BOID10

SAP BusinessObjects Information Design Tool

PARTICIPANT HANDBOOK
INSTRUCTOR-LED TRAINING

Course Version: 15
Course Duration: 5 Day(s)
Material Number: 50124038

SAP Copyrights and Trademarks

All other product and service names mentioned are the trademarks of their respective companies. Data contained in this document serves informational purposes only. National product specifications may vary.

These materials are subject to change without notice. These materials are provided by SAP SE and its affiliated companies ("SAP Group") for informational purposes only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group 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.

Typographic Conventions

American English is the standard used in this handbook.

The following typographic conventions are also used.

This information is displayed in the instructor's presentation	
Demonstration	
Procedure	
Warning or Caution	
Hint	
Related or Additional Information	
Facilitated Discussion	
User interface control	*Example text*
Window title	*Example text*

Contents

Course Overview

TARGET AUDIENCE
This course is intended for the following audiences:

- Application Consultant
- Data Consultant
- Developer
- Project Manager
- System Administrator

UNIT 1 Basic SAP BusinessObjects Universe Design

UNIT OBJECTIVES

- Discuss universes
- Identify local projects

Unit 1
Lesson 1

Describing Universes

LESSON OVERVIEW
This lesson introduces the purpose of the Information Design Tool - the SAP BusinessObjects metadata design environment used to create universes. In addition, this lesson provides an overview of the universe design process.

LESSON OBJECTIVES
After completing this lesson, you will be able to:

- Discuss universes

Universes

A universe is an organized collection of metadata objects that enable business users to analyze and report on corporate data in a nontechnical language. These objects include dimensions, measures, attributes, and navigation paths. The metadata object layer, called the business layer, is built on a relational database schema or an OLAP cube, so the objects map directly to the database structures via SQL or MDX expressions. A universe includes connections identifying the data sources so queries can be run on the data.

The role of the universe is to provide the business user with semantically understandable business objects. The user is free to analyze data and create reports using relevant business language regardless of the underlying data sources and structures.

Universes created using the information design tool can be used by the following SAP BusinessObjects data analysis and reporting applications starting with version BI 4:

Universes and SAP BusinessObjects Applications

Universes created using the information design tool can be used by the following SAP BusinessObjects data analysis and reporting applications starting with version BI 4:

- SAP BusinessObjects Web Intelligence
- SAP Crystal Reports for Enterprise
- SAP BusinessObjects Explorer
- SAP BusinessObjects Dashboard Design
- SAP BusinessObjects Design Studio
- SAP Lumira

To enable the designer to create universes, the information design tool provides the resources to:

- Create connections to data sources.

- Extract a complete OLAP cube schema.
- Extract tables and joins to build a relational schema called a data foundation.
- Create metadata objects from the cube or the data foundation. These objects are contained and organized in a business layer. The SQL and MDX expressions within objects can be validated and queries run against the target databases to test the business layer.
- Share resources to allow multiple designers to work on the same resources concurrently.
- Publish a universe, which compiles the business layer, the data foundation, and the connections into single universe file (.unx) either to a repository, to be implemented in deployments of SAP BusinessObjects data analysis and reporting applications, or locally, to be implemented by client applications in standalone mode (for example, Web Intelligence Rich Client).
- Create security profiles to define user access to universe data and metadata.

Table 1: Universe Terminology

Resource	Description
Project	A project is a named local workspace that contains the resources used to build one or more universes. A project can be shared so that multiple designers can work on the same resources. A project can contain any number of independent resources, for example data foundations, business layers, and connections. All resources contained within a project can be used interchangeably; for example, a connection can be used by several data foundations within the same project.
Connection	A connection is a named set of parameters that define how a universe can access a relational or OLAP data source. A universe is always associated with at least one connection. A connection is an independent resource and can be used by several universes. You can build a multisource-enabled universe that references one or more relational connections. Connections can be local (stored in a local file) or secured (an object in a shared repository that is referenced by a connection shortcut).
Connection shortcut	A connection shortcut is an object in the local project that references a secured connection in a repository. You use a connection shortcut to refer to secure connections when creating data foundations and business layers based on secure connections.
Data foundation	A data foundation is a schema that defines the relevant tables and joins from one or more relational databases. You enhance the data foundation by adding derived tables, alias tables, calculated columns, additional joins, contexts, prompts, lists of values, and other SQL definitions. The data foundation becomes the basis of one or more business layers.

Business layer	A business layer is a collection of metadata objects that provides an abstraction of relational database entities or OLAP cubes, understandable by a business user. Objects map via SQL expressions to an underlying data foundation, or via MDX expressions to an underlying OLAP cube. These objects include dimensions, navigation paths, measures, attributes, and predefined conditions. You can add or change dimensions, navigation paths, measures, attributes, and other objects as the business layer requires. You can validate the SQL or the MDX at any time. You can also create lists of values and parameters (also called prompts). When the business layer is complete, it is compiled with the connections or connection shortcuts and data foundation, published, and deployed as a universe.
Query	A query is a set of objects that define a request to the database for data. A query can be defined and saved in the business layer as a metadata object to be used to test objects in the business layer during universe development. These queries cannot be reused by any of the other analysis and reporting applications.
Parameters and lists of values	A parameter is a variable in the universe that requires a value at query time. Parameters are defined to prompt the user to supply a value, and in this case are referred to as prompts. A list of values is a collection of data values that can be associated with an object in the universe, allowing the user to choose values for a prompt. Parameters and lists of values can be defined in the data foundation. They are inherited by all business layers based on that data foundation. Parameters and lists of values can also be defined in the business layer.
Universe	A universe is a compiled file that includes all resources used in the definition of the metadata objects built in the design of the business layer. The universe is used by SAP BusinessObjects data analysis and reporting applications, where the business layer objects are visible for analysis and reporting.
Security Profiles	A security profile is a group of security settings that controls the data and metadata that is displayed to users and modifies the parameters defined in the data foundation and business layer. Security profiles are defined on published universes and stored in the repository.

The information design tool provides design resources that you use to first extract metadata from relational or OLAP data sources, and then to build a business layer of objects targeted to a specific user group.

You can create a universe from two types of data sources:

- One or more relational databases
- An OLAP cube

The process to create a universe is almost the same for both relational and OLAP data sources. The exception is for a universe based on a relational source. In this case, create a data foundation before you can create the business layer. This step is not necessary when you use a connection to an OLAP cube for the universe; the objects are presented from the cube structure, and you can select the objects directly for the universe.

Universe Design Process

Universe development is a cyclical process that includes planning, designing, building, distribution, and maintenance phases. The usability of any universe is directly related to how successfully the other phases in the development cycle interact with each other.

The universe development cycle presents an overview of a universe designing methodology that you can use to plan and implement a universe development project.

The proceeding diagram outlines the major phases in a typical universe development cycle:

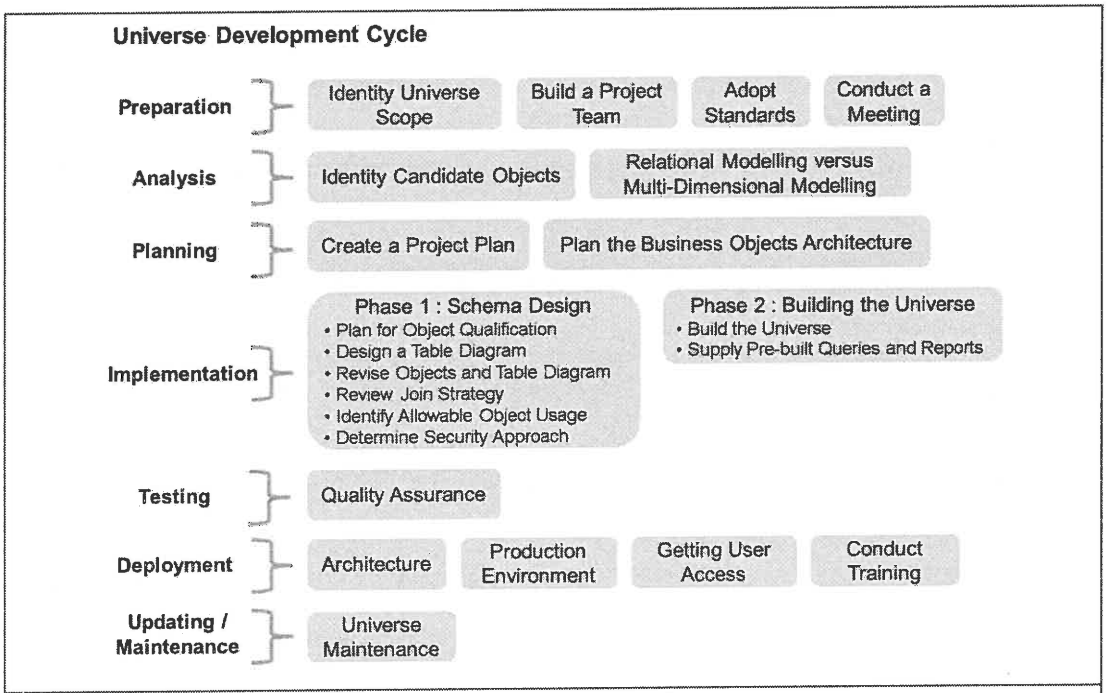


Figure 1: Universe Design Process

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

Implementation will be successful if the first three stages are carried out properly. SAP recommends that you spend 80% of the time allocated to the development of a universe on the first three stages:

- Preparing
- Analyzing
- Planning

Spending 80% of your time laying the foundation for your universe means that the remaining 20% of the time spent building your universe with the Information Design Tool will be much more productive.

Preparation Phase

During the preparation phase, the scope of a BusinessObjects universe is defined. The production and development architectures are identified and reviewed. Project teams are assembled and the initial task plan is defined.

Identify a Universe Scope

The definition and communication of project scope eliminates risk associated with deploying the universe to pilot users during the Implementation phase. The scope is defined in terms of intended functionality of the universe. Identification of target users of the universe also helps create a shared understanding of project objectives.

Key managers are involved in the scoping process. Once formulated, the objectives of the project are communicated to everyone involved, directly or indirectly.

Build a Project Team

In designating the team members, individuals are chosen to fill the following roles. One person may fill multiple roles.

Table 2: Project Team Roles

Role	Task
Sponsor	The sponsor is usually the individual funding the project. The project sponsor makes any final decisions regarding scope or unresolvable issues.
Project Leader	The project leader develops the project plan, assigns resources, tracks, and reports on progress.
Analyst	The analyst gathers requirements in the form of candidate objects.
Data Expert	The data expert is the individual who is familiar with the data structures.
Key User	The key user provides ongoing "business" perspective for developers.
Pilot Users	Pilot users work with the universe during the universe build and development phase.
QA Reviewer	The QA reviewer is an individual with BusinessObjects experience who is not part of the development process, and who performs a technical review of the final product.

In most cases, a single person is responsible for the bulk of the work, filling the roles of Analyst, BusinessObjects Administrator, and Data Expert. In designing and building the universe, this person maintains a relationship with the Key User, who should also be one of the Pilot Users.

This developer reports to a Manager or IS Director, who serves as Project Leader. The Leader maintains a close relationship with the Sponsor.

Other roles that will be impacted by the project include: the Database Administrator, the System Administrator, and the Data Administrator.

Adopt Standards

Standards for the components of a BusinessObjects universe help to guarantee consistency and stability in the final product. During preparation, the team adopts a set of standards for BusinessObjects components. Standards can be specified for:

- Universe names

- Object definition guidelines
- Folder names
- Alias names
- Help text

The standards may be revised during the first universe development project as the team becomes more familiar with the product.

Conduct a Meeting

Communicate the preparation phase strategy in a meeting. This meeting is your opportunity to gather all interested parties (developers, users, the sponsor) to ensure that everyone understands the scope of the endeavor. You can use this meeting to demonstrate BusinessObjects products and to help set expectations of the user community.

Analysis Phase

The primary objective of analysis activities is to identify user requirements for the on-demand query environment.

These requirements are captured in the form of candidate folders and objects.

Step 1: Identify Candidate Objects

There are many places to look for candidate objects. The best way to identify them is by talking to the end users. When interviewing end users, the type of questions to ask are: "What type of information do you need to do your job?", "How do you know you are doing well?", "How does your boss know you are performing well?", or "What kind of information do others ask you for?"

As users answer these questions, document their answers in terms of folder and object requirements. For example, if a user states, "We must retrieve information on employees by department and hire date," you have identified a potential folder ("information about employees") and an object or two ("department" and "hire date"). When you identify a potential folder, probe for objects. For example, "What kind of information about employees do they want?"

Candidate folders and objects can also be identified by reviewing existing reports.

Document your folders and objects. For example:

Type	Name	Description	Source
folder	Customer	Information on a customer, including location, credit ratings, and shipping preferences.	Interview number 1.
Object (Measure)	Total Revenue	This object can be combined with date ranges, customers, and products to provide meaningful measures.	Interview number 3, 4.

You can also try to document the qualification of objects (dimension, attribute, measure) and any potentially identified navigation paths.

Relational Modeling Versus Multi-Dimensional Modeling

The questions asked during BusinessObjects interviews are similar to those questions asked in the development of OLTP applications. However, what is done with the answers is different.

When conducting Analysis for an Online Transaction Processing (OLTP) application, analysts document data requirements in entity relationship diagrams. Rules of normalization are applied to the items that users request, breaking them down to an atomic level, or eliminating calculated objects. These activities optimize the data for storage in a relational database.

By contrast, requirements for an on demand query environment should be expressed in terms that are optimized for retrieval of the information.

A successful BusinessObjects universe presents information to a business person using user specific business terminology. The developer must "unlearn" analysis techniques used for the development of application systems. User requirements are taken at face value, remaining in business terms.

Basic rules of thumb:

- Do not normalize.
- Do not eliminate objects that can be derived from other objects.

For example, in an interview, a user states, "I must look at annual sales figures by region." Document this need at face value; identify the requirements, but do not attempt to transform them in a manner appropriate for storage in a relational database. You can identify three candidate objects: "Year of Sale," "Sales Amount," and "Region". Do not eliminate "Year of Sale" because you have already documented a "Date of Sale" object. Do not reduce "Sales" to the components from which it is calculated (perhaps "quantity" multiplied by "price"). Instead of normalizing object requirements, identify how they will support on-line analysis by end users.

Identifying candidate objects as dimensions, attributes, or measures facilitates end user reporting and analysis flexibility. You can also plan for scope of analysis (drill-down and drill-up options) by identifying dimensional navigation paths.

Once you have gathered and documented requirements in the form of candidate objects, you are ready to begin to plan the BusinessObjects universe requirements.

Planning Phase

The planning phase is used to identify a project strategy and determine resource requirements.

Step 1: Create a Project Plan

The project plan is the key to timely implementation. For each task, the plan should assign responsibility and target dates. Creation of the plan and the tracking of progress against the plan are the primary responsibilities of the project leader.

Step 2: Plan the BusinessObjects Architecture

Technical architecture requirements may have been briefly considered in the preparation phase. A review of the technical architecture is done during the planning phase of the project. Items to review include:

Table 3: Planning the BusinessObjects Architecture

Area	Task
Development Environment	Identify resources required to support a universe development environment.
Production Environment	Identify resources required for a universe production environment.
Computers	Review required computing resources for developer and user workstations.
Connectivity	Ensure that infrastructure is in place to support connectivity between users or developers and the repository and data stores, including appropriate middle-ware to support communication between clients and servers.
Configuration	Identify planned configuration for client software. Ensure that appropriate resources are available.
Security	Initiate a first look at security requirements.
Support Plan	Develop support policy for when the universe goes into production.
Change Management Plan	Identify procedures for the request, review, approval, and implementation of changes to the universe when in production.
Training Plan	Plan for a user training program.

