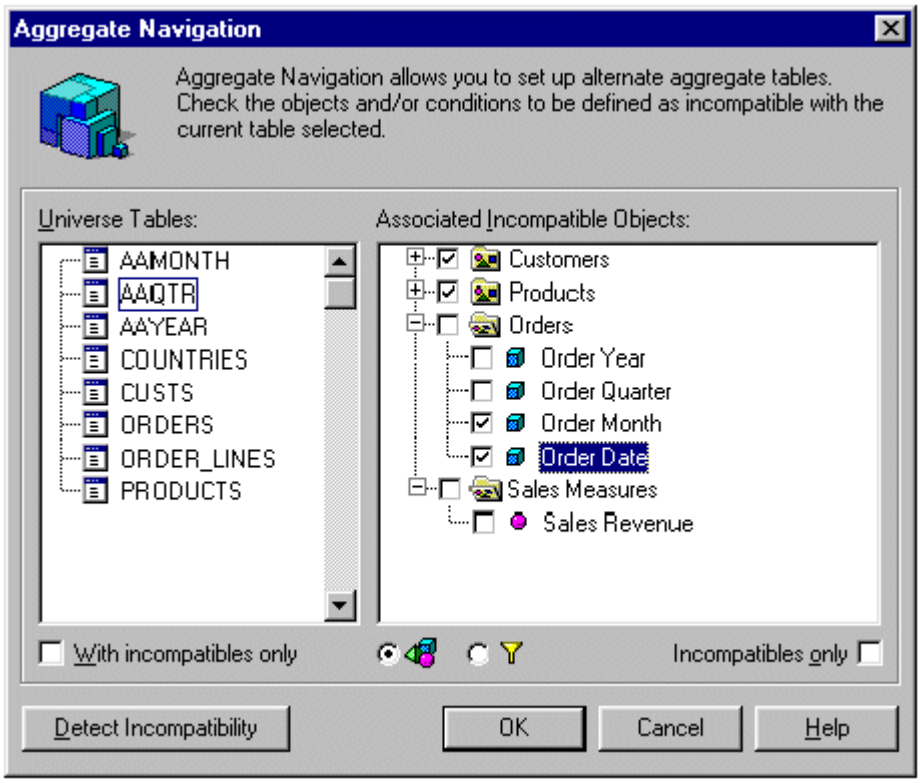
You specify incompatible objects using the *Aggregate Navigation* dialog as follows:

1. Select *Tools* > *Aggregate Navigation*.
   The *Aggregate Navigation* dialog box appears. It consists of two panes:
   - *Universe Tables*, which lists all the tables of the universe.
   - *Associated Incompatible Objects*, which lists all the objects of the universe.

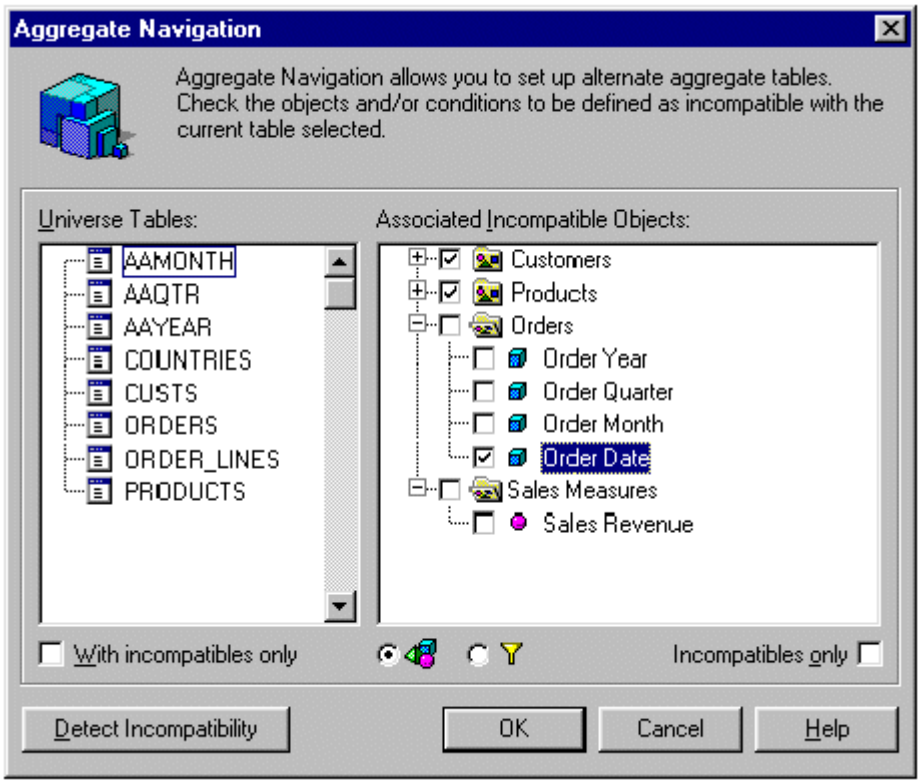
2. Click an aggregate table in the left pane.
3. In the right pane, select the check box for each incompatible object.
   For example, based on the matrix, for the AAYEAR table all the objects in the Customers class are incompatible. You select the check box beside the class name as follows:



4. Repeat the above steps for each aggregate table in your universe.
   For example, the incompatible objects for the AAQTR table are shown below.



For the AAMONTH table, only one object is incompatible.



5. Click *OK*, when all incompatible objects for all the tables are specified.

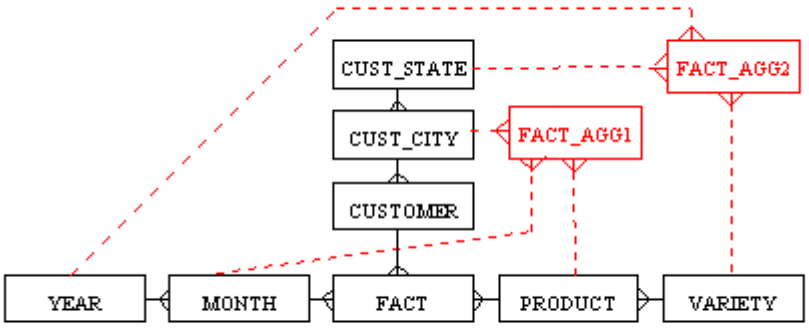
> **i** Note
>
> The dialog box also features a *Detect Incompatibility* button that can guide you in the process of specifying incompatible objects. When you click a table and then click this button, the universe design tool automatically checks those objects it considers as incompatible. You should view the incompatible objects proposed by *Detect Incompatibility* as suggestions, not final choices.

8.2.10 Resolving loops involving aggregate tables

When a database contains one or more aggregate tables, you should resolve any loops using contexts.

Example: Resolving a loop involving an aggregate table

A simple schema containing aggregate tables is shown below:



Note the following points in the schema:

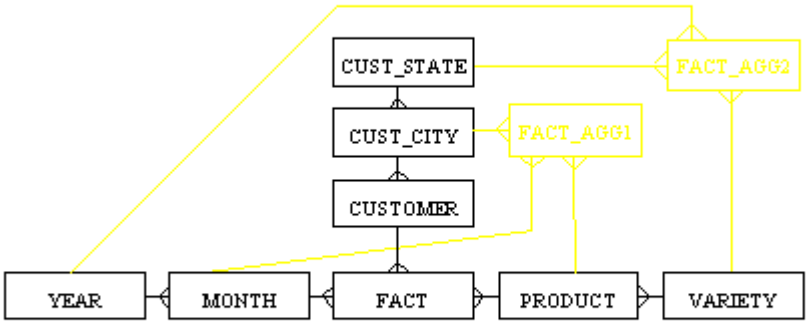
- FACT_AGG1 is an aggregate table that is nearly identical to the FACT table. It contains the (Customer) City Key, the Product Key, and the Month key in addition to a number of measures aggregated to Customer City, Product and Month.
- FACT_AGG2 is also an aggregate table similar to the FACT table. Its measures are aggregated to Customer State, Product and Year.
- The measures (the key performance indicators) are stored in all the fact tables. Sales Revenue is stored in FACT_AGG1, FACT_AGG2 and FACT, but is aggregated to the respective levels of each table.

For a query with sales Revenue and Customer State, you want to use the join between CUST_STATE and FACT_AGG2 rather than the join between CUST_STATE and CUST_CITY.

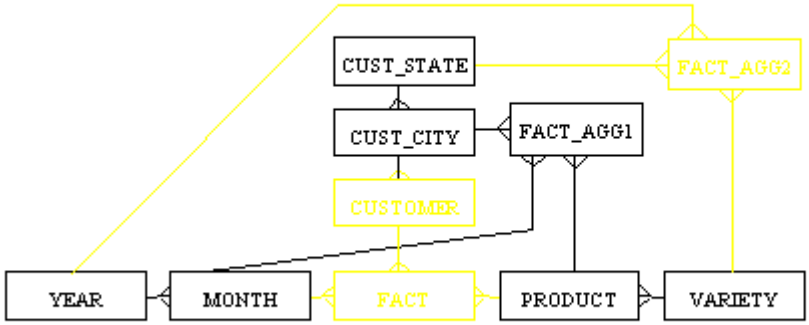
However, before you can run this query, you need to define three contexts, for example FACT, FACT_AGG1 and FACT_AGG2. You do not need to rename the context with more meaningful labels as they are transparent to the users.

The joins included in the three contexts are illustrated on the next page. In each schema, the darker set of joins represents the given context.

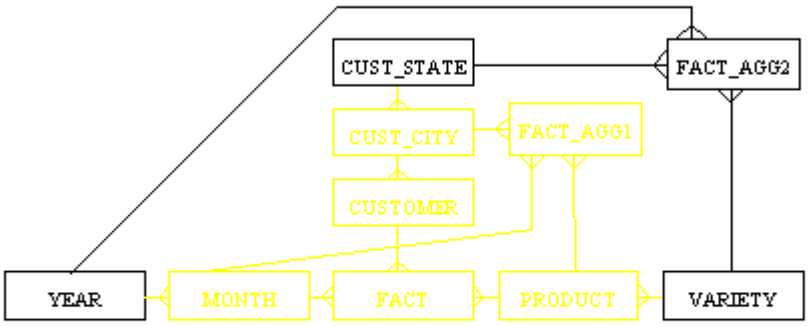
The FACT context



The FACT_AGG1 context



The FACT_AGG2 context



8.2.11 Testing aggregate awareness

The final step in setting up aggregate awareness is to test the results in Web Intelligence.

Based on the first example, we can run the following queries and then compare the different results:

- Order Year against Sales Revenue.
- Order Quarter against Sales Revenue.
- Order Month against Sales Revenue.
- Customer against Sales Revenue.
- Product against Sales Revenue.

8.3 Using @Functions in the SQL of an object

`@Functions` are special functions that provide more flexible methods for specifying the SQL for an object. `@Functions` are available in the *Functions* pane of the *Edit Select* box for an object.

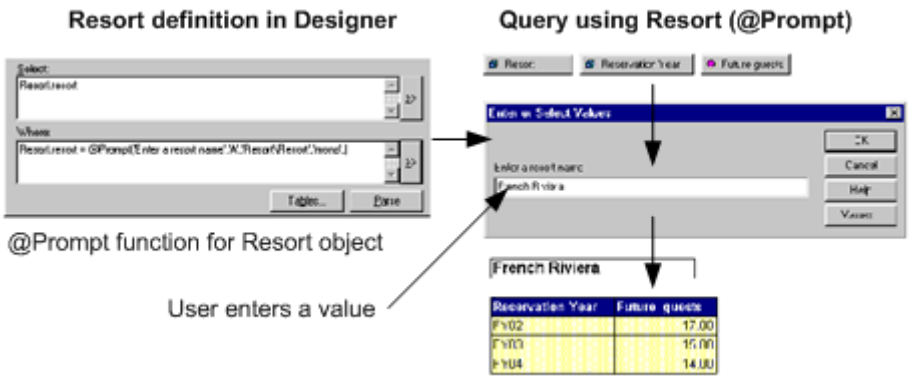
You can incorporate one or more `@Functions` in the SELECT statement or the WHERE clause of an object. The following `@Functions` are available:

@Function	Description	Usually used in object
@Aggregate_Aware	Incorporates columns containing aggregated and dimension data into objects.	SELECT statement
@Prompt	This inserts a prompt in the SQL. When the user runs the query, the user is prompted to enter a value for a restriction each time the object using the `@Prompt` function is included in a query.	SELECT statement WHERE clause
@Script	Runs a script each time the object using the `@Script` function is included is a query.	WHERE clause
@Select	Allows you to use the SELECT statement of another object.	SELECT statement
@Variable	Calls the value of a variable stored in memory. For example in a referenced text file.	WHERE clause
@Where	Allows you to use the WHERE clause of another object.	WHERE clause

Example: Using the `@Prompt` function to restrict returned values to entered prompt value

The `@Prompt` function is one of the `@Functions` available in the universe design tool. You can use the `@Prompt` function to display a message box when an object is used in a Web Intelligence query.

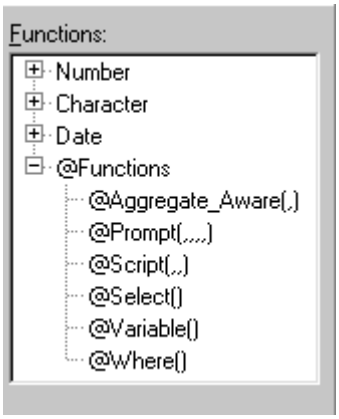
The message box prompts a user to enter a value for the object. The query returns values for the entered prompt value as shown below:



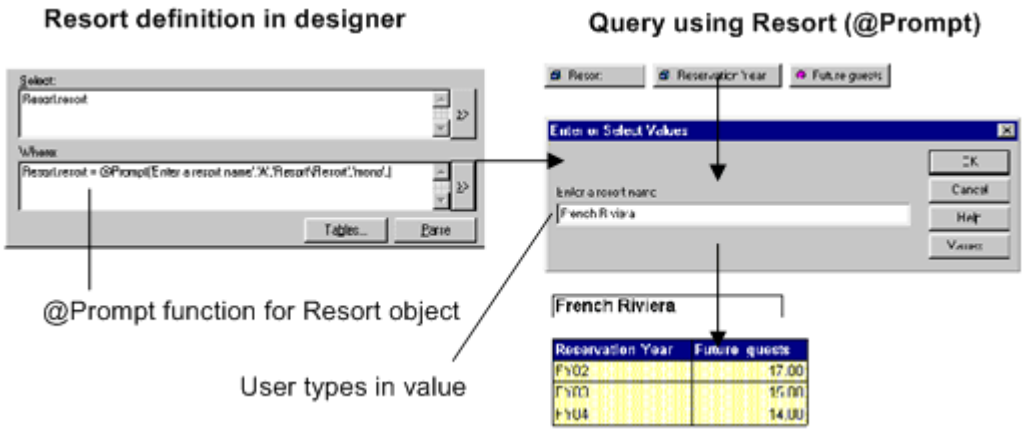
8.3.1 Inserting an `@Function` in an object

To insert an `@Function` in the SQL definition for an object:

1. Double-click an object in the *Universe pane*.
   The *Edit Properties* dialog box for the object appears.
2. Click the >> button next to the *Select* box.
   Or
   Click the >> button next to the *Where* box.
   The *Edit Select statement* or *Edit Where clause* dialog box appears.
3. Click in the `Select` statement or `Where` clause at the position where you want to add the `@Function`. If the box is empty as above, click anywhere in the box. The cursor automatically appears at the top left corner of the box.
4. Click the *@Functions* node in the *Functions* pane.
   The list of available `@Functions` appears.



5. Double-click an `@Function`.
   The syntax for the `@Function` is added to the SELECT statement or WHERE clause of the object. A description of the syntax appears in the *Description* box at the bottom of the dialog box. You can use this to help you type the parameters for the `@Function`.



6. Type the necessary parameters in the upper pane of the dialog box.
7. Click *Parse* to verify the syntax.
8. Click *OK* in each of the dialog boxes.

8.3.2 @Aggregate_Aware

The `@Aggregate_Aware` function allows an object to take advantage of tables containing summary data in the database. If your database contains summary tables and you are running queries that return aggregated data, it is quicker to run a SELECT statement on the columns that contain summary data rather than on the columns that contain fact or event data. Objects that are declared as not compatible with aggregate tables will not be able to use the aggregate tables, but will use the base tables for the query instead.

You can use the `@Aggregate_Aware` function to set up aggregate awareness in a universe. This process includes a number of other steps which are associated with the use of the `@Aggregate_Aware` function:

- Specify the incompatible objects for each aggregate table.
- Resolve any loops for the aggregate tables.
- Test the aggregate tables to ensure that they return the correct results.

8.3.2.1 Syntax for the @Aggregate_Aware function

The syntax of the `@Aggregate_Aware` function is as follows:

```
@Aggregate_Aware(sum(agg_table_1), ...
                         sum(agg_table_n))
```

You must enter the names of all aggregate tables as arguments. Place the names of the tables from left to right in descending order of aggregation.

Syntax	Description
`agg_table_1`	Is the aggregate with the highest level of aggregation.
`agg_table_n`	Is the aggregate with the lowest level of aggregation.

Example

```
@Aggregate_Aware(    R_Country.Revenue,
          R_Region.Revenue,
          R_City.Revenue,
          R_Customer.Revenue,
          R_Age_Range.Revenue,
          sum(Invoice_Line.days * Invoice_Line.nb_guests * Service.price)
          )
```

In the example, when an object using this function is declared as incompatible with any tables, those tables are ignored. For an object that is incompatible with the R_Country table and the R_Region table, then the SQL used becomes:

```
@Aggregate_Aware(    R_City.Revenue,
          R_Customer.Revenue,
          R_Age_Range.Revenue,
          sum(Invoice_Line.days * Invoice_Line.nb_guests * Service.price)
          )
```

8.3.3 @Prompt

Use the `@Prompt` function to insert a prompt in a query. Prompts can be used to restrict the data or to make large value objects easier to use when a user creates a report. You use the `@Prompt` function in the SELECT statement or WHERE clause for an object. It forces a user to enter one or more values for a restriction when that object is used in a query, or to select a value or a list of values. When the user runs the query, a prompt box appears asking for a value to be entered.

`@Prompts` are useful when you want to force a restriction in the inferred SQL but do not want to preset the value of the condition.

You can optionally define default values for prompts. A prompt containing default values behaves in the same way as a Web Intelligence prompt with default values.

You can edit the @Prompt definition in the following ways:

- Use the @Prompt Editor.
- Type the definition in the *SELECT* or *WHERE* field in the *Definition* pane of the *Edit properties* dialog for a condition.
- Type the definition in the *Advanced* edit pane of the *Edit properties* dialog.

> **Note**
>
> When you use two prompts that are very similar but use, for example, a different primary key, do not use the same question (prompt text). This is because the system will not be able to distinguish between the two prompts, and may provide an answer that is not appropriate.

> **Note**
>
> Due to the potential complexity of the `@Prompt` definition, especially when typing in a list of values, it is recommended to use the `@Prompt` editor.

> **Note**
>
> You can merge an `@Variable` function with an `@Prompt` function in the same query when the `@Prompt` function is monovalue.

Related Information

The @Prompt Editor [page 349]
Editing an existing @Prompt expression [page 354]
Syntax for the @Prompt function [page 361]
Manually defining the @Prompt function for an SQL statement [page 355]

8.3.3.1 The @Prompt Editor

The *@Prompt* editor allows you to define or edit prompts that appear when you are using BI client tools such as Web Intelligence to run a query on either relational or OLAP universes. The editor simplifies the process of defining or editing a prompt. The prompt definition is displayed at the bottom of the pane and is automatically updated when you define the different values for the prompt. When you right-click in an existing @Prompt expression, the *Edit @Prompt* menu item is enabled only when the syntax of the @Prompt string is correct.

You can also type the `@Prompt` definition directly in the *SELECT* or *WHERE* pane of the *Edit properties* dialog box.

Related Information

Syntax for the @Prompt function [page 361]
Manually defining the @Prompt function for an SQL statement [page 355]
Defining a static list of values for a prompt [page 352]
Selecting a universe object as a list of values for a prompt [page 352]
Selecting a custom list of values for a prompt [page 353]
Selecting a universe object as a list of values for a prompt [page 352]
Editing an existing @Prompt expression [page 354]

8.3.3.2 Properties of the @Prompt expression for the @Prompt editor

You can edit the following properties for the prompt:

Property	Description
Message	The prompt message that the user sees. For example, you can type 'Select a country'. The prompt text (question). Default = Enter values

Property	Description
Value Type	The data type that the user enters or selects. This ensures that the user enters or selects the correct type of data. Choose from: • Alphanumeric (A) • Numeric (N) • Date (D) Default = Alphanumeric
Key Type	When Primary_key is selected as the Selection mode, set the key type that the user enters or selects. Choose from: • Alphanumeric (A) • Numeric (N) • Date (D) Default = None
Allow multiple selection	When this option is selected, the user can enter or select more than one value. Default = Not selected - the user can only select or enter a single value
Selection mode	Define how the user selects the values required by the prompt. Select: • Free: The user can enter any value • Constrained: The user must choose from proposed values • Primary_key* Using a primary key value in a query dramatically increases the response time. You choose or enter the primary keys for the objects in the universe. At prompt time, the user selects the object name, but the query actually uses the corresponding primary key value. *You can only use the primary key setting if Index Awareness is set on the database tables. Default = Free
Keep last value selected	When selected, the last used values are proposed the next time the user runs the query. When this option is not selected, the default values are always used. Default = Not selected
Display Values	The user must select one or more items from your list of values. You can define the list of values in the following ways: • Type a list of values in the List of Values box (This is a static list) • Select an object from the universe • Import a file using the file input wizard Default = Static
Default values	You can declare default values for the prompt.

Related Information

Syntax for the @Prompt function [page 361]
Manually defining the @Prompt function for an SQL statement [page 355]
Defining a static list of values for a prompt [page 352]
Selecting a universe object as a list of values for a prompt [page 352]

Selecting a custom list of values for a prompt [page 353]
Selecting a universe object as a list of values for a prompt [page 352]
Editing an existing @Prompt expression [page 354]

8.3.3.3 Using the Prompt Editor to create an @Prompt expression

A universe is open in the universe design tool and you are creating or editing an object.

1. In the condition's *SELECT* or *WHERE* field, right-click at the place in the expression where you want to add a prompt and select *@Prompt editor* from the shortcut menu.
2. In the *Message* box, type the message that you want the user to see.
3. Select *Keep last value selected* if you want the prompt to propose the values used for the previous report.
   When the report is run for the first time, the default values (if any) are proposed.
4. Select *Allow multiple selection* if the user can enter or select more than one value.
5. Set the *Selection mode*. For *Free* selection, the user can enter any allowed value. For *Constrained* selection, the user must select a value from a list of values. For *Primary key*, the user selects the object name, but the object's primary key is used by the query. You can only use the *Primary key* setting if Index Awareness is set.

   > **i Note**
   >
   > When the *Constrained* option is selected and a List of Values is not specified, the corresponding Tab color turns to red, OK button is disabled, and on moving the cursor over the highlighted tab, the contextual tooltip displays a message indicating how to rectify the problem.

6. Set the *Value type* for the selection mode: *Alphanumeric*, *Number*, or *Date*.
7. If you have selected *Primary key* for the selection mode, set the *Key type*: *Alphanumeric*, *Number*, or *Date*.
8. If you are using multiple selection, define the list of values. Do one of the following: you can type or import a list of values, or you can select a universe object.
9. Click the *Default Values* tab and define any default values. When you allow multiple selection, you can set more than one default value.
10. Click *OK*.
    The *@Prompt* editor closes and the prompt expression is visible in the condition's statement in the *Edit Properties* pane.
11. Validate the prompt and click *Parse*.
    When the syntax is incorrect, a *Parse failed* message displays indicating the fragment of SQL that contains the error.

Related Information

Syntax for the @Prompt function [page 361]
Manually defining the @Prompt function for an SQL statement [page 355]
Defining a static list of values for a prompt [page 352]

Selecting a universe object as a list of values for a prompt [page 352]
Selecting a custom list of values for a prompt [page 353]
Selecting a universe object as a list of values for a prompt [page 352]
Editing an existing @Prompt expression [page 354]

8.3.3.4 Defining a static list of values for a prompt

The lower part of the *@Prompt* editor contains a table pane that you use to define a static list of values. You can use the up and down arrows to change the position of the values you enter. When the *Caption* title text is red, you must either complete values or correct values.

1. Enter the first value in the *Caption* field. If you have selected *Primary key* as the *Selection mode*, type an index value in the second field.
2. Click + to insert the value in the static list of values table.
3. Enter more values in the table to complete your list of values.
4. Enter any default values in the *Default values* pane.
   The default values are proposed when the user runs a report. When *Keep last value selected* is set, the values used the last time the report was run are proposed, otherwise the default values are proposed each time a report is run.

> **i** Note
>
> To edit a value in the static list, select it, then edit the value in the and click the update button.

> **i** Note
>
> To delete a value, select the value and click -.

Related Information

Syntax for the @Prompt function [page 361]
Manually defining the @Prompt function for an SQL statement [page 355]
Selecting a universe object as a list of values for a prompt [page 352]
Selecting a custom list of values for a prompt [page 353]
Selecting a universe object as a list of values for a prompt [page 352]
Editing an existing @Prompt expression [page 354]

8.3.3.5 Selecting a universe object as a list of values for a prompt

You can select a universe object as a list of values for a prompt.

> **Note**
>
> When creating row access restriction, you cannot use *Universe Object* in the *@Prompt* editor or function.

1. In the *@Prompt* editor, select *Universe Object* in the list of values pane.
2. Navigate to and select the universe object you want to use in the prompt.
3. Double-click on the object.
   The *@Prompt* editor closes and the prompt expression is visible in the condition's statement in the *Edit Properties* pane.

Related Information

Syntax for the @Prompt function [page 361]
Manually defining the @Prompt function for an SQL statement [page 355]
Defining a static list of values for a prompt [page 352]
Selecting a custom list of values for a prompt [page 353]
Selecting a universe object as a list of values for a prompt [page 352]
Editing an existing @Prompt expression [page 354]

8.3.3.6 Selecting a custom list of values for a prompt

The *Data File Input Wizard* allows you to import a custom list of values into the Caption table of the *@Prompt* editor for insertion into the `@Prompt` function. Only Excel and text files are supported.

1. In the *@Prompt* editor, select *File Input* in the list of values pane.
   The *Data File Input Wizard* appears.
2. Click *Browse* and browse to the input file you want to use.
3. Click *Open*.
4. Click *Next*.
5. Select *First row specifies column header* if the first row in your list of values file is used as the header or title values for the columns in the list.
6. Select the *File encoding* type
7. Set the *Delimiter* character that is used to separate data in your input file.
8. Click *Next*.
9. Click *Get unique column values* to ensure that only unique values are proposed.
10. Click *Get number of TOP records* to define the number of values that will be proposed to the user.
11. Use the *Column map* pane to select select the columns used for the caption and the primary key value.
12. Click *Sort on column* to sort the selected column, select *Ascending* to sort the column ascending order. If *Ascending* is not selected, the column is sorted in descending order.
13. Click *Finish*
    The *Column row append* confirmation message appears. Click *Yes* to confirm your choice.

14. The custom list of values is inserted in the list of values table in the *@Prompt* editor and the values are inserted in the `@Prompt` definition.
15. Click *OK* to insert the `@Prompt` in the *SELECT* or *WHERE* clause.

Related Information

Syntax for the @Prompt function [page 361]
Manually defining the @Prompt function for an SQL statement [page 355]
Defining a static list of values for a prompt [page 352]
Selecting a universe object as a list of values for a prompt [page 352]
Selecting a universe object as a list of values for a prompt [page 352]
Editing an existing @Prompt expression [page 354]

8.3.3.7 Editing an existing @Prompt expression

A universe is open in the universe design tool. The selected object expression contains an `@Prompt` function.

You want to edit an `@Prompt` function that already exists within the definition or where clause of an object or condition. The edit is done through the *@Prompt* editor.

1. Right-click on the `@Prompt` function and select *Edit prompt* from the shortcut menu.
2. Edit the prompt expression using the *@Prompt* editor.
3. Click *Parse* to check the syntax of the `@Prompt` .

Related Information

Syntax for the @Prompt function [page 361]
Manually defining the @Prompt function for an SQL statement [page 355]
Defining a static list of values for a prompt [page 352]
Selecting a universe object as a list of values for a prompt [page 352]
Selecting a custom list of values for a prompt [page 353]
Selecting a universe object as a list of values for a prompt [page 352]

8.3.3.8 Manually defining the @Prompt function for an SQL statement

> **Note**
>
> If you have not defined an @Prompt function before, you are advised to use the @Prompt editor. The syntax of the @Prompt function is complex, so please refer to the syntax definition page.

To define the `@Prompt` function manually for an object, do the following:

1. Open the of the *Edit properties* window: double-click a class, object or condition in the universe pane, or right-click an object and select *Object Properties*, or select *Edit* > *Propertes*.
2. Click in the SELECT or WHERE dialog of the *Definitions* pane and define the @Prompt values according to the syntax you require (see the links below).
3. Click *Parse* to check the syntax of the SQL.