Implementation Phase

The implementation phase can be split up into two stages:

1. Designing the data foundation.
2. Creating the business layer.

Implementation Phase 1: Data Foundation Design

Data foundation design comprises several tasks including the following:

Data Foundation Design Tasks

- Determine and document data sources
- Plan for object qualifications and drill-down functionality
- Design a table diagram
- Revise objects
- Review join strategy
- Identify allowable object usage
- Determine security approach

Determine and Document Data Sources

The first task during data foundation design is to determine and document the data source for each candidate object. If requirements were gathered in a tabular format, add a column to the

table where you can indicate the SQL fragment and source tables that are used to retrieve the object.

Table 4: Data Foundation Design: Data Source Determination

Type	Name	SQL Fragment	Description	Source
Class	Customer		Information on a customer, including location, credit ratings, and shipping preferences.	Interview number 1.
Object (Measure)	Total Revenue	SQL: `sum(order_lines.quantity* products.price)` Source Tables: • Order_Lines • Products	This object can be combined with date ranges, customers, and products to provide meaningful measures.	Interview number 3, 4.

Any candidate folders that were captured as general requirements without specific objects are expanded now. For example, suppose that there was a candidate folder called "Customer" and the specific objects within this folder were not identified. During the schema design stage, the developer must "fill out" this folder. The developer might fill it out based on knowledge of the business by including all columns from one or more tables, or the developer might go back to users for more detail.

There are several ways that objects can be mapped to enterprise data. Simple objects map back to a single column in the database. An example would be "Customer First Name", which maps back to the First_Name column in the Customers table. Complex objects use SQL to manipulate data that comes from one or more columns. For example, a "Customer Full Name" object might concatenate the First_Name and Last_Name columns from the Customers table.

Aggregate objects involve SQL GROUP functions. Counts, sums, and averages are all aggregate objects. The Total Revenue object is an aggregate object; it uses the SQL SUM function.

Plan for Object Qualifications and Drill-Down Functionality

As you design the universe, complete the process you began during analysis. Identify each object as a measure, a dimension, or an attribute. For each detail object, identify the dimension it is associated with.

Similarly, identify navigation paths within your dimensions. These navigation paths later enable users to "drill-down" and "drill-up".

Design a Table Diagram

Now that the objects are mapped back to data sources, the developer reviews all the objects and produces a table-diagram of the database objects that support the universe. Joins between the tables are then added to the diagram. The table diagram is a valuable tool for

resolving loops and SQL traps in the model. It also becomes an important reference for developers.

Hint:
If you find that you have documented a vast amount of folders and objects based on user requirements, you may consider designing data foundations that can be used to build multiple business layers, which cater to a specific function within the business, reducing the complexity and amount of folders and objects.

Revise Objects and Table Diagram

Once loops and SQL traps are resolved, some objects may require design modification. Any object based on a table that was replaced by an alias are updated. Consult your table of objects created in the preparation phase for such objects.

Some objects may be applicable in the context of more than one of the aliases; these objects will be split into multiple objects. Make sure that object names make it clear what each one represents.

Review Join Strategy

Where table relationships are optional, carefully choose the type of join to use. The use of standard (or inner) versus outer joins impacts the results of user queries. Using the wrong type of join may provide results that are not what users expect.

In SQL, a standard join between two tables returns only rows where both tables meet the join criteria. If one of the tables has no corresponding row in the second table, its data is not returned.

If one of the joined tables has no corresponding row in the other table, an outer join tells the database that is processing the SQL query to substitute a "null" row. With an outer join, information in one table that does not have corresponding data in the second table is returned with "blanks" in columns from the second table.

The developer must review join possibilities with a key user wherever optional relationships exist. The chosen solution should produce results that users are most likely to expect.

Identify Allowable Object Usage

The developer may identify certain objects that are not to be used in qualifications by end users. Certain complex objects may not be usable in qualifications for technical reasons, or there may be performance considerations.

Determine Security Approach

Security requirements are also addressed during the Implementation phase. Solutions to security requirements may involve complex object definition, reliance on database-level security, restriction sets, or the development of multiple universes. Chosen solutions may impact the database administrator and developers.

Implementation Phase 2: Business Layer Building

Once the data foundation is complete, the development team is ready to create the business layer.

Hint:
Remember that it is better to have several smaller less complex business layers than one large business layer. This method reduces maintenance, avoids potential security impacts, and will improve overall usability.

Pilot users then begin to use the universe. They provide feedback to developers who refine the universe until build is completed.

Supply Prebuilt Queries and Reports

During the build stage, the team may identify certain queries and reports that are of value to the entire enterprise. Created anytime throughout the build, these queries and reports are rechecked after the universe is finalized to ensure that objects used have not been renamed or removed. They are then exported to the repository so that they are available to all users.

Testing Phase

The pilot testing and refinement phase follows universe design implementation. Once an initial universe is built, it is deployed to the pilot users. These users work with the universe and provide feedback to the developers.

Testing Phase: Soliciting Feedback

- Better names for folders and objects
- Objects to be added to the universe
- Objects that can be removed
- Better ways to organize objects (For example, move an object from one folder to another, reclassifying a dimension as a detail, and so on.)
- Objects or queries that do not behave as expected

Based on this feedback, the universe is modified. The modified universe is made available to the pilot users for further evaluation. The testing phase can also address potential performance issues. As a developer you can look at implementing performance enhancements to the universe.

Quality Assurance

After the build is finalized, the universe is reviewed for quality assurance. An independent reviewer makes the following checks:

- Corporate standards for universe, object, folder, and alias naming are followed
- Objects are only defined with tables that are referenced in the SELECT or WHERE clauses
- Objects return results without syntactic error
- Objects return intended business results
- Objects are correctly classified as dimensions, attributes, or measures
- Defined navigation paths make sense
- Objects have help text
- Aliases are used appropriately

- Join syntax and foreign keys are accurate
- Standard and outer joins are used appropriately

These checks are best made by an individual who was not part of the development of the universe, guaranteeing an objective perspective. Any issues that are identified are reported to the developers for correction and review.

Deployment Phase

The universe has been built, and has passed all quality assurance checks. It is now ready for deployment.

The final deployment of the universe cannot begin until any architectural issues identified during planning phase have been addressed. These issues include the establishment of user connectivity, planning the installation configuration, preparation of a training program, and identification of support and change management processes.

Architecture

Architectural considerations identified during the planning phase are reviewed. Any issues that have not been resolved delay the deployment phase.

Production Environment

The production environment has been set up in accordance with the architecture and security plans identified during preparation and planning. The universe is modified to access data from production systems, rather than from development systems and is exported to the production repository.

Granting User Access

Any database accounts that are required for BusinessObjects users are created by the database administrator. These accounts are given appropriate access privileges to the data objects used by the universe. Users are also added to the Central Management System (CMS) and granted access to the universe.

Conduct Training

The release of the BusinessObjects universe to production users is coordinated with system and database administrators as appropriate. The user training program is executed with the roll-out of the universe. Without appropriate training, users will not derive benefits from the reporting and analysis applications, regardless of the quality of the universe.

Prepackaged Solutions

If you are designing a universe for Business Objects developers to develop precreated or prepackaged reports, consider:

- Predefining all filters and calculations that are used in standard documents to remain consistent throughout.
- Covering more than one business function to allow cross functional reporting. Precreated reports tend to cross reference reports against different business functions; therefore, the universe has to cover multiple business functions to provide end-to-end business reporting.

Information Design Tool Users

Typically, universe designers are IT or highly technical business users who understand SQL or MDX code. The universe designer may be a database administrator, an applications manager or developer, a project manager, or a report creator who has acquired enough technical skills

to create Business Layers for other users. A security administrator also uses the information design tool to define Business Layer security profiles.

There can be more than one designer in a company. The number of designers depends on the company's data requirements. For example, one designer could be appointed for each application, project, department, or functional area. When several people create universes, we recommend that you define a set of rules or guidelines for terminology so that objects are represented consistently.

LESSON SUMMARY

You should now be able to:

- Discuss universes

Unit 1
Lesson 2

Defining the Components of a Universe

LESSON OVERVIEW
This lesson outlines the first step in creating resources in the Information Design Tool: creating a local project.

LESSON OBJECTIVES
After completing this lesson, you will be able to:

- Identify local projects

Projects

A project is a named local workspace that contains the resources used to build one or more universes.

A project can be shared so that multiple designers can work on the same resources.

A project can contain any number of independent resources, for example data foundations, business layers, and connections. All resources contained within a project can be used interchangeably; for example, a connection can be used by several data foundations within the same project.

Local Project Creation

The first step in creating resources in the Information Design Tool is to create a local project in the Local Projects View. You create and edit all resources except secured connections and security profiles in a local project.

The resources and folders in a local project are stored as physical files and folders in the local file system. The Local Projects View lets you to navigate local projects and open resources in the information design tool.

Populating a Project with Resources

Once you have created a local project, there are several ways you can populate it with resources:

- Create resources using the wizards available on the *New* menu
- Convert a .unv universe that was created with the universe design tool, or migrated from an earlier version
- Retrieve a published universe

You edit the resources using the information design tool editors by double-clicking the resource name in the local project.

Operations Allowed on Local Resources

- Create a shared project so that you can share resources with other designers
- Check the integrity of data foundations and business layers
- Publish a business layer as a universe to the local file system or a repository
- Publish a connection to a repository
- Show dependent resources
- Save a resource as a report

- Create a shared project so that you can share resources with other designers
- Check the integrity of data foundations and business layers
- Publish a business layer as a universe to the local file system or a repository
- Publish a connection to a repository
- Show dependent resources
- Save a resource as a report

Unit 1
Exercise 1

Create a Local Project

Business Example

Your organization has chosen to use the Information Design Tool to create a Business Layer using a SQL Server database to provide nontechnical business users with semantically understandable business objects. These users can then analyze data and create reports using relevant business language regardless of the underlying data sources and structures.

The first step in the Data Foundation and Business Layer creation process is to create a local project.

Note:
In the data provided for you to complete the exercises, replace the xx with the group number the instructor provided you with at the beginning of the class.

Note:
The steps in the activity represent some of the steps demonstrated by the instructor in the classroom.

1. Launch the Information Design Tool.
2. Create a new local project called Motors_xx, where xx is the student number your instructor assigned to you.

Unit 1
Solution 1

Create a Local Project

Business Example

Your organization has chosen to use the Information Design Tool to create a Business Layer using a SQL Server database to provide nontechnical business users with semantically understandable business objects. These users can then analyze data and create reports using relevant business language regardless of the underlying data sources and structures.

The first step in the Data Foundation and Business Layer creation process is to create a local project.

> Note:
> In the data provided for you to complete the exercises, replace the xx with the group number the instructor provided you with at the beginning of the class.

> Note:
> The steps in the activity represent some of the steps demonstrated by the instructor in the classroom.

1. Launch the Information Design Tool.
   a) From the *Start* menu, choose *All Programs* → *_SAP Business Intelligence* → *SAP BusinessObjects BI platform 4 Client* → *Information Design*.
2. Create a new local project called Motors_xx, where xx is the student number your instructor assigned to you.
   a) From the menu bar, choose *File* → *New* → *Project*.
   What appears?
   b) In the *Project Name* box, enter `Motors_xx`.
   c) Choose *Finish*.
   The project is created in the local file system and displayed in the *Local Projects View*.

LESSON SUMMARY

You should now be able to:

- Identify local projects

Unit 1

Learning Assessment

1. Put the following Universe Design process steps in the correct order.

   Arrange these steps into the correct sequence.

   - [ ] Testing
   - [ ] Planning
   - [ ] Deployment
   - [ ] Analysis
   - [ ] Preparation
   - [ ] Implementation

2. Which are the operations you an do on the local resource with the Information Design Tool editors?

   Choose the correct answers.

   - [ ] A Save a resource as a report
   - [ ] B Visualize the resource design architecture
   - [ ] C Publish a report
   - [ ] D Publish a connection to a repository

3. You can populate a local project with resources by retrieving a published universe.

   Determine whether this statement is true or false.

   - [ ] True
   - [ ] False

Learning Assessment - Answers

1. Put the following Universe Design process steps in the correct order.

   Arrange these steps into the correct sequence.

   [5] Testing

   [3] Planning

   [6] Deployment

   [2] Analysis

   [1] Preparation

   [4] Implementation

2. Which are the operations you an do on the local resource with the Information Design Tool editors?

   Choose the correct answers.

   [X] A Save a resource as a report

   [ ] B Visualize the resource design architecture

   [ ] C Publish a report

   [X] D Publish a connection to a repository

3. You can populate a local project with resources by retrieving a published universe.

   Determine whether this statement is true or false.

   [X] True

   [ ] False

UNIT 2 Data Connections

Lesson 1

UNIT OBJECTIVES

- Define data connections

Unit 2
Lesson 1

Defining Connections

LESSON OVERVIEW
A connection is a named set of parameters that define how one or more SAP BusinessObjects applications can access relational or OLAP data sources. This lesson describes how to create a new connect to the database.

LESSON OBJECTIVES
After completing this lesson, you will be able to:

- Define data connections

Connections

A connection is a named set of parameters that define how one or more SAP BusinessObjects applications can access relational or OLAP data sources. A connection can be a local file or a remote object in a repository that is referenced by a local shortcut in the information design tool.

Table 5: Purpose of Connections

Connections used for...	Description
Relational data source for data foundation	Associate one or more relational connections to a data foundation and build a business layer on the data foundation. When you publish the business layer as a universe, the connections and the data foundation are integrated in the universe and provide the data for queries run against the universe.
OLAP data source for a business layer	For an OLAP data source, associate a business layer directly to a connection. When you publish, the business layer is published as a universe, and the connection provides direct access to the cube.
Access to an SAP BW BEx Query	Define SAP BW connections that use the SAP BICS Client middleware driver to provide access to the BEx Query. SAP BusinessObjects query and reporting applications connect directly to the BEx Query. You cannot use these connections as a source for business layers or universes.

Connections can be local or secured.

Local Connections

You create local connections in the Information Design Tool local project. Local connections are saved as independent objects on the local file system as .cnx files. When a connection is published, a connection object containing the same parameters as the local connection can be created in the `Connections` folder or subfolder in the repository. A local shortcut is created for the connection in the local project. You can use the shortcut in the same way as a local connection; however, the target connection is secured and subject to security restraints in the repository.

Purpose of Local Connections

Use local connections to:

- Access relational data sources when you are authoring a data foundation and relational business layer.

> Note:
> To create a multisource-enabled data foundation, you must reference secured connections.

- Access an OLAP cube when you are authoring an OLAP business layer.
- Run queries on a target database when you are testing modifications in the business layer or to build lists of values.

Local connections are primarily used in the authoring phase of data foundation and business layer development. They can be used by any user with access to the system running the Information Design Tool; therefore local connections have limited or no security.

After the business layer has been published as a universe to the repository, the connection is secured. Although the connection shortcut is available locally, the published connection can be edited only with appropriated repository system authorization.

Secured Connections

A secured connection is a connection that has been created in, or published to, an SAP BusinessObjects repository. It is stored in a dedicated Connections folder in the repository. You can create subfolders in the Connections folder to organize connections in the repository.

Secured connections cannot be copied to the local file system; however, they are available in the Local Projects View as connection shortcuts. A connection shortcut is an object that references a secured connection in a repository. The shortcut can be used in the same way as a local connection; however, the connection properties can be modified only by connecting to the repository system.

Purpose of Secured Connections

Use secured connections and connection shortcuts to:

- Retrieve data for universes published to a repository
- Retrieve data for SAP BusinessObjects reporting products directly accessing database middleware
- Provide direct access to an OLAP cube

You can create a secured connection by publishing a local connection to a repository, or by creating the connection directly in the repository. A secured connection is subject to the following general security constraints in the repository:

- Users must be authenticated.
- User rights can be defined at the user level to grant or deny access to connections or connection properties.
- Only authenticated users can use and share connections.