Related Information

Syntax for the @Prompt function [page 361]
The @Prompt Editor [page 349]
Editing an existing @Prompt expression [page 354]
Defining a static list of values for a prompt [page 352]
Selecting a universe object as a list of values for a prompt [page 352]

8.3.3.9 Properties of the @Prompt expression for manually defining a prompt

The Syntax for the @Prompt is as follows:

```
@Prompt('message','type','lov',Mono|Multi,free|constrained|primary_key
,persistent|not_persistent,{'default value'})
```

An example is as follows:

```
@Prompt('Displayed text ','A','Store\City',Mono,constrained,Persistent,{'Paris'})
```

The properties of the @Prompt expression values are described in the following table:

Property	Description
`'message'`	Mandatory Text of the prompt message. The text must be enclosed between single quotes, for example, 'Choose a Region', 'Pick a time period', or 'Choose a showroom'. The text appears in the prompt box when the user runs the query.
`'type'`	Mandatory, but can be empty (the 'A' setting is taken as default). The data type of the third parameter. It can be one of the following: • 'A' for alphanumeric • 'N' for number • 'D' for date The specified data type must be enclosed in single quotes. When using a hard-coded list of pairs of values, the syntax is: `'value_type:key_type'` , for example: `'A:N'` where the first value is the caption that the end user sees, and the second value is the primary key value that is actually used by the query in order to speed up the query. Each type (caption and primary key) can be A, N, or D as specified above. For example: 'A:A' or 'A:N'. In this case, the next parameter, 'lov', contains a list of pairs of parameters. Similarly, the 'default_value' parameter will contain pairs of values. Index awareness must be set when you use primary key.

Property	Description

`lov`	Mandatory, but can be empty. If you have an empty list, the comma is obligatory. When this parameter is a universe object, the fifth parameter (selection mode= free|constrained|primary key) must be `primary_key` and Index Awareness must be set in the universe.
	
	You can specify two types of list of values:
	
	• Pointer to a List of Values from an existing universe object. You invoke the target list of values by double-clicking on the object containing the list of values that you want to use in the *Classes and Objects* panel. This gives the Class name and the Object name, separated by a backslash. It must be enclosed in single quotes. For example: 'Client\Country'. When you are using Index Awareness, and you want to return the key values for an object, set the 5th value to `primary_key`
	• Hard-coded list of single values or value pairs. The values in a pair are separated by a colon. Each value is enclosed in single quotes. The pairs of values are separated by a comma. The whole list is enclosed in curly brackets. Set the constraint to primary_key.
	The syntax for a single default value:
	`{'value'}`
	The syntax for several single default values:
	`{'value1','value2',... ,'valuen'}`
	You can define pairs of default values.
	The syntax for a pair of default values is: `{'value':'key'}`.
	The colon (:) is the separator between the value and the key.
	The syntax for pairs of default values is:
	`{'value1':'key1','value2':'key2',...,'valuen':'keyn'}`
	For example: `{'Australia':'A', 'France':'F', 'Germany':'G', 'Japan':'J', 'Spain':'S', 'United Kingdom':'UK'}`

`Mono|Multi`	Mandatory, but can be empty (the `Mono` setting is taken as default). The comma is obligatory. Use `Mono` if the user can select only one value from the list of values. Use `Multi` if the user can select multiple values from the list of values.
`free|constrained|primary_key`	Mandatory, but can be empty (the `free` setting is taken as default). The comma is obligatory. Use `free` if the user can enter a value, or select one from the list of values. Use `constrained` if the user must select a value from the list of values. Use the `primary_key` parameter when you have set Index Awareness in the universe. The associated key value for the object is used rather than the entered or displayed value. When the lov parameter is used, `primary_key` is obligatory.

Property	Description
`persistent|not_persistent`	Optional. When this is set, end the argument with a comma. When this is not set, but there is a 7th parameter set (default values), you must also set the comma. **i Note** Note that this parameter has no effect in Desktop Intelligence. Use `persistent` if, when refreshing a document, the last values used in the prompt are displayed by default, even when default values are defined. Use `not_persistent` if, when refreshing a document, no values used are displayed in the prompt by default.
`'default value'`	Optional. The `'default value'` parameter is used to define default values presented to the user. When you use a hard-coded list, the default values entered here must be present in the `[lov]` list. For example, for a single value: {'France'} For a pair of values: {'France':'F'} For two pairs of values: {'France':'F','Germany':'G'} When refreshing a document, these values are displayed by default, but if the `persistent` option is set, then the last values used in the prompt are used instead of the default values. You can have single values or pairs of values. If you specify the `primary_key` parameter in the prompt definition, then you must provide the key values.

8.3.3.10 Examples: Using the @Prompt function

The following are examples of @Prompt syntax.

Minimal use of the @Prompt function:

```
@Prompt('Displayed text ','A',,,)
```

Using the @Prompt with a LOV without default values:

```
@Prompt('Displayed text ','A',{'Paris','London','Madrid'},,)
```

Using the @Prompt with a LOV and one default value:

```
@Prompt('Displayed text ','A',{'Paris','London','Madrid'},,,,{'Paris'})
```

Using the @Prompt with an object and a default value:

```
@Prompt('Displayed text ','A','Store\City',,,,{'Paris'})
```

Using the @Prompt with all possible settings:

```
@Prompt('Displayed text ','A','Store\City',Mono,Constrained,Persistent,{'Paris'})
```

Using the @Prompt with a LOV containing pairs of values, without default values:

```
@Prompt('Displayed text ','A:N',{'Paris':'12','London':'7','Madrid':'15'},,)
```

Using the @Prompt with a LOV containing pairs of values, and one default pair of values:

```
@Prompt('Displayed text ','A:N',{'Paris':'12','London':'7','Madrid':'15'},,,,
{'Paris':'12'})
```

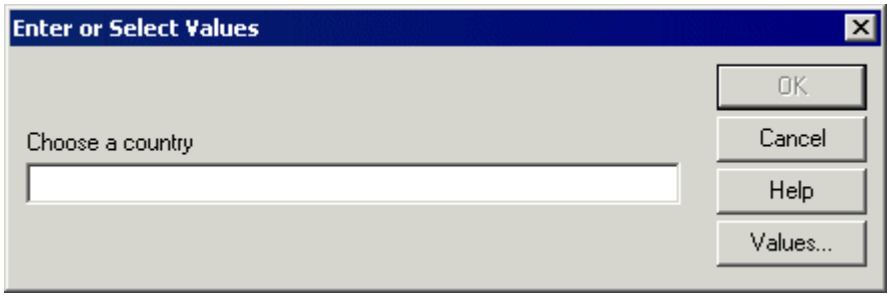
Example: Using @Prompt to restrict countries without default values

The object Country returns values for the countries of resorts. If you want to restrict the returned values to resorts for only one country, you would need a separate object for each resort country in the universe. However, using the `@Prompt`, you need only one object as follows:

```
Country.country=@prompt('Choose a country',
'A','Customer\Country of origin',Mono,Constant)
```

The user is prompted to enter the name of the country, and the returned values are the resorts from that particular country.

When a query is run in Web Intelligence, the following prompt box appears:



Example: @Prompt syntax with default values

```
@prompt('Enter value(s) for Customer with IA:',
'A','Customer\Customer with I A',
Multi,primary_key,,{'Baker','Brendt'})
```

Example: @Prompt syntax using a hard-coded list of values

The following example shows how to propose a list of countries, with a default value. When the object is index aware and the constraint is set to primary_key, then the default values can be a set of pairs (value, key) such as: {'England: 21', 'Scotland:39'}. The user must choose only one region, so the Mono parameter is set. The default value must appear in the list of values.

```
SELECT dbo.region.sales_region
FROM dbo.region
WHERE dbo.region.region_id = @Prompt('Choose a region','A:N',
{'England':'21', 'Scotland':'39', 'Wales':'14'},
Mono, primary_key, Persistent, {'Scotland':'39'})
```

This feature will also allow you to perform the behavior of the CASE WHEN ELSE clause on databases that do not support it, especially OLAP databases.

Example: @Prompt syntax to create a predefined condition using a match pattern prompt

The example below allows the user to select a client name by entering the first letter of the name. If the Web Intelligence user enters H%, then the report returns all the clients with the last name starting with H.

```
(@Select(Client\Client Name)
    LIKE (@Prompt('enter','A',,,)+%)
```

To allow the client to use either uppercase or lowercase letters, the syntax is:

```
(@Select(Client\Client Name)
    LIKE lower(@Prompt('enter','A',,,)+%) OR
 (@Select(Client\Client Name)
    LIKE upper(@Prompt('enter','A',,,)+%)
```

8.3.3.11 Syntax for the @Prompt function

The syntax for the `@Prompt` function is complex due to the versatility of this function. You write the prompt a message and specify the data type, if the data is a single value or multiple values, if the data is persistent, and you can specify default values. The syntax is shown below:

```
@Prompt('message','type',[lov],Mono|Multi,
free|constrained|primary_key,persistent|not_persistent,
[default_values])
```

Related Information

@Prompt [page 348]
Manually defining the @Prompt function for an SQL statement [page 355]
Properties of the @Prompt expression for manually defining a prompt [page 355]

8.3.4 @Select

You can use the `@Select` function to re-use the SELECT statement of another object. When the `@Select` function is used in the SELECT statement of an object, it specifies the path of another object in the universe as a parameter of the `@Select` function, in the form Class_Name\Object_Name. This then acts as a pointer to the SELECT statement of the referenced object.

Using the `@Select` function has the following advantages:

- You have to maintain only one instance of the SQL code.
- Ensures consistency of the code.

> **i** Note
>
> When you use `@Select` and `@Where` functions, one object now depends on another in the universe. You have created a new object dependency. When one object is deleted, the other object using the `@Select` or `@Where` function needs to be manually updated.

8.3.4.1 Syntax for the @Select function

The `@Select` function has the following syntax:

```
@Select(Classname\Objectname)
```

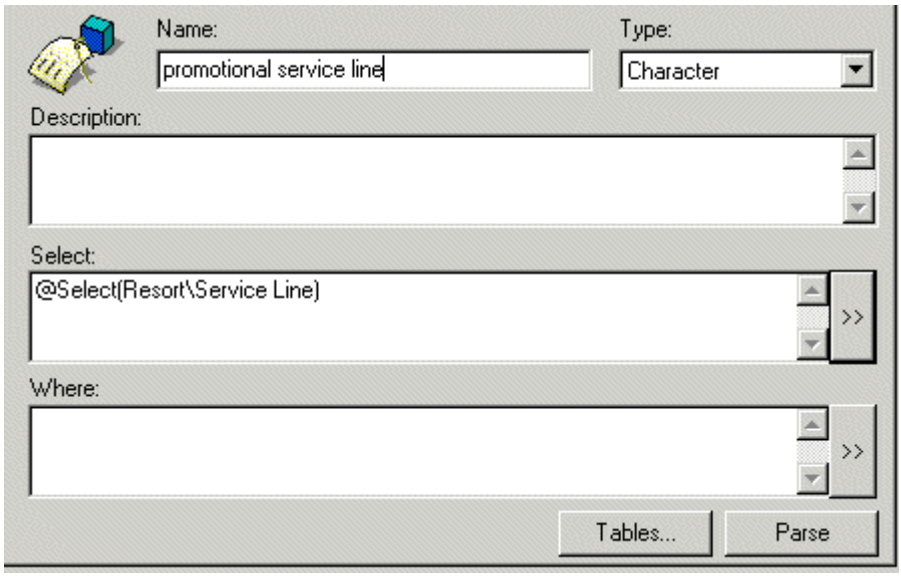
Syntax	Description
`Classname`	The name of the class that contains the referenced object.
`Objectname`	The name of the referenced object.

8.3.4.2 Example using the @Select function

Example: Using `@Select` to re-use the Service_line Select statement

You create an object called Promotional Service Line which is used to return service lines used in promotional campaigns for different resorts in the Club database. This object is in a new class called Promotions. You can use `@Select` to reference the existing SELECT statement for the Service_lines object.

The SELECT statement for Promotional Service Line appears below:



8.3.5 @Variable

The `@Variable` function is used, for example, in the WHERE clause to call the value assigned to one of the following types of variables:

- BusinessObjects system variables
- Report variables
- Language (Locale) variables
- Operating system variables

In most cases, you insert the `@Variable` on the operand side of the condition in the WHERE clause for an object from the *Definition* page of its *Edit properties* sheet. The query retrieves the value for the variable.

> **Note**
>
> The `@Variable` is a monovalue function and cannot be used with the IN or INLIST operators.

> **Note**
>
> When the same `@Variable` function is executed several times in a query, the prompt only appears once.

> **Note**
>
> The `@Variable` function is equivalent to a single value `@Prompt` function with the following settings:
>
> ```
> @Prompt('Question','A',,mono,free)
> ```
>
> You can merge an `@Variable` function with an `@Prompt` function in the same query when the `@Prompt` function is monovalue.

Related Information

Syntax for the @Variable function [page 363]
@Variable property descriptions [page 364]

8.3.5.1 Syntax for the @Variable function

The `@Variable` function has the following syntax:

```
@Variable('<Variablename>')
```

> **Note**
>
> The variable name must be inside single quotes.

Example: @Variable syntax to return the BOUSER value

```
@Variable('BOUSER')
```

Related Information

@Variable [page 362]
@Variable property descriptions [page 364]

8.3.5.2 @Variable property descriptions

In all cases, the variable name must be enclosed by single quotes.

Variable name	Description
BusinessObjects system variables • BOUSER - user login • DBUSER - database user name • DBPASS - database user password	Values for the BusinessObjects system variables. The returned data is then restricted based on that BusinessObjects user's login. Values for the BusinessObjects declared database user.
Report variables • DOCNAME - the name of the document • DPNAME - the name of the Data Provider • DPTYPE - the type of the Data Provider • UNVNAME - the name of the universe • UNVID - the ID of the universe used	These variables can be referenced in, for example, the Begin_SQL parameter that will be executed before the SELECT statement. This can be used for audit purposes concerning the use of the database (For example: To determine which report query or which universe is used most frequently).
Language variables • PREFERRED_VIEWING_LOCALE • DOMINANT_PREFERRED_VIEWING_LOCALE	Language variables • PREFERRED_VIEWING_LOCALE - User's Preferred Viewing Locale. This is the same locale chosen by the user to display universe metadata. • DOMINANT_PREFERRED_VIEWING_LOCALE - The Dominant Locale of the user's Preferred Viewing Locale. This prevents the users translating data in all locales (fr_FR, fr_BE, fr_CA, ...). If translations are available in fr_FR, then if the user locale is fr_BE or fr_CA, since they share the same dominant locale, then they can reuse translations in fr_FR.
Operating system variables	You can enter Windows environment variables in order to obtain information about your installation.

Related Information

@Variable [page 362]
Syntax for the @Variable function [page 363]

8.3.5.3 Using the BusinessObjects system variable

You can use the `@Variable` function with BusinessObjects™ system variables to restrict data according to the identity of the currently logged in BusinessObjects™ user.

> **i** Note
>
> The BusinessObjects™ login parameters must be the same as the database login parameters.

The User Name assigned to each BusinessObjects™ user is held as the following BusinessObjects™ system variable:

- `BOUSER` - the username

This variable appears in the User Identification box when the user performs a logon to a Business Objects product.

You use the `@Variable` function in the WHERE clause for an object to restrict data access for a user and their database profile when the object is used in the query.

You insert the `@Variable` on the operand side of the condition in the WHERE clause for an object from the *Definition* page of its *Edit properties* sheet.

Example

Using `@Variable` to restrict employee access to employee data.

In the universe for a human resources database, you have an object called Employee name. You want to restrict the returned data for Employee name to the values authorized in the database for each user. This would allow you to control what employee information each user is allowed to see. This information is defined by their database profile.

You insert the `@Variable` function in the WHERE clause as follows:

```
Employees.Employee_Name = @Variable('BOUSER')
```

When the object Employee name is used in a query, the data is returned only for the value in the tables that matches the `BOUSER` value.

Related Information

@Variable [page 362]
Syntax for the @Variable function [page 363]
@Variable property descriptions [page 364]

8.3.5.4 Using locale variables

Use the locale variables of the `@Variable` function to define the locale settings so Web Intelligence retrieves reports and displays information in the appropriate locale. Your database tables must contain a column declaring the languages for rows that contain translations of the data. A locale defines a language and a geographical area, the way data is sorted, and how dates are formatted and other particular formats. Use the `@Variable` function in the WHERE clause for an object. It forces a user to select a locale when that object is used in a query. When the user runs the query, a prompt box appears prompting the user to enter the locale. There is a list of Local Codes and Dominant Locale Codes in the translation management tool Guide.

The settings you can define are:

- `@Variable('PREFERRED_VIEWING_LOCALE')`
- `@Variable('DOMINANT_PREFERRED_VIEWING_LOCALE')`

Example

The PRODUCT table below has been translated in many languages. The user wants to list the product names in a specific locale.

PRODUCT Table

Product ID	LOCALE	Product_Name
DC1212	en_GB	Digital camera
DC1212	fr_FR	Appareil photo numérique
DC1212	de_DE	Digitalkamera
DC1212	es_ES	Cámaras digitales
...	...	...

```
SELECT Product_Name

FROM PRODUCT

WHERE PRODUCT.LOCALE = @Variable('PREFERRED_VIEWING_LOCALE')
```

At query time the user replaces the variable by the correct locale value and Web Intelligence retrieves the information in that locale.

Related Information

@Variable [page 362]
Syntax for the @Variable function [page 363]
@Variable property descriptions [page 364]

8.3.5.5 Using report variables

You use the `@Variable` function in the WHERE clause of an object to include report variables in the request.

These variables can be referenced in the `Begin_SQL` parameter that will be executed before the SELECT statement. This can be used for audit purposes concerning the use of the database (For example: To determine which report query or which universe is used most frequently).

The variables can be referenced in:

- The definition of an object: SELECT, WHERE clauses, etc ...
- Filters
- The Join expression
- The Begin_SQL parameter

Related Information

@Variable [page 362]
Syntax for the @Variable function [page 363]
@Variable property descriptions [page 364]

8.3.5.6 Using operating system variables

You can enter windows environment variables in order to obtain information about your installation. For example, NUMBER_OF_PROCESSORS, USERNAME.

Example

If you include the following `@Variable(NUMBER_OF_PROCESSORS)` in a query, the response will include the number of processors on the machine you are using.

Related Information

@Variable [page 362]
Syntax for the @Variable function [page 363]
@Variable property descriptions [page 364]

8.3.5.7 Using custom variables with Desktop Intelligence

With Desktop Intelligence, you can use `@Variable` function in the WHERE clause of an object to reference a variable in an associated text file. This allows you to define user specific conditions on an object.

To use this variable, BusinessObjects needs to be launched by a command line that includes the `-vars` parameter. You will need to change the command line in Windows shortcuts on all PCs that use this feature.

> **i** Note
>
> Ensuring that BusinessObjects is launched by a command line makes using the `@Variable` function difficult to maintain for universe deployments of more than a few users. If you have more than a few users, or a geographically diverse user base, you should not use `@Functions` with associated text files to implement restrictions.

Advantages of using the `@Variable` function with text file variables:

The principle advantage for using the `@Variable` function with text file variables is that you can update the values for the variables in the text file without making any changes to the universe.

Disadvantages of using the `@Variable` function with text file variables:

- The command string must be changed on every client post to include the `- vars <textfile.txt>` argument
- Security can be a problem, as a text file on a PC can be modified locally.

Given the number of potential problems using the `@Variable` function with text variables, if you are using Business Objects products in an enterprise environment, then you should use the security options available in Supervisor to control access to data.

Related Information

To use custom variables with Desktop Intelligence [page 368]
@Variable [page 362]
Syntax for the @Variable function [page 363]
@Variable property descriptions [page 364]

8.3.5.7.1 To use custom variables with Desktop Intelligence

Perform the following steps to use one or more predefiend variable values.

1. Create a text file that contains a list of variables with the corresponding values. Use the following format: Variable name = value
2. Add the following to a command line used to start BusinessObjects: `Busobj.exe -vars <textfile.txt>` For example, if you have a text file called Bovars.txt, you would type the following: `C:\BusinessObjects\Busobj.exe -vars Bovars.txt` The `-vars` syntax is a switch that tells the operating system to load the text file into memory for use by BusinessObjects.

3. Open the *Edit Properties* sheet for the object that you want to reference the text variable.
4. Insert the `@Variable` on the operand side of the condition in the WHERE clause. For example: COUNTRY.COUNTRY_NAME = `@Variable`('Country'). Country is the name of the variable in the text file. The term must be enclosed in single quotes.
5. Click *OK* and save the universe.

8.3.6 @Where

You can use the `@Where` function to re-use the WHERE clause of another object. When the `@Where` function is used in the WHERE clause of an object, it specifies the path of another object in the universe as a parameter of the `@Where` function, in the form `Class_Name\Object_Name`. This then acts as a pointer to the WHERE clause of the referenced object.

Using the WHERE clause creates a dynamic link between two objects. When the WHERE clause of the original object is modified, the WHERE clause of the referencing object is automatically updated.

Using the `@Where` function allows you to use existing code. This has the following advantages:

- You have to maintain only one instance of the SQL code.
- Ensures consistency of the code.

When you use `@Select` and `@Where` functions, one object now depends on another in the universe. You have created a new object dependency. When one object is deleted, the other object using the `@Select` or `@Where` function needs to be manually updated.

8.3.6.1 Syntax for the @Where function

The syntax of this function is the following:

```
@Where(Classname\Objectname)
```

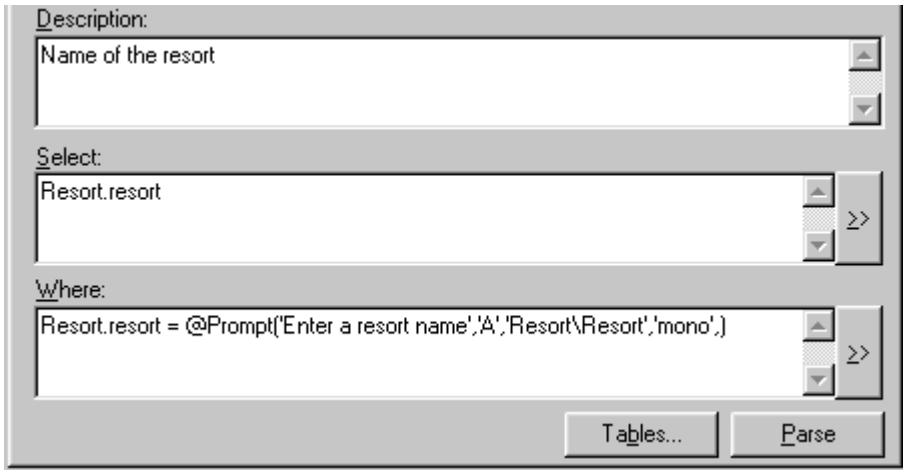
Syntax	Description
`Classname`	The name of a class.
`Objectname`	The name of the referenced object.

8.3.6.2 Example: Using the @Where function to re-use a WHERE clause

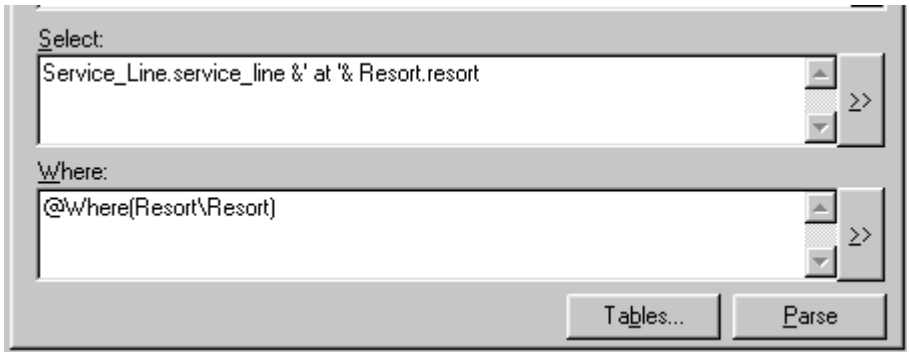
Example: Using `@Where` to re-use the Resort WHERE clause

You create an object called Resort Service Lines which is used to return service lines available at each resort. You want to reuse the `@Prompt` function defined in the Resort object, so that users are prompted to enter a resort name when they query the services available at that particular resort.

The SQL for the Resort object (the object that you want to reference) appears as follows:



The new object Resort Service Lines uses the `@Prompt` function in the WHERE clause for Resort as follows:



When you run a query with Resort Service Line, you are prompted to type the name of a resort. When you modify the WHERE clause for Resort, the change is automatically made in the Resort Service Line object.

8.4 Using external strategies to customize universe creation

The universe design tool uses built-in automated routines to automatically create universe components based on the database structure. These routines are called strategies and are available from the Strategies page of the Parameters dialog box (Files > Parameters > Strategies). These strategies are built-in to the universe design tool. You cannot access or modify them. The use and activation of strategies is described in the section Selecting strategies [page 83].

You can also create SQL scripts that follow a defined output structure to perform customized automatic universe creation tasks. You can select these from the Strategies page with the other strategies. These user defined and customized scripts are called External strategies.

This section describes external strategies and their use.

8.4.1 Migrating external strategies to the universe design tool

External strategies in the universe design tool versions previous to Universe Designer 6.5 were defined in an external text file called the st<xxxx>.txt file.This file is no longer supported.

> **i** Note
>
> If you are migrating from Universe Designer 6.5, external strategies are treated in the same way in the universe design tool.

To ensure that your customized and user defined external strategies used in previous versions are available from the universe design tool, you must do the following:

- Edit the new external strategy file (<RDBMS>.STG) as follows:
  - Open the external strategy file for your target RDBMS in an XML editor.
  - Create a new entry for each strategy.
  - For each strategy, copy the SQL script directly into the STG file using the SQL tag.

  Or
  - Enter a file path to reference the data in an external text file using the FILE tag.

  Both methods are described fully in the section Creating an external strategy [page 381].
- Copy the Help text to a second XML file (<RDBMS><language>.STG). This is described in the section Creating a Help entry for an external strategy [page 374].
- Verify that the external strategy file is declared in the general parameters file (SBO), not the parameters file (PRM), as was the case for previous versions of Universe Designer. This is described in the section Verifying that the external strategy file is declared [page 375].

8.4.2 External strategies overview

The table below provides an overview of the files used and their role in the creation and management of external strategies.

Roles and files in external strategies management process	Description
External strategies stored and created in External strategy file (<RDBMS>.STG).	XML file contains external strategy name, type, SQL script, or file reference to external text file containing data. File is stored here: $INSTALLDIR/dataAccess/RDBMS/connectionServer/<RDBMS>/<RDBMS>.stg. One file for each RDBMS. Uses the strategy.dtd file here: $INSTALLDIR/dataAccess/RDBMS/connectionServer/strategy.dtd Related sections: • How is the strategy file (STG) structured? [page 376] • Creating an external strategy [page 381]
Help text for external strategies stored and created in External strategy language file (<RDBMS><language>.STG)	XML file contains Help text for each external strategy in the external strategy file. This is the text that appears under an external strategy when it is selected on the Strategies page. File is stored here: $INSTALLDIR/dataAccess/RDBMS/connectionServer/<RDBMS>/<RDBMS><language>.stg. Uses the strategy_localization.dtd file located here: $INSTALLDIR/dataAccess/RDBMS/connectionServer/strategy_localization.dtd. Related section: Creating a Help entry for an external strategy [page 374].
External strategy file is declared in the general data access file (SBO) for the target RDBMS.	XML file contains the general data access parameters for a target RDBMS. The name of the external strategy file is set as the value for the parameter External Strategies by default. Related section: Verifying that the external strategy file is declared [page 375]

8.4.3 What is an external strategy?

An external strategy is an SQL script stored externally to the .UNV file, and structured so that it can be used by the universe design tool to automate object or join creation, and table detection tasks in a universe. External strategies are stored in an external strategy file with the extension STG. External strategy files are in XML format. There is one for each supported RDBMS.

External strategy files are stored in the following directory:

$INSTALLDIR/dataAccess/RDBMS/connectionServer/<RDBMS>/<rdbms>.stg

> **Note**
>
> You should use an XML editor to edit the external strategy file.

8.4.3.1 Accesing external strategies

External strategies appear in the drop-down list boxes that also list the built-in strategies on the Strategies page. Each drop-down list box corresponds to a strategy type category in the XML file. An external strategy appears in the list with External Strategy prefixing the strategy name as follows:

```
External Strategy:<strategy name>
```

For example, an external strategy for join creation called Constraints in the Strategy file, appears as External Strategy: Constraints in the joins drop-down list on the Strategies page

8.4.4 Creating Help text for external strategies

On the Strategies page, a commentary note appears under each selected strategy. This is the Help text for the strategy. For built-in strategies the Help text cannot be accessed or edited. However, you can access and edit the Help text for external strategies.

> **Note**
>
> In previous versions of the universe design tool the Help text was included in the strategy text file in the section [HELP]. The text in this section is now stored in a separate file, the external strategy language file described below.

8.4.4.1 External strategy Help text is stored in a separate file

The Help text for external strategies is stored in a separate external strategy language file called <RDBMS><language>.stg. For example, oaracleen.stg is the Help text file for the strategies in the oracle.stg file.

You can edit and customize Help text entries. The Help text should describe briefly what the strategy does to help designers who may not be familiar with the strategy.

For each external strategy that appears in the external strategy file, you should ensure that a corresponding entry with Help text appears in the external strategy language file.

There is a strategy language file for each language version of the universe design tool that you have installed. The external strategy language file is in the same directory as the external strategy file. For example, if you have a French version of the universe design tool, the external strategy language file for Oracle is oraclefr.stg. The English version is oracleen.stg.

When you create a new external strategy in the external strategy file, you also create an entry for the Help text in the external strategy language file. This provides information about the external strategy to other designers using the universe.

Example: Help text entry for the strategy shipped with Oracle data access driver

The Help text for the strategy Classes and Objects listed in the oracleen.stg file is shown below. This is the Help text for the Classes and Strategies external strategy defined in the file oracle.stg.

```
<Strategy Name="Classes_and_Objects">
        <Message id="Help">This strategy reads the database structure. It
associates tables with classes, and columns with objects.</Message>
        <Message id="Name">External Strategy: Classes and Objects</Message>
```

8.4.4.2 Creating a Help entry for an external strategy

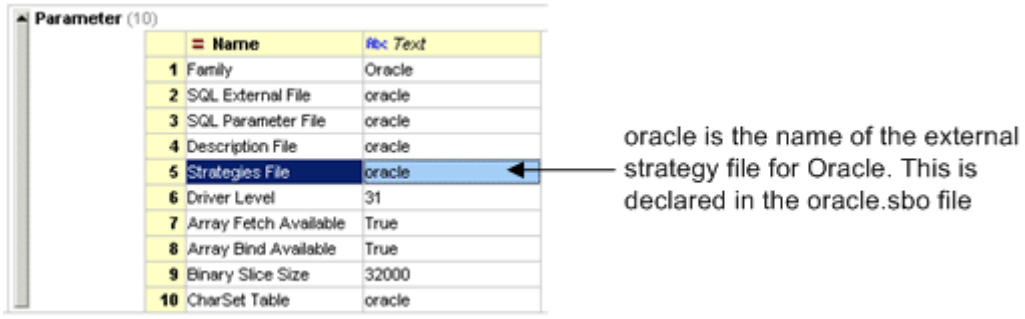
To create a Help entry for an external strategy:

1. Open the external strategy language file for the target RDBMS in an XML editor. The external strategy language file for a target RDBMS is located here:
   `$INSTALLDIR/dataAccess/RDBMS/connectionServer/<RDBMS>/<RDBMS><language>.stg.`
   For example:
   `$INSTALLDIR/dataAccess/RDBMS/connectionServer/oracle/oracleen.stg.`
2. Create a new Name element.
3. Enter the name of the strategy. This is the strategy for which you are creating Help text.
4. Create a Message ID called Help. This tag contains the Help text.
5. Enter the Help text.
6. Create a Message ID called Name. This tag contains the name that you want to appear in the strategy drop down list when the external strategy is selected.
7. Enter a strategy name.
   Validate, save, and close the file.
   When you next start up the universe design tool , the Help text appears under the selected external strategy.

> → Tip
>
> An easy way to create and set parameters for a new Name element is to copy an existing Name element and fill in the new values for the new strategy.

8.4.5 Verifying that the external strategy file is declared

An external strategy file is declared in the Parameter section of the general parameter (SBO) file for the target RDBMS. For example, the external strategy file for Oracle is oracle.stg. It has the value oracle in the oracle.sbo file as shown below:



8.4.5.1 Verifying that the strategy file is declared in the SBO file

To verify that an external strategy file is declared correctly:

1. Open the SBO file for the target RDBMS.
2. Ensure that the parameter Strategies Name is set to the name of the external strategies file. This is the default setting.
3. If the name is not set correctly, enter the correct name of the external strategies file.
4. If you have made modifications, save and close the file.
   Or
5. If you have not made any modifications, close the file without saving.

> **i Note**
>
> External strategies in previous versions of the universe design tool were declared in the PRM file. This is no longer the case for Universe Designer 6.5. The Strategies File parameter in the SBO file is set to the name of the external strategies file for the target RDBMS by default. Refer to the section Creating Help text for external strategies [page 373] for full information on migrating external strategies to Universe Designer 6.5.

8.4.6 Using example external strategies

All external strategy files contain a number of existing strategies delivered with Business Objects products. For example, a file may contain one object strategy, one join strategy, and one table browser strategy, or multiple strategies of each type.

You can customize an example file, or use it as the basis to create a new external strategy. You can customize an existing strategy or create your own.

Save a copy of each file before modifying it.

8.4.7 How is the strategy file (STG) structured?

There is an external strategy file (STG) file in XML format for each supported RDBMS. You migrate existing or create new external strategies to this file. All external strategy files use the strategy dtd (<RDBMS>.dtd) file in the following directory:

$INSTALLDIR/dataAccess/RDBMS/connectionServer

The elements in the external strategy XML file are defined in the external strategy DTD file. If you are using certain XML editors, for example XML SPY, the available parameters are listed in a drop down list when you create a new strategy element.

The external strategy file contains one main section called Strategies. All the external strategies are defined in this section. The Strategies section has the following elements and parameters:

File element	Description
Strategy	Main element. All external strategies are created within this element.
Name	Name of the external strategy. This name appears in the drop down list on the Strategies page. Default element.
Type	The list that the external strategy appears in on the Strategy page. There are 3 values: • JOIN: Join strategy appears in the Joins list. • OBJECT: Classes and objects strategy appears in the Classes and Objects list. • STRUCT: Table detection strategy appears in the Tables list.
SQL	The SQL code for the script. This is the SQL script that the universe design tool runs when the strategy is selected. The SQL script must follow a specific output format for object and join creation, and table detection routines to run correctly. See the section The output format of object strategies (OBJECT) [page 378] for information on structuring the SQL for an external strategy.
Connection	Specify a database connection. The connection type must be personal.

File element	Description
SkipMeasures	When set to Y, it skips the screen in the Quick Design wizard that deals with the creation of measures:
File	File path of an external text file that contains data organized in a specific output format that creates a universe automatically. See the section Creating a text file for data [page 382] for more information.

Example: Classes and Objects external strategy in oracle.stg

The external strategy file for Oracle is oracle.stg. It is stored in the directory $INSTALLDIR/dataAccess/RDBMS/connectionServer/oracle/oracle.stg. This file contains a number of example external strategies shipped with the universe design tool. You can customize these strategies, or use them as templates for new ones.

An external strategy from the oracle.stg file that automatically associates tables with classes, and columns with objects is shown below:

```
<Strategy Name="Classes_and_Objects">
        <Type>OBJECT</Type>
        <SQL>SELECT
       U1.table_name,'|',
       U1.column_name,'|',
       translate(initcap(U1.table_name),'_',' '),'|',
       translate(initcap(U1.column_name),'_',' '),'|',
       U1.table_name||'.'||U1.column_name,'|',
       '  ','|',
decode(SUBSTR(U1.DATA_TYPE,1,1),'N','N','F','N','D','D','C'),'|',
       SUBSTR(U2.comments,1,474),'|',
       'O','|'
FROM USER_TAB_COLUMNS U1,USER_COL_COMMENTS U2
WHERE
    U1.table_name=U2.table_name
and U1.column_name=U2.column_name
UNION
SELECT
       S.SYNONYM_NAME,'|',
       U1.column_name,'|',
       translate(initcap(S.SYNONYM_NAME),'_',' '),'|',
       translate(initcap(U1.column_name),'_',' '),'|',
       S.SYNONYM_NAME||'.'||U1.column_name,'|',
       '  ','|',
decode(SUBSTR(U1.DATA_TYPE,1,1),'N','N','F','N','D','D','C'),'|',
       SUBSTR(U2.comments,1,474),'|',
       'O','|'
FROM ALL_TAB_COLUMNS U1, ALL_COL_COMMENTS U2, ALL_OBJECTS O, USER_SYNONYMS S
WHERE
      S.table_owner=O.owner
AND   S.table_name=O.object_name
AND   (O.OBJECT_TYPE='TABLE' OR O.OBJECT_TYPE='VIEW')
AND   O.owner=U1.owner
AND   O.object_name=U1.table_name
AND   U1.owner=U2.owner
AND   U1.table_name=U2.table_name
AND   U1.column_name=U2.column_name</SQL>
    </Strategy>
```

8.4.8 The output formats of strategies

You write or copy the SQL script within the <SQL> tag in the external strategies file. The order and type of information returned by the SQL script depends on whether you are creating an object, join, or table strategy. The universe design tool has different information needs for each of the different types of strategies.

When you create the SQL script for a strategy, you must ensure that the generated output for the script matches the output formats described below.

The script output is formatted as a series of columns. Each column corresponds to a unit of generated information used to create the object, join, or table components.

This section presents the output formats for:

- Object strategies
- Join strategies
- Table browser strategies.

8.4.8.1 The output format of object strategies (OBJECT)

The output format of an object strategy contains nine columns. You must ensure that your output includes all these columns even if they contain null values. All returned values have to be separated with a pipe '|' character. The pipe has to be appended at the end of the returned values.

Column number	Column contains...	Description
1	Table	Table name format is [Qualifier.] [Owner.]Table where each name can have up to 35 characters. If you leave this column empty, then the tables are obtained from the Select (fifth column) and Where (sixth column).
2	Column Name	Name of the column.
3	Class Name	Name of a class. Subclasses are written as follows: Class\Subclass format.
4	Object Name	Name of the object or condition. If the object name is empty, then a class and its description are created.
5	Select	Select statement.

Column number	Column contains...	Description
6	Where:	If you leave the Select column empty, but include a Where clause, then a pre-defined condition and its description are created.
7	Type	C (Character), N (Numeric), D (Date), T (Long Text). If the column is left empty, the default is N.
8	Description	Description of the object.
9	Qualification	D (Dimension), M (Measure), or I (Detail). If the column is left empty, the default is D.

Example: External object strategy that copies column comments to object descriptions

The example below does not contain a Where clause. The output column for the Where clause is empty.

<Strategies>

<Strategy Name="Read Column descriptions">

<Type>OBJECT</Type>

<SQL>Select

	Col	Description
Table_name, '|',	1	Table name
Column_name, '|',	2	Column name
Replace (Table_name,'_',' '), '|',	3	Replace underscores in table name with blanks in Class name
Replace (Column_name, '_', ' '),'|',	4	Replace underscore in column name with blanks in Object name.
Table_name||'.'||Column_name, '|',	5	Concatenate table name to column name separated by a period. This is the Select statement.
,'|',	6	No Where clause

Column_type,'|',	7	Get column type from system tables
Column_Desc,'|',	8	Get column description from system tables
' ','|'	9	Object type null will default to a Dimension.

</SQL>

8.4.8.2 The output format of join strategies (JOIN)

The output format of a join strategy contains the following columns:

Column number	Column contains...	Description
1	Table1	Name of first table in join
2	Table2	Name of second table in join.
3	Join Definition	The actual definition of the join in a table1.column1=table2.column2 form
4	Outertype	Outer join type. L=outer left, R=outer right. If the column is left empty, there is no outer join.
5	Cardinality (optional)	valid values are 11, 1N, N1.

8.4.8.3 The output format of table browser strategies (STRUCT)

The output format of a table browser strategy contains the following columns:

Column number	Column contains...	Description
1	Qualifier	RDBMS dependant. The Table Qualifier is the database name or some other identification.
2	Owner	RDBMS dependant
3	Table	Name of the table, view, or synonym.

Column number	Column contains...	Description
4	Column	Column name.
5	Data Type	C (Character), N (Numeric), D (Date), T (Long Text). If the column is left empty, the default is C.
6	Nullable Y (Yes) or N (No)	Indicates whether there can be null values in columns

8.4.9 Creating an external strategy

You can create an external strategy in two ways:

Create external strategy by...	Tag in XML file	Description
Inserting SQL script directly.	SQL	You insert the SQL script for the strategy directly in the external strategy file using the SQL tag.
Referencing data in an external file	FILE	You enter the file path and name for an external text file that contains the data for the strategy.

Both methods are described in the following procedure.

8.4.9.1 Creating an external strategy

To create an external strategy directly:

1. Open the external strategy file for the target RDBMS in an XML editor. The strategy file for a target RDBMS is located here:
   $INSTALLDIR/dataAccess/RDBMS/connectionServer/<RDBMS>/<RDBMS>.stg.
2. Create a new strategy element.
   This is the new strategy. If you are using an XML editor for example XML Spy, the Name, Type, and SQL elements for the strategy are created automatically.
3. Enter a strategy name.
   The name of the strategy is visible in the Strategies tab of the Universe Parameters dialog box and in the Quick Design wizard.
4. Enter a TYPE parameter: OBJECT, JOIN, or STRUCT.
   For example, TYPE=OBJECT.
5. Enter the SQL statement of the strategy. The SQL format is described in the section The output formats of strategies [page 378].

Or

If you want to reference a text file containing data, replace the SQL element with the File element. Enter the file path for the data file, for example C:\Path\Filename.txt

6. Add optional elements and set values if necessary.
7. Check the validity of the XML file, then save and close the file.
8. Verify that the external strategy file is declared in the general data access file for the target RDBMS (<RDBMS>.SBO). Do this as follows:
   - Open the general data access file (SBO) in the directory:

   $INSTALLDIR/dataAccess/RDBMS/connectionServer/<RDBMS>/
   - Ensure that the Strategies File element is set to the name of the external strategies file. This is the default value.
   - If you have modified the SBO file, save and close the file.

   The external strategy appears in the Join, Objects, or Tables drop down lists on the Strategies page of the Parameters dialog box. You must close and restart the universe design tool for a newly created external strategy to be visible.

> **i Note**
>
> If you want to add Help text that appears under the external strategy when it is selected on the Strategies page, you add this text to a separate file, the external <RDBMS><language>.STG file, located in the same directory as the external strategy file. Adding Help text for an external strategy is described in the section Creating Help text for external strategies [page 373].

8.4.10 Creating a text file for data

You can create a text file that contains the data for an external strategy. When you create an external strategy, you can enter the file path and name for the text file instead of directly inserting the SQL. You insert the FILE element in the external strategy file, and set the value to the file path and name.

The output of the SQL script must adhere to the correct format for the type of strategy, object, join, or table. The output formats are described in the section The output formats of strategies [page 378].

All formats consist of columns of information separated by tabulations.

8.4.11 Applying external strategies in the universe design tool

You apply external strategies as follows:

1. Ensure that the external strategy that you want to use is selected in the Strategies page of the Parameters dialog box.

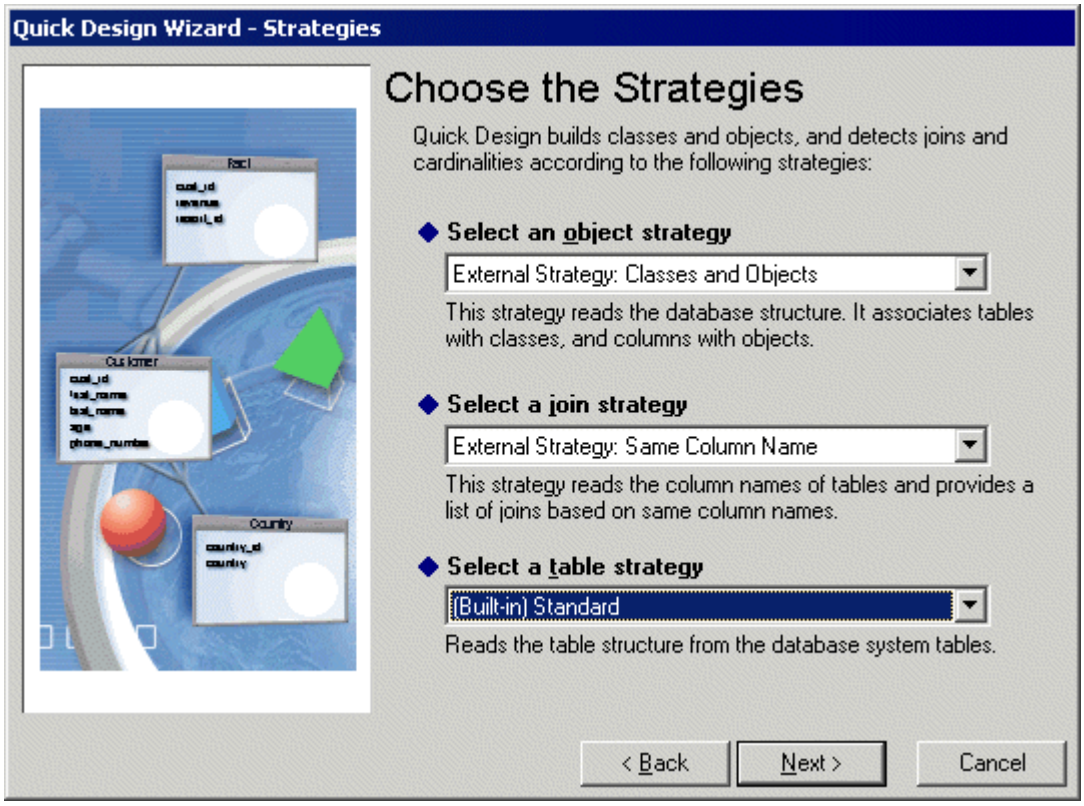
   For example,
   - To insert objects extracted with an object strategy, you select the Candidate Objects command from the Insert menu.
   - To insert joins derived from a join strategy, select the Detect Joins command from the Tools menu.

- To insert tables extracted with a table browser strategy, you select the Tables command from the Insert menu.

> **i Note**
>
> When you select a join strategy, the universe design tool will use the strategy to detect candidate joins and cardinalities. You can choose to apply the suggested joins or cardinalities. If you want the candidate join and cardinalities to be automatically applied based on the selected strategy, you must select the corresponding creation options on the database page of the Options dialog box (Tools > Options > database). See the section Using the automatic creation functions of a strategy [page 85] for more information.

8.4.11.1 Selecting strategies in the Quick Design Wizard

You can select an external strategy you set up from the Quick Design wizard. To do so, you must click the option Click here to choose strategies from the welcome window of the wizard.



8.5 Using analytic functions

The universe design tool supports the use of analytic functions for specific RDBMS. Analytic functions are called RISQL functions in RedBrick, and OLAP functions in Teradata. You can use the universe design tool to define analytic functions for objects in a universe.

Web Intelligence users can also use analytic functions to perform data analysis that is not normally possible within the reporting capabilities of InfoView. For more information about how these and other functions are used in Web Intelligence, please refer to section *Calculating values with Smart Measures* in the document *Using Functions, Formulas and Calculations in Web Intelligence*.

This section describes how you can define Analytic, RISQL, and OLAP functions for objects in a universe for the following RDBMS:

- IBM DB2 UDB and Oracle [page 385]
- RedBrick (RISQL functions) [page 389]
- Teradata (OLAP functions) [page 391]

8.5.1 What are analytic functions?

An analytic function is a function that performs an analytical task on a result set that can be divided into ordered groups of rows or partitions.

In the universe design tool you can define objects with analytic functions to calculate rankings, cumulative aggregates, and ratios within one or more partitions. Depending on your RDBMS, you can also define the range of rows on which you want to apply the analysis within the partition.

For a full description of analytic functions refer to your RDBMS documentation.

8.5.2 What are the advantages of using analytic functions?

Defining objects using analytic functions in the universe design tool has the following benefits for Web Intelligence users:

- Reduced work. An object defined with an analytic function can perform data analysis that would normally require the use of extended syntax at the report level.
- Added functionality. A number of data analysis tasks such as calculating rolling averages and applying advanced aggregate processing are not normally available in InfoView. Objects that use analytic functions now allow Web Intelligence users to conduct advanced data analysis that was not previously possible.
- Improved query performance. The calculations are done on the server.

8.5.3 Which analytic function families are supported?

You can define analytic functions for the following function families:

- Ranking
- Accumulative aggregation
- Ratio, Ratio to Report, or Reporting Aggregate

8.5.4 How are analytic functions used in the universe design tool?

You use analytic functions by defining the analytic function in the SELECT statement for an object.

The RDBMS section in each Parameters (PRM) file lists the analytic functions that can be used in a SELECT statement. This list may not contain all the functions available for each family in each of the RDBMS supported for analytic functions.

8.5.4.1 What is a PRM file?

The PRM file is a parameter file used to configure universe creation and SQL query generation in Web Intelligence products. There is a PRM file for each supported RDBMS. PRM files are located in the following folder:

<INSTALLDIR>\dataAccess\RDBMS\connectionServer\<rdbms>\

See the *Data Access Guide* for full information on modifying parameter files.

Before using an analytic function, you should verify that it is listed in the PRM file. If it is not listed, you can add the name of the function to the list. The universe design tool will then support its use in the Select statement for an object. See the section Verifying and Adding Analytic Function Support to the PRM File [page 387] for more information.

8.5.4.2 Using analytic functions for each RDBMS

Using analytic functions will be described for each of the following RDBMS:

- Syntax that you can use for analytic, RISQL, and OLAP functions in the Select statement.
- How you can verify and modify PRM files to ensure the support of unlisted analytic functions.
- RDBMS specific rules and restrictions for the use of analytic functions.
- Inserting analytic function syntax automatically when editing Select statements.

8.5.5 IBM DB2 UDB and Oracle

You can use the same syntax for analytic functions for both RDBMS.

8.5.5.1 Defining the SELECT statement for DB2, UDB, and Oracle RDBMS

You define an analytic function in the SELECT statement for an object. You need to type the syntax in one of the edit boxes for the Select statement.

> **i** Note
>
> You can automate syntax entry by adding analytic functions to the Functions list in the Edit Select Statement dialog box. To make a function available in the Functions list, you need to add the analytic function to the [FUNCTIONS] section of the .prm file. See the section Inserting syntax automatically in Select statements [page 393] for more information.

Analytic functions are identified by the keyword OVER; for example:

```
RANK() OVER (PARTITION BY calender.cal_year ORDER BY SUM(telco_facts.total_billed_rev)DESC)
```

The clause that follows the OVER keyword defines the partition, and how the rows are ordered in the result table.

The syntax for each family of analytic functions is described as follows:

Function family	Syntax	Description
Ranking	`RANK() OVER(PARTITION BY arg1 ORDER BY arg2 ASC/DESC)`	• arg1 is optional. If no argument is included, then the partition is by default the whole result set. • arg2 is required. The rank is based on this argument value. • ASC/DESC determines whether values are sorted in ascending or descending order. ASC is the default value.
Windows Aggregate	`SUM(arg1) OVER(PARTITION BY arg2 ORDER BY arg3)`	• arg1 is the argument on which the cumulative aggregation is based. • arg2 is the reset clause. It is optional. • arg3 is the group clause. It is optional.
Reporting Aggregate	`RATIO_TO_REPORT(arg1) OVER(PARTITION BY arg2)`	• arg1 is the argument on which the ratio is based. • arg2 is the reset clause. It is optional.

Using a Window clause

For the Windows Aggregate family, you can also define a <window clause> which defines the range for the window size after arg3. For example;

```
<window frame units> ::= ROW |RANGE <window frame start>::= UNBOUNDED PRECEDING
|<window frame preceding> |CURRENT ROW <window frame between>
```

For the BETWEEN clause syntax and other window size definitions, refer to your RDBMS documentation.

8.5.5.2 Verifying and Adding Analytic Function Support to the PRM File

The PRM files for IBM DB2 UDB and Oracle have been updated to support the use of analytic functions.

However, the PRM file may not contain all the analytic functions available in the target RDBMS.Before using an analytic function, you should verify that it is listed in the RDBMS section of the PRM file, and if necessary, add it to the list.

You can do this as follows:

To add support for an analytic function to the Oracle or IBM DB2 PRM file:

1. Browse to the Data Access directory in the Business Objects path.
2. Open the PRM file for your RDBMS in a text editor.
3. Scroll to the RDBMS section of the PRM file.
4. Verify that the following parameters and values are present:

Parameter and value in PRM	Description
OVER_CLAUSE = Y	Generates the appropriate SQL (OVER_CLAUSE).
RISQL_FUNCTIONS = <list of functions used>	Analytic functions available.

5. If you want to use an analytic function that is not listed, type the name of the function at the end of the list. For example, to use RATIO_TO_REPORT you need to add it to the list as follows:

```
[RDBMS]
(GENERAL)
...
OVER_CLAUSE=Y
RISQL_FUNCTIONS=RANK,SUM,AVG,COUNT,MIN,MAX,
VARIANCE,STDDEV,RATIO_TO_REPORT
```

6. Save any modifications and close the file.
   You need to restart the universe design tool for any changes to the PRM file to take effect.

8.5.5.3 Rules For Using DB2, UDB, and Oracle RDBMS Analytic Functions

The following rules apply when using analytic functions for DB2 UDB and Oracle:

Rule	Description
Analytic functions cannot appear in a GROUP BY clause.	Aggregate functions such as SUM defined in the analytic function are used in the GROUP BY clause, but an analytic function such as RANK will not be used. To ensure that analytic functions are not used in GROUP BY clause, they are listed after the RISQL FUNCTIONS parameter in the PRM file. The OVER_CLAUSE preceding it must be set to Y. This is the default setting.
Analytic functions must not generate a GROUP BY clause.	If you add an analytic function to the Functions section in the PRM file (to populate the list of functions in the Edit SQL dialog box), you must ensure that the GROUP CLAUSE is set to N. This will prevent it from generating a GROUP BY clause. See the section Inserting syntax automatically in Select statements [page 393] for more information.
If an analytic function uses an aggregate function, all the dimensions used by the analytic function will appear in the GROUP BY clause.	For example; RANK() OVER (PARTITION BY year ORDER BY SUM(sales). The GROUP BY clause will contain the dimension year even if the rank function is used alone in a query.

8.5.5.4 Restrictions for using analytic functions in Oracle and DB2

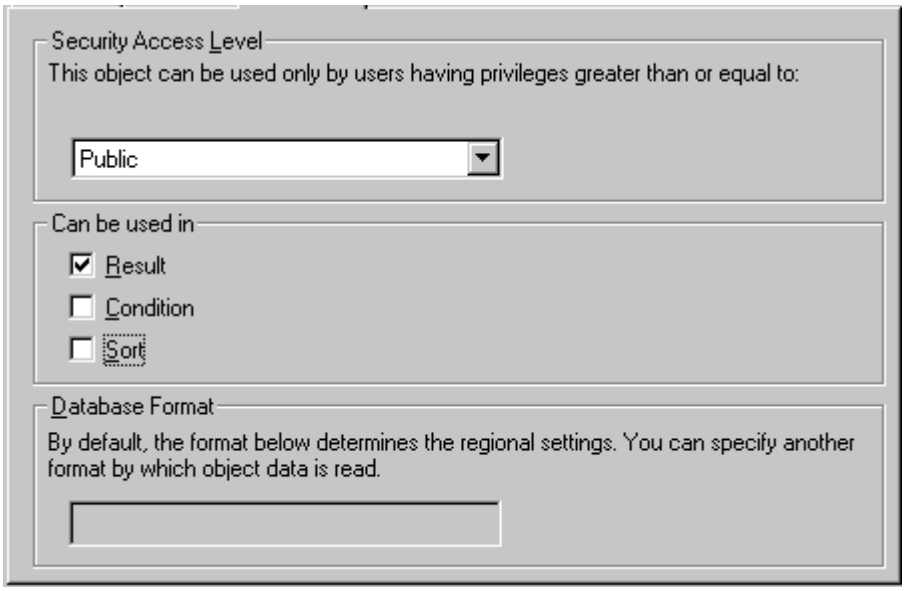
You have the following restrictions when using analytic functions with IBM DB2 UDB v7.1 and Oracle 8.1.6:

- You can not use the functions @Prompt and @Variable in the definition of an object that also uses analytic functions.
- Analytic functions are not supported as user objects. If you add an analytic function to the Functions section in the PRM file (to populate the list of functions in the Edit SQL dialog box), you must ensure that IN MACRO is set to N.
- Objects that use analytic functions cannot be used as a condition or in a sort. If end users try to use these objects to define a condition, they will receive a SQL error message. You can prevent the end user from using an object in either a condition or a sort by editing the object properties as follows:

Preventing use of an analytic object in a condition or sort

To prevent the use of an analytic function in a condition or sort:

1. Right-click the object in the universe design tool.
2. Select *Object Properties* from the contextual menu.
   The *Edit Properties* dialog appears.
3. Clear the *Condition* and *Sort* check boxes in the *Can be used in* group box.



4. Click *OK*.

8.5.6 RedBrick (RISQL functions)

The following sections describe how you can use RISQL functions in the universe design tool.

8.5.6.1 Defining the SELECT Statement for RedBrick RISQL Functions

You define an analytic function in the SELECT statement for an object. You need to type the syntax in one of the edit boxes for the Select statement.

> **i** Note
>
> You can automate syntax entry by adding RISQL functions to the Functions list in the Edit Select Statement dialog box. To make a function available in the Functions list, you need to add the RISQL function to the [FUNCTIONS] section of the PRM file. See the section Inserting syntax automatically in Select statements [page 393] for more information.

The syntax for each family of RISQL functions is described as follows

Function family	Syntax	Description
Ranking (RANK)	RANK(arg1) For example: `RANK(SUM(telco_facts.total_billed_rev))`	arg1 is required. The rank is based on this argument.
Aggregate Families (CUME, MOVINGAVG, MOVINGSUM)	MOVINGSUM(arg1,Number) For example: `MOVINGSUM (COUNT(complants.id),2)`	• arg1 is required. The cumulative aggregation is based on this argument. • Number is optional. This is the number of preceding lines used for the sum.
Ratio (RATIOTOREPORT)	RATIOTOREPORT(arg1) For example: `RATIOTOREPORT (SUM(telco_facts.total_billed_rev))`	arg1 is required. The ratio is based on this argument.

8.5.6.2 Verifying and Adding RISQL Function Support To The PRM File

The PRM file may not contain all the RISQL functions available. Before using an RISQL function, you should verify that it is listed in the RDBMS section of the PRM file, and if necessary, add it to the list. You can do this as follows:

To add support for an analytic function to the Redbrick PRM file:

1. Browse to the Data Access directory in the Business Objects path.
2. Open the PRM file for your RDBMS in a text editor.
3. Scroll to the RDBMS section of the PRM file.
4. Verify that the following parameters and values are present:

Parameter and value in PRM	Description
OLAP_CLAUSE = WHEN	Applies the condition.
RISQL_FUNCTIONS = <list of functions used>	Analytic functions available.