Unit 2
Exercise 2

Create a Relational Connection

Business Example.

Now that you have created a local project, create a connection to a data source. Your organization uses SQL Server 2008 as its primary database with an OLE DB connection.

Note:
To complete the exercises, replace the xx with the group number the instructor provided you with at the beginning of the class.

1. Using the login credentials in the following table, create a relational connection called Motors_xx. As the database driver, use the MS SQL Server 2008 OLE DB Providers.

Field Name	Enter
Authentication Mode	User specified username and password
User name	Provided by instructor
Password	Provided by instructor
Server	Provided by instructor
Database	Motors

The Motors_xx connection is created.

Unit 2
Solution 2

Create a Relational Connection

Business Example.
Now that you have created a local project, create a connection to a data source. Your organization uses SQL Server 2008 as its primary database with an OLE DB connection.

> Note:
> To complete the exercises, replace the xx with the group number the instructor provided you with at the beginning of the class.

1. Using the login credentials in the following table, create a relational connection called Motors_xx. As the database driver, use the MS SQL Server 2008 OLE DB Providers.

Field Name	Enter
Authentication Mode	User specified username and password
User name	Provided by instructor
Password	Provided by instructor
Server	Provided by instructor
Database	Motors

   a) In the *Local Projects* View, choose your `Motors_xx` project folder.

   b) Choose *File* → *New* → *Relational Connection*.
   The *New Relational Connection* wizard opens.

   c) As the name of your connection, enter **`Motors_xx`**, where xx is the student number your instructor assigned to you.

   d) Choose *Next*.
   The *Database Middleware Driver Selection* screen appears.

   e) On the *Database Middleware Driver Selection* screen, expand to *Microsoft* → *MS SQL Server 2008*.

   f) Choose *OLE DB Providers*.

   g) Choose *Next*.

> Note:
> There might be a slight delay before the *Next* button is enabled.

h) Enter the login credentials listed in the table in step 1.

i) Choose *Next*.

> Note:
> If the *Next* button is disabled, click elsewhere on the desktop before choosing *Next*.

j) Accept the default parameters and choose *Finish*.

The Motors_xx connection is created.

LESSON SUMMARY
You should now be able to:

- Define data connections

Learning Assessment

1. What is a secured connection?

2. A secured connection can be copied to the local file system.

   Determine whether this statement is true or false.

   ☐ True

   ☐ False

Learning Assessment - Answers

1. What is a secured connection?

   A secured connection is a connection that has been created in, or published to, an SAP BusinessObjects repository.

2. A secured connection can be copied to the local file system.

   Determine whether this statement is true or false.

   ☐ True

   ☒ False

UNIT 3 Data Foundations

Lesson 1

Lesson 2

UNIT OBJECTIVES

- Create data foundations
- Use joins
- Create an equi-join
- Create an outer join
- Create a theta join
- Create column filters to restrict data

Unit 3
Lesson 1

Creating Data Foundations

LESSON OVERVIEW
Data foundation contains a schema of relevant tables and joins from one or more relational databases that are used as a basis for one or more business layers. This lesson explores how to create data foundations.

LESSON OBJECTIVES
After completing this lesson, you will be able to:

- Create data foundations

Data Foundations

A data foundation contains a schema of relevant tables and joins from one or more relational databases that are used as a basis for one or more business layers.

You reference relational connections in the data foundation. You insert tables and joins from the databases referenced in the connections.

Using the Data Foundation Editor, you can enhance the data foundation by adding derived tables, alias tables, calculated columns, additional joins, contexts, prompts, and lists of values. The availability of some features depends on the data foundation type.

You can build any number of business layers on the same data foundation. In this case the data foundation becomes the basis for multiple universes.

Single-source and multisource-enabled are the two types of data foundations that allow you to take advantage of different data foundation features.

Data Foundation Editor

The Data Foundation Editor is divided into a data foundation view and browsing panes.

The data foundation view is a graphical representation of the tables and joins. The Master view contains all tables and joins and cannot be deleted. You can define custom views that contain subsets of the tables. Access the views by the tabs at the bottom of the view pane. In the data foundation view, you can work on tables and joins using commands in the Insert and Detect menus, or by clicking objects directly in the view.

Data Foundation Editor Tabs

The browsing panes allow you to work with different elements of the data foundation. Access the panes by clicking the corresponding tab:

- Connections
- Data Foundation (displays a tree view of the tables and joins)
- Aliases and Contexts

- Parameters and Lists of Values
- Properties

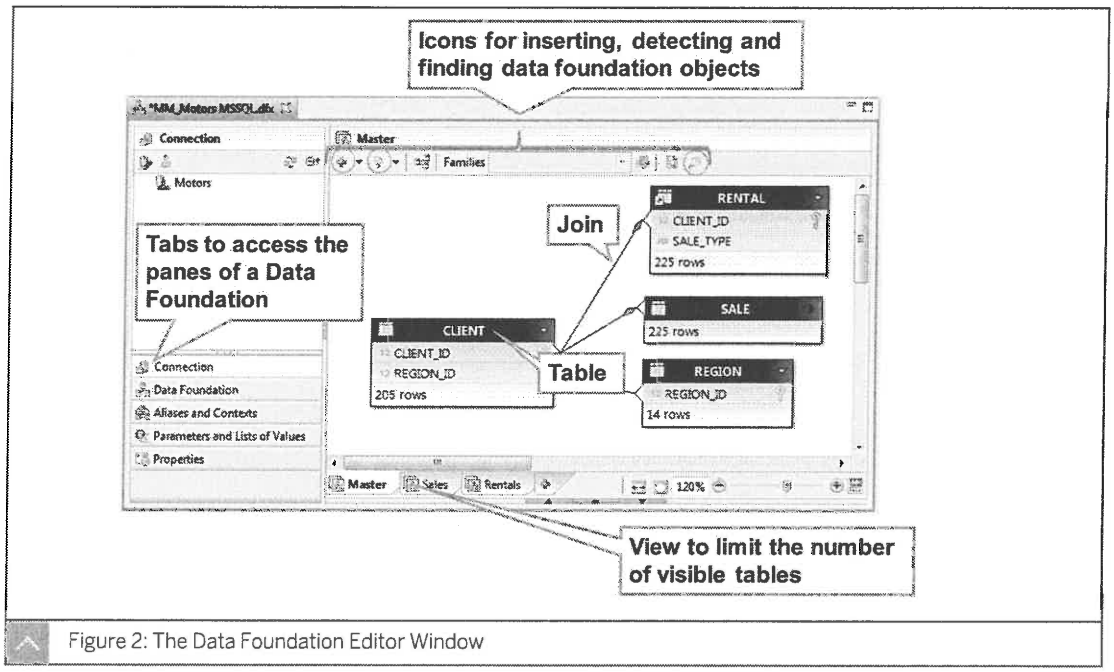


Figure 2: The Data Foundation Editor Window

Data Foundation View Navigation

To access a menu of commands that are available on tables, right-click the table header in the data foundation view. To select multiple tables, click the table headers while holding down the CTRL key.

To access commands that are available on columns, right-click the column name in the table in the data foundation view.

Data Foundation Navigation Commands

Several commands in the table right-click menu are available to help you locate related tables in the data foundation:

- **Select Related Tables** automatically selects all tables linked by joins to the selected table.
- **Highlight Related Tables** grays any table that is not linked to the selected table by a join.
- **Highlight Aliases** grays all tables except the selected standard table and its alias tables.
- **Highlight Original Table** grays all tables except the selected alias table and the standard table on which it is based.

You can also use the search panel to perform advanced searches on the data foundation. To open the search panel, click .

Families

A family allows you to define a certain color and font to associate with tables on the data foundation. This formatting can be helpful in easily identifying tables that have a specific relationship on the data foundation. Families are stored in XML format and can be imported or exported.

After the families have been created, you can associate them with zero or multiple tables in the data foundation. To associate a family with a table, select one or more tables and then select the appropriate family in the dropdown list in the toolbar above the data foundation schema.

Unit 3
Exercise 3

Insert Tables in a Data Foundation

Business Example

You need to create the data foundation to identify the tables in the SQL Server database that contain data the business users need for their reports.

Create a Data Foundation

Create a data foundation.

> Note:
> In the data provided for you to complete the exercises, replace the xx with the group number the instructor provided you with at the beginning of the class.

1. Prepare the design of your data foundation by defining business requirements for the business layers to which it will provide data. In other words, determine what data is needed for business users' reports.

> Note:
> This preparation has already been done for you for this course. But, in a real-world scenario, it is always the first step of creating a data foundation.

2. Create a new Data Foundation called Motors_xx, where xx is the student number your instructor assigned to you.

Insert Tables

Now that you have a data foundation, called Motors_xx data, you can insert the required tables:

1. Insert the following tables in the Motors_xx data foundation:
   - CLIENT
   - COLOR
   - COUNTRY
   - FINANCE_PERIOD
   - MAKER
   - MODEL
   - REGION
   - SALE

- SALES_PRICE_RANGE
- SALE_MODEL
- SHOWROOM
- STYLE

Unit 3
Solution 3

Insert Tables in a Data Foundation

Business Example

You need to create the data foundation to identify the tables in the SQL Server database that contain data the business users need for their reports.

Create a Data Foundation

Create a data foundation.

> Note:
> In the data provided for you to complete the exercises, replace the xx with the group number the instructor provided you with at the beginning of the class.

1. Prepare the design of your data foundation by defining business requirements for the business layers to which it will provide data. In other words, determine what data is needed for business users' reports.

> Note:
> This preparation has already been done for you for this course. But, in a real-world scenario, it is always the first step of creating a data foundation.

2. Create a new Data Foundation called Motors_xx, where xx is the student number your instructor assigned to you.
   a) In the *Local Projects* View, choose your `Motors_xx` project folder.
   b) Choose *File* → *New* → *Data Foundation*.
      The *New Data Foundation* wizard appears.
   c) As the name of your data foundation, enter **`Motors_xx`**, where xx is the student number your instructor assigned to you.
   d) Choose *Next*.
   e) Since the Data Foundation will use only a single connection, choose *Single Source*.
   f) Choose *Next*.
   g) To use the connection for this data foundation, beside your Motors_xx data connection, check the box.
   h) Choose *Finish*.
      The *Master* tab appears. In the next task, Insert Tables, you continue from this step to add tables to your data foundation.

Insert Tables

Now that you have a data foundation, called Motors_xx data, you can insert the required tables:

1. Insert the following tables in the Motors_xx data foundation:
   - CLIENT
   - COLOR
   - COUNTRY
   - FINANCE_PERIOD
   - MAKER
   - MODEL
   - REGION
   - SALE
   - SALES_PRICE_RANGE
   - SALE_MODEL
   - SHOWROOM
   - STYLE

   a) On the right of the *Master* tab, right-click the open space.

   b) Choose *Insert* → *Tables*.

   c) To choose the table names that you want to insert, in the *DBO pane*, expand the Motors_xx connection and choose the tables listed previously.
   You can also use CTRL+click to choose multiple tables.

   d) Choose *Finish*.

LESSON SUMMARY
You should now be able to:

- Create data foundations

Unit 3
Lesson 2

Using Joins

LESSON OVERVIEW
A join is a condition that links tables in the data foundation. This lesson describes how to define and create joins.

LESSON OBJECTIVES
After completing this lesson, you will be able to:

- Use joins
- Create an equi-join
- Create an outer join
- Create a theta join
- Create column filters to restrict data

Joins

A join is a condition that links tables in the data foundation. A join restricts the data returned when the two tables are queried.

Tables that are joined have a parent-child relationship. If tables are not joined, then a query run on the two tables can return a result set that contains all possible row combinations. Such a result set is known as a cartesian product and is rarely useful.

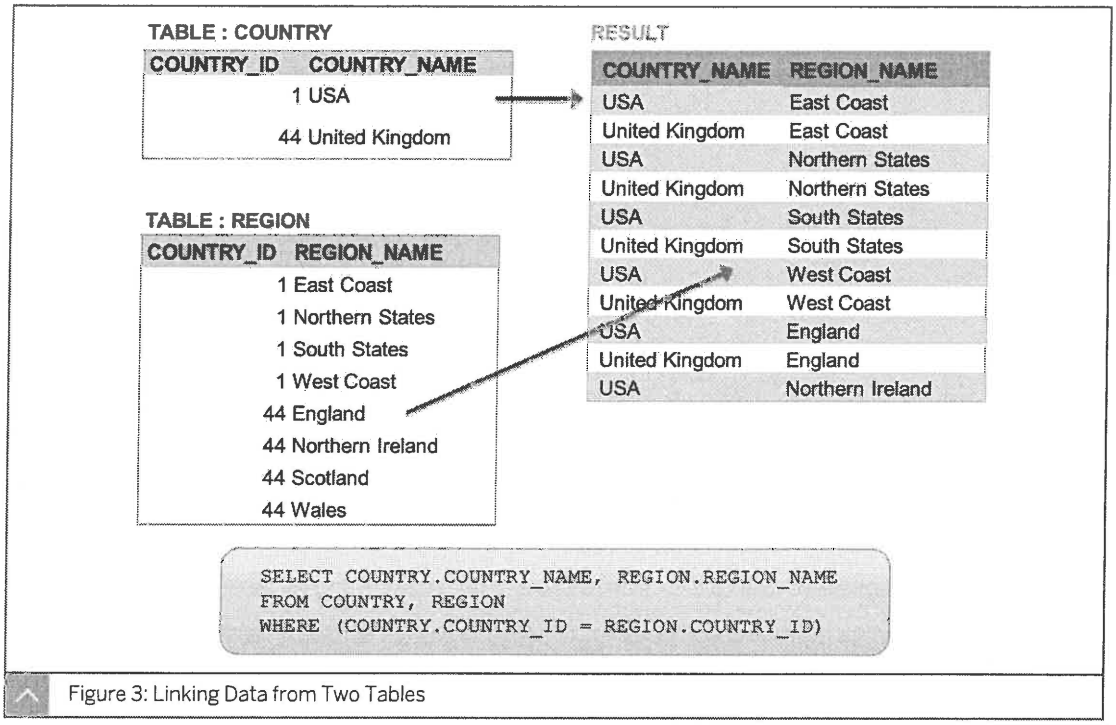


Figure 3: Linking Data from Two Tables

Joins are defined by linking a column in one table to a column in a second table. Self-joins are used to define column filters on one table.

Table 6: Types of Joins

Join Type	Description
Equi-join (default) (including the complex equi-join)	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 ssame column is present in both tables, the join synchronized the two tables. You can also create complex equi-joins where one join links multiple columns between tables. Equi-join in the join type created by default between two tables.
Outer join	Link two tables, one of which has rows that may not match those in the common column of the other table.
Theta join (conditional join)	Link tables based on a relationship other than equality between two columns, as in, for example a BETWEEN join.
Shortcut join	Join providing an alternate 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
Column filter	Single-table join used to set a restriction on the table.

To insert a join, select the *Insert Join* command from the Insert menu in the data foundation view.