An example appears below:

```
[RDBMS]
(GENERAL)
...
OVER_CLAUSE=WHEN
RISQL_FUNCTION= RANK, CUME, MOVINGSUM, MOVINGAVG, RATIOTOREPORT,TERTILE
```

5. If you want to use an RISQL function that is not listed, type the name of the function at the end of the list.
6. Save any modifications and close the file.
   You need to restart the universe design tool for any changes to the PRM file to take effect.

8.5.6.3 Rules for using RISQL functions

The following rules apply when using RISQL functions:

Rule	Description
RISQL functions cannot appear in a GROUP BY clause.	Aggregate functions such as SUM defined in the RISQL function are used in the GROUP BY clause, but an analytic function such as RANK will not be used. To ensure that RISQL functions are not used in the GROUP BY clause, they are listed after the RISQL FUNCTIONS parameter in the PRM file. The OVER_CLAUSE preceding it must be set to WHEN. This is the default setting.
RISQL functions must not generate a GROUP BY clause.	If you add an RISQL function to the Functions section in the PRM file (to populate the list of functions in the Edit SQL dialog box), you must ensure that the GROUP CLAUSE is set to N. This will prevent it from generating a GROUP BY clause. See the section Inserting syntax automatically in Select statements [page 393] for more information.
You can use an RISQL function in a condition	A WHEN clause is generated

8.5.6.4 Restrictions for using analytic functions in RedBrick

You have the following restrictions when using RISQL functions:

- RESET BY clause is not supported.
- SORT BY clause not supported. See the section *Restrictions for using analytic functions in Oracle and DB2* for the procedure describing how you can prevent the end user from using an object in a sort by editing the object properties.

8.5.7 Teradata (OLAP functions)

The following sections describe how you can use OLAP functions in the universe design tool.

8.5.7.1 Defining the Select statement for Teradata OLAP Functions

Ratio functions are not available in Teradata V2R3. You define an OLAP function in the Select statement for an object. You need to type the syntax in one of the edit boxes for the Select statement.

For information on how to make a function available in Functions list to automate syntax entry, see the section *Restrictions for using analytic functions in Oracle and DB2*.

The syntax for each family of OLAP functions is described as follows:

Function family	Syntax	Description
Ranking (RANK)	RANK(arg1 DESC/ASC) For example: `RANK(invoice_line.nb_guests)`	• arg1 is required. The rank is based on this argument. The argument can be an object or a list of objects. **i Note** You cannot use an object that uses an aggregate object (sum, avg, min, count) as arg1. • DESC/ASC specifies the ranking order. ASC is the order by default.
Aggregate Families (CSUM, MAVG, MDIFF, MLINREG, MSUM	CSUM(arg1 DESC/ASC) For example: `CSUM(invoice_line.nb_guests)`	• arg1 is required. The cumulative aggregation is based on this argument. The argument can be an object or a list of objects. • DESC/ASC specifies the order of result rows. ASC is the order by default.

8.5.7.2 Verifying and adding OLAP function support in the PRM file

The PRM file for Teradata has been updated to support the use of OLAP functions. However, the PRM file may not contain all the OLAP functions available. Before using an OLAP function, you should verify that it is listed in the RDBMS section of the PRM file, and if necessary, add it to the list. You can do this as follows:

To add support for an analytic function to the Teradata PRM file

1. Browse to the Data Access directory in the Business Objects path.
2. Open the PRM file for your RDBMS in a text editor.
3. Scroll to the RDBMS section of the PRM file.
4. Verify that the following parameters and values are present:

Parameter and value in PRM	Description
OLAP_CLAUSE = QUALIFY	Applies the condition.
RISQL_FUNCTIONS = <list of functions used>	Analytic functions available.

An example appears below:

```
[RDBMS]
(GENERAL)
...
OVER_CLAUSE=QUALIFY
RISQL_FUNCTION= RANK, CSUM, MAVG, MDIFF,MLINREG,MSUM,QUANTILE
```

5. If you want to use an RISQL function that is not listed, type the name of the function at the end of the list.
6. Save any modifications and close the file.
   You need to restart the universe design tool for any changes to the PRM file to take effect.

8.5.7.3 Rules for using OLAP functions

The following rules apply when using OLAP functions:

- OLAP functions cannot appear in a GROUP BY clause. To ensure that OLAP functions are not used in GROUP BY clause, they are listed after the RISQL FUNCTIONS parameter in the PRM file. The OVER_CLAUSE preceding it must be set to QUALIFY. This is the default setting.
- You cannot combine an object using an OLAP function with an object using an aggregate function in the same query.
- You can use OLAP functions in a condition. A QUALIFY clause is generated.
- You can use OLAP functions in a SORT BY clause.

8.5.7.4 Restrictions for using analytic functions in Teradata

You have the following restrictions when using OLAP functions:

- RESET BY clause is not supported.
- OLAP functions cannot be used in a sub-query.
- An OLAP function cannot be used in the same Select statement as another function.
- An OLAP function cannot be based on another function.
- OLAP functions are not supported as user objects.

8.5.8 Inserting syntax automatically in Select statements

You can automate the insertion of analytic function syntax by adding the analytic function to the Functions list box in the Edit Select Statement dialog box.

You populate the Functions list box by adding the analytic function to the list of functions under the [FUNCTION] section in the appropriate PRM file for the target RDBMS.

Once added to the PRM file, the function becomes available in the Functions list box in the Edit Select Statement dialog box. When you double click the function syntax, the defined syntax is inserted in the edit box.

When you add the analytic function to the PRM file, you must set the following values:

Parameter	Description
GROUP = N	Analytic, RISQL, and OLAP functions cannot generate a GROUP BY clause. By setting the value N, you prevent the analytic function from being used in a GROUP BY clause.
For IBM DB2 UDB v.7.1 and ORACLE 8.1.6 only: IN_MACRO = N	This prevents the analytic function for DB2 UDB and Oracle from being used in user objects. For RedBrick and Teradata, this value can be set at Y.

You can add an analytic function to the [FUNCTION] section in the PRM file as follows:

To add an analytic function to the PRM file:

1. Browse to the Data Access directory in the Business Objects path.
2. Open the PRM file for your RDBMS in a text editor.
3. Scroll to the [FUNCTION] section of the PRM file.
4. Copy an existing function and paste it at the end of the list.
5. Type a unique number for the newly pasted function, and modify the values as appropriate for the analytic function that you want to add to the list.
6. Set the GROUP value to N.
   If you are using IBM DB2 UDB, or ORACLE, set the IN_MACRO value to N.
   For example:

```
(n)
NAME: RANK
TRAD:
HELP: Return the rank of
TYPE=N
IN_MACRO=N
GROUP=N
SQL=
```

7. Save and close the PRM file.
   You need to restart the universe design tool for the changes to be applied.

> **i Note**
>
> When you restart the universe design tool, the syntax for the added analytic function appears under the appropriate Type node (Number, Character, or Date).

8.6 Using the SQL prefix function

The SQL prefix function inserts SQL orders before a BusinessObjects-generated SQL statement by using the universe parameter `Begin_SQL`. This ensures that the orders are executed before all generated SQL statements. This function works with any database that supports passing parameters before the select statement. The following are some examples:

- Teradata: using 'QUERY_BAND' for a transaction (refer to the Teradata documentation)
- Oracle: turn on read only transaction
- Netezza: trigger optimization options

To set the SQL prefix function, set the SQL_prefix parameter for the universe.

Related Information

SQL Parameters that you set in the user interface [page 93]

8.6.1 To prefix SQL statements with the BEGIN_SQL universe parameter

Your database supports passing of parameters before the SELECT statement.

The `BEGIN_SQL` parameter allows you to prefix SQL statements with the same parameters each time a SQL statement is generated.

1. Open the *Universe Parameters* dialog box.
2. Click the *Parameter* tab.
3. In the *Parameter* list, select the `BEGIN_SQL` parameter and enter the appropriate prefix commands.
4. Save the settings.
5. Save the universe.

Example

This example uses the `BEGIN_SQL` parameter with Teradata. The query has the user ID and the application ID bound to the query for reporting purposes. In the *Parameter* pane of the *Universe Parameters* dialog box, the `BEGIN_SQL` parameter is set as follows:

```
BEGIN_SQL = SET QUERY_BAND = 'UserId=DG12234;AppId=TRM;' FOR TRANSACTION;
```

Below, when running a query, two SQL statements are executed:

1) The `BEGIN_SQL` statement:

```
SET QUERY_BAND = 'UserId=DG12234;AppId=TRM;' FOR TRANSACTION;
```

2) The main query SQL result statement:

```
SELECT
    RESORT_COUNTRY.COUNTRY, sum(INVOICE_LINE.DAYS * INVOICE_LINE.NB_GUESTS
      * SERVICE.PRICE)
    FROM
    COUNTRY RESORT_COUNTRY, INVOICE_LINE, RESORT_COUNTRY.COUNTRY
```

8.7 Optimizing the array fetch parameter

The Array Fetch parameter in the CS.CFG file allows you to set the maximum number of rows that are permitted in a FETCH procedure. The CFG file is an XML file that specifies default values for certain parameters used by Business Objects products when queries are run against a database.

The Array Fetch parameter determines the packet size on the network. For example, if you set the Array Fetch at 20, and you plan to retrieve 100 rows, then five fetches will be executed to retrieve the data.

Some data sources do not allow modifying the FETCH size. In this case all rows will be returned in a single FETCH. If you want to retrieve binary long-objects (BLOB), you should set the Array Fetch size as 1.

If you have a network that allows you to send a large array fetch, then you can set a new larger value (values can be set from 1 to 999). This will speed up the FETCH procedure, and reduce your query processing time.

8.7.1 Modifying the array fetch parameter

To modify the Array Fetch parameter:

1. Open the CS.CFG file in an XML editor.
   The CFG file is stored in the following directory:
   <INSTALDIR>\dataAccess\RDBMS\connectionServer.
2. Search for the parameter Array Fetch.
3. Set the parameter value. Save and close the CFG file.
4. Restart the universe design tool.

8.8 Allocating table weights

Table weight is a measure of how many rows there are in a table. Lighter tables have less rows than heavier tables. By default BusinessObjects sorts the tables from the lighter to the heavier tables (those with the least amount of rows to those with the most). This determines the table order in the FROM clause of the SQL statement.

The order in which tables are sorted at the database level depends on your database. For example, Sybase uses the same order as BusinessObjects, but Oracle uses the opposite order. The SQL will be optimized for most databases, but not for Oracle where the smallest table is put first in the sort order.

So, if you are using an Oracle database, you can optimize the SQL by reversing the order that BusinessObjects sorts the tables. To do this you must change a parameter in the relevant PRM file of the database.

8.8.1 Modifying the PRM file to allocate table weights

To modify the PRM file to allocate table weights:

1. Open the PRM file for your database in an XML editor.
   The PRM file is stored in the following directory:
   <INSTALLDIR>\dataAccess\RDBMS\connectionServer\<rdbms>\
   For example, the file for Oracle is oracle.prm in here:
   <INSTALLDIR>\dataAccess\RDBMS\connectionServer\oracle\oracle.prm
2. Find the REVERSE_TABLE_WEIGHT parameter in the Configuration section of the file.
3. Change the Y to an N.
   For example the parameter appears as REVERSE_TABLE_WEIGHT=N.
   If the line is not in the file, the default is Y.
4. This forces BusinessObjects to sort the tables from those with the most rows to those with the least rows.
5. Save and close the .PRM files.
6. Restart the universe design tool to apply the changes to the .PRM file.

8.9 Modifying the number of returned rows for a table

You can also manually change the number of rows for any table in the universe design tool. To view the number of rows in any table, select View > Number of rows in tables. The number of rows appears at the bottom left of each table symbol. You can modify this number as follows:

1. Open a universe in the universe design tool.
2. Right-click the relevant table
3. Select Number of Rows in Table from the contextual menu.
   The Table Row Count dialog box appears.
4. Select the Modify manually tables row count radio button.
   A text box appears at the left of the dialog box.
5. Type a number in the text box. This is the number of rows that you want to use for the table.
6. Click OK, then save the universe.

8.10 Using shortcut joins

A shortcut join links two tables that are already joined in a common path. You can use a shortcut join to reduce the number of tables that are used in a query. Refer to the section Restrictions for the use of outer joins [page 162] for more information.

> **i** Note
>
> Shortcut joins will not create loops.

9 Working with OLAP universes

9.1 About OLAP universes

9.1.1 What is an OLAP universe?

An OLAP universe is a Business Objects universe that has been generated from an OLAP cube or query. The universe is created automatically from a selected connection to an OLAP data source.

Once the universe has been created, it can be exported to the Central Management Server (CMS) as any other universe. The universe is then available to Web Intelligence users to run queries and create reports.

You generate and maintain an OLAP universe in the following way:

- To generate an OLAP universe, first select the OLAP data source.

  > **i Note**
  >
  > For a secured connection to an OLAP data source, any users who need to generate the universe or refresh its structure need to have the *Download* right on the connection. This right is set in the CMC by the administrator.

- Define a connection to the data source using the New Connection wizard, and select the connection for a new universe.
  The universe design tool generates the universe automatically. OLAP structures are mapped directly to classes, measures, dimensions, details, and filters in the universe. The universe structure appears in the Universe pane.
- You can save and export the OLAP universe to the CMS.
- You can modify any of the OLAP universe components.
- The Update OLAP Universe wizard allows you manage the lifecycle of the OLAP universe. The wizard automatically refreshes the universe structure with changes made in the OLAP data source. The wizard can differentiate generated objects from objects added or modified manually, allowing you to preserve the manual changes made in the universe design tool.

Related Information

Which OLAP data sources can be used to create a universe? [page 400]
About connections to OLAP data sources [page 406]
Universe design tool features supported for OLAP universes [page 416]
About OLAP universe lifecycle management [page 429]

9.1.2 Which OLAP data sources can be used to create a universe?

You can create OLAP universes automatically from the following OLAP data sources:

- SAP Business Warehouse (BW)
- Microsoft Analysis Services (MSAS) 2000
- Microsoft Analysis Services (MSAS) 2005
- Hyperion Essbase

> **i Note**
>
> In order to connect to an Essbase OLAP data source from SAP BusinessObjects OLAP products including universe design tool, Web Intelligence Rich Client and Web Intelligence, ensure that Essbase Client middleware is properly installed and configured on machines hosting those SAP BusinessObjects OLAP products. Specifically, ensure that Essbase Client environment variables ARBORPATH and ESSBASEPATH are created and set as Windows system environment variables (as opposed to Windows user environment variables).

One universe is generated automatically from one cube or query. OLAP universes support a single cube in the universe.

Related Information

SAP Business Warehouse (BW) data sources [page 400]
How SAP BW objects are mapped and used in a universe [page 459]
MSAS features supported for OLAP universes [page 404]
How MSAS cubes are mapped to universe components [page 468]
Essbase features supported for OLAP universes [page 405]
How Essbase cubes are mapped to universe components [page 467]

9.1.2.1 SAP Business Warehouse (BW) data sources

When creating an OLAP universe based on a BW data source, you can build the universe based directly on an InfoCube/MultiCube, or based on a BEx Query enabled on top of any InfoProvider. An InfoProvider can be:

- an InfoCube
- a MultiCube or Multi-InfoProvider
- an Operational Data Store (ODS)
- an InfoSet

Related Information

SAP Business Warehouse (BW) InfoCubes as data sources [page 401]
SAP BW Queries as data sources [page 401]
Queries as recommended data sources [page 402]

9.1.2.1.1 SAP Business Warehouse (BW) InfoCubes as data sources

The following types of InfoCubes are supported as data sources for building OLAP universes:

- Standard and Transactional InfoCubes: Data and metadata are physically stored in the same SAP Business Warehouse (BW) system
- Remote InfoCube: Data is physically stored on a remote system

> **i Note**
>
> While fully supported, building and deploying universes on remote InfoCubes is not recommended for ad-hoc query-, reporting-, and analysis-use scenarios. Such architecture is generally not expected to meet query performance expectations with interactive queries.

- MultiCubes and Multi-InfoProviders

> **i Note**
>
> Building and deploying a Business Objects universe on top of a MultiCube or Multi-InfoProvider is identical to building and deploying a universe on top of an InfoCube.

All the characteristics, hierarchies, key figures, including time and unit, in the InfoCube are visible in the universe.

9.1.2.1.2 SAP BW Queries as data sources

SAP BW customers use BEx Queries to access SAP Business Explorer front-ends.

> **i Note**
>
> In order to serve as a data source and become available through the OLAP interface to Business Objects universes, BEx queries must be released for OLE DB for OLAP. You allow external access to the BEx Query in the SAP BW Query Designer, on the *Extended mode* tab of the *Query Properties* dialog box.

All InfoObjects in the BEx query selected as rows, columns, and free characteristics are visible in the universe. This includes characteristics, hierarchies, key figures, structures, and variables.

Both InfoSets and Operational Data Stores (ODS) can be exposed to universes via BEx Queries.

Queries based on an ODS

An ODS can be exposed to a universe via a BEx Query.

ODS objects are often used to manage detailed transaction-level data before it is aggregated into InfoCubes. Including ODS objects in the SAP NetWeaver technology platform data store design is a way to minimize InfoCube size and improve loading and querying performance.

> **i Note**
>
> An ODS is usually a large, detailed relational structure. Accessing an ODS via the OLAP BAPI interface does not deliver ideal query performance. Consider these alternatives to meet end-user expectations for fast report delivery:

- Create direct access to an ODS via BAPI calls
- Access the ODS tables using direct SQL in Web Intelligence

Queries based on an InfoSet

An InfoSet can be exposed to a universe via a BEx Query.

InfoSets are sometimes defined in SAP BW to report master data.

> **i Note**
>
> You can report master data by basing the universes on InfoCubes, eliminating the requirement to go through InfoSets and BEx Queries. The key difference between the two approaches is that master data reported off InfoCubes limits data to valid transactions.

Related Information

Queries as recommended data sources [page 402]

9.1.2.1.3 Queries as recommended data sources

BEx Queries are recommended as data sources for generating Business Objects universes for the following reasons:

- Not all SAP BW metadata features can be retrieved on an InfoCube level, as summarized in the following table.

BW metadata feature	SAP OLAP Business Application Programming Interface (BAPI) support level
Characteristics (incl. Time and Unit)	InfoCube/BEx Query
Hierarchies	InfoCube/BEx Query
Basic Key Figures	InfoCube/BEx Query
Navigational Attributes	BEx Query only
Display Attributes	InfoCube/BEx Query
Calculated Key Figures / Formulas	BEx Query only
Restricted Key Figures	BEx Query only
Custom Structures	BEx Query only
Variables	BEx Query only

- BEx Queries offer a flexible extension to the data modeling environment. InfoCubes require more effort to change.
- BEx Queries offer significant functionality to create customized data sources that meet end-user requirements.

Although BEx Queries have advantages as data sources, you do not need a BEx Query for every report, nor do you need a universe for every existing BW Query. To minimize maintenance costs, focus the implementation strategy on limiting the final number of BEx Queries and universes required to meet all the ad-hoc query and reporting needs. Keep in mind the following points to reduce the number of universes needed:

When Web Intelligence is the front-end tool, you are not restricted by the output format in the BEx Query. There is no direct impact on performance when working with OLAP universes created from large BEx Queries. OLAP universe objects not inserted in the Web Intelligence query have no direct impact on the query performance.

> **i** Note
>
> Business Objects recommends having a few BEx Queries – from a single one to a handful of them – for every InfoCube or MultiCube that is in scope for ad-hoc query and reporting. Then build a universe on top of each of these BEx Queries.

9.1.2.1.4 SAP BW multilingual universes

With Web Intelligence, it is possible to leverage the multilingual capabilities of SAP BW. In order to implement a multilingual environment, the BW system must include multilingual metadata and multilingual data.

You must create a universe for each language supported by the solution. The language in which the universe connection is created determines the language in which the universe is generated.

The user's SAP authentication determines the language of the data returned to the query. The user must log into InfoView using SAP authentication and specify the desired language for results returned from the SAP server.

The result-set language is dependent on SAP's Unicode support. If the SAP system does not contain the data in the desired language, the data is not available in Web Intelligence in this language. Web Intelligence reverts to displaying technical names instead of descriptions when the descriptions are not translated in BW.

9.1.2.1.5 Prerequisites to using SAP BW in the universe design tool

When creating a universe from an SAP BW data source, you can enable SSO (Single Sign On) for view time. SSO allows users to log on to SAP BusinessObjects Enterprise with their SAP Credentials and leverage SAP Authentication.

To enable SSO for OLAP Universes on top of SAP, you must install the SAP Integration and configure the SAP Security Plug In.

Once SAP security integration is configured, you can use SAP credentials to launch universe design tool. Create the BusinessObjects Enterprise user name by concatenating the SAP System ID and SAP Client ID as defined when configuring the security integration with the SAP user ID.

For more information, see the *Business Objects XI Integration for SAP Solutions Installation Guide*, and the *Business Objects XI Integration for SAP Solutions User Guide*.

9.1.2.2 MSAS features supported for OLAP universes

The following table summarises the support level for MSAS features for universes generated from an MSAS data source.

MSAS metadata feature	OLAP universe support level
Cube	Supported
Local cube	Supported
Virtual cube (MSAS 2000)	Supported
Perspective (MSAS 2005)	Supported
Dimensions	Supported
Virtual dimensions (MSAS 2000)	Supported
Hierarchies	Supported
Levels	Supported
Level Property	Supported
Attributes (MSAS 2005)	Supported
Measures	Supported
Measure group (MSAS 2005)	Supported
Calculated measures	Supported

MSAS metadata feature	OLAP universe support level
Display folder (MSAS 2005)	Supported
KPI (MSAS 2005)	Not supported
Action	Not supported
Database sort order	Must define custom sort order in Web Intelligence
Write back	Not supported

Related Information

How MSAS cubes are mapped to universe components [page 468]

9.1.2.3 Essbase features supported for OLAP universes

The following table summarizes the support level for Essbase features for universes generated from a Hyperion Essbase data source.

Essbase metadata feature	OLAP universe support level
Block storage mode	Supported
Aggregate storage mode	Supported
Hybrid mode	Not supported
Alias tables	Supported
Dimensions	Supported
Attribute dimensions	Supported
Duplicate members	Supported
Generations	Supported
Levels	Not supported
User Define Attributes (UDA)	Not supported
Dynamic Time Series (DTS)	Not supported
Essbase Integration Services (EIS) Drill-through	Not supported
Substitution variables	Not supported
Linked partitions	Not supported
Linked Reporting Objects (LRO)	Not supported
Database sort order	Must define custom sort order in Web Intelligence
Write back	Not supported

Related Information

How Essbase cubes are mapped to universe components [page 467]

9.2 Defining connections to OLAP data sources

9.2.1 About connections to OLAP data sources

To generate an OLAP universe, you must first define a connection to the OLAP data source. Define one connection for each cube or query that you want to use to create a universe.

Use the New Connection wizard to define the connection. The wizard leads you through the following steps in creating a connection:

- Starting the New Connection wizard in the universe design tool.
- Naming the connection and selecting the database middleware.
- Defining the login parameters for the connection. These parameters vary depending on the database middleware that you selected.
- Selecting the cube or query to use to create the universe.
- Defining the connection life-time.
- Defining custom parameters. These parameters vary depending on the database middleware that you selected.

Defining a connection is the first step in creating an OLAP universe. Once you have defined the connection, the universe design tool generates the universe automatically.

> **i** Note
>
> If you define a connection from the connections list on the Tools menu, you must create the universe as a separate step.

Related Information

To start the New Connection wizard [page 407]
To select database middleware for an OLAP connection [page 407]
Login parameters for SAP BW OLAP connections [page 410]
Login parameters for MSAS OLAP connections [page 411]
Defining login parameters for Essbase connections [page 412]
To select source cube or query for OLAP connections [page 412]
To define configuration parameters for OLAP connections [page 413]
Defining custom parameters for Essbase connections [page 414]

9.2.2 To start the New Connection wizard

To start the New Connection wizard, do one of the following:

Start from...	Do...
New Universe icon	Click the *New Universe* icon, then click *New...* on the *Definition* page of the *Universe Parameters* box.
File menu	From an empty session, select *File* > *Parameters*, then click *New...* on the *Definition* page of the *Universe Parameters* box.
Quick Design wizard	If the Quick Design wizard is enabled, it starts automatically when you start the universe design tool. Click *New...* in step one of the Quick Design wizard. **i Note** If the wizard has been disabled, select *File* > *New*. If the wizard does not start when you select *File* > *New*, select *Tools* > *Options*. On the *General* page of the *Options* dialog box, select the *File/New starts Quick Design wizard* check box. Click *OK* and select *File* > *New*.
Tools menu	Select *Tools* > *Connections*. Click *Add...* in the *Wizard Connection* dialog box.

9.2.3 To select database middleware for an OLAP connection

On the *Database Middleware Selection* page of the New Connection wizard, enter a name for the connection, select the type and the database middleware for the connection.

Database Middleware Selection parameter	Description
Connection Type	Select *Secured* for controlled access to the connection (recommended). Select *Shared* to allow uncontrolled access to the connection to all users. Select *Personal* to restrict access to the universe creator. You can use personal connections to access personal data on a local machine only.
Connection Name	Enter a name for the connection.
Filter Stored Procedures Network Layers	The *Filter Stored Procedures Network Layers* parameter is not used for OLAP connections.

Database Middleware Selection parameter	Description
List of available data access drivers.	The page lists the databases and middleware that correspond to your data access driver key. Expand the node for your target database to show the supported middleware for that database. Expand the middleware node to show the Business Objects data access driver for the OLAP middleware. Select the data access driver.

9.2.4 Login parameters for relational connections - UDT

The following login parameters apply to most relational connections. In order to use UDT, you have to install the Business Objects clients in 64-bit system instead of 32 bits system.

Parameter	Description
Authentication Mode	The method used to authenticate the user's login credentials when accessing the data source: • *Use specified user name and password*: Uses the *User Name* and *Password* parameters defined for the connection. • *Use BusinessObjects credential mapping*: Uses the database credentials associated with the user's account defined on the Central Management Server (CMS) to connect to the data source. The database credentials are set in the User's Properties in the Central Management Console. For more information, see the *SAP BusinessObjects Business Intelligence platform Administrator's Guide*. • *Use Single Sign On*: This authentication mode is used to support end-to-end single sign on defined in the Central Management Server (CMS). If you use an external authentication source (for example LDAP), the CMS and the data source must be configured to use this external authentication source. For more information on single sign on, see the *SAP BusinessObjects Business Intelligence platform Administrator Guide*. • *Use single sign-on when refreshing reports at view time*: If you have opened a session using Windows Active Directory single sign-on (SSO), and if some connections have been defined to use SSO, choose this mode of authentication when connecting to the CMS repository. For more information, see the *SAP BusinessObjects Business Intelligence platform Administrator's Guide*.
User Name	The user name to access the data source if *Authentication Mode* is *Use specified user name and password*.
Password	The password to access the data source if *Authentication Mode* is *Use specified user name and password*.

Parameter	Description
Use SSL	This parameter applies only to SAP HANA connections. If selected, uses SSL protocol to connect to the server.
Single Server	This parameter applies only to SAP HANA connections. Check this option if you are connecting to only one SAP HANA database server. • *Host Name*: the name of the server hosting the data source. Do not include the port number. • *Instance Number*: the SAP HANA instance number, which represents the second and third digits of the port number. Must be set between **00** and **99**. For example, if the port number is `30215`, the instance number is **02**.
Multiple Servers	This parameter applies only to SAP HANA connections. Check this option to take advantage of the SAP HANA failover mechanism. *Server (host:port{;host:port})*: a list of servers separated by a semi-colon, for example: **`(host1:30015;host2:30015;host3:30015)`**. The JDBC driver will choose one of these hosts for connecting. If a host is not available, the driver chooses the next host from the list. You can also enter the host and port of only one server in the field.
Server (<host>:<port>)	The name and port of the server hosting the data source. For connections to Oracle, you can enter a list of servers separated by commas: (<host>:<port>, <host>:<port>).
Server	The name of the server hosting the data source.
Database	The database name.
Data Source Name	For ODBC connections, the name of the data source that you have defined using your operating system's data source manager.
Alias	For DB2 connections, this is the alias of the database that you have created in the DB2 configuration assistant.
Net Service	For Oracle connections using JDBC middleware, the Oracle Net Service name.
Service	For Oracle connections, the alias that contains the server IP and Net Service information.
JDBC_URL *JDBC_CLASS*	For generic JDBC connections, the JDBC URL and class used to connect to the database.
Informix Server	For Informix connections, the name of the Informix server that you have defined.
OLE DB Provider Name	For generic OLE DB providers, the provider name.