When you insert a join, define the following join properties in the *Edit Join* dialog box:

Table 7: Join Properties

Join Property	Description
Table 1 Column	The column in the first table to use for the join.
Table 2 Column	The column in the second table or the column to join to.
Join Operators	Between Table 1 and Table 2, a list box of join operators lets you choose how to compare the values of the columns in the join. The list includes an equi-join (=) and operators for joins that are not based on equality between columns (>, >=, <, <=, !=) You can also choose to create a complex join, which is a join that contains sub queries.
Shortcut join	Select the *Shortcut join* check box to create a shortcut join. Shortcut joins provide alternative paths between two tables and improve the performance of queries. Shortcut joins do not take into account intermediate tables so longer join paths are shortened.
Outer join	Select the *Outer join* check box to create outer joins. An outer join allows rows to be returned even when there is no matching row in the joined table. To create a left outer join, select the *Outer join* check box under Table 1. This join returns all the rows in Table 1 even when they do not have a match in Table 2. To create a right outer join, select the *Outer join* check box under Table 2. This join returns all the rows in Table 2 even when they do not have a match in Table. To create a full outer join, select the *Outer join* check box under both tables. This join returns all rows from both tables with null values when there is no match.
Expression	Based on the columns and operators you select, an SQL expression is automatically generated to define the join. You can type a custom expression for the join. For help editing the join expression, click the *SQL Assistant* icon.

Join Property	Description
Cardinality	Select the cardinality for the join from the *Cardinality* list box. You can also click the *Detect* button to automatically detect the cardinality defined for the join the database.

Approaches to Creating Joins

- Defining joins manually in the schema
- Defining join properties directly in the *Edit Join* dialog box
- Allowing the Information Design Tool to detect the joins for you

Join Detection

Join detection looks at the data foundation tables and proposes appropriate joins.

Methods of Detecting Joins

- Join detection based on column name. This method looks for identical column names in different tables. It also checks that the data type of the two columns is the same. If more than one column matches between two tables, joins are proposed for each column.

Hint:
Joins between a table and its alias are not proposed.

- Join detection based on database keys. This method looks for relationships defined in the database between primary keys and foreign keys.
- For data foundations with an SAP BW connection, join detection is based on the joins in the database schema referenced in the connection.

To detect joins in the data foundation, select *Detect Joins* from the *Detect* menu in the data foundation view and select the join detection method.

For a multisource-enabled data foundation, select a method for each connection. This method is used to detect joins between tables referenced by the connection. You can also detect joins between tables from different connections. In this case, the method used is by column name.

Once you have selected the join detection method, the joins are detected and proposed in a dialog box.

You can then select the joins to be inserted into the data foundation.

When inserting tables, joins can be detected and inserted automatically. Set the defaults for automatic join detection, and the default detection method to use, in the application preferences page. From the main menu, select *Window* → *Preferences* → *Information Design Tool* → *Data Foundation Editor* → *Automatic Detections*.

Cardinality

Cardinality further describes how tables are joined by stating how many rows in one table match the rows in another table. Cardinalities are needed when detecting aliases and contexts to resolve loops in the data foundation.

Expressing Cardinality

- Cardinality of a join is expressed as a pair of numbers: the number of rows in one table that match the number of rows in the joined table.
- The number of rows that match can be none (0), one (1), or many (n) for each table.
  - For example, the tables <Customer> and <Reservations> are joined.
  - For each customer, there can be one or more reservations, so the cardinality of the <Customer> table is one-to-many, or 1,n.
  - For each reservation, there can be one and only one customer, so the cardinality of the <Reservations> table is one-to-one, or 1,1.

Cardinalities can be detected for joins automatically and stored in the data foundation. The detection method first detects primary and foreign keys. Cardinalities are set according to the key status of the column in the two tables as follows:

Table 8: Key Table

First Table Column	Second Table Column	Cardinality
Primary key	Foreign key	1, n
Foreign key	Primary key	n, 1

If no keys are detected, the cardinality is set using table row counts.

To detect or set cardinalities, select *Detect Cardinalities* in the *Detect* menu. The *Detect Cardinalities* dialog box lists the current cardinalities stored for all joins in the data foundation. From this list, you can set manually, or detect cardinality for a selection of joins.

To detect cardinality for one join, right-click the join line in the data foundation view and select *Detect Cardinality*. The cardinality of the selected join is updated.

Hint:
To select multiple joins, click the join lines while holding down the CTRL key.

You can also manually set cardinality when editing the join details with the *Edit Join* command.

When inserting joins, cardinality can be detected and set automatically. Set the defaults for automatic cardinality detection in the application preferences page. From the main menu, select *Window* → *Preferences* → *Information Design Tool* → *Data Foundation Editor* → *Automatic Detections*.

Manual Cardinality Setting

When you set cardinalities manually, consider each individual join. This consideration helps you to become aware of potential join path problems in your schema.

If you rely on automatically detected cardinalities, you may not find these problems. Such problems include 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.

Equi-Joins

An equi-join is a join based on column values between two tables. In a normalized database, the columns used in an equi-join are frequently the primary key from one table and the foreign key in the other. A primary key of a relational table uniquely identifies each record in the table. Primary keys may consist of a single attribute or multiple attributes in combination. A foreign key is a field in a relational table that matches the primary key column of another table.

When a SELECT statement is run, the SELECT and FROM clauses are now properly defined and prevent a cartesian product.

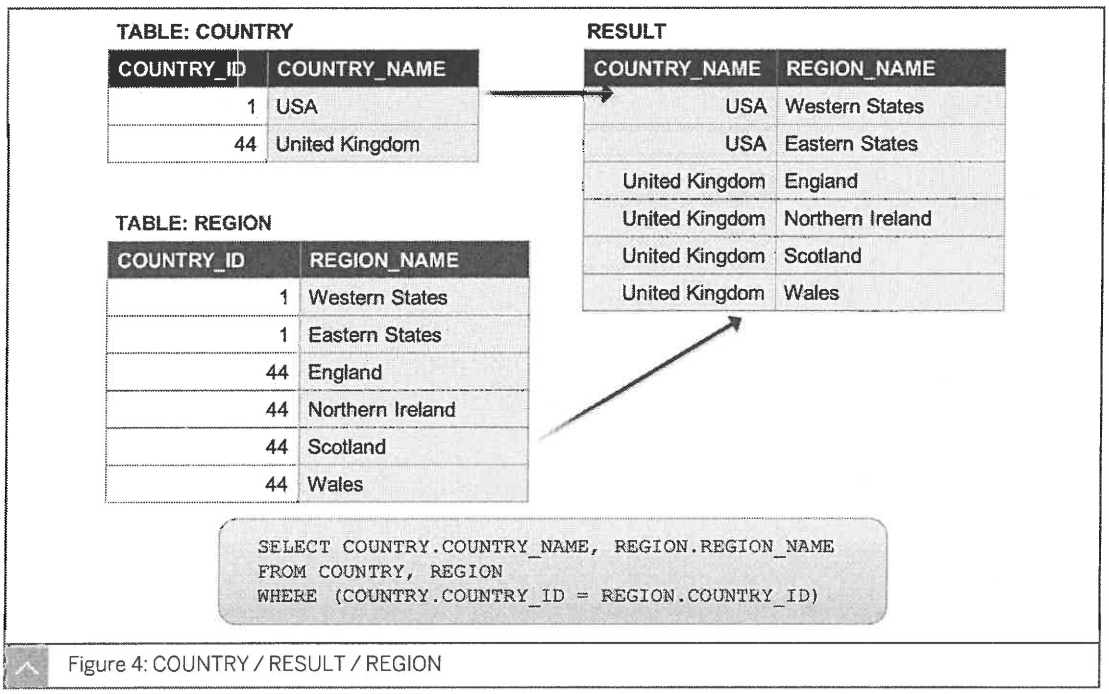


Figure 4: COUNTRY / RESULT / REGION

Outer Joins

An outer join is a join that links two tables, one of which has rows that may not match those rows 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.

For instance, the example illustrated previously shows the Country and Region tables from a database. Note that there are three different values in the primary key of the Country table, and only two distinct values in the corresponding foreign key of the Region table. If you were to apply an equi-join, the result set of a query would only show information on US and UK. However, you may wish to show all three countries irrespective of equivalent foreign key values in the Region table. To create this view, use an outer join.

In specifying an outer join in a standard SQL SELECT statement, you are required to identify which of the two tables is the outer. Using straight SQL (as opposed to generating it using a universe), the problem is that different RDBMS define outer differently and the syntax of the statement also differs. For example, depending on the underlying RDBMS, the outer join maybe on the left or right.

In a universe, the outer join is always placed on the table that contains all the data. That is, on Country in the previous example. To do this join, place a check against the table that contains all the data in the *Edit Join* dialog box.

Hint:
A good way to find out where to place your outer join is by reading the description that shows up in the Cardinality zone. If you select the outer join check box for Country, the description reads:

Each Country has ZERO or more Regions, AND each Region has one and only one Country.

When you check the outer join box for the Country table, you retrieve all countries whether they have a region or not.

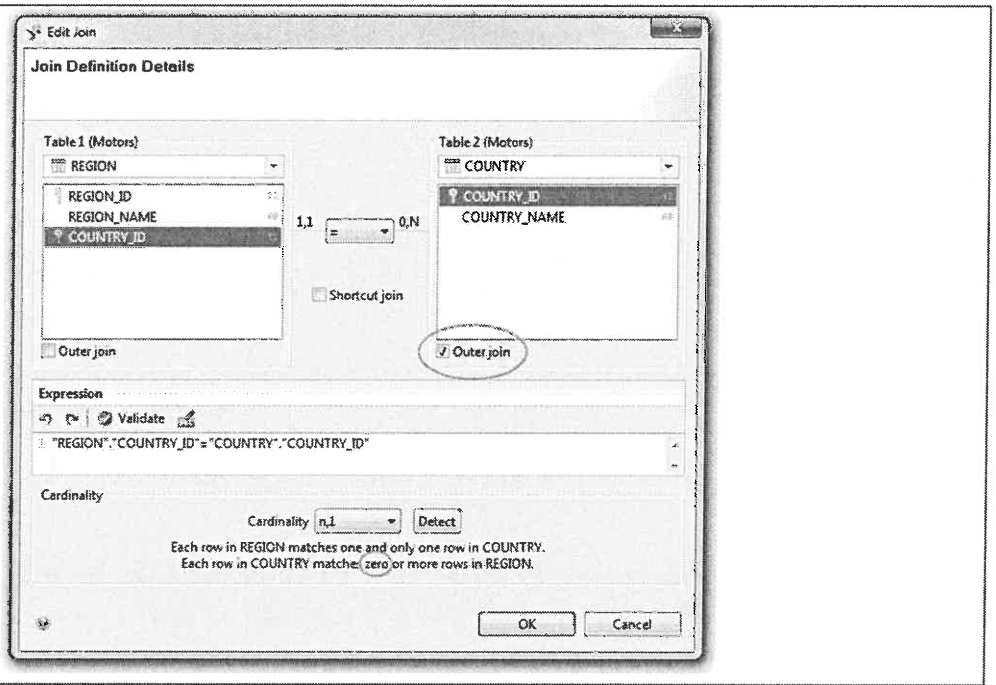


Figure 5: Creating an Outer Join

Once this step is done, the correct outer join is inferred when used in the generated query, and the syntax is correctly inferred for the appropriate RDBMS (assuming you have the correct SQL inference driver).

An outer join is shown by a small circle on the join line in the universe structure at the end that points to the table that may have missing values.

Outer Join Best Practices

We recommend that outer joins be placed at the end of the flow of data; otherwise, ambiguous outer join errors may occur. Potentially, these errors could cause the SQL to try to match on the equality of a NULL value, which it cannot do.

If you do place outer joins in the middle of a table path, make the subsequent joins in the path outer to avoid errors.

Always remember that outer joins may cause the query to run slower than a query with no outer joins.

This problem can be resolved by using aliases and aggregate aware, discussed later in this course.

Theta Joins

A theta join is a "between-type" join that links tables based on a relationship other than equality between two columns. It is used to demonstrate ranges, such as start date and end date, or minimum and maximum. A theta join can use any operator other than the equal operator.

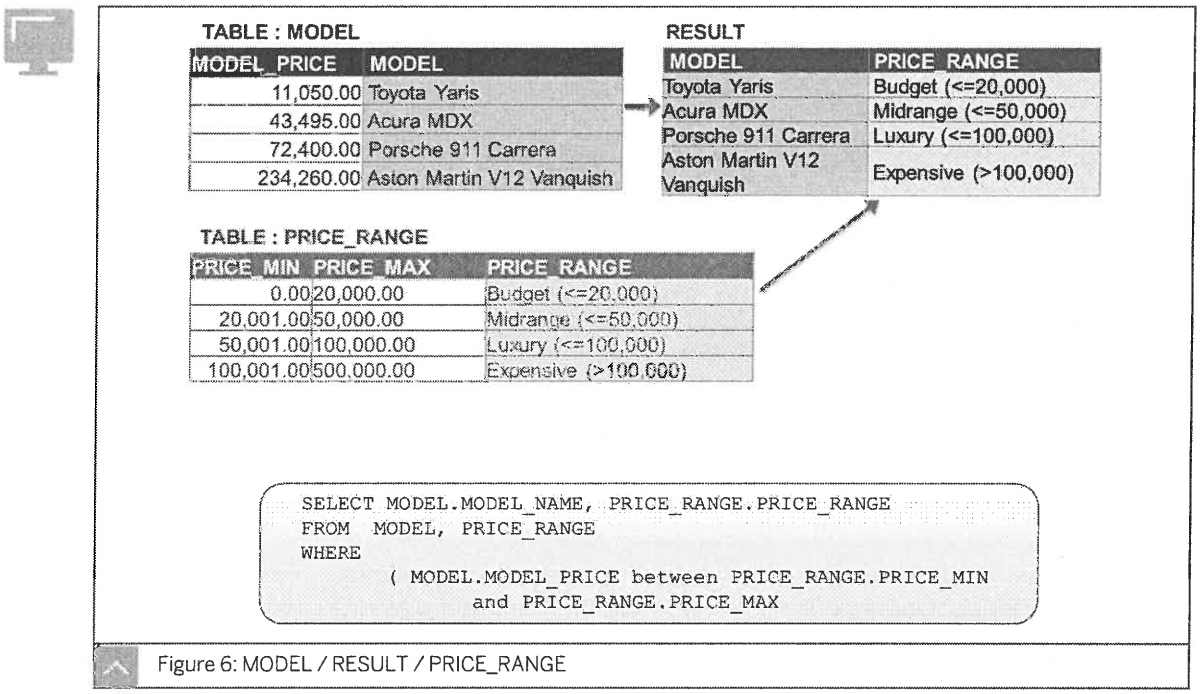


Figure 6: MODEL / RESULT / PRICE_RANGE

For instance, there is a table in the Motors database called Sales_Price_Range. This table contains a number of rows defining fixed price ranges by which you may wish to analyze data as illustrated in the previous figure.

To do this analysis, include the table in the universe structure and a join set. The obvious table to join it to is the Model table, which includes the price of a model. However, there is no common column between the Sales_Price_Range and Model tables, so an equi-join cannot be used. Instead, infer that a join exists where the value in a row of the Model_Price column in the Model table is between the values in a row for the Price_Range_Min and Price_Range_Max columns of the Sales_Price_Range table.

Column Filters

A column filter lets you restrict the values returned whenever the table is used in a query.