9.2.5 Login parameters for SAP BW OLAP connections

The *New Connection* wizard's *Login Parameters* dialog box can contain the following parameters:

Login parameter	Description
Authentication mode	• *Use specified user name and password*: uses the login details as authentication. • *Use Business Objects credential mapping*: the user is prompted for a database user password associated with his BusinessObjects account to refresh a report. This is set using the parameters *dbuser* and *dbpass*. These are set at the administrative level. Refer to *SAP Business Objects Enterprise Administrator's Guide* for information on setting up this option. • *Use Single Sign On when refreshing reports at View Time*: When selected, the username and password used to access the CMS are automatically used as the database login parameters. See the *Business Objects Enterprise Administration Guide* for information on setting Single Sign-On (SSO).
Use SNC if available	Select this check box to use the SNC.
Client	The number used to identify the client on SAP BW system (required).
User name	The user name to use to access the OLAP server if the *Authentication mode* is *Use specified user name and password*.
Password	The password to use to access the OLAP server if the *Authentication mode* is *Use specified user name and password*.
Language	The language that will be used for the connection. **i Note** The connection language determines the language in which the universe is generated.
Save Language	Which language will be used for the connection: • If you check *Save Language*, the value from the *Language* field will be used. • If you clear *Save Language*, the value from the user's session will be used.
Login mode or *Server Type*	Select *Application Server* to connect directly to the SAP server without using load balancing. Select *Message Server* to benefit from SAP load balancing capabilities.
Application Server	Select or enter the name or IP address of the SAP application server (required for Application server login mode).
System Number	Enter the system number, for example 00 (required for Application server login mode) .
System ID	Enter the *Message Server*, *Logon Group*, and if required, the *System ID* when using *Message Server* login mode.
Logon Group	

Login parameter	Description
Message Server	

9.2.6 Login parameters for MSAS OLAP connections

The *New Connection* wizard's *Login Parameters* dialog box can contain the following parameters:

Login parameter	Description
Authentication mode	• *Use specified user name and password*: uses the login details as authentication. • *Use Business Objects credential mapping*: the user is prompted for a database user password associated with his BusinessObjects account to refresh a report. This is set using the parameters *dbuser* and *dbpass*. These are set at the administrative level. Refer to *SAP Business Objects Business Intelligence platform Administrator's Guide* for information on setting up this option. • *Use Single Sign On when refreshing reports at View Time*: When selected, the username and password used to access the CMS are automatically used as the database login parameters. See the *SAP Business Objects Business Intelligence platform Administrator's Guide* for information on setting Single Sign-On (SSO).
Server	Enter one of the following: • The URL for the MSAS library exposed and configured in the MSAS server. • The server name for the MSAS data source • The full path file name of an MSAS cube file. Enter the full path file name between double quotes, for example: "Z:\All cubes\test.cub" **i Note** If the cube file is located on a host system different from the SAP BusinessObjects host, the two machines must have a shared connection. You must create the connection to a cube file directly on the SAP BusinessObjects host.
User name	The user name to use to access the OLAP server if the *Authentication mode* is *Use specified user name and password*.
Password	The password to use to access the OLAP server if the *Authentication mode* is *Use specified user name and password*.
Language	The language that will be used for the connection.

9.2.7 Defining login parameters for Essbase connections

On the *Login Parameters* page of the New Connection wizard, specify the login details for connecting to the Essbase database.

Login parameter	Description
Authentication mode	Select *Use specified username and password* to require the user to enter login information when using the connection. To synchronize Essbase security with BusinessObjects Enterprise, enter the Essbase DBuser and DBpass for *User name* and *Password*. Select *Use BusinessObjects credential mapping* to use the user's BusinessObjects Enterprise login credentials for the connection. **i Note** The *Use Single Sign On when refreshing reports at view time* option is not supported for Essbase connections.
User name	Enter the Essbase DBuser.
Password	Enter the Essbase DBpass.
Server	Enter the Essbase server name.

9.2.8 To select source cube or query for OLAP connections

The cube browser shows the OLAP cubes available to the target server.

Expand the cube nodes to display the cubes and queries available. The browser has the following tools to help you search:

Cube browser tool	Description
Favorites	A folder that holds links to cubes that you select for quick access. To add a cube to Favorites, right click a cube in the OLAP cube browser and select Add to Favorites from the contextual menu.
Search	Searches the names of available cubes or queries for a text string. Type a string in the text box and click Search. Each found instance is highlighted. Click Search to continue searching.
$INFOCUBE folder	For SAP BW data sources, the InfoCubes and MultiCubes are grouped in a folder called $INFOCUBE.

Select the cube or query to use to create the universe.

9.2.9 To define configuration parameters for OLAP connections

On the *Configuration Parameters* page of the New Connection wizard, define the connection parameters to manage the connection life-time. You can accept the default settings when you create the connection and modify them later.

Configuration parameter	Description
Connection Pool Mode *Pool timeout*	You define the life-time with the *Connection Pool Mode* and *Pool timeout* parameters. By default, the *Connection Pool Mode* is set to *Keep the connection alive for*, with a default *Pool timeout* of 10 minutes. **Note** Business Objects recommends you keep the default connection life-time. The universe building process slows significantly if the *Connection Pool Mode* is set to disconnect after each transaction. Disconnecting after each transaction also impacts key end-user workflows such as working with hierarchical list of values. The connection life-time can have a significant impact when working with SAP BW. However, connection life-time can also impact updating existing universes with changes in the BEx Query. This is because the OLAP BAPI interface builds a metadata cache on the client side every time a connection to SAP BW is established. This cache is only emptied when the connection closes. To minimize the risk of metadata cache being desynchronized with SAP BEx Query updates, you can change the *Pool timeout* from 10 minutes to 1 minute. When working in parallel editing BW Queries and mapping new universes to these queries, it is recommended that you close the universe design tool (so that universe connections are also closed and the metadata cache is emptied) before building any new universes to take into account changes that were just made on the BEx Query side.
Array fetch size	The *Array fetch size* parameter allows you to set the optimum number of rows that are permitted in a FETCH procedure.
Array bind size *Login timeout*	The *Array bind size*, and *Login timeout* parameters are not used for OLAP connections.
Maximum Parallel Queries	The *Maximum Parallel Queries* parameter allows you to set the maximum number of concurrent tasks for the connection, ranging from 1 to 64. **Note** If the number of maximum parallel queries per connection exceeds the maximum concurrent jobs set in the Cental Management Console for Web Intelligence, the minimum value is considered.

9.2.10 Defining custom parameters for Essbase connections

On the *Custom Parameters* page of the New Connection wizard, specify the alias table and select the dimension to be used as the Measure dimension when generating the universe.

Login parameter	Description
Alias table	To generate the universe on an alias table other than the default, select the alias table from the list.
Measure dimension	Select the dimension to use as the Measure dimension. the universe design tool generates the members of the dimension you select as measures in the universe.

9.3 Customizing OLAP universes

9.3.1 Creating OLAP Universes with additional parameters

This feature applies to OLAP universes only and allows you to define additional metadata parameters when you create an OLAP universe with MSAS, SAP Business Warehouse (BW), or Essbase.

When you create an OLAP universe you can define the following parameters:

Generic OLAP options	**Description**
Generate technical names as details	You can set the application to generate the technical name as a detail object of the dimensions in the universe. When the universe is generated, this creates detail objects that point to the technical names.

SAP OLAP options	**Description**
Set measures aggregation to delegated	You can set the application to set the aggregation function of measures to database delegated.
Replace prefixes L00, L01,	Universe level prefixes indicate the level in the hierarchy of an object. Level L00 is the top or root level, L01 is the next level down. In the "New Universe Wizard", you can replace OLAP universe level prefixes with a different prefix. The level numbering is retained, but the prefix 'L' can be replaced by Level, for example. Type your own prefix in the *New prefix* field. This prefix is prepended to all levels throughout the OLAP universe.
Rename level 00 to All	This option is disabled if *Generate level 00* is set to *No*. You can rename the top level (root level) L00 to All the next time the universe is generated.
Generate level 00	This option only applies to SAP Characteristics. You can disactivate this option for Characteristics and hierarchies. Level 00 is always generated for hierarchy variables. You

SAP OLAP options	Description
	can regenerate the Level numbers (L00, L01, L02...) when you generate or update a universe. The level numbers are appended to the level names (e.g. "Monthly Sales_L01"). This is useful for Web Intelligence reports, where the All level is used to aggregate results for a query. This avoids having to create the aggregate field in the Web Intelligence report.

> **i** Note
>
> When the universe is created with "Generate Level 00" disabled, root level is not generated for hierarchies.

9.3.2 Defining OLAP options for your OLAP universe

Use the OLAP options to define how specific universe metadata is generated from an OLAP source. You can select OLAP options from the *OLAP* page of the *Options* dialog box (*Tools* > *Options* > *OLAP*). All content of the OLAP source is extracted and created in the universe depending on selected options. You can select the following OLAP universe generation options:

Generic OLAP options	Description
Generate technical names as details	You can set the application to generate the technical name of the universe as a property. When the universe is generated, this creates an object that points to the technical name.

SAP OLAP options	Description
Set measures aggregation to delegated	You can set the application to generate delegated measures for measures that use the aggregation function. When the universe is generated, any measures using the aggregation function are set to database delegated.
Replace prefixes L00, L01,	Universe level prefixes indicate the level in the hierarchy of an object. Level L00 is the top or root level, L01 is the next level down. In the "New Universe Wizard", you can replace OLAP universe level prefixes with a different prefix. The level numbering is retained, but the prefix 'L' can be replaced by Level, for example. Type your own prefix in the *New prefix* field. This prefix is prepended to all levels throughout the OLAP universe.
Rename level 00 to All	This option is disabled if Generate level 00 is set to No. You can rename the top level (root level) L00 to ALL when the universe is next generated. This is useful for Web Intelligence reports, where the All level is used to aggregate results for a query. This avoids having to create the aggregate field in the Web Intelligence report.

SAP OLAP options	Description
Generate level 00	This option only applies to SAP Characteristics. You can disactivate this option for Characteristics only. Level 00 is always generated for hierarchies and hierarchy variables. You can regenerate the Level numbers (L00, L01, L02...) when you generate or update a universe. The level numbers are appended to the level names (e.g. "Monthly Sales_L01")

9.3.3 Defining objects in OLAP universes

You can use the SQL editor to define a Select statement or a Where clause for an object, and to insert MDX operators and functions for OLAP universe objects. The options and functions available in the SQL editor depend on the underlying database.

9.3.4 Universe design tool features supported for OLAP universes

OLAP universes are created automatically. Once you have created the OLAP universe, you can modify any of the universe components.

These universe design tool features are supported for generated OLAP universes:

- Hide, duplicate, and rename classes and objects (dimension, detail, and measure)
- Insert new classes and objects (dimension, detail, and measure)
- Edit an object's format
- Edit an object's data type
- Define primary and foreign keys
- Parse dimension, detail, and measure object MDX syntaxes
- Check universe integrity
- Edit hierarchies
- Create cascading list of values
- Define a delegate search for a list of values, allowing users to limit loading of the list of values at query run time
- Use default values for variables
- Define measures with database delegated projection function (smart measures)
- Refresh the universe structure

In addition, the following features are available uniquely for OLAP universes:

- Create calculated measures (SAP BW and MSAS only)
- Create pre-defined conditions
- Define optional prompts

All objects based on OLAP universes are generated with index awareness. If there are duplicate values in a hierarchy for an object, index awareness eliminates inconsistency in the list of values. For example, say that

Paris occurs twice in the hierarchy, once under the parent France and once under the parent Texas. If the user selects Paris under France, only rows for Paris, France are returned.

The following universe design tool features are not supported for OLAP universes:

- You cannot set row-level security authorizations in an OLAP universe.
- You cannot edit a list of values in an OLAP universe.
- You cannot view and edit the universe entity-relationship schema as no schema is generated for OLAP universes.

9.3.5 Database delegated projection function

In a universe, any measure can hold a projection function (*Sum*, *Min*, *Max*, *Count*, and *Avg*). The projection function is used to aggregate the measure locally in Web Intelligence when the number of dimensions displayed in a report is smaller than the number of dimensions in the query result set.

Non-additive measures, such as ratio, average, and weight, can only be shown at the same aggregation level as the query result set. Therefore, non-additive measures generally have their projection function set to *None* in the universe.

The projection function *Database delegated* allows you to delegate the aggregation of a non-additive measure to the database server. These are called smart measures in Web Intelligence. A smart measure has its projection function set to *Database delegated* on the properties page of the object properties. For more information about how these and other functions are used in Web Intelligence , please refer to section *Calculating values with Smart Measures* in the document *Using Functions, Formulas and Calculations in Web Intelligence*.

> **i Note**
>
> For OLAP universes based on MSAS and Essbase data sources, all measures are created in the universe with the projection function set to *Database delegated* by default.

> **i Note**
>
> Please be aware of the following limitation when using a smart measure based on a measure that has aggregate aware set: It is strongly recommended to ensure that the aggregate tables used in the measure definition have consistent data (the aggregate values are accurate regarding detail values) otherwise the smart measures can produce inconsistent data. For example, if a year aggregate table and a day aggregate table are used for a smart measure, the year aggregate table is consistent with the day aggregate table for complete years, but for the current year, the year table can be empty while the day table is accurate on a daily basis. In this case, a report that uses smart measures based on the current year and daily tables can give incoherent results.

Example: Smart measure

In this example, a query contains two dimensions: Country and Region, and three measures: Order Amount, Delivered Amount, and % of Delivered Amount.

L01 Region	Amount Delivered	Order Quantity	% Delivered
Reg1	497,318,880	497,332,680	99.997
Reg2	199,463,776	199,466,536	99.998
Reg3	198,927,552	198,933,072	99.997
		Sum:	299.992

The sum for % Delivered is incorrect because it is a summation of the % Delivered column.

If this measure has a projection function set to *Database delegated* in the universe, when the user refreshes the report, Web Intelligence connects to the database to calculate the correct value.

L01 Region	Amount Delivered	Order Quantity	% Delivered
Reg1	497,318,880	497,332,680	99.997
Reg2	199,463,776	199,466,536	99.998
Reg3	198,927,552	198,933,072	99.997
		Sum:	299.992
		Total:	99.997

> **i** Note
>
> The some functions such as the ratio function (Average) must be used with caution. When calculating an average from a column, the behavior of this function can be unexpected when it is not configured correctly.
>
> For example, the SQL function `sum(Shop_facts.Margin)/sum(Shop.facts.Quantity_sold)` can have unexpected results. When incorrectly configured, it will calculate the average for each cell and return the sum of those averages. To correct this behavior, the parametrization of the function must be performed as follows:
>
> 1. Go to the *Edit Properties* option for the function.
> 2. For the option *Choose how this measure will be projected when aggregated*, select the Function *Db delegated* from the Function dropdown list.
> 3. Save your changes.

Related Information

Setting aggregate projection for a measure [page 419]

9.3.6 Setting delegated measures for OLAP universes

You can set the application to generate delegated measures for measures that use the aggregation function. When the universe is generated, any measures using the aggregation function are set to database delegated...

Related Information

Replacing OLAP universe level prefixes [page 435]
Regenerating Level 00 for OLAP universes [page 434]
Renaming level L00 to ALL [page 435]

9.3.7 Setting aggregate projection for a measure

When you create a measure you must specify the way the aggregate function will be projected onto a report.

Returned values for a measure object are aggregated at two levels of the query process:

- Query level. Data is aggregated using the inferred SELECT statement.
- Microcube to block level. When data is projected from the microcube to the block in a report. This projection function of measures allows local aggregation in the microcube.

> **i** Note
>
> A microcube is a conceptual way to present the data returned by a query before it is projected onto a report. It represents the returned values held in memory by a Business Objects reporting product. The block level is the 2 dimensional report that a user creates with the returned data. A user can choose to use all, or only some of the data held in the microcube to create a report. A user can also do aggregate functions on the returned values in the microcube (local aggregation) to create new values on a report.

The two levels of aggregation fit into the query process as follows:

- User creates a query in Web Intelligence.
- Web Intelligence infers the SQL from the query and sends a SELECT statement to the target database.
- The data is returned to the microcube. This is the first aggregation level.
- The microcube projects the aggregated data onto the report. Data is split out in the Query pane requiring aggregation to lower levels. This is the second aggregation level.

When you initially make a query the result set of the Select statement is stored in the microcube, and all data then held in the microcube is projected into a block. As data is projected from the lowest level held in the microcube no projection aggregation is taking place.

However, when you use the Query pane to project only partial data from the microcube, aggregation is required to show measure values at a higher level.

For example, in the previous example, if you do not project the year data into the block, the three rows related to Year need to be reduced to one row to show the overall Sales Revenue for that resort, so a sum aggregation is used.

You set projection aggregation on the *Properties* page of the *Edit Properties* sheet for a measure (right-click Object > Object Properties > Properties).

Projection aggregation is different from SELECT aggregation.

9.3.8 Calculated measures in OLAP universes

You can create calculated measures in the universe to restrict queries. Calculated measures in OLAP universes are like definitions on the measure object in non-OLAP universes, except that you define the restriction using MDX functions embedded in XML tags rather than using SQL.

Calculated measures are available for these OLAP data sources:

- SAP Business Warehouse (BW)
- MSAS 2000 and 2005

Calculated measures can be used in filters or the where clause.

Syntax for calculated measure expressions

The syntax for a calculated measure consists of calculations embedded in the tags <EXPRESSION></EXPRESSION>.

Universe design tool functions are allowed in calculated measure expressions, such as:

- @Select
- @Prompt
- @Variable
- @Where

> **i Note**
>
> Expressions for calculated measures cannot include the @Aggregate_Aware function. The check integrity function validates the XML syntax and any of the above listed @Functions, including those inserted in the MDX statements. However, the MDX statements are not parsed.

Constants are allowed in the expression, for example "10" or "ABC".

Calculated measures can reference any OLAP metadata:

- measures
- dimensions
- dimension levels
- MDX expressions

Recommendations for calculated measure expressions

Use @Select (Measure name) rather than the measure definition for the following reasons:

- @Select is resolved at query time.
- Calculated measures can reference other calculated measures if they are inside a @Select function.
- The validity of objects inside the @Select function is checked.

Generate and set the index awareness for each object definition.

Use a reference to an object or detail whose definition refers to the Technical Name or Unique Name of the level or attribute.

Example: Calculated measure expression

```
<EXPRESSION>@Select(Key Figures\Order Amount)*@Select(Key Figures\Order
Quantity)</EXPRESSION>
```

Related Information

To create a calculated measure in an OLAP universe [page 421]

9.3.8.1 To create a calculated measure in an OLAP universe

To create a calculated measure in an OLAP universe:

1. In the universe design tool, open an OLAP universe.
2. Insert a new measure object in the universe.
3. In the *Where:* box, enter or paste the object definition as an XML/MDX expression.
4. Click *Parse* to review the object definition and fix any errors.
5. Click *OK* to save the object definition.
6. Select *Tools* > *Check Integrity*.
   The integrity check validates the XML syntax and any universe design tool @FUNCTIONS.

Related Information

Calculated measures in OLAP universes [page 420]

9.3.9 About MDX functions for cube queries

Use the MDX editor to define your cube queries.

When adding a new object or a predefined filter to an OLAP universe, there is a list of supported MDX expressions corresponding to the specific data source connection.

A library of available expressions is stored in the `.prm` connection file. When you open the Edit Properties pane for an object and opend the Edit Select pane for the query, the available expressions are displayed in the Functions pane. To insert the expression in the SELECT or WHERE statement, click in the position in the expression where you want to insert the expression and double-click on the appropriate expression.

OLAP Universe MDX Dictionary - List of Functions (PRM file)

When adding a new object or a predefined filter to an OLAP universe, an explicit list of MDX functions (mainly member functions) and operators is made available in the object and filter editors for the appropriate OLAP connection (SAP or MSAS) that you can use in the expression. For a description of how to set the connectivity for SAP or mySQL (sap.prm, sqlsrv_as.prm), refer to the Data Access Guide. The available functions and operators depend on the connection type of the universe. This list of functions is provided by the PRM file for each connectivity. It does not provide the whole list of supported functions, only the most frequently used functions.

The following MDX operators are available for queries.

- Equal
- NotEqual
- InList
- NotInList
- Greater
- GreaterOrEqual
- Less
- LessOrEqual
- Between
- NotBetween
- Like
- NotLike

The list below shows examples of some of the available MDX folder functions when editing a condition. The available functions depend on the underlying database.

- Set functions (ADDCALCULATEDMEMBERS, ALLMEMBERS ...)
- Statistical/Numeric functions (AGGREGATE, AVG ...)
- Navigation/Member functions (ANCESTOR, ASCENDANTS...)
- Metadata functions (AXIS, HIERARCHY...)

9.3.10 XML syntax for filter and WHERE statements

This section describes the XML syntax for defining the WHERE clause or filter statements in your OLAP universe. You must add the FILTER or FILTER EXPRESSION tags manually, and then enter your expression between the tags either manually or with the universe design tool MDX editor.

- Use <FILTER= "your_object_definition"> when using a single object definition. Enter your object definition inside the double quotes.
- Use <FILTER EXPRESSION= "yourcomplexMDX_expression"> when using a complex MDX expression containing one or more objects. Enter your expression inside the double quotes.

The syntax for a single filter object is as follows:

```
<FILTER = "your_object_definition"><CONDITION
OPERATORCONDITION="yourOperator"><CONSTANT VALUE="your_Value"/></CONDITION></
FILTER>
```

Where:

- yourMDX_expression is the single object definition, enclosed in double quotes.
- CONSTANT VALUE is either CONSTANT CAPTION or CONSTANT TECH_NAME
- yourOperator is the filter expression operator (equals, inlist...). When the InIist operator is used, you must insert a CONSTANT CAPTION or CONSTANT TECH_NAME element for each item in the list.
- your_Value is the defined filter value when CONSTANT CAPTION is used, or the object identifier if CONSTANT TECH_NAME is used.

The syntax for a single filter object using the InList operator, where three countries are listed, is as follows:

```
<FILTER= "your_object_definition "><CONDITION OPERATORCONDITION="InList"><CONSTANT
CAPTION="England"/><CONSTANT CAPTION="France"/><CONSTANT CAPTION="Germany"/></
CONDITION></FILTER>
```

The syntax for a complex filter expression and the TECH_NAME for the filtered value is as follows:

```
<FILTER EXPRESSION="yourComplex_MDX_Expression"><CONDITION
OPERATORCONDITION="Equal"><CONSTANT TECH_NAME="1"/></CONDITION></FILTER>
```

Example: Filter with a calculated member in the filter expression

```
<FILTER EXPRESSION="IIF([0CALYEAR].CurrentMember > "2000", 1,0)"><CONDITION
OPERATORCONDITION="Equal"><CONSTANT CAPTION="1"/></CONDITION></FILTER>
```

9.3.11 Predefined conditions in OLAP universes

Predefined conditions in OLAP universes are like conditions in non-OLAP universes except that you define the WHERE clause using XML rather than SQL. You can declare filters manually, or by using the Predefined filter editor.

9.3.11.1 XML Syntax for predefined filter options

Syntax for predefined conditions

A single predefined condition may contain multiple filters combined with the AND and OR operators. By default, all filters are combined with the AND operator. To include filters using OR, you must use the AND and OR operator tags.

The functions @Select, @Prompt and @Variable are allowed in the predefined filter definition.

Predefined filters can include one or multiple prompts. Prompts can be mandatory or optional.

Example: Using AND and OR tags for pre-defined conditions

```
<OPERATOR VALUE="AND">
    <FILTER "[Level Object definition]">
        <CONDITION OPERATORCONDITION="Operator">
           <CONSTANT Level Attribute="Value"/>
        </CONDITION>
    </FILTER>
    <OPERATOR VALUE="OR">
        <FILTER "[Level Object definition]">
            <CONDITION OPERATORCONDITION="Operator">
                <CONSTANT Level Attribute="Value"/>
            </CONDITION>
        </FILTER>
        <FILTER "[Level Object definition]">
            <CONDITION OPERATORCONDITION="Operator">
                <CONSTANT Level Attribute="Value"/>
                </CONDITION>
            </FILTER>
    </OPERATOR>
</OPERATOR>
```

9.3.11.2 Manually creating pre-defined conditions in an OLAP universe

To create a pre-defined condition:

1. In universe design tool, open an OLAP universe and click the conditions radio button at the bottom of the Universe pane.
   The conditions view of the Universe pane appears. It contains a tree view of the classes in the universe.
2. Right-click a class and select *Condition...* from the contextual menu.
3. In the *Where:* box, edit the XML template filter.

   The template filter has the format:

   ```
   <FILTER "[Level Object definition]">
       <CONDITION OPERATORCONDITION="Operator">
   ```

```
        <CONSTANT Level Attribute="Value"/
    </CONDITION>
</FILTER>
```

Replace the elements in the template as follows:

Template element:	Possible Values:
Level Object definition	Enter the dimension level or measure implied in the filter. Enter the object definition not the object name.
Operator	Enter one of: ○ Equal ○ NotEqual ○ Greater ○ Less ○ GreaterOrEqual ○ LessOrEqual ○ Between ○ NotBetween ○ InList ○ NotInList ○ Like ○ NotLike
Level Attribute	Enter one of: ○ NAME ○ CAPTION ○ TECH_NAME ○ DESCRIPTION
Value	Enter the value or a prompt. Define one value per CONSTANT tag.

An example of an edited predefined condition:

```
<FILTER KEY="[0D_DIV].[LEVEL01]">
    <CONDITION OPERATORCONDITION="InList">
        <CONSTANT CAPTION="Internal"/>
        <CONSTANT CAPTION="Service"/>
    </CONDITION>
</FILTER>
```

4. Click *Parse* to review the syntax and fix any errors.
5. Click *OK* to save the condition.

Related Information

Predefined conditions in OLAP universes [page 423]
Optional prompts in OLAP universes [page 428]

9.3.11.3 About the Pre-defined Filter editor

The *Pre-defined filter* editor is for editing pre-defined filters in OLAP universes. Use it to select objects, operators, lists of values, prompts, functions, and other optional elements that can be used to define a filter for your OLAP universe.

In the condition properties panel of a filter, you can manually type the filter expression or click >> to open the *Pre-defined filter* editor. When the editor is open, you can insert an `@Prompt` in the filter expression: Right-click at the appropriate point in the filter expression and select *New @Prompt* from the shortcut menu. The pre-defined filter editor inserts the filter expression in the query/object definition.

Example: Restriction on dimension Customer at the country level to restrict country to Canada

<FILTER KEY="[Customer].[Country].[Country]"> <CONDITION OPERATORCONDITION="Equal">
<CONSTANT CAPTION="Canada" /> </CONDITION> </FILTER>

Related Information

About the options for the Pre-defined Filter editor [page 426]
Editing a pre-defined filter with the pre-defined filter editor [page 427]
About MDX functions for cube queries [page 422]

9.3.11.4 About the options for the Pre-defined Filter editor

The *Pre-defined Filter* editor allows you to easily define a universe filter for an OLAP universe. You can define the following options:

Option	Description
Select an Operator	Select an operator from the available list. Default = *Equal*
Base the filter on	Filter either on an existing universe object or on a free definition (for example: [Measures].[Internet Sales Amount]). Default = *Universe object*.
Select a LoV	Select a list of objects in the current universe when filter based on existing object. Default selection = The Root class in the list of objects.
Comparison values	Define values to compare the object/expression to. Depending on selected operator, there are one or two sets of values to enter. Values can either be static or based on a prompt. Default = *Static values*.

Option	Description
Insert a Prompt	Edit a prompt manually, or use the *@Prompt* editor. Click >> to open the *@Prompt* editor.
Set Index Awareness	Enable the index awareness function. Primary keys must be declared for this to work correctly. When index awareness is set up in the universe design tool, the primary and foreign key columns are used to speed up data retrieval and to allow the universe design tool to generate more efficient SQL filters. Default = Not selected.
Use calculated expression	When selected, this encloses the filter expression inside <EXPRESSION> </EXPRESSION> tags. Default = Not selected.
Optional	Set the current filter expression as optional. This applies to the current filter expression in the filter editor only and not to the entire pre-defined condition object. Default = Not selected.

> **i** Note
>
> "Optional" tags cannot be used for pre-defined filters in Web Intelligence. If these tags are used, they are treated as a mandatory part of the query, therefore preventing the query from running.

Related Information

Editing a pre-defined filter with the pre-defined filter editor [page 427]

9.3.11.5 Editing a pre-defined filter with the pre-defined filter editor

You are editing a filter in an OLAP universe.

The *Pre-defined Filter* editor is updated as you select or enter the values. You can right-click in the filter expression to insert an `@Prompt` expression in the filter expression. When you right-click and select *New @Prompt*, the *Prompt* editor opens.

1. In the *Properties* pane of the condition (filter) pane, click >>.
   The *Pre-defined Filter* editor appears.
2. To base the filter on a universe object, select *Universe object* and choose an object from the *Available Objects* pane. To base the predefined filter on your own expression, select *Free definition* and type the expression in the *Available Objects* pane.
3. Select an operator from the *Operators* list. Multiple values (right operand) are allowed for the In List and Not In List operators only.
4. Select *Static value* to define one or more fixed values, or select *Prompt* to insert a prompt expression. When you select *Prompt* , the *Edit* button is activated. Click *Edit* to open the *@Prompt* editor and define the prompt expression as required.

5. Click *OK* to validate the filter definition.
   The parser checks the syntax for errors, including the integrity check. If errors are found a warning message with error message is shown. When no errors are found, the new condition object is added to the universe with the filter definition.

Related Information

About the options for the Pre-defined Filter editor [page 426]
About the Pre-defined Filter editor [page 426]

9.3.12 Optional prompts in OLAP universes

Universes generated from OLAP data sources support optional prompts.

For SAP BW optional variables, filters with the optional conditions are automatically generated in the universe.

In pre-defined conditions or in an object's WHERE clause, to make a prompt optional, embed the XML filter expression between the two XML tags: <OPTIONAL> and </OPTIONAL>.

Example: Optional prompt in a pre-defined condition

```
<OPTIONAL>
    <FILTER KEY="[Products].[Family]" >
        <CONDITION OPERATORCONDITION="InList" >
            <CONSTANT CAPTION="@prompt('Enter value(s) for Product
family:','A','Products\Family',Multi,primary_key,persistent)"/>
        </CONDITION>
     </FILTER>
</OPTIONAL>
```

Related Information

Manually creating pre-defined conditions in an OLAP universe [page 424]

9.3.13 To improve performance of certain queries on SAP BW universes

For queries on SAP BW universes that include only the key and medium name detail objects of a dimension, you can modify the generated syntax of the objects to improve query performance.