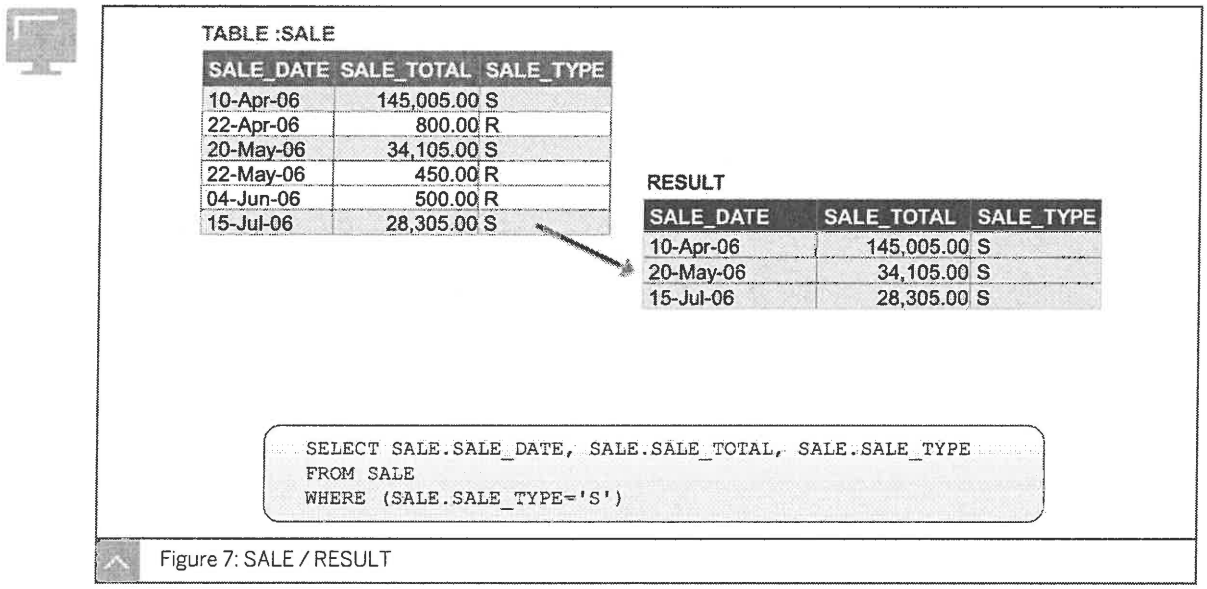


Figure 7: SALE / RESULT

The following rules apply to column filters:

Rules for Column Filters

- Only one filter per column is allowed.
- You can insert a filter on a calculated column.
- The expression can contain subqueries.
- The following @functions are allowed in the expression: @Prompt and @Variable.
- If you insert a filter into a standard table, and then create an alias from the table, the filter is not inserted into the alias table.
- If you insert a filter into an alias table, the filter is not automatically inserted into the original standard table.

You can view the join expression underlying a join line, or all the join expressions for a table, using List Mode as well as the *Edit Join* dialog box.

Equi-Join Creation

The Equi-Join creation links 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. Equi-join is the join type created by default between two tables.

Outer Join Creation

Outer Join creatiion links two tables, one which has rows that may not match those in the common column of the other table.

Theta Join Creation

Theta Join creation links tables based on a relationship other than equality between two columns, as in for example, a BETWEEN join.

Column Filter Creation

Column filter creation is a single-table join used to set a restriction on the table.

Unit 3
Exercise 4

Create Joins on the Data Foundation

Business Example

After you have inserted your tables in your data foundation, you can set the joins you require. In this exercise, you set equi and theta joins.

> Note:
> To complete the exercises, replace the value *xx* with the group number the instructor provided you with at the beginning of the course.

Create Equi-Joins

In the data foundation, define the equi-joins you require to structure the data.

1. Insert the equi-joins listed in the following table, set all cardinalities, and save your data foundation.

Join	Type	Cardinality
CLIENT.CLIENT_ID = SALE.CLIENT_ID	Equi	1:N
SHOWROOM.SHOW-ROOM_ID = SALE.SHOW-ROOM_ID	Equi	1:N
SALE_MODEL.MODEL_ID = MODEL.MODEL_ID	Equi	N:1
COUNTRY.COUNTRY_ID = REGION.COUNTRY_ID	Equi	1:N
REGION.REGION_ID = CLI-ENT.REGION_ID	Equi	1:N
MODEL.STYLE_ID = STYLE.STYLE_ID	Equi	N:1
MODEL.MAKER_ID = MAK-ER.MAKER_ID	Equi	N:1
SALE_MODEL.COLOR_ID = COLOR.COLOR_ID	Equi	N:1

Create Theta-Joins

In the data foundation, create the equi-joins you require to structure the data.

1. In the data foundation, define the theta joins.

Insert joins from the following table, set all cardinalities, and save your data foundation.

Join	Type	Cardinality
SALE.SALE_DATE between FINANCE_PERIOD.FP_START and FINANCE_PERIOD.FP_END	Theta	N:1
MODEL.MODEL_PRICE between SALES_PRICE_RANGE.PRICE_RANGE_MIN and SALES_PRICE_RANGE.PRICE_RANGE_MAX	Theta	N:1

Insert a Column Filter

To manage the sales types displayed, insert a filter to display all sales types with a value of 'S;.

1. In the Data Foundation, define a filter. Insert the filter listed in the following table, set its cardinalities, and save your data foundation.

Filter	Type
To set the SALE.SALE_TYPE ='S' column filter, in the data foundation view, right-click the column SALE.SALE_TYPE and choose *Insert Filter*.	Column

Unit 3
Solution 4

Create Joins on the Data Foundation

Business Example

After you have inserted your tables in your data foundation, you can set the joins you require, In this exercise, you set equi and theta joins.

Note:
To complete the exercises, replace the value *xx* with the group number the instructor provided you with at the beginning of the course.

Create Equi-Joins

In the data foundation, define the equi-joins you require to structure the data.

1. Insert the equi-joins listed in the following table, set all cardinalities, and save your data foundation.

Join	Type	Cardinality
CLIENT.CLIENT_ID = SALE.CLIENT_ID	Equi	1:N
SHOWROOM.SHOW-ROOM_ID = SALE.SHOW-ROOM_ID	Equi	1:N
SALE_MODEL.MODEL_ID = MODEL.MODEL_ID	Equi	N:1
COUNTRY.COUNTRY_ID = REGION.COUNTRY_ID	Equi	1:N
REGION.REGION_ID = CLI-ENT.REGION_ID	Equi	1:N
MODEL.STYLE_ID = STYLE.STYLE_ID	Equi	N:1
MODEL.MAKER_ID = MAK-ER.MAKER_ID	Equi	N:1
SALE_MODEL.COLOR_ID = COLOR.COLOR_ID	Equi	N:1

a) In the data foundation view of the Motors_xx data foundation, from the *Insert* menu choose the *Insert Join* command.
The *Edit Join* dialog box appears.

b) In the *Edit Join* dialog box define the join properties by clicking on the table and column of the first table, and the table and column of the second table.
You can use this method for all the joins that have a Table1.Column = Table2.Column syntax.

c) Choose *Detect* and choose *OK*.

Create Theta-Joins

In the data foundation, create the equi-joins you require to structure the data.

1. In the data foundation, define the theta joins.
   Insert joins from the following table, set all cardinalities, and save your data foundation.

Join	Type	Cardinality
SALE.SALE_DATE between FINANCE_PERIOD.FP_START and FINANCE_PERIOD.FP_END	Theta	N:1
MODEL.MODEL_PRICE between SALES_PRICE_RANGE.PRICE_RANGE_MIN and SALES_PRICE_RANGE.PRICE_RANGE_MAX	Theta	N:1

   a) To define the SALE.SALE_DATE between FINANCE_PERIOD.FP_START and FINANCE_PERIOD.FP_END theta join, in the *Edit Join* dialog box, click on the column SALE.SALE_DATE.

   b) From the *Table2* drop-down list, choose the table FINANCE_PERIOD.

   c) Press and hold down the Ctrl key and in the *Table2* column list box, choose the two columns FP_START and FP_END.

   d) Repeat steps a-c for the join MODEL.MODEL_PRICE between SALES_PRICE_RANGE.PRICE_RANGE_MIN and SALES_PRICE_RANGE.PRICE_RANGE_MAX.

Insert a Column Filter

To manage the sales types displayed, insert a filter to display all sales types with a value of 'S;.

1. In the Data Foundation, define a filter. Insert the filter listed in the following table, set its cardinalities, and save your data foundation.

Filter	Type
To set the SALE.SALE_TYPE ='S' column filter, in the data foundation view, right-click the column SALE.SALE_TYPE and choose *Insert Filter*.	Column

   a) To set the column filter, in the data foundation view, right-click the column SALE.SALE_TYPE and choose *Insert Filter*.

The *Edit Join* dialog appears.

b) In the *Edit Join* dialog, replace SALE.SALE_TYPE after the = sign with 'S'.

c) Choose *Save*.

LESSON SUMMARY

You should now be able to:

- Use joins
- Create an equi-join
- Create an outer join
- Create a theta join
- Create column filters to restrict data

Learning Assessment

1. What does the Data Foundation browsing pane of the Data Foundation Editor show?

   Choose the correct answers.

   - ☐ A Business Layer architecture
   - ☐ B List of users
   - ☐ C Tree view of the tables and joins
   - ☐ D Schema architecture

2. What is a join?

   _____

   _____

   _____

3. What is the term for the result of a query that was run on two tables that are not joined?

   Choose the correct answer.

   - ☐ A Caesarean
   - ☐ B Cardinality
   - ☐ C Cartesian
   - ☐ D Unjoined

4. Multiple filter names may be applied to each column.

   Determine whether this statement is true or false.

   - ☐ True
   - ☐ False

5. What is an equi-join?

   _____

   _____

   _____

6. What is an outer join?

7. What does a theta join do?

Choose the correct answer.

- ☐ A Demonstrate ranges
- ☐ B Demonstrate parallels
- ☐ C Demonstrate report filters
- ☐ D Demonstrate table inconsistencies

Learning Assessment - Answers

1. What does the Data Foundation browsing pane of the Data Foundation Editor show?

   Choose the correct answers.

   - [ ] A Business Layer architecture
   - [ ] B List of users
   - [x] C Tree view of the tables and joins
   - [ ] D Schema architecture

2. What is a join?

   A join is a condition that links tables in the data foundation. A join restricts the data returned when the two tables are queried.

3. What is the term for the result of a query that was run on two tables that are not joined?

   Choose the correct answer.

   - [ ] A Caesarean
   - [ ] B Cardinality
   - [x] C Cartesian
   - [ ] D Unjoined

4. Multiple filter names may be applied to each column.

   Determine whether this statement is true or false.

   - [ ] True
   - [x] False

5. What is an equi-join?

   An equi-join is a join that is based on column values between two tables.

6. What is an outer join?

An outer join is a join that links two tables, one that has rows that may not match those rows in the common column of the other table.

7. What does a theta join do?

Choose the correct answer.

- [X] A Demonstrate ranges
- [ ] B Demonstrate parallels
- [ ] C Demonstrate report filters
- [ ] D Demonstrate table inconsistencies

UNIT 4 Business Layers

UNIT OBJECTIVES

- Access business layer data
- Create a business layer
- Populate the business layer
- Add business layer folders
- Create business layer dimension objects
- Create time dimension objects
- Create attribute objects

- Verify business object accuracy and integrity
- Create a measure object
- Create a delegated measure object
- Create shortcut joins

Unit 4
Lesson 1

Accessing Data through the Business Layer

LESSON OVERVIEW
The role of the Business Layer is to present a business-focused front end to the SQL structures in the database. The data used in a universe schema depends greatly on the end-user requirements.

LESSON OBJECTIVES
After completing this lesson, you will be able to:

- Access business layer data
- Create a business layer
- Populate the business layer

Business Layer

A business layer is a collection of metadata objects that map to SQL or MDX definitions in a database, for example, columns, views, database functions, or preaggregated calculations. The metadata objects include dimensions, navigation paths, measures, attributes, and predefined conditions. Each object corresponds to a unit of business information that can be manipulated in a query to return data. Business layers can be created directly on an OLAP cube, or on a data foundation that is built on a relational database.

When a business layer is complete, it is published to the Central Management Server as a universe. A universe is a published .unx file that includes a business layer and its connection to an OLAP cube, or a business layer and its corresponding data foundation. The universe is available in the repository to SAP BusinessObjects data analysis and report creation applications.

The principle role of the business layer is to define and organize metadata before it is published as a universe. Think of it as a metadata workbench that a designer uses to assemble and modify a metadata set before publishing as a universe for data analysis and report creation applications.

Use the *New Business Layer* wizard to create a business layer based on a data foundation or an OLAP cube. Before you can create a business layer, you need the following:

- A project folder in the Local Projects View
- If you are basing the business layer on a data foundation, a data foundation saved in the same project folder
- If you are basing the business layer on an OLAP cube, an OLAP connection or connection shortcut saved in the same project folder

Business Layer Editor

You use the Business Layer Editor to create and edit business layer objects and properties.

The Business Layer Editor is divided into browsing panes on the left, an editing pane on the upper right, and a data source pane on the lower right.

The browsing panes allow you to work with different elements of the business layer. Access the panes by clicking the corresponding tab:

- Business Layer
- Queries
- Parameters and Lists of values
- Navigation Paths

The Business Layer is the default browsing pane. It displays a tree view of the objects in the business layer. The following options are available for displaying and navigating the business layer tree view:

- Filter by business view
- Search for an object
- Change display options: Show or hide objects, show technical names

The editing pane lets you edit the properties of the object or element selected in the browsing pane.

The data source pane displays the data foundation or the OLAP connection information:

- The data foundation master view containing all tables and joins displays by default. Tabs for other data foundation views, if defined, appear at the bottom of the data source pane. To change views, click the tab.
- The OLAP metadata in the connection displays in the left side of the data source pane. Select a metadata object to display its properties in the right side of the pane.

Business Layer Properties

The following properties and restrictions are defined for the entire business layer. The restrictions are applied in the published universe.

Table 9: Business Layer Properties and Restrictions

Property	Restriction	Description
Name		Identifies the business layer and the universe when the business layer is published.
Description		Description of the universe purpose and content. This description is available to display in the query and reporting tools that use the published universe.

Property	Restriction	Description
Query Limits	Limit result set size to	Specifies the number of rows that are returned in a query. This property limits the number of rows returned, but does not restrict the RDBMS from processing all the rows in the query. This property only limits the number once the RDBMS has started to send rows.
	Limit execution time to	Specifies the number of minutes to limit the time passed for query execution, but does not stop the process on the database.
	Warn if estimate exceeds	Specifies to send a message when an estimate of the execution time exceeds the specified number of minutes.
Query (apply to business layers based on data foundations)	Allow use of subqueries	Permits subqueries in a query.
	Allow use of union, intersect, and minus operators	Enables universe user to combine queries using the data set operators union, intersect, and minus to obtain one set of results.
	Allow complex operands in Query Panel	Allows complex operands in the list of operands available when defining a filter in the *Query Panel*.
Summary		Displays the number of each type of object defined in the business layer. Also displays the number of data foundation objects defined in the underlying data source.
Change Data Foundation	Data Source	Specifies the data source for the business layer; either a data foundation or OLAP connection. For OLAP data sources, other properties are defined.

Property	Restriction	Description
Parameters (apply to business layers based on data foundations)		Specifies custom values for SQL generation parameters that override the default value or any customized value in the data foundation properties.
Comments		Comments about the business layer.

Business Layer Objects

The Business Layer objects pane contains the metadata objects you use to build the business layer.

Depending on the type of data source for the business layer, you can create and edit the following types of objects in a business layer:

- Dimensions
- Measures
- Navigation paths (OLAP only)
- Analysis dimensions (OLAP only)
- Attributes
- Filters
- Named member sets (OLAP only)
- Calculated members (OLAP only)
- Folders

In BusinessObjects products, an object is a named component in a universe that represents a column or function in a database.

In the Information Design Tool, objects appear as icons in the business layer pane. Each object represents a meaningful entity, fact, or calculation used in an end-user's business environment. The objects that you create in the business layer pane are the objects that end users see and use in the Business Objects end-user reporting tools.

In a business layer, you can qualify an object as being one of three types:

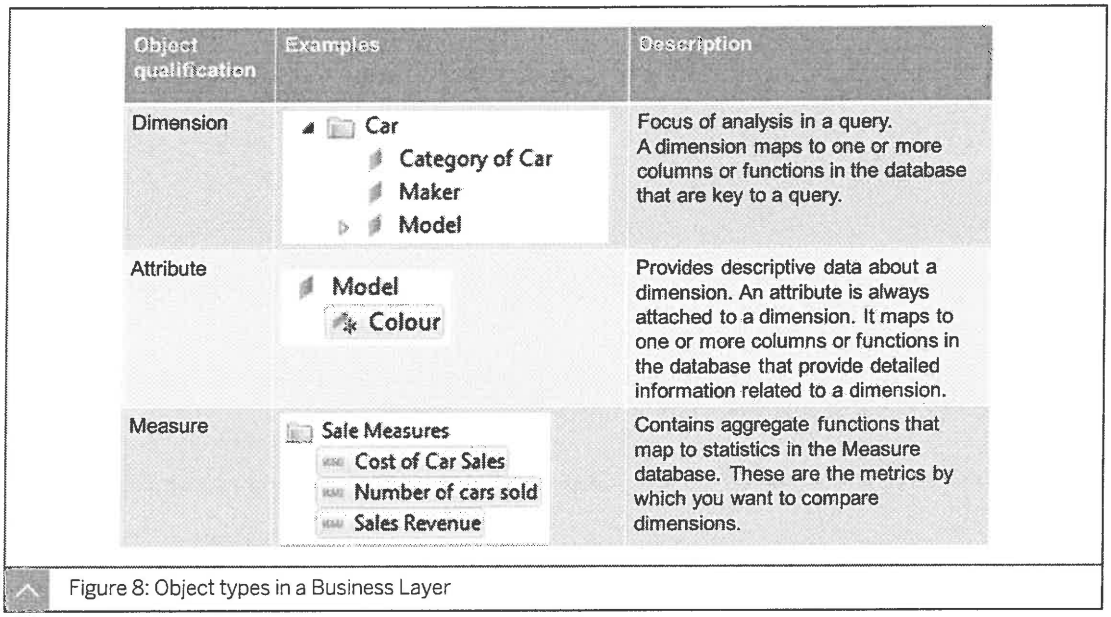


Figure 8: Object types in a Business Layer

Each object in the business layer has properties that can be defined and modified at any time. The properties that you set for objects in the business layer are applied in the published universe.

Table 10: Business Layer Object Properties

Property	Definition
Name	The name of the object. Choose names the correspond to the query and data analysis culture of the target user profile.
Description	Comment describing the object.
Active/Hidden/ Deprecated	**Active**: Object is visible in the Query Panel **Hidden**: Object is valid but not available in the Query Panel. **Deprecated**: Object is hidden and not valid; for example, when the target database field no longer exists but you want to keep the object for possible future use.

LESSON SUMMARY

You should now be able to:

- Access business layer data
- Create a business layer
- Populate the business layer

Unit 4
Lesson 2

Integrating the Business Layer Components

LESSON OVERVIEW
This lesson explains how to indentify and integrate business layer components — folders and objects — to provide an easy-to-use interface for end users.

LESSON OBJECTIVES
After completing this lesson, you will be able to:

- Add business layer folders
- Create business layer dimension objects
- Create time dimension objects
- Create attribute objects

Business Layer Folders

Within a universe, objects are grouped into folders. This grouping is done to provide a structure for the universe and makes it easier for users to locate particular objects. A strategy that is frequently employed is to group related dimension and detail objects into one folder and place measure objects into a unique and single-measures folder.

This strategy can be extended by introducing subfolders to break down the objects into further subsets.

Figure 9: Folders and Subfolders

Each object in a business layer is contained within a folder. You can create new folders and edit the properties of existing folders. Folders are represented as folders on a tree hierarchy in the Universe pane.

Business Layer Dimension Objects

Dimension objects, where possible, tend to be organized hierarchically within a folder. This organization is important if you intend to use default hierarchies for drilling in the reporting tools. Attribute objects are organized below their associated dimension objects.

For example, in the Web Intelligence Rich Client Query Panel, users drag objects from the Data tab into the Result Objects pane to run queries and create reports that display the data returned by the query.

Each object maps to a column or function in the target database, and when an object is selected in the Query Panel, the object infers a SELECT statement. When multiple objects are

combined, a SELECT statement is run on the database which includes the SQL inferred by each object and a default WHERE clause.

You can create objects for use only in the Information Design Tool, so that they are hidden in the Business Objects end-user querying tools.

SQL Definition

The SQL definition tab in business layer object properties lets you define the SQL statement for the selected object. You can enter the following properties:

Table 11: SQL Definition Properties

Property	Description
Select:	Lets you enter the SELECT statement directly in the text box> You can also click the *SQL Assistant* button and use the SQL editor to build the statement.
Where:	Lets you enter the WHERE statement directly in the text box> You can also click the *SQL Assistant* button and use the SQL editor to build the statement
Extra Tables:	Lets you select tables related to the object to include in the query at runtime . Click the button at the end of the text field to open a list of related tables. Select or clear the extra tables.

Checking object integrity

Always check the integrity of your business layer after defining folders and objects.

Use the *Check Integrity* option to detect any errors in the SQL syntaxes used in the created objects.

Note:
Ensure that the *Check Cardinalities* box is cleared. With a large database, not checking cardinalities saves time running the integrity check.

Keys Properties

On the *Keys* tab in business layer object properties, you specify for a dimension object which database columns are primary and foreign keys. This specification allows the query to take advantage of indexes on key columns. Defining keys speeds up data retrieval by optimizing the SQL that is generated for the query.

For example, in a star schema database if you build a query that filters on a value in a dimension table, the filter can be applied directly on the fact table by using the dimension table foreign key. This filter eliminates unnecessary and inefficient joins to dimension tables. Key awareness is discussed in detail later in this course.

Hint:
Defining keys is only available for dimensions built on a data foundation.

Advanced Properties

The *Advanced* tab in business layer object properties lets you define advanced properties for the selected object. You can enter the following properties:

Table 12: Business Layer Object: Advanced Properties

Property	Description
Access level	Defines the security access level of the object. You can select a security level that restricts use of the object to end users with the appropriate security level.
Object can be used in Result	When selected, the object can be used in a query.
Object can be used in Condition	When selected, the object can be used in the condition of a query.
Object can be used in Sort	When selected, return values can be sorted.
Database format	Available on 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 the date format to use the target database format for storing dates. For example, the date format can be US format or European format.
List of Values	Lets you associate a list of values (LOV) to the object. The LOV applies when defining a filter on the object in the *Query Panel*.
Display	Lets you set display options for the data returned by the object in a query. Click *Edit display format* to select a predefined format or define a custom format. You can also select to display the data returned by the object as HTML or Hyperlink.

The *Advanced* tab allows you to set the security access level of the object, how it can be used in a query, options for the list of values and the formatting of the object. You can select a security level that restricts use of the object to users with the appropriate security level.

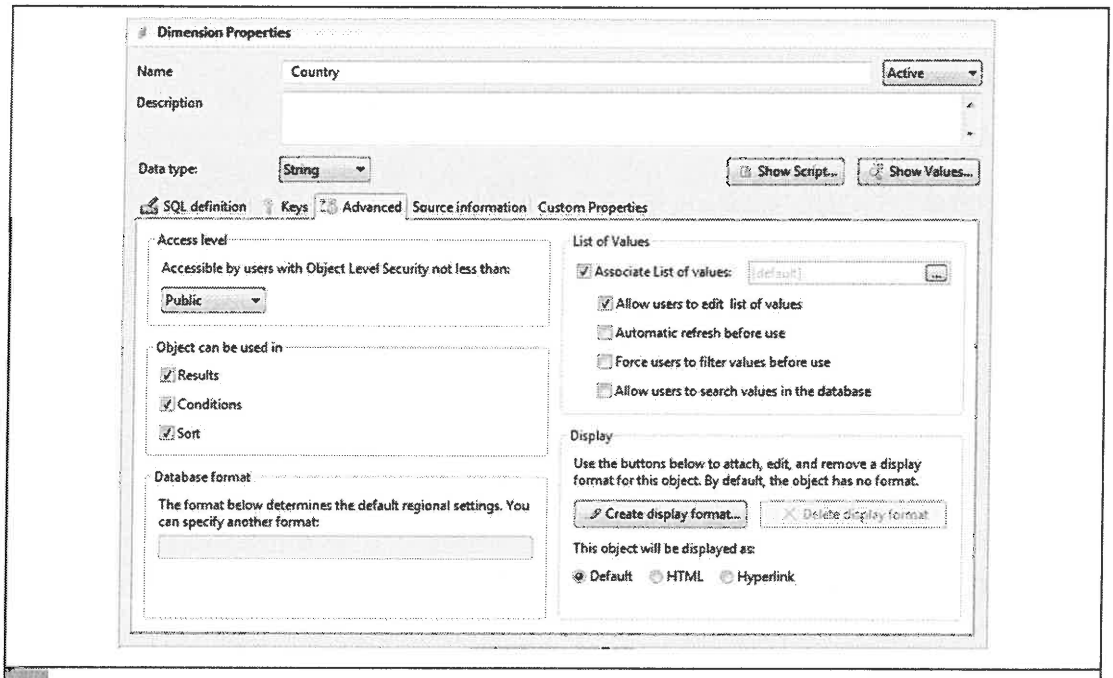


Figure 10: Edit Dimension Properties: Advanced

In the Security Access Level area of the Advanced properties, 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. The user profile is set by the SAP BusinessObjects administrator in the SAP BusinessObjects repository.

If the Object can be used in the Advanced properties area, select one of the following options to define how this object can be used in a query:

- The Result check box – use this object to return results in a query.
- The Condition check box – use this object to apply a condition or query filter in a query.
- The Sort check box – specify the object in the ORDER BY clause of a SELECT statement.

Hint:
This option can increase the processing speed of a query. However, in certain edited List of values (LOV) situations, it is not useful to sort at query level, because block-level sorting in the query tools overrides any row order of data that is stored in the microcube.

In the List of Values area of the Advanced properties, you define list of values (LOV) query options.

A list of values (LOV) is a list that contains the data values associated with an object. A LOV allows a user to chose values as a response to a prompt when an associated object is included in a query. The LOV allows a data set to be restricted to the selected values. A LOV is an independent component in the business layer or data foundation and is available to all business objects in the business layer. A LOV can be associated with an object at any time. Only use a list of values with an object if it provides something useful for the user. LOVs are useful where there is a limited set of distinct values for the database columns underlying the object.

Hint:
If the object is likely to refer to many distinct rows in the database, it is advisable not to associate an LOV or change its configuration to avoid performance issues.

Source Information Properties

The *Source Information* tab in business layer object properties contains descriptive fields that only apply to objects used by Data Services.

Table 13: Business Layer Object: Source Information Properties

Property	Description
Technical Information	Displays information about a column, for example, the original database name of the concerned column for the object.
Mapping	Displays initial formula information describing how a column has been specified (used in Data Integrator), for example revenue = column calculated from several sources.
Lineage	Displays source columns for the formula used to calculate the column in the database.

Pre-Formatted Objects

All objects have a default display format that is based on the object type. This format can be changed by the universe designer so that end users do not need to change the format in their respective application (for example: Web Intelligence, Crystal Reports for Enterprise, or SAP BusinessObjects Dashboards).

You can change the default display format through the object's *Advanced* tab. Click the *Create Display Format* button, and choose an appropriate format from the list of predefined formats. You can also create a custom format by clicking the *Customer Format* button and defining the format.

Unit 4
Exercise 5

Create and Populate the Business Layer

Business Example

Create and populate the business layer.

> Note:
> To complete the exercise, replace the xx with the group number that the instructor provided you with at the beginning of the class.

Create the Business Layer

To create folders and objects, first create a business layer.

1. Create a business layer in your project folder Motors_xx. Since there are many objects to rename, many subfolders in your business layer, foreign key columns in your tables, and some concatenations in your objects, do not create folders and objects automatically.

Create Folders and Subfolders

Create the folders and subfolders.

1. Create the following folders and subfolders:
   - Sales
   - Client
   - Showroom
   - Car
   - Finance Period
   - Sale Prices (subfolder of Car)
   - Sales Details (subfolder of Sales)
   - Sales Dates (subfolder of Sales)

Create Objects for the Subfolders

After you create the folders and subfolders, you can create the objects.

1. Create objects for each of the subfolders as identified in the tables. Some of the properties for each object have been specified for you. Qualify the attributes as indicated (dimension or attribute).

   The following table lists the objects for the Client folder:

Object Name	SELECT Statement	Object Description
Country	COUNTRY.COUN-TRY_NAME	Country in which client re-sides

Object Name	SELECT Statement	Object Description
Region	REGION.REGION_NAME	Region of country in which client resides
Area	CLIENT.CLIENT_AREA	Area of region in which client resides (for example, county or state)
Client Town	CLIENT.CLIENT_TOWN	Town or city in which client resides
Client Name	CLIENT.CLIENT_LASTNAME + ',' + CLIENT.CLIENT_FIRSTNAME	Last name, first name

The following table lists the objects for the Car folder:

Object Name	SELECT Statement	Object Description
Maker	MAKER.MAKER_NAME	Car manufacturer
Category of car	STYLE.STYLE_NAME	The style group into which a car fits (for example, coupe or 4x4)
Model	MODEL. MODEL_NAME +' ' + MODEL. MODEL_TRIM +' ' + MODEL. MODEL_ENGINE	Model name, trim, and engine size

The following table lists the objects for the Showroom folder:

Object Name	SELECT Statement	Object Description
Showroom town	SHOWROOM.SHOWRROM_TOWN	Town in which showroom exists
Showroom	SHOWROOM.SHOWRROM_NAME	Name of showroom

The following table lists the objects for the Financial Period folder:

Object Name	SELECT Statement	Object Description
Financial year	FINANCE_PERIOD.FP_YEAR	For example, FY 03-04
Financial quarter	FINANCE_PERIOD.FP_QUARTER	For example, Q1
Financial month	FINANCE_PERIOD.FP_MONTH	For example, Month 01

2. Create objects for each of the subfolder as identified in the following tables. Some of the properties for each object have been specified for you.

   The following table lists the objects for the `Sales Prices` subfolder:

Object Name	SELECT Statement	Object Description
Price range	SALES_PRICE_RANGE.PRICE_RANGE	Description of price range banding
Model price	MODEL.MODEL_PRICE	Manufacturer recommended retail price

The following table lists the objects for the `Sales Details` subfolder:

Object Name	SELECT Statement	Object Description
Invoice ID number	SALE.SALE_ID	Unique invoice ID number

The following table lists the objects for the `Sales Dates` subfolder:

Object Name	SELECT Statement	Object Description
Sale date	SALE.SALE_DATE	Date of sale

The Business Layer pane should appear similar to the following folder structure:

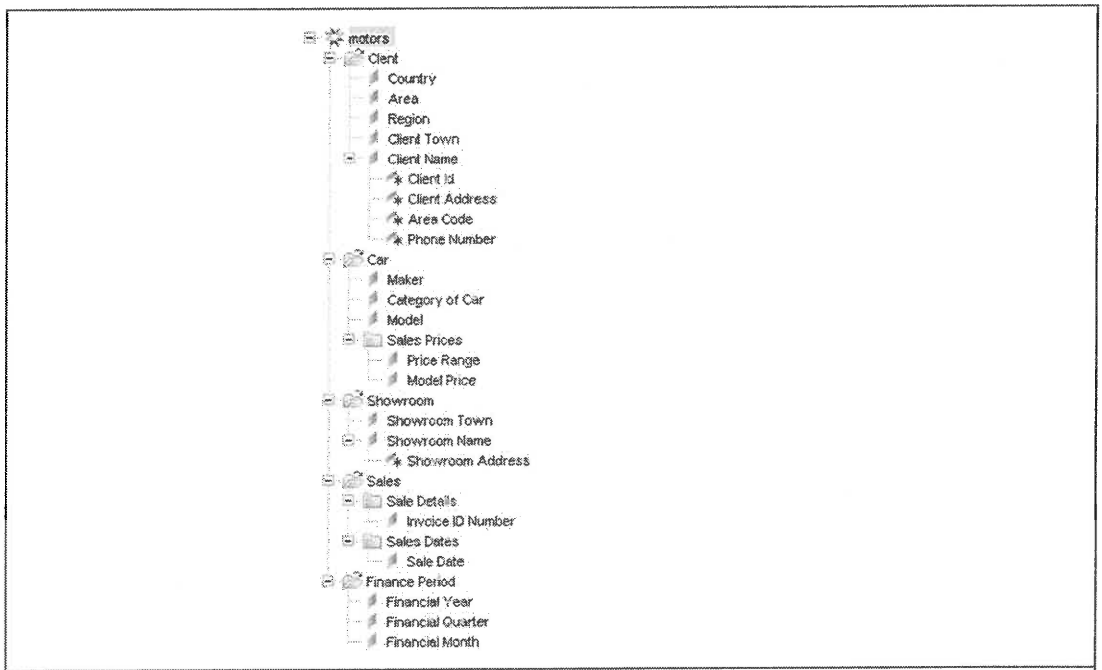


Figure 11: Business Layer all folders and Objects

Unit 4
Solution 5

Create and Populate the Business Layer

Business Example

Create and populate the business layer.