To modify the syntax:

1. Open the universe in universe design tool.
2. Double click the key detail object you want to modify.
3. In the Select text box on the *Definition* tab of the *Edit Properties* dialog box, change the syntax to refer to the NAME attribute of the SAP characteristic.
   For example, for the object *L01 Customer Key*, change the generated select syntax:

   ```
   [Z_CUSTOM].[LEVEL01].[[2Z_CUSTOM]].[Value]
   ```

   to refer to the NAME attribute:

   ```
   [Z_CUSTOM].[LEVEL01].[NAME]
   ```
4. Click *OK* to save the changes.
5. Follow the same steps for the name object. Change the syntax to refer to the DESCRIPTION attribute of the SAP characteristic.
   For example, for the object *L01 Customer Medium Name*, change the generated select syntax:

   ```
   [Z_CUSTOM].[LEVEL01].[[5Z_CUSTOM]].[Value]
   ```

   to refer to the DESCRIPTION attribute:

   ```
   [Z_CUSTOM].[LEVEL01].[DESCRIPTION]
   ```

9.4 OLAP universe lifecycle management

9.4.1 About OLAP universe lifecycle management

> **i Note**
>
> When you open a universe created with a version of Universe Designer prior to XIR3.1 SP2, you must refresh and save the universe before making any changes in the universe or in the OLAP source.

OLAP universes are generated automatically from an OLAP data source (for example, an SAP BEx Query or an MSAS 2005 cube). In the universe design tool, you can create and change objects in the existing OLAP universe.

The *Update OLAP Universe* wizard allows you to refresh the structure of OLAP universes automatically with changes made in the OLAP data source. The wizard compares the universe with the updated data source. The wizard can differentiate generated objects from objects added or modified manually, allowing you to keep the changes made manually in the universe design tool. The wizard does not update objects that were manually added in the universe design tool.

What can be detected and updated depends on the items and the data source, as shown in the table below.

What the wizard can detect	New items can be detected in	Modified items can be detected in	Deleted items can be detected in
Dimensions	All data sources	All data sources	All data sources
Hierarchies	SAP BW and MSAS only	All data sources	All data sources
Levels	All data sources	All data sources	All data sources
Properties	MSAS only	MSAS only	MSAS only
Measures	All data sources	All data sources	All data sources
SAP BW Variables	SAP BW only	SAP BW only	SAP BW only
Sub-classes	All data sources	All data sources	All data sources

> **i** Note
>
> When you update a universe created with a version of Universe Designer earlier than XIR3.1 SP2, if the name of a dimension has changed in the SAP cube, the refresh of the dimension does not work: The dimension is duplicated in the universe. You must manually update the classes in the universe.

Related Information

To refresh an OLAP universe [page 432]
Overview: relationship between universe object status and OLAP object status [page 430]
How dimensions are managed in OLAP universe updates [page 436]
How hierarchies or characteristics are managed in OLAP universe updates [page 440]
How levels are managed in OLAP universe updates [page 447]
How SAP variables are managed in OLAP universe updates [page 450]
How key figures or measures are managed in OLAP universe updates [page 453]
How SAP key dates are managed in OLAP universe updates [page 457]

9.4.2 Overview: relationship between universe object status and OLAP object status

The table below gives a brief overview of the relationship between an SAP OLAP object state and universe object state. There are specific notes concerning most of the actions, for more information refer to the more detailed sections in this chapter.

		Universe object status				
OLAP metadata		**Unchanged**	**Updated***	**Deleted**	**Moved**	**Hidden**
Dimension		Universe equivalent = class				
Status	Unchanged	NoC	Upd	NoC	NoC	NoC

OLAP metadata		Universe object status				
		Unchanged	**Updated***	**Deleted**	**Moved**	**Hidden**
	Updated*	Upd	Upd	NoC	Upd	Upd
	Deleted	Del/Ob	Del/Ob	NoC	Del/Ob	NoC
	Moved	Move	NoC	NoC	NoC	Move
	Characteristic created	CreS	CreS	N/A	CreS	CreS
	Created	Cre	Cre	N/A	Cre	Cre
Hierarchy or Characteristic		Universe equivalent = sub-class				
Status	Unchanged	NoC	Upd	NoC	NoC	NoC
	Updated*	Upd	Upd	NoC	Upd	Upd
	Changed	UpdMDX	UpdMDX	NoC	UpdMDX	UpdMDX
	Display Attribute	Cre	Cre	Cre	Cre	Cre
	Navigation Attribute	Del/Ob	Del/Ob	NoC	Del/Ob	Del/Ob
	Deleted	Del/Ob	Del/Ob	NoC	Del/Ob	Del/Ob
	Moved	Move	Move	NoC	Move	Move
	New	Cre	Cre	Cre	Cre	Cre
Level		Universe equivalent = level				
Status	Unchanged	NoC	NoC	NoC	NoC	NoC
	Updated*	Upd	Upd	NoC	Upd	Upd
	Deleted	Del/Ob	Del/Ob	NoC	Del/Ob	Del/Ob
	Moved	Move	Move	NoC	Move	Move
	New	Cre	Cre	Cre	Cre	Cre
Variable		Universe equivalent = filter				
Status	Unchanged	NoC	NoC	NoC	NoC	NoC
	Updated*	Upd	Upd	Cre	Upd	Upd
	Deleted	Del/Ob	Del/Ob	NoC	Del/Ob	Del/Ob
	New	Cre	Cre	Cre	Cre	Cre
Key figure		Universe equivalent = measure				
Status	Unchanged	NoC	NoC	NoC	NoC	NoC
	Updated*	Upd	Upd	NoC	Upd	Upd
	Deleted	Del/Ob	Del/Ob	NoC	Del/Ob	Del/Ob
	Moved	Move	Move	NoC	Move	Move
	New	Cre	Cre	Cre	Cre	Cre
Key date		Universe equivalent = parameter				

		Universe object status				
OLAP metadata		**Unchanged**	**Updated***	**Deleted**	**Moved**	**Hidden**
Status	Unchanged	NoC	N/A	Cre	N/A	N/A
	Deleted	Del	N/A	N/A	N/A	N/A
	New	Cre	N/A	Cre	N/A	N/A

LEGEND:

- *:The one of the object properties (name, description...) has changed.
- Cre: Create the equivalent object
- CreS: Create the equivalent sub-class object
- Del/Ob: Deleted or Obsolete (obsolete objects are hidden and their names prefixed with ##)
- Move: The object is moved
- N/A: Does not apply
- NoC: No change
- Upd: Updated
- UpdMDX: Update the MDX definition

Related Information

To refresh an OLAP universe [page 432]
How dimensions are managed in OLAP universe updates [page 436]
How hierarchies or characteristics are managed in OLAP universe updates [page 440]
How levels are managed in OLAP universe updates [page 447]
How SAP variables are managed in OLAP universe updates [page 450]
How key figures or measures are managed in OLAP universe updates [page 453]
How SAP key dates are managed in OLAP universe updates [page 457]

9.4.3 To refresh an OLAP universe

To refresh the structure of an OLAP universe:

- In the universe design tool, open the universe you want to update.
- Select *View* > *Refresh Structure*.
  The Update OLAP Universe wizard appears.
- Click *Begin*.

  The Updated metadata definitions page appears.

  Select keep options when you want to preserve modifications to objects in the universe that were made manually. All keep options are selected by default. You can choose to keep the following attributes:

Option	Description
Keep business name	Class, Dimension, Measure, Detail and Condition names.
Keep type of objects	If the object has changed in the universe (for example, a detail is changed to a dimension, the update will not reintroduce the initial object type). This concerns Dimensions, Measures, and Details.
Keep description of objects	When this is selected, if the description is updated in the OLAP source, the universe is not updated with this information.
Keep data type of objects	Character, Numeric, Date, and Long text objects.
Keep list of values options of objects	You can keep the options that were initially set: ○ Associate a list of values ○ Automatic refresh before use ○ Hierarchical display ○ Export with universe ○ Delegate search
Keep advanced options of objects	The options are: Security access level The object can be used in ○ Used in result ○ Used in condition ○ Used in sort
Delete obsolete objects	Items that are no longer in the data source will be deleted from the universe.
Hide obsolete objects	Items that are no longer in the cube will be hidden in the universe and prefixed by /##/.

- Select the desired options and click *Next*.

  The *Change Management Results* page appears showing added/deleted/hidden objects. Hidden objects are moved to a separate class in the universe and appear in italic font preceded by /##/.
- In the Added metadata options pane, set the options for how added metadata is treated

Generic OLAP options	Description
Generate technical names as details	You can set the application to generate the technical name of the universe as a property. When the universe is generated, this creates an object that points to the technical name.
Regenerate all manually deleted objects	Any manually deleted universe objects will be regenerated.

SAP OLAP options	Description
Set measures aggregation to delegated	You can set the application to set the aggregation function of measures to database delegated.

SAP OLAP options	Description
Replace prefixes L00, L01,	Universe level prefixes indicate the level in the hierarchy of an object. Level L00 is the top or root level, L01 is the next level down. In the "New Universe Wizard", you can replace OLAP universe level prefixes with a different prefix. The level numbering is retained, but the prefix 'L' can be replaced by Level, for example. Type your own prefix in the *New prefix* field. This prefix is prepended to all levels throughout the OLAP universe.
Rename level 00 to All	This option is disabled if *Generate level 00* is set to *No*. You can rename the top level (root level) L00 to All the next time the universe is generated.
Generate level 00	This option only applies to SAP Characteristics. You can disactivate this option for Characteristics only. Level 00 is always generated for hierarchies and hierarchy variables. You can regenerate the Level numbers (L00, L01, L02...) when you generate or update a universe. The level numbers are appended to the level names (e.g. "Monthly Sales_L01"). This is useful for Web Intelligence reports, where the All level is used to aggregate results for a query. This avoids having to create the aggregate field in the Web Intelligence report.

- On the Change Management results page, select one of:

Option	Description
OK	If you are not satisfied with the results, you can click *OK*, then close the Universe without saving or exporting.
Export	If you are satisfied with the changes, click *Export* to save and export the updated universe to the CMS.
Check Integrity	Click *Check Integrity* to perform an integrity check. It checks the structure, parses objects, parses joins, parses conditions, and checks cardinality. When the check is complete, the *Integrity Check Results* page appears. From this page you can print the results of the check.

If you do not see all changes to the universe that you expect, stop, then restart the universe design tool before trying the update again. This makes a new connection to the data source and clears the cache.

Related Information

Synchronizing the universe and the OLAP cube [page 435]
About OLAP universe lifecycle management [page 429]

9.4.4 Regenerating Level 00 for OLAP universes

You can regenerate the Level numbers (L00, L01, L02...) when you generate or update a universe. The level numbers are appended to the level names (e.g. "Monthly Sales_L01")

Related Information

Setting delegated measures for OLAP universes [page 419]
Replacing OLAP universe level prefixes [page 435]
Renaming level L00 to ALL [page 435]

9.4.5 Renaming level L00 to ALL

You can rename the top level (root level) L00 to ALL when the universe is next generated. This is useful for SAP BusinessObjects Web Intelligence reports, where the ALL level is used to aggregate results for a query. This avoids having to create the aggregate field in the Web Intelligence report.

Related Information

Setting delegated measures for OLAP universes [page 419]
Replacing OLAP universe level prefixes [page 435]
Regenerating Level 00 for OLAP universes [page 434]

9.4.6 Replacing OLAP universe level prefixes

Universe level prefixes indicate the level in the hierarchy of an object. Level L00 is the top or root level, L01 is the next level down. In the *New Universe Wizard*, you can replace OLAP universe level prefixes with a different prefix. The level numbering is retained, but the prefix 'L' can be replaced by Level, for example. Type your own prefix in the *New prefix* field. This prefix is prepended to all levels throughout the OLAP universe.

Related Information

Setting delegated measures for OLAP universes [page 419]
Regenerating Level 00 for OLAP universes [page 434]
Renaming level L00 to ALL [page 435]

9.4.7 Synchronizing the universe and the OLAP cube

When you update a universe, the objects in the universe are compared with the objects in the OLAP cube. The comparison ensures that changes made in the cube do not adversely affect the universe. This means that any objects used (and even deleted) in the universe must always be available. Any new objects in the OLAP cube

are made available to the universe. To see how the different objects are affected by changes, see the links below.

When object properties are updated, only certain properties are updated in the universe, other properties should not change. The following table shows what happens.

Related Information

To refresh an OLAP universe [page 432]
How dimensions are managed in OLAP universe updates [page 436]
How hierarchies or characteristics are managed in OLAP universe updates [page 440]
How levels are managed in OLAP universe updates [page 447]
How SAP variables are managed in OLAP universe updates [page 450]
How key figures or measures are managed in OLAP universe updates [page 453]
How SAP key dates are managed in OLAP universe updates [page 457]

9.4.8 How dimensions are managed in OLAP universe updates

This applies to SAP, MSAS, and Essbase data sources. The universe class is the equivalent of the OLAP dimension. How the universe objects are managed with respect to the OLAP objects depends on the type of change. Refer to the topics listed below to see how the universe objects are impacted by specific OLAP object changes.

Related Information

To refresh an OLAP universe [page 432]
When a dimension is unchanged [page 436]
When a dimension is updated (name, description) [page 437]
When a dimension is deleted [page 438]
When a dimension is moved [page 439]
When a hierachy or characteristic is created [page 439]
When a dimension is new [page 440]

9.4.8.1 When a dimension is unchanged

The following table shows what happens to the equivalent universe class in the different possible situations when the dimension is unchanged:

When the universe class	What happens to the universe class
Is unchanged	The universe class is unchanged
Is updated	Update the business name if the option *Keep business name* is not selected. Update the description if the option *Keep description* is not selected. Remain unchanged if these options are selected.
Is deleted	No change to the universe class. Create the object if the option *Regenerate Objects manually deleted* is selected. Do not regenerate children that had not been deleted
Is moved	No change to the universe class.
Is hidden	No change to the universe class.

Related Information

To refresh an OLAP universe [page 432]
When a dimension is updated (name, description) [page 437]
When a dimension is deleted [page 438]
When a dimension is moved [page 439]
When a hierachy or characteristic is created [page 439]
When a dimension is new [page 440]

9.4.8.2 When a dimension is updated (name, description)

The following table shows what happens to the equivalent universe class in the different possible situations when the name or description of the dimension is updated:

When the universe class	What happens to the universe class
Is unchanged	Update the business name if the option *Keep business name* is not selected. Update the description if the option *Keep description* is not selected. Remain unchanged if these options are selected.
Is updated	Update the business name if the option *Keep business name* is not selected. Update the description if the option *Keep description* is not selected. Remain unchanged if these options are selected.
Is deleted	No change to the universe class. Create if the option *Regenerate Objects manually deleted* is selected. Do not regenerate children that had not been deleted.

When the universe class	What happens to the universe class
Is moved	Update the business name if the option *Keep business name* is not selected. Update the description if the option *Keep description* is not selected. Remain unchanged if these options are selected.
Is hidden	Update the business name if the option *Keep business name* is not selected. Update the description if the option *Keep description* is not selected. Remain unchanged if these options are selected.

Related Information

To refresh an OLAP universe [page 432]
When a dimension is unchanged [page 436]
When a dimension is deleted [page 438]
When a dimension is moved [page 439]
When a hierachy or characteristic is created [page 439]
When a dimension is new [page 440]

9.4.8.3 When a dimension is deleted

The following table shows what happens to the equivalent universe class in the different possible situations when the dimension is deleted:

When the universe class	What happens to the universe class
Is unchanged	Delete if the option *Delete obsolete objects* is selected. Make the sub-class obsolete if the option *Hide obsolete objects* is selected. Do not delete if the class contains custom objects
Is updated	Delete if the option *Delete obsolete objects* is selected. Make the sub-class obsolete if the option *Hide obsolete objects* is selected. Do not delete if the class contains custom objects
Is deleted	No change to the universe class.
Is moved	Delete if the option *Delete obsolete objects* is selected. Make the sub-class obsolete if the option *Hide obsolete objects* is selected. Do not delete if the class contains custom objects
Is hidden	No change to the universe class

Related Information

To refresh an OLAP universe [page 432]
When a dimension is unchanged [page 436]
When a dimension is updated (name, description) [page 437]
When a dimension is moved [page 439]
When a hierachy or characteristic is created [page 439]
When a dimension is new [page 440]

9.4.8.4 When a dimension is moved

The following table shows what happens to the equivalent universe class in the different possible situations when the dimension is moved:

When the universe class	What happens to the universe class
Is unchanged	Move the class accordingly
Is updated	No change
Is deleted	No change. Create if the option *Regenerate Objects manually deleted* = *Yes* Do not regenerate children that had not been deleted
Is moved	No change
Is hidden	Move the class accordingly

Related Information

To refresh an OLAP universe [page 432]
When a dimension is unchanged [page 436]
When a dimension is updated (name, description) [page 437]
When a dimension is deleted [page 438]
When a hierachy or characteristic is created [page 439]
When a dimension is new [page 440]

9.4.8.5 When a hierachy or characteristic is created

Hierachy applies to MSAS or Essbase data sources, characteristic applies to SAP data sources. The following table shows what happens to the equivalent universe class in the different possible situations when the SAP characteristic is created:

When the universe class	What happens to the universe class
Is unchanged	Create sub-class
Is updated	Create sub-class
Is deleted	Does not apply
Is moved	Create sub-class
Is hidden	Create sub-class

Related Information

To refresh an OLAP universe [page 432]
When a dimension is unchanged [page 436]
When a dimension is updated (name, description) [page 437]
When a dimension is deleted [page 438]
When a dimension is moved [page 439]
When a dimension is new [page 440]

9.4.8.6 When a dimension is new

The universe class is created when the dimension is created.

Related Information

To refresh an OLAP universe [page 432]
When a dimension is unchanged [page 436]
When a dimension is updated (name, description) [page 437]
When a dimension is deleted [page 438]
When a dimension is moved [page 439]
When a hierachy or characteristic is created [page 439]

9.4.9 How hierarchies or characteristics are managed in OLAP universe updates

This section applies to hierarchies for MSAS and Essbase data sources, and characteristics for SAP data sources. The universe sub-class is the equivalent of the OLAP characteristic. How the universe objects are managed with respect to the OLAP objects depends on the type of change. Refer to the topics listed below to see how the universe objects are impacted by specific OLAP object changes.

Related Information

To refresh an OLAP universe [page 432]
When a hierarchy or characteristic is unchanged [page 441]
When a characteristic business name or description is updated [page 442]
When a characteristic's active hierarchy is changed [page 442]
When a characteristic display attribute is changed to a navigation attribute [page 443]
When a hierarchy or characteristic is deleted [page 445]
When a hierarchy or characteristic is moved [page 446]
When a hierarchy or characteristic is new [page 446]

9.4.9.1 When a hierarchy or characteristic is unchanged

The following table shows what happens to the equivalent universe sub-class in the different possible situations when the hierarchy or characteristic is unchanged:

When the universe sub-class	What happens to the universe sub-class
Is unchanged	No change
Is updated	No change
Is deleted	No change. Create if Option: *Regenerate Objects manually deleted* = *Yes* Do not regenerate children levels that had not been deleted
Is moved	No change
Is hidden	No change

Related Information

To refresh an OLAP universe [page 432]
When a characteristic business name or description is updated [page 442]
When a characteristic's active hierarchy is changed [page 442]
When a characteristic display attribute is changed to a navigation attribute [page 443]
When a hierarchy or characteristic is deleted [page 445]
When a hierarchy or characteristic is moved [page 446]
When a hierarchy or characteristic is new [page 446]

9.4.9.2 When a characteristic business name or description is updated

The following table shows what happens to the equivalent universe sub-class in the different possible situations when the characteristic is updated:

When the universe sub-class	What happens to the universe sub-class
Is unchanged	Update the business name if the option *Keep business name* is not selected. Update the description if the option *Keep description* is not selected. Remain unchanged if these options are selected.
Is updated	Update the business name if the option *Keep business name* is not selected. Update the description if the option *Keep description* is not selected. Remain unchanged if these options are selected.
Is deleted	No change. Create if Option: *Regenerate Objects manually deleted* = *Yes* Do not regenerate children levels that had not been deleted
Is moved	Update the business name if the option *Keep business name* is not selected. Update the description if the option *Keep description* is not selected. Remain unchanged if these options are not selected.
Is hidden	Update the business name if the option *Keep business name* is not selected. Update the description if the option *Keep description* is not selected. Remain unchanged if these options are selected.

Related Information

To refresh an OLAP universe [page 432]
When a hierarchy or characteristic is unchanged [page 441]
When a characteristic's active hierarchy is changed [page 442]
When a characteristic display attribute is changed to a navigation attribute [page 443]
When a hierarchy or characteristic is deleted [page 445]
When a hierarchy or characteristic is moved [page 446]
When a hierarchy or characteristic is new [page 446]

9.4.9.3 When a characteristic's active hierarchy is changed

This applies to SAP data sources only. The following table shows what happens to the equivalent universe sub-class in the different possible situations when the characteristic's active hierarchy has changed:

When the universe sub-class	What happens to the universe sub-class
Is unchanged	Update the MDX definitions of existing objects in the sub-class to reference the new active hierarchy. Reports built before the refresh continue to work
Is updated	Update the MDX definitions of existing objects in the sub-class to reference the new active hierarchy. Reports built before the refresh continue to work
Is deleted	No change. Create if the option *Regenerate Objects manually deleted* = *Yes* Does not regenerate children levels that were not deleted.
Is moved	Update the MDX definitions of existing objects in the sub-class to reference the new active hierarchy. Reports built before the refresh continue to work.
Is hidden	Update the MDX definitions of existing objects in the sub-class to reference the new active hierarchy.

Related Information

To refresh an OLAP universe [page 432]
When a hierarchy or characteristic is unchanged [page 441]
When a characteristic business name or description is updated [page 442]
When a characteristic display attribute is changed to a navigation attribute [page 443]
When a hierarchy or characteristic is deleted [page 445]
When a hierarchy or characteristic is moved [page 446]
When a hierarchy or characteristic is new [page 446]

9.4.9.4 When a characteristic display attribute is changed to a navigation attribute

This applies to SAP data sources only. The following table shows what happens to the equivalent universe sub-class in the different possible situations when the characteristic display attribute is changed to a navigation attribute:

When the universe sub-class	What happens to the universe sub-class
Is unchanged	Create
Is updated	Create
Is deleted	Create

When the universe sub-class	What happens to the universe sub-class
Is moved	Create
Is hidden	Create

Related Information

To refresh an OLAP universe [page 432]
When a hierarchy or characteristic is unchanged [page 441]
When a characteristic business name or description is updated [page 442]
When a characteristic's active hierarchy is changed [page 442]
When a hierarchy or characteristic is deleted [page 445]
When a hierarchy or characteristic is moved [page 446]
When a hierarchy or characteristic is new [page 446]

9.4.9.5 When a characteristic navigation attribute is changed to a display attribute

This applies to SAP data sources only. The following table shows what happens to the equivalent universe sub-class in the different possible situations when the hierarchy or characteristic navigation attribute is changed to a display attribute:

When the universe sub-class	What happens to the universe sub-class
Is unchanged	Delete if the option *Delete obsolete objects* is selected. Make the sub-class hidden if the option *Hide obsolete objects* is selected. Does not delete if the class contains custom objects.
Is updated	Delete if the option *Delete obsolete objects* is selected. Make the sub-class hidden if the option *Hide obsolete objects* is selected. Does not delete if the class contains custom objects.
Is deleted	No change
Is moved	Delete if the option *Delete obsolete objects* is selected. Make the sub-class hidden if the option *Hide obsolete objects* is selected. Does not delete if the class contains custom objects.
Is hidden	Delete if the option *Delete obsolete objects* is selected. Make the sub-class hidden if the option *Hide obsolete objects* is selected. Does not delete if the class contains custom objects.

Related Information

To refresh an OLAP universe [page 432]

When a hierarchy or characteristic is unchanged [page 441]
When a characteristic business name or description is updated [page 442]
When a characteristic's active hierarchy is changed [page 442]
When a hierarchy or characteristic is deleted [page 445]
When a hierarchy or characteristic is moved [page 446]
When a hierarchy or characteristic is new [page 446]

9.4.9.6 When a hierarchy or characteristic is deleted

The following table shows what happens to the equivalent universe sub-class in the different possible situations when the hierarchy or characteristic is deleted:

When the universe sub-class	What happens to the universe sub-class
Is unchanged	Delete if the option *Delete obsolete objects* is selected. Make the sub-class obsolete if the option *Hide obsolete objects* is selected. Do not delete if the sub-class contains custom objects.
Is updated	Delete if the option *Delete obsolete objects* is selected, make the sub-class obsolete if the option *Hide obsolete objects* is selected. Do not delete if the sub-class contains custom objects.
Is deleted	No change
Is moved	Delete if the option *Delete obsolete objects* is selected, make the sub-class obsolete if the option *Hide obsolete objects* is selected. Do not delete if the sub-class contains custom objects.
Is hidden	Delete if the option *Delete obsolete objects* is selected, make the sub-class obsolete if the option *Hide obsolete objects* is selected. Do not delete if the sub-class contains custom objects.

Related Information

To refresh an OLAP universe [page 432]
When a hierarchy or characteristic is unchanged [page 441]
When a characteristic business name or description is updated [page 442]
When a characteristic's active hierarchy is changed [page 442]
When a characteristic display attribute is changed to a navigation attribute [page 443]
When a hierarchy or characteristic is moved [page 446]
When a hierarchy or characteristic is new [page 446]

9.4.9.7 When a hierarchy or characteristic is moved

If the characteristic is moved within the same dimension, there is no change: ignore the table below. The following table shows what happens to the equivalent universe sub-class in the different possible situations when the hierarchy or characteristic is moved to another dimension:

When the universe sub-class	What happens to the universe sub-class
Is unchanged	Move the sub-class accordingly.
Is updated	Move the sub-class accordingly.
Is deleted	No change. Create if the option: *Regenerate Objects manually deleted* = *Yes* Do not regenerate children levels that had not been deleted.
Is moved	No change.
Is hidden	Move the sub-class accordingly.

Related Information

To refresh an OLAP universe [page 432]
When a hierarchy or characteristic is unchanged [page 441]
When a characteristic business name or description is updated [page 442]
When a characteristic's active hierarchy is changed [page 442]
When a characteristic display attribute is changed to a navigation attribute [page 443]
When a hierarchy or characteristic is deleted [page 445]
When a hierarchy or characteristic is new [page 446]

9.4.9.8 When a hierarchy or characteristic is new

The universe sub-class created when the hierarchy or characteristic is created.

Related Information

To refresh an OLAP universe [page 432]
When a hierarchy or characteristic is unchanged [page 441]
When a characteristic business name or description is updated [page 442]
When a characteristic's active hierarchy is changed [page 442]
When a characteristic display attribute is changed to a navigation attribute [page 443]
When a hierarchy or characteristic is deleted [page 445]

When a hierarchy or characteristic is moved [page 446]

9.4.10 How levels are managed in OLAP universe updates

> **i** Note
>
> In the universe, do not move levels to other hierarchies. If you want to move a level, copy and paste the level to the new hierarchy.

The universe level or dimension object is the equivalent of the OLAP level. How the universe objects are managed with respect to the OLAP objects depends on the type of change. Refer to the topics listed below to see how the universe objects are impacted by specific OLAP object changes.

Related Information

To refresh an OLAP universe [page 432]
When a level is unchanged [page 447]
When the name or description of a level is updated [page 448]
When a level is deleted [page 449]
When a level is moved [page 449]
When a level is new [page 450]

9.4.10.1 When a level is unchanged

The following table shows what happens to the universe level in the different possible situations when the level is unchanged:

When the universe level	What happens to the universe level
Is unchanged	No change
Is updated	No change
Is deleted	No change. Create if the option *Regenerate Objects manually deleted* = *Yes*
Is moved	No change
Is hidden	No change

Related Information

To refresh an OLAP universe [page 432]

When the name or description of a level is updated [page 448]
When a level is deleted [page 449]
When a level is moved [page 449]
When a level is new [page 450]

9.4.10.2 When the name or description of a level is updated

The following table shows what happens to the universe level in the different possible situations when the name or description of a level is updated:

When the universe level	What happens to the universe level
Is unchanged	Update the business name if the option *Keep business name* is not selected. Update the description if the option *Keep description* is not selected. Remain unchanged if these options are selected.
Is updated	Update the business name if the option *Keep business name* is not selected. Update the description if the option *Keep description* is not selected. Remain unchanged if these options are selected.
Is deleted	No change. Create if the option: *Regenerate Objects manually deleted* = *Yes*
Is moved	Update the business name if the option *Keep business name* is not selected. Update the description if the option *Keep description* is not selected. Remain unchanged if these options are selected.
Is hidden	Update the business name if the option *Keep business name* is not selected. Update the description if the option *Keep description* is not selected. Remain unchanged if these options are selected.