> Note:
> To complete the exercise, replace the xx with the group number that the instructor provided you with at the beginning of the class.

Create the Business Layer

To create folders and objects, first create a business layer.

1. Create a business layer in your project folder Motors_xx. Since there are many objects to rename, many subfolders in your business layer, foreign key columns in your tables, and some concatenations in your objects, do not create folders and objects automatically.
   a) In the *Local Projects View*, choose your Motors_xx project folder.
   b) Choose *File* → *New* → *Business Layer*.
   The *New Business Layer* wizard opens.
   c) Since the data foundation will use a relational data source, choose *Relational Data Source* and choose *Next*.
   d) As the name of your business layer, enter Motors_xx, where xx is the student number your instructor assigned to you.
   e) Choose *Next*.
   f) On the right in the *Data Foundation Selection* dialog, choose the ellipsis (...) button to choose Motors_xx.dfx.
   g) Choose *OK*.
   h) Uncheck *Automatically create folders and objects* and choose *Finish*.

Create Folders and Subfolders

Create the folders and subfolders.

1. Create the following folders and subfolders:
   - Sales
   - Client
   - Showroom
   - Car

- Finance Period
- Sale Prices (subfolder of Car)
- Sales Details (subfolder of Sales)
- Sales Dates (subfolder of Sales)

a) At the top of the *Business Layer* pane, choose the *Insert object* icon and choose the *Insert Folder* icon.

b) In the *Folder Properties* pane, in the *Name* property, enter `Client`.

c) Repeat steps a and b for the next 4 folders.

d) Click on the *Car* folder.

e) At the top of the *Business Layer* pane, choose the *Insert Object* icon and choose the *Insert Folder* icon.

f) In the *Folder Properties* pane, in the *Name* property, enter `Sales Prices`.

g) After choosing the appropriate parent folder (in parentheses in the preceding bullet list) repeat steps d through f for the remaining two subfolders.

Create Objects for the Subfolders

After you create the folders and subfolders, you can create the objects.

1. Create objects for each of the subfolders as identified in the tables. Some of the properties for each object have been specified for you. Qualify the attributes as indicated (dimension or attribute).

   The following table lists the objects for the Client folder:

Object Name	SELECT Statement	Object Description
Country	COUNTRY.COUNTRY_NAME	Country in which client resides
Region	REGION.REGION_NAME	Region of country in which client resides
Area	CLIENT.CLIENT_AREA	Area of region in which client resides (for example, county or state)
Client Town	CLIENT.CLIENT_TOWN	Town or city in which client resides
Client Name	CLIENT.CLIENT_LASTNAME + ',' + CLIENT.CLIENT_FIRSTNAME	Last name, first name

   The following table lists the objects for the Car folder:

Object Name	SELECT Statement	Object Description
Maker	MAKER.MAKER_NAME	Car manufacturer

Object Name	SELECT Statement	Object Description
Category of car	STYLE.STYLE_NAME	The style group into which a car fits (for example, coupe or 4x4)
Model	MODEL. MODEL_NAME +' ' + MODEL. MODEL_TRIM +' ' + MODEL. MODEL_ENGINE	Model name, trim, and engine size

The following table lists the objects for the Showroom folder:

Object Name	SELECT Statement	Object Description
Showroom town	SHOWROOM.SHOWRROM_TOWN	Town in which showroom exists
Showroom	SHOWROOM.SHOWRROM_NAME	Name of showroom

The following table lists the objects for the Financial Period folder:

Object Name	SELECT Statement	Object Description
Financial year	FINANCE_PERIOD.FP_YEAR	For example, FY 03-04
Financial quarter	FINANCE_PERIOD.FP_QUARTER	For example, Q1
Financial month	FINANCE_PERIOD.FP_MONTH	For example, Month 01

a) For each object in the `Client` folder, choose *New* → *Dimension*, or simply drag the appropriate column from the table indicated into the folder.

b) For each object in the `Car` folder, choose *New* → *Dimension*, or simply drag the appropriate column from the table indicated into the folder.

c) For each object in the `Showroom` folder, choose *New* → *Dimension*, or simply drag the appropriate column from the table indicated into the folder.

d) For each object in the `Financial Period` folder, choose *New* → *Dimension*, or simply drag the appropriate column from the table indicated into the folder.

2. Create objects for each of the subfolder as identified in the following tables. Some of the properties for each object have been specified for you.

The following table lists the objects for the `Sales Prices` subfolder:

Object Name	SELECT Statement	Object Description
Price range	SALES_PRICE_RANGE.PRICE_RANGE	Description of price range banding
Model price	MODEL.MODEL_PRICE	Manufacturer recommended retail price

The following table lists the objects for the `Sales Details` subfolder:

Object Name	SELECT Statement	Object Description
Invoice ID number	SALE.SALE_ID	Unique invoice ID number

The following table lists the objects for the `Sales Dates` subfolder:

Object Name	SELECT Statement	Object Description
Sale date	SALE.SALE_DATE	Date of sale

a) For each object in the `Sales Prices` subfolder folder, choose *New* → *Object*, or simply drag the appropriate column from the table indicated into the folder.

b) For each object in the `Sales Details` subfolder folder, choose *New* → *Object*, or simply drag the appropriate column from the table indicated into the folder.

c) For each object in the `Sales Dates` subfolder folder, choose *New* → *Object*, or simply drag the appropriate column from the table indicated into the folder.

The Business Layer pane should appear similar to the following folder structure:

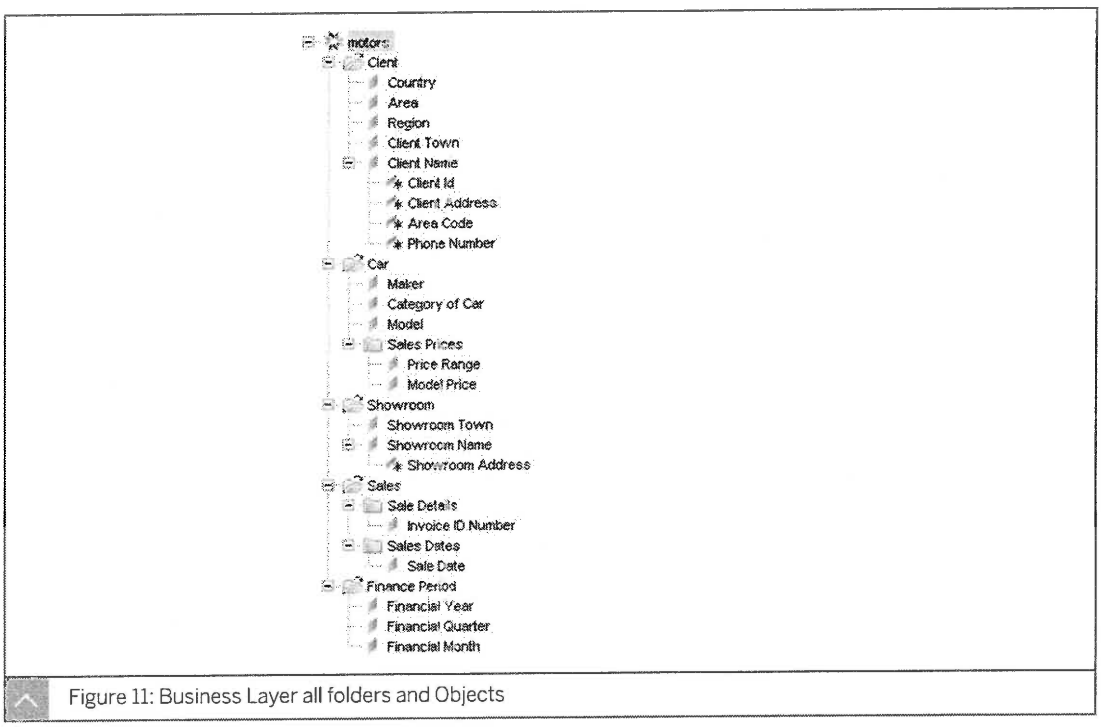


Figure 11: Business Layer all folders and Objects

Time Dimension Objects

One of the most common requirements for data analysis is to provide facilities for analysis against different time periods. Time is a special case because all the information for the dimension objects can come from a single column in a database. In most other cases, each dimension object infers a separate database column.

Creating time dimension objects is achieved by using SQL date scalar functions to extract the day, month, year, and possibly quarter from a single database column of a date type.You create the standard time structure by ordering the objects Year, Quarter, Month, and Week.

Note:
The scalar functions used to extract elements of a database varies with each RDBMS.

To create a time object using database functions:

1. Create a new folder or subfolder with an appropriate name. Repeat the remainder of this procedure to create each time object required within the folder that is based on the date scalar functions.
2. Drag the database column from the Structure pane table that contains the date required, and drop it on the newly created folder.
3. Double click the object to open the *Edit Properties* dialog box.
4. Edit the properties of the object so that it will infer an element of the date as required.
   - Change the object name to reflect the scalar function used.
   - Change the data type if required, depending on the scalar function used.
   - Change the SELECT statement to the relevant scalar function, string conversation, and required string value, depending on the RDBMS used.
5. Click *OK* to close the *Edit Properties* dialog box.
6. Repeat steps 2 through 5 for each of the other time objects required within the folder, based on scalar functions.

Hint:
Since you are creating a similar time object to the previous one, it is more efficient to edit a copy of the previous object, rather than creating a new one as suggested in step 2.

The *Universe* pane now contains a folder or subfolder with multiple time related objects that are based on a single date column of a database table.

Unit 4
Exercise 6

Create Time Dimension Objects

Business Example

Create time dimension objects.

1. Create the following dimension objects manually in the `Sales Dates` subfolder, using alphanumeric database scalar functions and formatting: Sales Year, Sales Quarter, and Sales Month.

Hint:
Since you are creating a similar time object to the previous one, it can be more efficient to edit a copy of the previous object, rather than creating a new one as suggested in step 2.

Unit 4
Solution 6

Create Time Dimension Objects

Business Example

Create time dimension objects.

1. Create the following dimension objects manually in the `Sales Dates` subfolder, using alphanumeric database scalar functions and formatting: Sales Year, Sales Quarter, and Sales Month.

> Hint:
> Since you are creating a similar time object to the previous one, it can be more efficient to edit a copy of the previous object, rather than creating a new one as suggested in step 2.

a) In the business layer, drag the SALE_DATE column from the SALE table in the data foundation pane within the business layer and drop it in the Sales Dates folder.

> Note:
> You cannot have two objects with the same name in the same folder. We will change the new object's name later.

b) Highlight the object to edit its properties.

c) Edit the SELECT statement to the relevant scalar function, string conversion, and required concatenated string value, depending on the RDBMS used:

- Sales Year:

```
'Calendar Year ' + datename(YYYY,SALE.SALE_DATE)
```

d) Change the object name to **`Sales Year`**.

e) Change the data type if required, depending on the scalar function used.

f) For each of the other time objects required within the folder based on scalar functions, repeat steps a to e.

- Sales Quarter:

```
'Q ' + datename(Q,SALE.SALE_DATE)
```

- Sales Month:

```
datename(mm,SALE.SALE_DATE)
```

g) Test the new objects by running a few queries using them and other objects in the *Query* drawer.

Attribute Objects

Unit 4
Exercise 7

Create Attribute Objects

Business Example

After you create the objects, you can create the attributes required for existing objects and populate the subfolders with attribute objects.

1. Create the following objects as attribute objects of the Client Name object in the `Client` folder:

Attribute Name	SELECT Statement	Attribute Description
Client Address	CLIENT.CLIENT_ADDRESS	Address of client
Area Code	CLIENT.CLI- ENT_AREA_CODE	Postal or zip code
Phone Number	CLIENT.CLI- ENT_PHONE_NO	Phone number of client
Client ID	CLIENT.CLIENT_ID	Unique client ID number

2. Create the following object as an attribute object of the Showroom object in the Showroom folder:

Attribute Name	SELECT Statement	Attribute Description
Showroom address	SHOWROOM.SHOW- RROM_ADDRESS	Address of showroom

Unit 4
Solution 7

Create Attribute Objects

Business Example

After you create the objects, you can create the attributes required for existing objects and populate the subfolders with attribute objects.

1. Create the following objects as attribute objects of the Client Name object in the `Client` folder:

Attribute Name	SELECT Statement	Attribute Description
Client Address	CLIENT.CLIENT_ADDRESS	Address of client
Area Code	CLIENT.CLI- ENT_AREA_CODE	Postal or zip code
Phone Number	CLIENT.CLI- ENT_PHONE_NO	Phone number of client
Client ID	CLIENT.CLIENT_ID	Unique client ID number

   a) For the client address, area code, phone number, and client ID, right-click the client name dimension and choose *New* → *Attribute*.

2. Create the following object as an attribute object of the Showroom object in the Showroom folder:

Attribute Name	SELECT Statement	Attribute Description
Showroom address	SHOWROOM.SHOW- RROM_ADDRESS	Address of showroom

   a) For the showroom address attribute, right-click the showroom name dimension and choose *New* → *Attribute*.

LESSON SUMMARY

You should now be able to:

- Add business layer folders
- Create business layer dimension objects
- Create time dimension objects
- Create attribute objects

Unit 4
Lesson 3

Validating Objects

LESSON OVERVIEW
After defining folders and objects, you need to check the integrity of your business layer. This lesson describes how to validate the accuracy and integrity of business objects.

LESSON OBJECTIVES
After completing this lesson, you will be able to:

- Verify business object accuracy and integrity

Business Object Integrity Check

Always check the integrity of your business layer after defining folders and objects.

Use the *Check Integrity* option to detect any errors in the SQL syntaxes used in the created objects.

Note:
Ensure that the *Check Cardinalities* box is cleared. With a large database, not checking cardinalities saves time running the integrity check.

Object Testing

When you create objects in the business layer, test them in Business Objects end-user querying tools or by building and running queries with the Information Design Tool. There are three things to test:

- Do the objects exist?

  If not, you may have forgotten to save your universe since the object you are testing was created.
- Does the SQL appear correct?
- Are the results of the query correct?

Note:
Remember that you must also test the joins already created in the structure.

Unit 4
Exercise 8

Test Objects

Business Example

After you create objects, and ensure that they are working as expected, you verify and test them.

Verify Business Objects

To ensure that your objects are sound, verify their integrity,

1. Check the integrity of the objects, and alter anything required.

> Note:
> Test the validity of the joins also.

Save Your Business Layer

Now it is time to save your business layer.