Related Information

To refresh an OLAP universe [page 432]
When a level is unchanged [page 447]
When a level is deleted [page 449]
When a level is moved [page 449]
When a level is new [page 450]

9.4.10.3 When a level is deleted

The following table shows what happens to the universe level in the different possible situations when the level is deleted:

When the universe level	What happens to the universe level
Is unchanged	Delete if the option *Delete obsolete objects* is selected. Make the sub-class obsolete if the option *Hide obsolete objects* is selected.
Is updated	Delete if the option *Delete obsolete objects* is selected. Make the sub-class obsolete if the option *Hide obsolete objects* is selected.
Is deleted	No change.
Is moved	Delete if the option *Delete obsolete objects* is selected. Make the sub-class obsolete if the option *Hide obsolete objects* is selected.
Is hidden	Delete if the option *Delete obsolete objects* is selected. Make the sub-class obsolete if the option *Hide obsolete objects* is selected.

Related Information

To refresh an OLAP universe [page 432]
When a level is unchanged [page 447]
When the name or description of a level is updated [page 448]
When a level is moved [page 449]
When a level is new [page 450]

9.4.10.4 When a level is moved

The following table shows what happens to the universe level in the different possible situations when the level is moved:

When the universe level	What happens to the universe level
Is unchanged	Move the level accordingly (within the same hierarchy).
Is updated	Move the level accordingly (within the same hierarchy).
Is deleted	No change. Create if the option *Regenerate Objects manually deleted* = *Yes*.
Is moved	No change. Create if the option *Regenerate Objects manually deleted* = *Yes*.
Is hidden	Move the level accordingly (within the same hierarchy).

Related Information

To refresh an OLAP universe [page 432]
When a level is unchanged [page 447]
When the name or description of a level is updated [page 448]
When a level is deleted [page 449]
When a level is new [page 450]

9.4.10.5 When a level is new

The universe level is created when the OLAP level is created.

Related Information

To refresh an OLAP universe [page 432]
When a level is unchanged [page 447]
When the name or description of a level is updated [page 448]
When a level is deleted [page 449]
When a level is moved [page 449]

9.4.11 How SAP variables are managed in OLAP universe updates

This section only concerns SAP data sources. The universe filter and associated list of values objects are the equivalent of the OLAP variable. How the universe objects are managed with respect to the OLAP objects depends on the type of change. Refer to the topics listed below to see how the universe objects are impacted by specific OLAP object changes.

Related Information

To refresh an OLAP universe [page 432]
When an SAP variable is unchanged [page 451]
When the name or description of an SAP variable is updated [page 451]
When an SAP variable is deleted [page 452]
When an SAP variable is new [page 453]

9.4.11.1 When an SAP variable is unchanged

The following table shows how universe filters are managed in the different possible situations when the SAP source variable is unchanged:

When the universe filter	What happens to the universe filter
Is unchanged	No change
Is updated	No change
Is deleted	Create. If the characteristic referenced in the variable is not in the universe, then create a sub-class for that characteristic too.
Is moved	No change
Is hidden	No change

Related Information

To refresh an OLAP universe [page 432]
When the name or description of an SAP variable is updated [page 451]
When an SAP variable is deleted [page 452]
When an SAP variable is new [page 453]

9.4.11.2 When the name or description of an SAP variable is updated

The following table shows how universe filters are managed in the different possible situations when either the name or description of the SAP source variable is updated:

When the universe filter	What happens to the universe filter
Is unchanged	Update the business name if the option *Keep business name* is not selected. Update the description if the option *Keep description* is not selected. Remain unchanged if these options are selected.
Is updated	Update the business name if the option *Keep business name* is not selected. Update the description if the option *Keep description* is not selected. Remain unchanged if these options are selected.
Is deleted	Create. If a characteristic referenced in the variable is not in the universe, then create a sub-class for that characteristic too.

When the universe filter	What happens to the universe filter
Is moved	Update the business name if the option *Keep business name* is not selected. Update the description if the option *Keep description* is not selected. Remain unchanged if these options are selected.
Is hidden	Update the business name if the option *Keep business name* is not selected. Update the description if the option *Keep description* is not selected. Remain unchanged if these options are selected.

Related Information

To refresh an OLAP universe [page 432]
When an SAP variable is unchanged [page 451]
When an SAP variable is deleted [page 452]
When an SAP variable is new [page 453]

9.4.11.3 When an SAP variable is deleted

The following table shows how universe filter is managed in the different possible situations when the SAP variable is deleted:

When the universe filter	What happens to the universe filter
Is unchanged	Delete if the option *Delete obsolete objects* is selected. Make the sub-class hidden if the option *Hide obsolete objects* is selected. When made obsolete: also change from Compulsory to Optional to avoid automatic application in queries
Is updated	Delete if the option *Delete obsolete objects* is selected. Make the sub-class hidden if the option *Hide obsolete objects* is selected. When made obsolete: also change from Compulsory to Optional to avoid automatic application in queries
Is deleted	No change.
Is moved	Delete if the option *Delete obsolete objects* is selected. Make the sub-class hidden if the option *Hide obsolete objects* is selected. When made obsolete: also change from Compulsory to Optional to avoid automatic application in queries
Is hidden	Delete if the option *Delete obsolete objects* is selected. Make the sub-class hidden if the option *Hide obsolete objects* is selected. When made obsolete: also change from Compulsory to Optional to avoid automatic application in queries

Related Information

To refresh an OLAP universe [page 432]

When an SAP variable is unchanged [page 451]
When the name or description of an SAP variable is updated [page 451]
When an SAP variable is new [page 453]

9.4.11.4 When an SAP variable is new

The following table shows how universe filters are managed in the different possible situations when the SAP variable is new:

When the universe filter	What happens to the universe filter
Is unchanged	Create. If a characteristic referenced in the variable is not in the universe, then create a sub-class for that characteristic too.
Is updated	Create. If a characteristic referenced in the variable is not in the universe, then create a sub-class for that characteristic too.
Is deleted	Create. If a characteristic referenced in the variable is not in the universe, then create a sub-class for that characteristic too.
Is moved	Create. If a characteristic referenced in the variable is not in the universe, then create a sub-class for that characteristic too.
Is hidden	Create. If a characteristic referenced in the variable is not in the universe, then create a sub-class for that characteristic too.

Related Information

To refresh an OLAP universe [page 432]
When an SAP variable is unchanged [page 451]
When the name or description of an SAP variable is updated [page 451]
When an SAP variable is deleted [page 452]

9.4.12 How key figures or measures are managed in OLAP universe updates

SAP data souces use key figures, MSAS and Essbase data sources use measures. The universe measure is the equivalent of the OLAP key figure. How the universe objects are managed with respect to the OLAP objects depends on the type of change. Refer to the topics listed below to see how the universe objects are impacted by specific OLAP object changes.

Related Information

To refresh an OLAP universe [page 432]
When a key figure or measure is unchanged [page 454]
When the name, description, or data type of a key figure or measure is updated [page 454]
When a key figure or measure is deleted [page 455]
When a key figure or measure is moved [page 456]
When a key figure or measure is new [page 457]

9.4.12.1 When a key figure or measure is unchanged

The following table shows what happens to the universe measure in the different possible situations when the SAP key figure or MSAS/Essbase measure is unchanged:

When the universe measure	What happens to the universe measure
Is unchanged	No change
Is updated	No change
Is deleted	No change. Create if the option *Regenerate Objects manually deleted* = *Yes*
Is moved	No change
Is hidden	No change

Related Information

To refresh an OLAP universe [page 432]
When the name, description, or data type of a key figure or measure is updated [page 454]
When a key figure or measure is deleted [page 455]
When a key figure or measure is moved [page 456]
When a key figure or measure is new [page 457]

9.4.12.2 When the name, description, or data type of a key figure or measure is updated

The following table shows what happens to the universe measure in the different possible situations when the SAP key figure or MSAS/Essbase measure is updated:

When the universe measure	What happens to the universe measure
Is unchanged	Update the business name if the option *Keep business name* is not selected. Update the description if the option *Keep description* is not selected. Update the data type if the option *Keep data type of objects* is not selected. Remain unchanged if these options are selected.
Is updated	Update the business name if the option *Keep business name* is not selected. Update the description if the option *Keep description* is not selected. Update the data type if the option *Keep data type of objects* is not selected. Remain unchanged if these options are selected.
Is deleted	No change. Create if the option *Regenerate Objects manually deleted* = *Yes*
Is moved	Update the business name if the option *Keep business name* is not selected. Update the description if the option *Keep description* is not selected. Update the data type if the option *Keep data type of objects* is not selected. Remain unchanged if these options are selected.
Is hidden	Update the business name if the option *Keep business name* is not selected. Update the description if the option *Keep description* is not selected. Update the data type if the option *Keep data type of objects* is not selected. Remain unchanged if these options are selected.

Related Information

To refresh an OLAP universe [page 432]
When a key figure or measure is unchanged [page 454]
When a key figure or measure is deleted [page 455]
When a key figure or measure is moved [page 456]
When a key figure or measure is new [page 457]

9.4.12.3 When a key figure or measure is deleted

The following table shows what happens to the universe measure in the different possible situations when the SAP key figure or MSAS/Essbase measure is deleted:

When the universe measure	What happens to the universe measure
Is unchanged	Delete if the option *Delete obsolete objects* is selected. Make the sub-class obsolete if the option *Hide obsolete objects* is selected.

When the universe measure	What happens to the universe measure
Is updated	Delete if the option *Delete obsolete objects* is selected. Make the sub-class obsolete if the option *Hide obsolete objects* is selected.
Is deleted	No change.
Is moved	Delete if the option *Delete obsolete objects* is selected. Make the sub-class obsolete if the option *Hide obsolete objects* is selected.
Is hidden	Delete if the option *Delete obsolete objects* is selected. Make the sub-class obsolete if the option *Hide obsolete objects* is selected.

Related Information

To refresh an OLAP universe [page 432]
When a key figure or measure is unchanged [page 454]
When the name, description, or data type of a key figure or measure is updated [page 454]
When a key figure or measure is moved [page 456]
When a key figure or measure is new [page 457]

9.4.12.4 When a key figure or measure is moved

The following table shows what happens to the universe measure in the different possible situations when the SAP key figure or MSAS/Essbase measure is moved:

When the universe measure	What happens to the universe measure
Is unchanged	Move the object accordingly.
Is updated	Move the object accordingly.
Is deleted	No change. Create if Option: *Regenerate Objects manually deleted* = *Yes*.
Is moved	No change.
Is hidden	Move the object accordingly.

Related Information

To refresh an OLAP universe [page 432]
When a key figure or measure is unchanged [page 454]
When the name, description, or data type of a key figure or measure is updated [page 454]
When a key figure or measure is deleted [page 455]
When a key figure or measure is new [page 457]

9.4.12.5 When a key figure or measure is new

The universe measure is created when the OLAP key figure or measure is created.

Related Information

To refresh an OLAP universe [page 432]
When a key figure or measure is unchanged [page 454]
When the name, description, or data type of a key figure or measure is updated [page 454]
When a key figure or measure is deleted [page 455]
When a key figure or measure is moved [page 456]

9.4.13 How SAP key dates are managed in OLAP universe updates

This section applies to SAP data sources only. The universe parameter is the equivalent of the OLAP Key Date. How the universe objects are managed with respect to the OLAP objects depends on the type of change. Refer to the topics listed below to see how the universe objects are impacted by specific OLAP object changes.

Related Information

To refresh an OLAP universe [page 432]
When a SAP key date is unchanged [page 457]
When a SAP key date is deleted [page 458]
When a SAP key date is new [page 458]

9.4.13.1 When a SAP key date is unchanged

The universe parameter is the equivalent of the OLAP Key Date. The following table shows what happens to the universe parameter in the different possible situations when the SAP key date is unchanged:

When the universe parameter	What happens to the universe parameter
Is unchanged	No change
Is updated	Not applicable
Is deleted	Not applicable

When the universe parameter	What happens to the universe parameter
Is moved	Not applicable
Is hidden	Not applicable

Related Information

To refresh an OLAP universe [page 432]
When a SAP key date is deleted [page 458]
When a SAP key date is new [page 458]

9.4.13.2 When a SAP key date is deleted

The universe parameter is the equivalent of the OLAP Key Date. The following table shows what happens to the universe parameter in the different possible situations when the SAP key date is deleted:

When the universe parameter	What happens to the universe parameter
Is unchanged	Delete
Is updated	Not applicable
Is deleted	Not applicable
Is moved	Not applicable
Is hidden	Not applicable

Related Information

To refresh an OLAP universe [page 432]
When a SAP key date is unchanged [page 457]
When a SAP key date is new [page 458]

9.4.13.3 When a SAP key date is new

The universe parameter is the equivalent of the OLAP Key Date. The following table shows what happens to the universe parameter in the different possible situations when the SAP key date is new:

When the universe parameter	What happens to the universe parameter
Is unchanged	Create
Is updated	Not applicable
Is deleted	Create
Is moved	Not applicable
Is hidden	Not applicable

Related Information

To refresh an OLAP universe [page 432]
When a SAP key date is unchanged [page 457]
When a SAP key date is deleted [page 458]

9.5 How the different OLAP cubes are mapped to universes

9.5.1 How SAP BW objects are mapped and used in a universe

When you create a universe from either an InfoCube or a BEx Query, the universe design tool maps SAP BW OLAP structures to equivalent classes and objects in the universe.

All InfoObjects in the BEx Query set as rows, columns, free characteristics, and filters are exposed to the universe. This includes characteristics, hierarchies, key figures, structures, and variables.

Hierarchies are mapped, allowing Web Intelligence users to drill down according to BW hierarchies.

For InfoCubes, all the dimensions, key figures, and hierarchies are mapped.

The following table shows the universe objects created for each BW object.

SAP BW object:	Universe objects created:
Dimension Group	Class
Characteristic	Subclass with dimension and detail objects
Characteristic with hierarchy	If data source is a BEx Query: Subclass containing dimension and detail objects for each hierarchy level in the currently defined hierarchy If data source is an InfoCube: Subclasses containing dimension and detail objects for each hierarchy level for all hierarchies defined for the characteristic

SAP BW object:	Universe objects created:
Structure based on Characteristics (BEx Queries only)	Class with single dimension object for the structure
Navigation attribute	Subclass with dimension and detail objects (same as Characteristic)
Display attribute	Detail object for the dimension
Key Figure structure	Class
Key Figure	Measure object in the class for the Key Figure structure with dimension objects for units/currency.
Calculated Key Figure (BEx Queries only)	Measure and dimension objects (same as Key Figure)
Restricted Key Figure (BEx Queries only)	Measure and dimension objects (same as Key Figure)
Variables (BEx Queries only)	Filter mandatory in query In the class for the dimension to which the variable applies, two dimension objects supporting the list of values, one for caption, one for description.
Key date variable (BEx Queries only)	Universe parameters defining key date variable in the universe

Characteristics in the Filters section of the BEx Query are not mapped. However, the filtering applies to the universe. If the filter has a fixed value, the filter is applied transparently when running the Web Intelligence query. If the characteristic has a variable defined, the variable is mapped with these limitations:

- The variable always behaves like a mandatory variable
- Hierarchy and hierarchy node variables are supported except for the hierarchy version variable

To avoid these limitations, move the characteristic from the Filter section to the Free section of the BEx Query.

Related Information

How characteristics are mapped and used [page 460]
How key figures are mapped and used [page 461]
How hierarchies are mapped and used [page 461]
How variables are supported in universes [page 462]
How variables are mapped to a universe [page 464]

9.5.1.1 How characteristics are mapped and used

When no hierarchy is defined on the characteristic in the BEx Query or InfoCube, the universe design tool creates a class containing the characteristic as two dimension objects: Level 00 and Level 01. The Level 00

dimension represents the aggregation of the characteristic when all members are selected (the member returned from SAP NetWeaver technology platform is *All members*). The Level 01 dimension contains all members for the characteristic as a flat list of values.

For each dimension object, the universe design tool creates a detail object for the key, up to three detail objects for the description (short, medium, and long descriptions), and a detail object for each display attribute.

The SELECT clause is defined using the technical name of the characteristic.

Navigation attributes defined in the BW Query are mapped in the parent object class in the same way as characteristics are mapped.

> **i** Note
>
> A large number of navigation attributes defined in the universe negatively impacts the performance of the query in Web Intelligence.

Stuctures defined in the BEx Query that are based on characteristics are included in the universe as single-dimension objects with the elements of the structure as dimension members.

9.5.1.2 How key figures are mapped and used

All key figures in the InfoCube or defined in the BEx Query are included in the universe under a single object class called Key Figures.

Most key figures are defined in BW with either a currency or a unit characteristic. For each key figure, the universe design tool creates:

- A measure object with numeric format corresponding to the key figure without the unit.
- A dimension object with character format that contains the unit or currency. For example, 'USD', '€', 'km'.
- A dimension object with character format that contains the key figure and the unit (formatted value) based on user preferences configured on the SAP server. For example, '200 USD', '345 €', '25 km'.

The Key Figures class includes the calculated key figures and restricted key figures defined in the BEx Query. The original calculation and restrictions are applied to the query, but are not exposed in the universe.

9.5.1.3 How hierarchies are mapped and used

Hierarchies are mapped to allow Web Intelligence users to drill down with SAP BW hierarchies in the same way as custom-made universe hierarchies.

> **i** Note
>
> The *Use Query Drill* option in the Web Intelligence Document Properties dialog box significantly improves drill down performance.

When a hierarchy is defined on a characteristic in the BEx Query, the universe design tool creates one hierarchical structure in the universe, with a subclass for each level in the hierarchy. The structure depends on the current BEx Query definition:

- If a hierarchy is defined in the BEx Query, the universe design tool creates this hierarchy structure in the universe.
- If a hierarchy variable is defined in the BEx Query that allows the user to choose a hierarchy at run time, the universe design tool creates a generic hierarchy in the universe. The structure has the highest number of levels defined for any of the hierarchy structures available for the characteristic.

When building a universe on top of an InfoCube, all hierarchies defined on the characteristic are exposed in the resulting universe. The universe design tool creates subclasses for each hierarchical structure, each containing subclasses for the levels in that hierarchy.

In the universe, Level 00 of a hierarchy represents the top node of the structure. When multiple tops exist for the hierarchical structure, the Level 00 dimension contains all top nodes as a list of values. When the hierarchy attribute is set to not filter unassigned nodes, it is necessary to include Level 00 with the top node for unassigned members. Unassigned members are grouped at the lowest level of the hierarchy.

> **i** Note
>
> Most often SAP BW hierarchies have only one top node. You can delete the Level 00 objects from the default universe to make the universe simpler to use. Generally, you only need to keep Level 00 when there is a need to query/report unassigned members.

If the number of levels in a hierarchy changes in the BEx Query, you must update the universe.

Related Information

About OLAP universe lifecycle management [page 429]

9.5.1.4 How variables are supported in universes

SAP variables can be interpreted as user prompts defined in the BW Query. Variables can be mandatory or optional, and can have default values.

Variables for characteristics are used to filter values for a characteristic. Variables are populated with values when a query is executed. They can store characteristic values, hierarchies, hierarchy nodes, texts, and formula elements.

SAP BW variables apply to BEx Queries only.

> **i** Note
>
> When defining the variable in the Query Designer, on the SAP BW Variables Wizard Other Settings dialog box, the Ready for Input option must be selected.

The following types of SAP BW variables are supported in universes:

Characteristic variables
Hierarchy variables, except for the hierarchy version variable
Hierarchy node variables

Currency variables
Formula variables
Text variables (as replacement path)
Key date variables

The following table shows universe support for user entry BW variables. User entry variables can be mandatory or optional, and can have default values.

Variable Type		Support Level
Characteristic (including key date and currency)	single value prompt	supported
	multiple single value prompt	supported
	interval prompt	supported this is not supported for the key date variable which is a single value variable
	selection option prompt	supported as interval prompt this is not supported as an interval prompt for the key date variable which is a single value variable
	pre-calculated value set	not supported
Text		supported
Formula		price, quota, and numeric values supported
Hierarchy		supported except for version variable
Hierarchy node		supported

The following table shows universe support for other processing types of BW variables.

	Processing Type			
Variable type	Replacement path	Authorization	Customer exit	SAP exit
Characteristic	supported	supported	supported, no prompt is created in the universe	supported
Text	supported	N/A	supported	N/A
Formula	supported	N/A	supported	supported without user entry
Hierarchy	N/A	N/A	supported	supported
Hierarchy node	N/A	N/A	supported	supported without user entry

The Exclude operator is supported, however Web Intelligence does not specify that the selected value be excluded from the query. Other operators, such as Less than and Greater than, can only be used with Selection option entry type. The selection option type is turned into an interval for Web Intelligence prompting.

> **i Note**
>
> To process BW variables in Web Intelligence, you must include at least one measure in the Web Intelligence query.

Related Information

How variables are mapped to a universe [page 464]
How key date variables are supported in a universe [page 466]
How hierarchy and hierarchy node variables are supported in a universe [page 467]

9.5.1.4.1 How variables are mapped to a universe

The user needs to be prompted for all optional and mandatory variables, even when the dimension is not used in the result set, allowing the user to limit the result set. Therefore, a variable defined in the BEx Query is mapped, even if the corresponding characteristic is not in the query.

The user needs to know if a variable is mandatory or optional, and be able to ignore optional variables. Optional variables are defined as optional in the universe, and become optional prompts in Web Intelligence. Mandatory variables become mandatory prompts in Web Intelligence.

For characteristic variables, the universe design tool creates a mandatory filter in the universe. A mandatory filter is a pre-defined query filter object which is hidden to Web Intelligence users, but is applied systematically and transparently to all Web Intelligence queries built on the universe.

Variable Type	Mapped to
Characteristic variable, including currency and formula variable	Universe mandatory filter
Hierarchy variable	Universe mandatory filter
Hierarchy node variable	Class mandatory filter
Key date variable	Universe parameters

For each mandatory filter, two dimension objects are created as reference objects for the @Prompt function to display the expected list of values. The list of values dimensions are hidden in the universe. They are necessary for the correct functioning of the prompt so must not be deleted and must be moved or modified carefully.

Default values for variables are defined in the @Prompt function in the filter using the primary key, persistent/not persistent, and default values parameters. The @Prompt function syntax can be seen in the Properties page of the filter in the universe.

To avoid conflict between BW variables and filters defined by Web Intelligence users, objects involved in an SAP variable definition are generated with the option *Can be used in Condition* unchecked in the *Advanced* page of the object properties. This restricts Web Intelligence users from including dimensions involved in SAP variables in the Filter pane.

Example: WHERE clause generated for an SAP BW variable

This example shows the WHERE clause generated for a BW variable on dimension object Customer2. The syntax for the generated WHERE clause for a variable can be seen on the Properites page of the filter.

```
<FILTER KEY="[Z_VAR002]">
    <CONDITION OPERATORCONDITION="Equal">
        <CONSTANT TECH_NAME="@Prompt(
            'Customer Variable Single Value Mandatory',
            'A',
            'Customer2\LovCustomer Variable Single Value MandatoryBase',
            mono,
            primary_key)"/>
    <CONDITION>
</FILTER>
```

The prompt text is generated from the BW variable name. You can edit the text to make it more descriptive.

`Customer2\LovCustomer Variable Single Value MandatoryBase` is the name of the hidden universe object that is used to build the list of values.

> **i** Note
>
> If you rename the class or move the list of values object to another folder, you must update the syntax in the filter key.

9.5.1.4.2 How variables and lists of values are supported

A BEx Query can contain more than ten variables, which means that ten or more lists of values can be loaded. Loading and refreshing lists of values can have an important impact on performance. The following options are available for improving query performance for queries with variables:

- At universe generation time, all SAP BW variables (except key date) are mapped to mandatory filters. By default, the filter objects are not associated with a list of values (except for hierarchy node variables). You must explicitly associate a list of values in the object properties page.
- Optional variables are generated as optional prompts. An optional prompt does not automatically load the list of values at query run time.
- The delegate search option on the list of values properties presents the user with an empty list of values at query run time. The user enters search criteria to limit the number of values returned in the list of values. To activate the delegated search option for a list of values, edit the list of values properties on the object properties page of the object to which the list of values applies.

  > **i** Note
  >
  > The delegated search is not supported for cascading lists of values.

Related Information

Optional prompts in OLAP universes [page 428]

9.5.1.4.3 How key date variables are supported in a universe

A key date variable in a BEx Query allows you to specify a date for time-dependent data. Key dates can influence the data retrieved for a dimension, for example, a product description can change over time. A key date can influence a hierarchy structure, for example, a specific cost center can be on Level 01 in one year, and on Level 02 in a different year.

The key date variable is a special SAP BW variable because the date value entered by the user is not contained in any dimension of the BW Query. The key date is a property of the query.

In a BEx Query, the key date variable can be defined for two uses:

- To specify the valid date for a specific hierarchy, impacting only that hierarchy.
- To specify a date for the complete query. In this case, the key date that is set in a query influences the following:
  - time-dependent master data
  - currency exchange rates
  - the list of hierarchies
  - time-dependent hierarchy structures

> **i Note**
>
> In the universe, the use of a key date is limited to the whole universe. Therefore, the key date generated in a universe impacts all other SAP variables and data.

SAP BW supports only one key date variable per BW Query, therefore a universe contains only one key date variable.

Key date variables can be mandatory or optional, and can have a default value. If no default value is defined and the user does not enter a value, the query uses the current system date.

The key date variable properties of the query are mapped to five universe parameters, described in the following table.

Parameter	Description
KEYDATE_ENABLED	Set to *Yes* if a key date is enabled on the universe.
KEYDATE_NAME	Technical name of the key date variable.
KEYDATE_CAPTION	Caption for the key date variable presented when prompting the user for a value.
KEYDATE_DEFAULT_VALUE	Default value for the key date, if it exists.
KEYDATE_MANDATORY	Set to *Yes* if a user must enter a value or use the default.

At query run time, Web Intelligence proposes the same key date for all queries. The user can modify the key date. A *Keydate Properties* dialog box is available to manage which key date is used. The user is prompted for the key date before any other type of variable prompt.

9.5.1.4.4 How hierarchy and hierarchy node variables are supported in a universe

A hierarchy variable is used to prompt the user for the hierarchy to be used in the query. Web Intelligence users can create queries and reports to retrieve and display members from any hierarchy.

If the hierarchy variable is optional and the user leaves the prompt empty, no hierarchy is used in the report.

A report contains the largest number of hierarchy levels independent of the hierarchy that is selected. Hierarchy levels that are not returned in the result set are empty in the report.

A hierarchy node variable is used to prompt the user for the node to be defined as top node for the hierarchy in the query.

When a query contains both a hierarchy and hierarchy node variable, the Web Intelligence user must first select a hierarchy in the list of available hierarchies. Next, the user selects the hierarchy node. The list of hierarchy nodes available shows hierarchy nodes for all hierarchies. The list is not filtered based on the hierarchy selected. The user is responsible for selecting a node from the correct hierarchy. Selecting a hierarchy node from a different hierarchy can cause the report to be empty.

Related Information

How hierarchies are mapped and used [page 461]

9.5.2 How Essbase cubes are mapped to universe components

The universe design tool creates a universe from an Essbase cube by mapping Essbase outlines to equivalent classes and objects. You identify the cube data source when you create the connection.

Essbase alias tables define a set of alternate names for dimensions, levels, and members in the outline. The universe design tool generates the universe using the names according to the alias table you select when you create the connection to the Essbase data source.

In an Essbase outline, measures are defined as dimensions. You select the dimension to use as the measures dimension when you create the connection to the Essbase data source. The universe design tool generates the members of that dimension as measures in the universe.

Any dimension supports hierarchies with multiple levels. A maximum of one hierarchy can be defined for each dimension.

The following table shows which objects are created in the universe for each Essbase outline element.

Essbase Object	Universe object created:
Dimension	A class containing the generations for the dimension.

Essbase Object	Universe object created:
Generation	An object in the dimension class with two detail objects: one for caption, one for name.
Measures dimension	A class named according to the dimension selected as the measures dimension in the universe connection (usually Measures class or Accounts class).
Measure	A measure object in the measure class or subclass. The measures are created with a structure of class and subclass that matches the structure in the Essbase outline.

Measures are generated with the aggregation projection function set to `Database delegated` by default. When refreshing the Web Intelligence report, the aggregation of the measure is delegated to the database server.

9.5.3 How MSAS cubes are mapped to universe components

The universe design tool creates a universe from MSAS cubes by mapping MSAS structures to equivalent classes and objects. You specify the cube data source when you create the connection.

The following table shows which objects are created in the universe structures for each MSAS object. This mapping applies to MSAS virtual cubes and local cubes (.cub files) as well as MSAS standard cubes.

MSAS Object:	Universe object created:
Dimension	A class containing objects for that dimension.
Display Folder (MSAS 2005)	A subclass in the dimension class.
Hierarchy	A subclass in the corresponding dimension class or a sub-subclass in the corresponding display folder class.
Attribute (MSAS 2005)	A subclass in the corresponding dimension class or a sub-subclass in the corresponding display folder class.
Measures	A Measure class containing all measure objects. Measure objects are created in the Measure class, or the subclass for the Measure Group.
Measure Group (MSAS 2005)	A subclass in the Measure class.
Level	An object in the dimension class or subclass, and a Level All object which represents the aggregate of all sub-levels.
Level Property	A detail in the level object to which it applies.

Measures are generated with the aggregation projection function set to `Database delegated` by default. When refreshing the Web Intelligence report, the aggregation of the measure is delegated to the database server.

10 Deploying universes

10.1 Overview

This chapter is about universe deployment and management. It describes the following:

- How are universes deployed? [page 469]
- Setting access restrictions on a universe [page 470]
- Managing users and logins [page 479]

10.2 How are universes deployed?

Deploying a universe is the action of making the universe available to Web Intelligence users or other designers. You deploy a universe by exporting it to the Central Management Server (CMS) repository.

You test a universe by exporting to a test repository and running tests in Web Intelligence before exporting to the production repository.

You should only deploy a universe to Web Intelligence users when it has completed the design, build, and test phases.

See the following sections for information on deploying universes by importing to and exporting universes from the respository:

- Importing a universe [page 39]
- Exporting a universe [page 41]

10.2.1 Identifying a universe in the repository

A universe is identified by the following parameters:

Identifier	Description
File name	Maximum 100 characters and a .unv extension.
Long name	Consists of up to 35 characters. This is the name by which end users identify the universe in Web Intelligence, so it should be a name that describes the purpose of the universe.

Identifier	Description
Unique system identifier	Identifier assigned by the CMS.

10.2.1.1 Universe identifier rules

The following rules apply to the universe identifiers for universes stored in universe folders in the repository:

- A universe identifier is unique in the CMS.
- The combination of file name and folder location (path). A universe is unique in the parent folder.

10.3 Giving all users access to a universe

You can make a universe accessible to all universe design tool users in both workgroup and enterprise mode, by saving a universe in workgroup mode. The connection for the universe cannot be a secured connection. If you want to make a universe available to all users, you must save the universe with an unsecured connection.

To make a universe accessible to all universe design tool users:

1. Verify that the universe that you want to make available to all users does not have a secured connection.
2. Secured connections are required to export universe to the CMS. If a universe has a secured connection, select or create a new shared connection. See the section Modifying universe identification parameters [page 73] for more information.
3. Select *File* > *Save As*.
   A *File Save* box appears.
4. Select the *Save For All Users* check box.
5. Click *OK*.

10.4 Setting access restrictions on a universe

You can apply restrictions to defined user and groups who use a universe.

Universe security is managed at two levels:

Security level	Description
CMS	From the Central Management Console you can set restrictions that apply to universes stored in a CMS. You can set what universes users can access, and depending on the rights defined for a user group, you can restrict viewing, editing, deleting, and other actions in a universe. **This guide does not cover setting restrictions at the CMS level, see the BusinessObjects Enterprise Administrator's Guide for information on using the Central Management System.**
Universe	You can define restrictions for users allowed to use a universe. A restriction can include object access, row access, query and SQL generation controls, and connection controls. This guide describes the types of restrictions you can define on a universe.

10.4.1 What is a restriction?

A restriction is a named group of restrictions that apply to a universe. You can apply a restriction to a selected group or user account for a universe. When users connect to a universe, the objects, rows, query types, and connection that they use in the universe are determined by their applied restriction.

You assign a restriction to a BusinessObjects user or group. This limits the access to universe objects or resources based on the profile of the user group.

10.4.2 What restrictions can be applied in a universe?

Access restrictions that apply to a user group are defined in a restriction. You can define multiple restrictions for a universe. Restrictions can be edited, or deleted at any time.

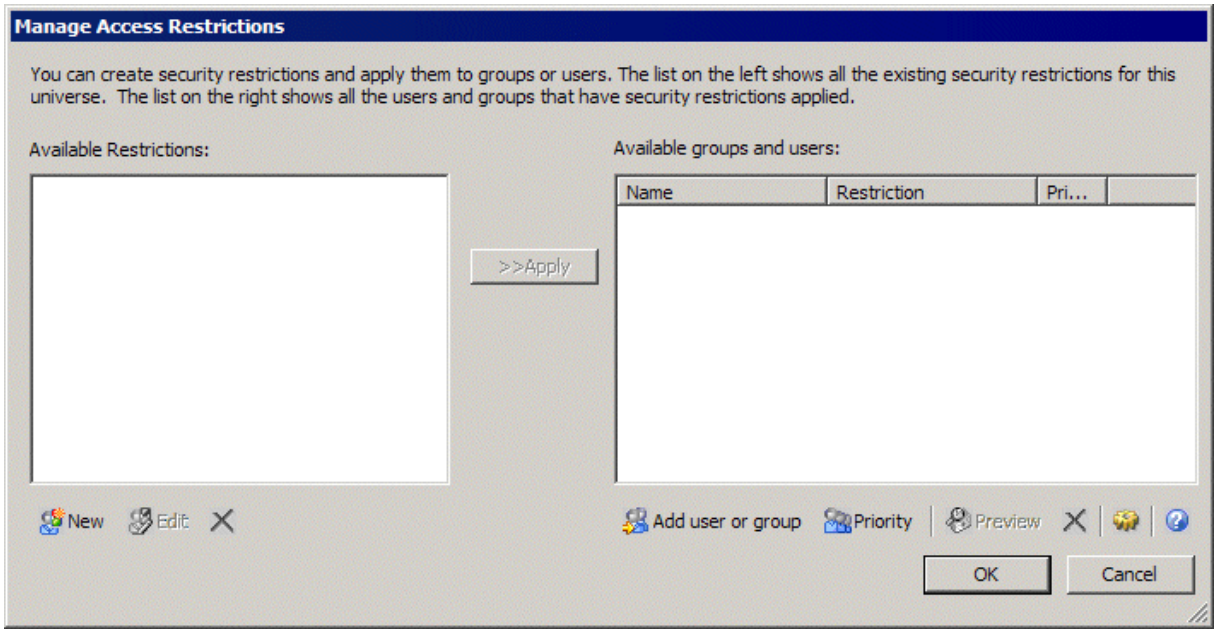
A restriction can define the following types of restrictions to apply to a selected user or group for a universe:

Type of restriction	Description
Connection	Universe connection to the data source. You can select an alternative connection for the universe. Creating and editing connections is described in the section Modifying universe identification parameters [page 73].
Query controls	Options to limit the size of the result set and query execution time. See the section Indicating SQL restrictions [page 88] for more information.

Type of restriction	Description
SQL generation options	Options to control the SQL generated for queries. See the section Setting SQL generation parameters [page 90] for more information.
Object access	You can select objects that are not available to the universe.
Row access	You can define a WHERE clause that restricts access to row and limits the result set returned by a query.
Alternative table access	You can replace a table referenced in the universe by another table in the database.

10.4.3 How are access restrictions managed?

Access restrictions are managed from the Manage Access Restrictions dialog box. You access this box by selecting Tools > Manage Security > Manage Access Restrictions. The dialog box appears below.



The restrictions currently available to the universe are listed in the Available Restriction pane.

The users and user groups that are defined for each restriction appear in the Available Groups and Users pane.

The options available to manage access restrictions are described below.

Restriction option	Description
New	Define a new restriction.
Edit	Modify an existing restriction.
Delete selected restriction 	Remove a restriction from the list.
Add user or group	Add a user or group from the list of BusinessObjects users and groups defined for the BusinessObjects system.
Priority	Allows you to set a priority level for one or more user groups.
Preview	Allows you to view all users and groups defined for the BusinessObjects system.
Remove security option from selected users or groups 	Removes all restrictions that are set on the selected user or groups.
Restriction options 	Allows you to choose whether row restrictions are implemented with AND or OR operators.

10.4.4 Creating a restriction

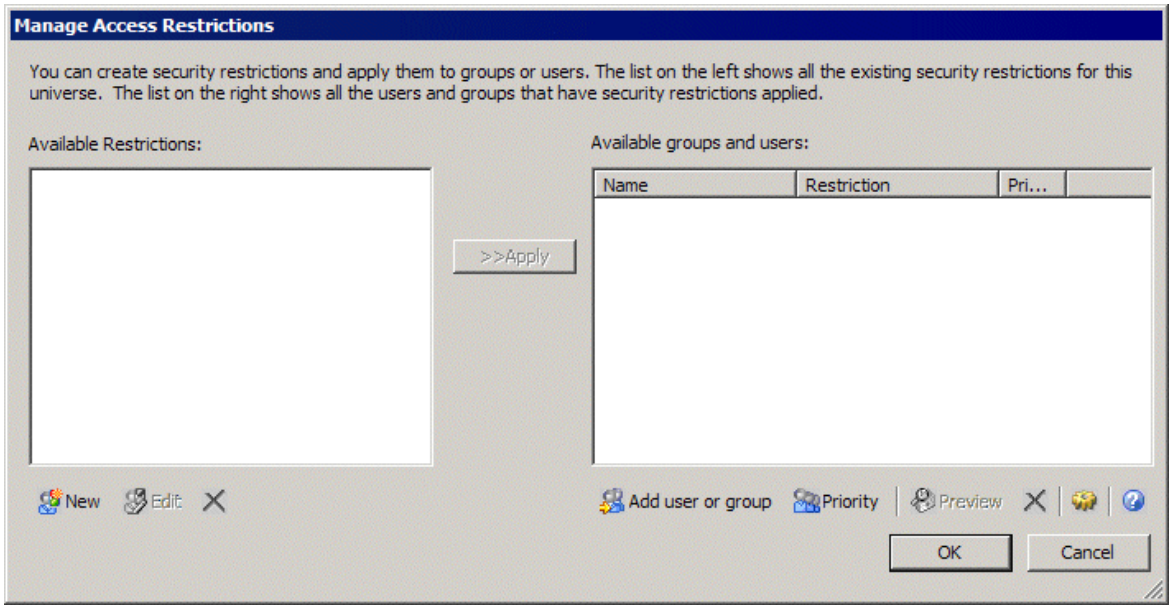
You can create, edit, and delete a restriction at any time once the universe has been exported to the CMS.

You can create multiple restrictions depending on the query needs of a target user group.

10.4.4.1 To create a restriction

1. Select Tools > Manage Security > Manage Access Restrictions.

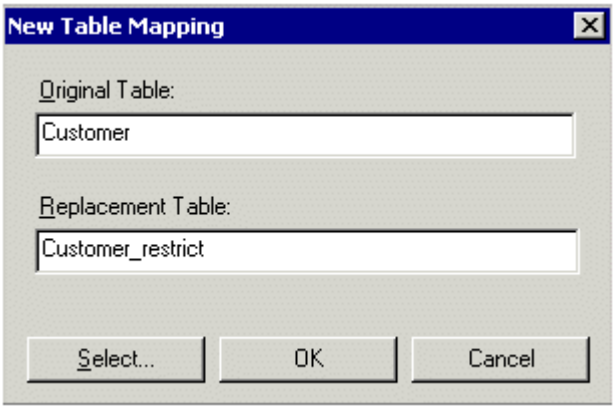
   The Manage Access Restrictions box appears.



2. Click *New*.

   The *Edit Restriction* dialog box appears. You can choose a replacement connection for the universe's connection, or edit the connection properties by using the buttons next to the *Connection* dialog.
3. Do one or more of the following:

To set...	Do this...
a new connection	Select a connection name from the Connection list box.
query controls	○ Click the Controls tab. ○ Select a query option and enter a value for each option.
SQL generation options	○ Click the SQL tab. ○ Select check boxes as appropriate for Query, Multiple paths, or Cartesian product options.
object access restrictions	○ Click the Objects tab. ○ Click Add. The Restricted Objects box appears. ○ Click Select. The Object browser appears. ○ Select an object to restrict. ○ Click OK to close the Objects Browser and Restricted Object boxes.

To set...	Do this...
row access restrictions	○ Click the Rows tab. ○ Click Add. ○ Click the Browse button next to the Table box. ○ Click a table name and click OK. ○ Click the Browse button next to the Where Clause box. ○ Type a WHERE clause in the definition box. Or Build a WHERE clause by selecting a column, an operator, and function from the SQL editor. See the section Using the Join SQL Editor [page 145] for information on using this editor.
a reference to an alternative table	○ Click the Table Mapping tab. ○ Click Add. The New Table Mapping box appears. ○ Put the cursor in the Original Table box and click Select. A table browser appears. ○ Select a table and click OK. ○ Put the cursor in the Replacement Table box and click Select. ○ Select a table in the table browser and click OK. 

4. Click OK.

   The new restriction appears in the list.
5. Click OK.

10.4.5 Applying universe access restrictions

You set access restrictions on a universe by applying a restriction to one or more users or user groups.

10.4.5.1 To apply a restriction to a universe user

You can choose a replacement connection for the universe's connection.

1. Select Tools > Manage Security > Manage Access Restrictions.

   The Manage Access Restrictions dialog box appears.
2. Click a restriction in the Available Restrictions pane.
3. Click a user or group in the Available Users and Groups pane.

   Or

   If you want to select multiple users or groups; click multiple users or groups while holding down the CTRL key.
4. Click Apply.
5. Click OK.

10.4.6 Adding a user group to the list of available users for a universe

You apply a restriction to user groups defined for your BusinessObjects system. These users are defined in the Manage Groups and User Accounts section of the BusinessObjects Administration Console. See the BusinessObjects Enterprise XI 3.0 Administrator's Guide for information on setting up users and groups for the BusinessObjects system.

If you need to apply a restriction to a user group that is not in the Available Groups and Users pane, you can add the user group to the list as follows:

10.4.6.1 To add a user group to the "Available Groups and Users" pane

1. From the Manage Access Restrictions box (Tools > Manage Security > Manage Access Restrictions), click the Add user or Group icon. 

   The Select Users and Groups dialog box appears. It lists all the user groups defined in the BusinessObjects Administration Console that have access to the BusinessObjects system. If the list of users is too large to easily find a target user or group, you can search the list as follows:
   - Select Name or Description from the drop down list.
   - Type in a text string in the For Text box to search the list for the text string in either Name or Description fields selected above.
   - Click the Search icon to start searching.

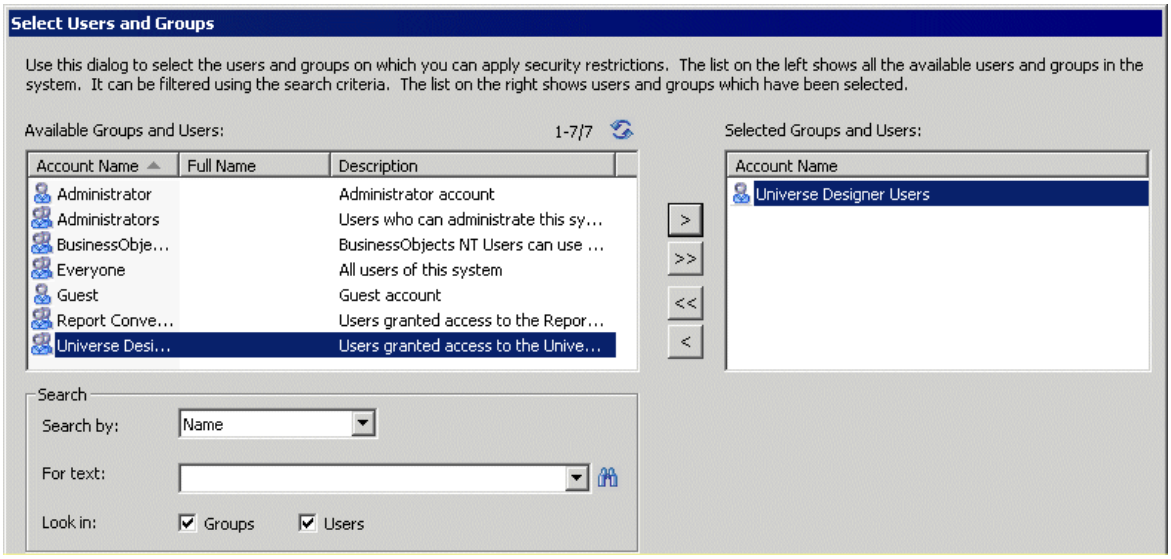
   To filter the list, you can also select the Groups or Users check box to display only groups or users in the list.

2. Click a user or group.

   Or

   Click multiple user or groups while holding down the CTRL key.
3. Click the right head arrow.

   The user or group appears in the Selected Groups and Users list pane on the right of the dialog box.



4. Click OK.

   The user or group now appears in the Available Groups and Users list in the Manage Access Restrictions dialog box.

10.4.7 Setting restriction group priority

You can specify which restriction to apply to a user that belongs to multiple groups using a universe. For example a user belongs to two groups, Sales with a restriction to view 5000 rows of data, and Marketing to view 10000 rows. When the user refreshes a report, the restriction associated with the lowest level group is applied. In the example above, if the Sales group had order 1 and Marketing had order 2, the restriction from marketing (10000) would be used.

You can arrange user groups in order. The restriction for the lowest group in the listed order is used.

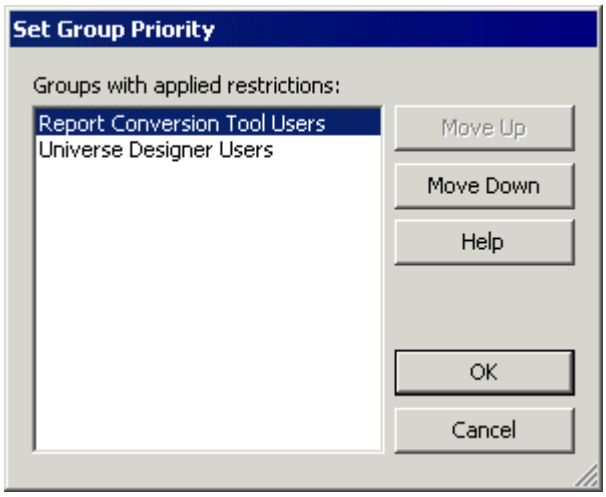
> **i** Note
>
> This only applies to exclusive restrictions such as connection, table mapping, or SQL controls. If object restrictions are set on both groups, they will ALL be applied.

10.4.7.1 To set user group priority for multiple restriction use

1. Select Tools > Manage Security > Manage Access Restrictions.

   The Manage Access Restrictions dialog box appears.
2. Click a user or group in the Available Groups and Users pane.
3. Click the Priority icon.

   The Set Group Priority box appears.



4. Select a user or group, and click the Move Up or Move Down buttons to change the priority level.
5. Click OK.

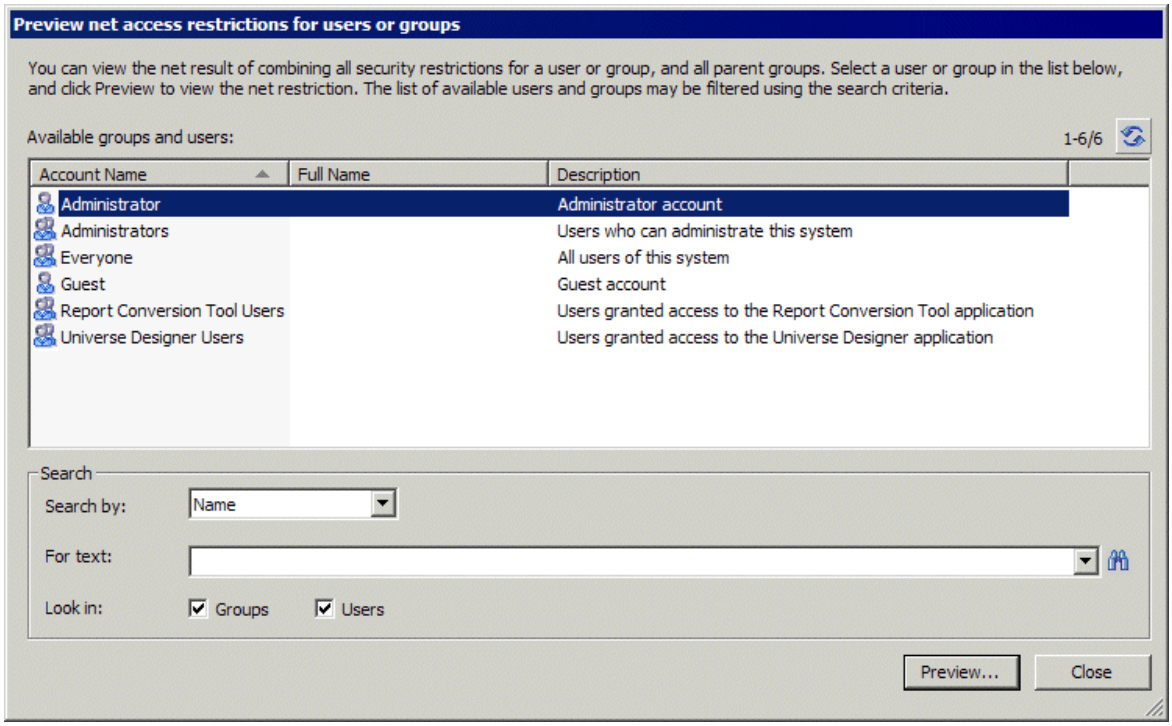
10.4.8 Viewing users and groups security restrictions

You can view the restrictions applied to all user and groups.

10.4.8.1 To view restrictions for all universe users and groups

1. Select Tools > Preview Security Restrictions.

   The preview users and groups dialog box appears.



2. Click a user account name in the list.
3. Click Preview.

   The security restriction that applies to that user account appears. Parameters and options that appear in red are those that have been modified and apply specifically to the restriction.
4. Click OK to close the box.

10.5 Managing users and logins

You can log into the universe design tool as a different user and also change your login. User accounts must be valid accounts for the target repository.

You can also log in to the universe design tool in stand alone mode. You can use the universe design tool, create universes, personal and shared connections, but you can not access connections and universes in the CMS.

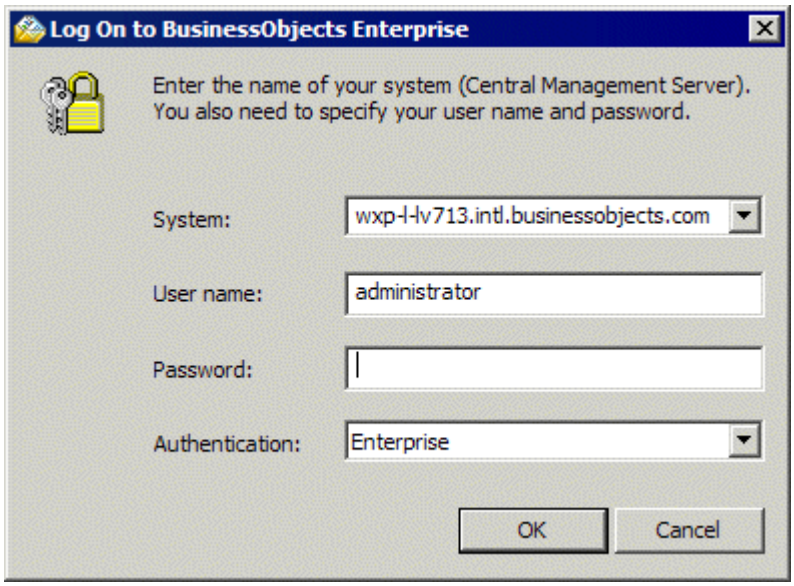
10.5.1 Managing logins

You can log into the universe design tool as a different user without quitting your work session. User accounts must be valid for the target repository. You can log in as another user only if you know the corresponding user name and password.

10.5.1.1 To log in as a different user

1. Select Tools > Login As.

   If there are open universes, the universe design tool closes them automatically. The User Identification dialog box appears.



2. Type a valid user name in the User Name box.
3. Type a new user name and password.
4. Click OK.

   When you log in as another user, you are automatically entitled to all the rights of that user; however, you may also be prohibited from certain operations as a result of restrictions set on the user profile.

10.5.2 Managing passwords

During a session, you can change the password with which you logged provided that your user account has the rights to do so. You cannot, however, change your user name.

10.5.2.1 To change your password

1. Select Tools > Change Password.

   The Change Password dialog box appears.
2. Type your existing password in the Enter Old Password box.
3. Type your new password in the Enter New Password box.

4. Confirm your new password by typing it again in the Confirm New Password box.
5. Click OK.

   The password is changed.

11 Using the sample materials

11.1 Overview

This appendix provides detailed information on the structure of the Club database built with Microsoft Access. This is the database from which most of the examples and illustrations in this guide are derived.

You can find the database file, Club.mdb, in the \Samples\<language>\Databases subfolder in the Business Objects path. Also in this folder, you will find the efashion demo database.

11.2 The Club database

The Club database is used in most of the examples given in this guide.

11.2.1 The structure of the tables

The Club database is used by the sales manager of Island Resorts, a fictitious business specializing in packaged holidays. Based on the information in this database, the sales manager can perform sales and marketing analysis. The database is made up of the following tables:

- Age_group
- City
- Country
- Customer
- Invoice_Line
- Region
- Region_Sline
- Reservation_Line
- Reservations
- Resort
- Sales
- Sales_Person
- Service
- Service_Line

The next sections describe each of the above tables and their columns.

11.2.1.1 The Age_group table

The Age_group table stores information on the age ranges of customers.

Column Name	Description
age_min	the lower limit of the age range
age_max	the upper limit of the age range
age_range	the age range of customers

11.2.1.2 The City table

The City table stores information on the city in which the customers reside.

Column Name	Description
city_id	system-generated city number
city	the city in which the customer resides (Albertville, Amsterdam, Augsburg...Versailles, Washington D.C., Yokohama)
region_id	system-generated region number

11.2.1.3 The Country Table

The Country table relates to the country in which the customer resides.

Column Name	Description
country_id	system-generated country number
country	The name of the country in which the customer resides (Australia, France, Germany, Holland, Japan, UK, US.)

11.2.1.4 The Customer table

The Customer table contains information relating to customer identification such as name and address.

Column Name	Description
cust_id	system-generated customer number
first_name	first name of the customer
last_name	last name of the customer
age	age of the customer
phone_number	phone number of the customer
address	first line of the customer's address
city_id	system-generated city number
sales_id	system-generated sales person number (the person who sold the packaged holiday).
sponsor_id	system-generated sponsor number (optional)

11.2.1.5 The Invoice_Line table

This table includes invoice information; it is used to bill the customer.

Column Name	Description
inv_id	system-generated invoice number
service_id	system-generated service number
days	Number (3-15) representing the length of the stay at the resort in days. For billing purposes, a stay can be up to 15 days. Beyond 15 days, the system considers the remaining days to be a new stay.
nb_guests	number of guests for which the invoice is drawn up

11.2.1.6 The Region table

The Region table stores information on the geographical region in which the customer resides.

Column Name	Description
region_id	system-generated region number
region	geographical region in which the customer resides (Bavaria, East Coast, East Germany...Wales, West, West Japan)
country_id	system-generated country number

11.2.1.7 The Region_Sline table

This table enables calculation of a sales revenue aggregate in the universe. Aggregate awareness is covered in Chapter 5 of this guide.

Column Name	Description
sl_id	system-generated service line number (service line information is given in the Service_Line table)
region_id	system-generated region number
sales_revenue	the total sales revenue by region.

11.2.1.8 The Reservation_Line table

Information relating to customer reservations is stored in the Reservation_Line table.

Column Name	Description
res_id	system-generated reservation number
service_id	system-generated service number
res_days	days of the week reserved (1 - 7)
future_guests	number of future guests (1 - 5)

11.2.1.9 The Reservation table

The Reservation table contains information on the date of the customer reservation.

Column Name	Description
res_id	system-generated reservation number
cust_id	system-generated customer number
res_date	the date on which the customer reserved

11.2.1.10 The Resort table

The Resort table contains information on each resort.

Column Name	Description
resort_id	system-generated resort number
resort	the name of the resort: Australian Reef, Bahamas Beach, French Riviera, Hawaiian Club, Royal Caribbean
country_id	system-generated country number

11.2.1.11 The Sales table

The Sales table contains sales information.

Column Name	Description
inv_id	system-generated invoice number
cust_id	system-generated customer number
invoice_date	date of the invoice

11.2.1.12 The Sales_Person table

The Sales_Person table stores information on the sales persons of the Island Resorts business.

Column Name	Description
sales_id	system-generated sales person number

Column Name	Description
sales_person	name of the sales person (Andersen, Barrot, Bauman... Moore, Nagata, Schmidt)

11.2.1.13 The Service table

The Service table includes information on the price and types of services available in a given resort.

Column Name	Description
service_id	system-generated service number
service	services available in a resort (see the query results below)
sl_id	system-generated service line number (service line information is given in the next table)
price	the price of the service

11.2.1.14 The Service_Line table

The Service_Line table stores information on the service line of resorts. Service line means simply the category in which the service falls.

Column Name	Description
sl_id	system-generated service line number
service_line	Service line includes: accommodation, food and drinks, recreation
resort_id	system-generated resort number (values 1 to 5)

Important Disclaimers and Legal Information

Hyperlinks

Some links are classified by an icon and/or a mouseover text. These links provide additional information.
About the icons:

- Links with the icon : You are entering a Web site that is not hosted by SAP. By using such links, you agree (unless expressly stated otherwise in your agreements with SAP) to this:
  - The content of the linked-to site is not SAP documentation. You may not infer any product claims against SAP based on this information.
  - SAP does not agree or disagree with the content on the linked-to site, nor does SAP warrant the availability and correctness. SAP shall not be liable for any damages caused by the use of such content unless damages have been caused by SAP's gross negligence or willful misconduct.
- Links with the icon : You are leaving the documentation for that particular SAP product or service and are entering a SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

Videos Hosted on External Platforms

Some videos may point to third-party video hosting platforms. SAP cannot guarantee the future availability of videos stored on these platforms. Furthermore, any advertisements or other content hosted on these platforms (for example, suggested videos or by navigating to other videos hosted on the same site), are not within the control or responsibility of SAP.

Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.
The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

Example Code

Any software coding and/or code snippets are examples. They are not for productive use. The example code is only intended to better explain and visualize the syntax and phrasing rules. SAP does not warrant the correctness and completeness of the example code. SAP shall not be liable for errors or damages caused by the use of example code unless damages have been caused by SAP's gross negligence or willful misconduct.

Gender-Related Language

We try not to use gender-specific word forms and formulations. As appropriate for context and readability, SAP may use masculine word forms to refer to all genders.

www.sap.com/contactsap

© 2020 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company. The information contained herein may be changed without prior notice.

Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies.

Please see https://www.sap.com/about/legal/trademark.html for additional trademark information and notices.

THE BEST RUN SAP