1. Save your business layer.

Test Objects

Finally, test the objects to make sure that they are working as expected.

1. Test a few objects in the *Query* drawer.
   View the SQL and run queries to make sure that the correct data is retrieved.

Unit 4
Solution 8

Test Objects

Business Example

After you create objects, and ensure that they are working as expected, you verify and test them.

Verify Business Objects

To ensure that your objects are sound, verify their integrity,

1. Check the integrity of the objects, and alter anything required.

> Note:
> Test the validity of the joins also.

   a) Right-click on your business layer, and choose *Check Integrity*.

   b) In the *Check Integrity* dialog, choose *Tables, Joins, and Business Layer*.

   c) Choose *Check Integrity*.

   d) To exit, choose *OK*.

Save Your Business Layer

Now it is time to save your business layer.

1. Save your business layer.
   a) To save your business layer, from the *File* menu, choose *Save*.

Test Objects

Finally, test the objects to make sure that they are working as expected.

1. Test a few objects in the *Query* drawer.
   View the SQL and run queries to make sure that the correct data is retrieved.
   a) In the business layer, choose the *Query* drawer.

   b) From the `Client` folder, drag the Country, Region, and Client Name objects to the *Result Objects* pane.

   c) To view the SQL, choose *View Script*.

   d) In the *Data Preview* pane, choose *Refresh*.

LESSON SUMMARY

You should now be able to:

- Verify business object accuracy and integrity

Unit 4
Lesson 4

Creating Measure Objects

LESSON OVERVIEW
In addition to dimension and detail objects, you can define measure objects. This lesson explains how to identify and create measure objects.

LESSON OBJECTIVES
After completing this lesson, you will be able to:

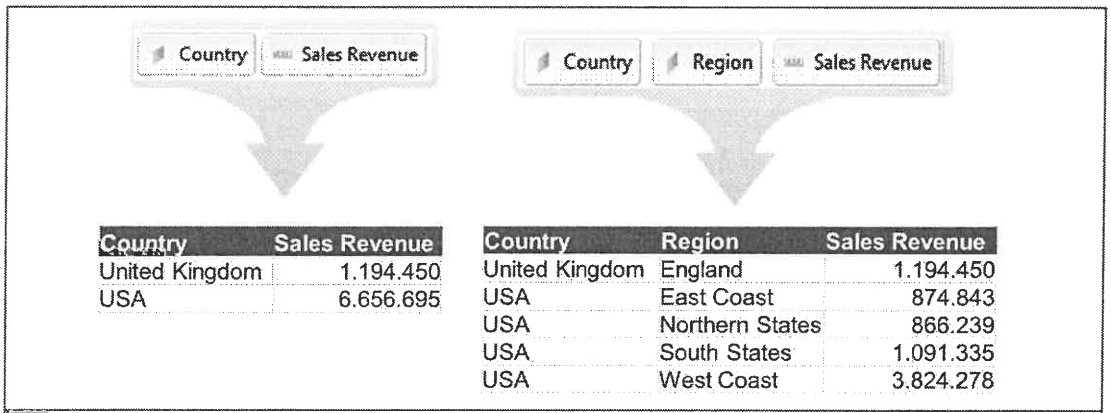
- Create a measure object
- Create a delegated measure object

Measure Objects

In addition to dimension and attribute objects, you can define measure objects. Measure objects contain aggregate functions that map to statistics in the database. They represent the metrics by which you want to compare dimensions.

A measure object returns numeric information. Measure objects are flexible because they are semantically dynamic. This means that the values they return in a query vary depending on the dimension and detail objects that are used with them.

You can see from the illustration that two separate queries, using the same Sales Revenue measure object, but different dimension objects, results in the measure returning different values.



Country	Sales Revenue
United Kingdom	1.194.450
USA	6.656.695

Country	Region	Sales Revenue
United Kingdom	England	1.194.450
USA	East Coast	874.843
USA	Northern States	866.239
USA	South States	1.091.335
USA	West Coast	3.824.278

Figure 12: Different Levels of Measure Aggregation

You create a measure object by using an aggregate function in the SELECT definition of the object.

Basic Aggregate Functions

- Sum
- Count
- Average
- Maximum
- Minimum

A measure object returns numeric data from the database that aggregates up or down according to the dimension objects in the query. The most regularly used aggregates are shown in the preceding bullet list. However, there are others that can be used. The full set of aggregate functions is held in the **Numeric** functions selection list of the **SQL expression editor** dialog box.

Measure Object SQL Inference

When a query uses a measure object with a dimension or attribute object, the query definition automatically infers a GROUP BY clause in the SELECT statement.

Inference of the GROUP BY clause is dependent on the SQL rule: When the SELECT clause line contains an aggregate, everything outside of that aggregate in the clause must also appear in the GROUP BY clause. This SQL rule is why dimension and attribute objects must not contain aggregates. Any dimension or attribute that is used in the same query as a measure object is always included in an inferred GROUP BY clause.

When a query includes only a measure object, the SQL inferred is the same as when the query uses a dimension object:

When Queries Include Measure Objects

The query result shows one row of the total revenue value.

In this example, the query includes only the Sales Revenue measure so the inference engine does not include a GROUP BY clause in the SQL statement.

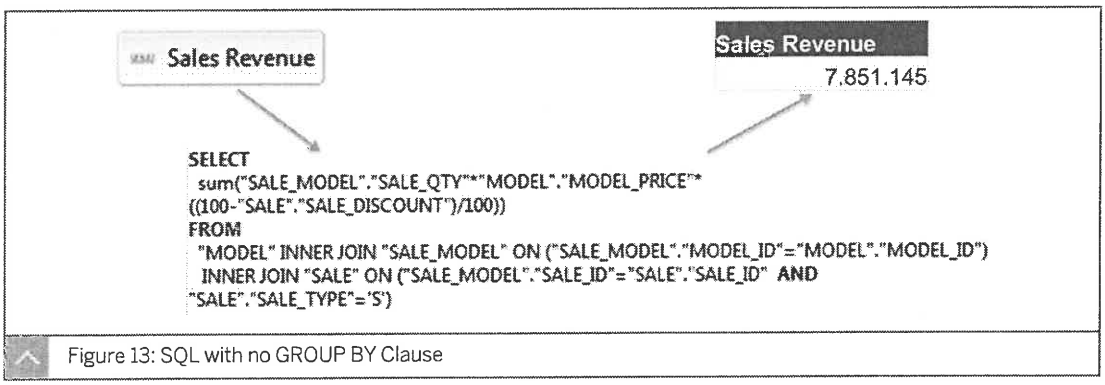


Figure 13: SQL with no GROUP BY Clause

When a query uses at least one dimension or attribute object and a measure, the inference engine includes a GROUP BY clause with all the objects except the measure in the SQL Statement:

- The SELECT clause shows the objects and measure selected in the query with the syntax including the aggregate function.
- The GROUP BY clause includes all the objects except the aggregate.

In this example, the query includes the Sales Revenue measure and the Country object so the inferred SQL statement includes a GROUP BY clause with the Country.

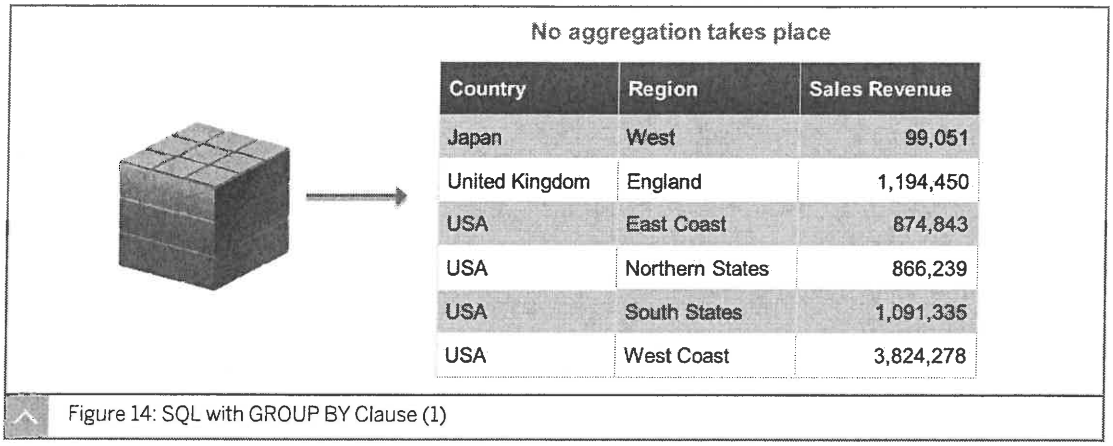


Figure 14: SQL with GROUP BY Clause (1)

In this example, the query includes two dimensions (Country and Region) so the inference engine includes both dimensions in the GROUP BY clause. As a result, the values returned for the Sales Revenue measure object are aggregated to a lower level, the Region. This mechanism in the inference engine allows the measure objects to adapt dynamically to other associated objects.

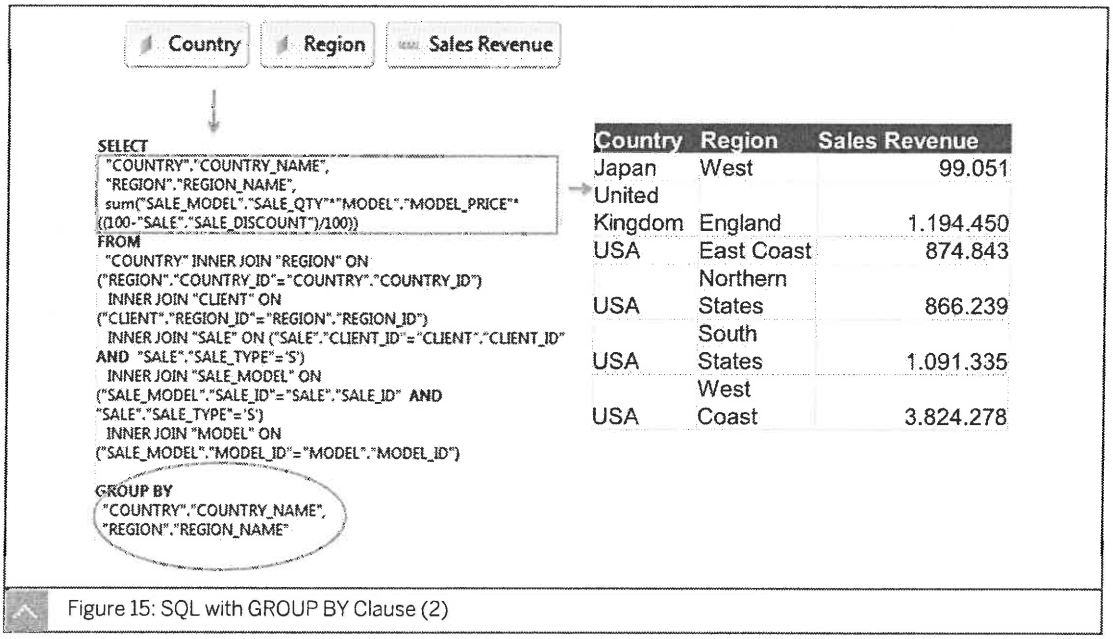


Figure 15: SQL with GROUP BY Clause (2)

Measure Object Behavior

Query Process

How do the Business Objects end-user querying tools process the measure object in a query, and how are the values projected?

There are two levels of aggregation in the query process:

- Aggregation at SELECT level
- Aggregation at projection level

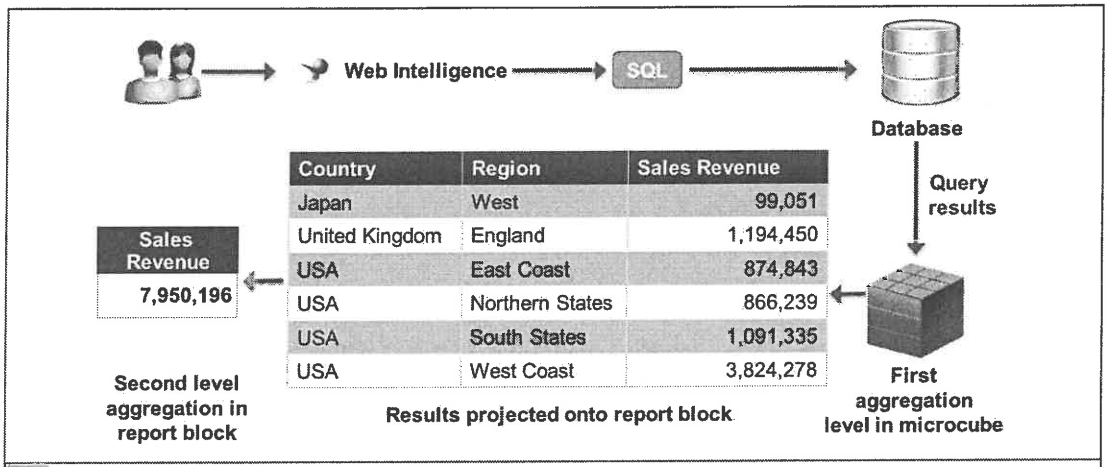


Figure 16: Aggregation at SELECT and at Projection Level

SELECT Level Aggregation

Aggregation at SELECT level occurs first in the query process:

- The user creates a query.
- Web Intelligence defines the SQL from the query and sends a SELECT statement to the target database.
- The data is returned to the microcube. The first level of aggregation now takes place in the microcube.
- The results are projected into the report.

Aggregation at projection level

When you run a query in certain SAP BusinessObjects reporting and analysis applications, the result set of the SELECT statement is stored in the microcube. All data then held in the microcube is projected into a block (the table or chart in the report). Therefore, because data is projected from the lowest level held in the microcube, no projection aggregation takes place.

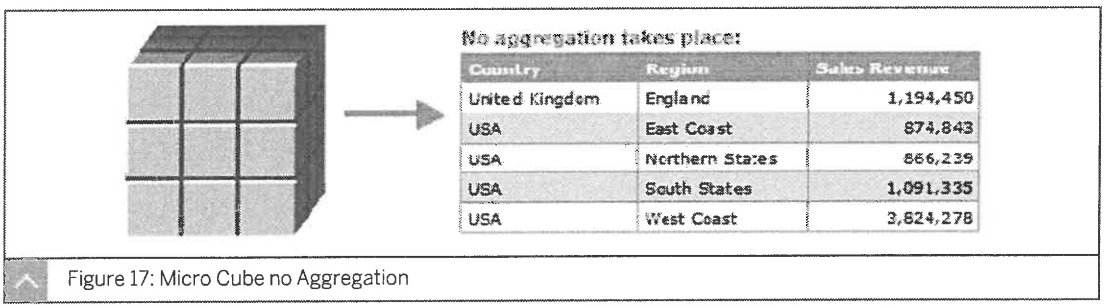
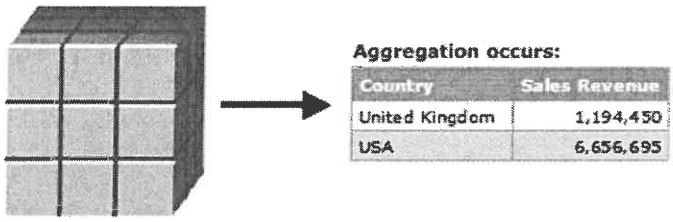


Figure 17: Micro Cube no Aggregation

However, when you edit the table, for example, by removing a column, and therefore project only partial data from the microcube, aggregation is required to show measure values at a higher level. The data in the microcube remains unchanged.

For instance, if you do not project the region data into the block, the four rows related to USA are reduced to one to show the overall Sales Revenue for that country. In this instance, a sum aggregation is required.



Aggregation occurs:

Country	Sales Revenue
United Kingdom	1,194,450
USA	6,656,695

Figure 18: Micro Cube with Aggregation

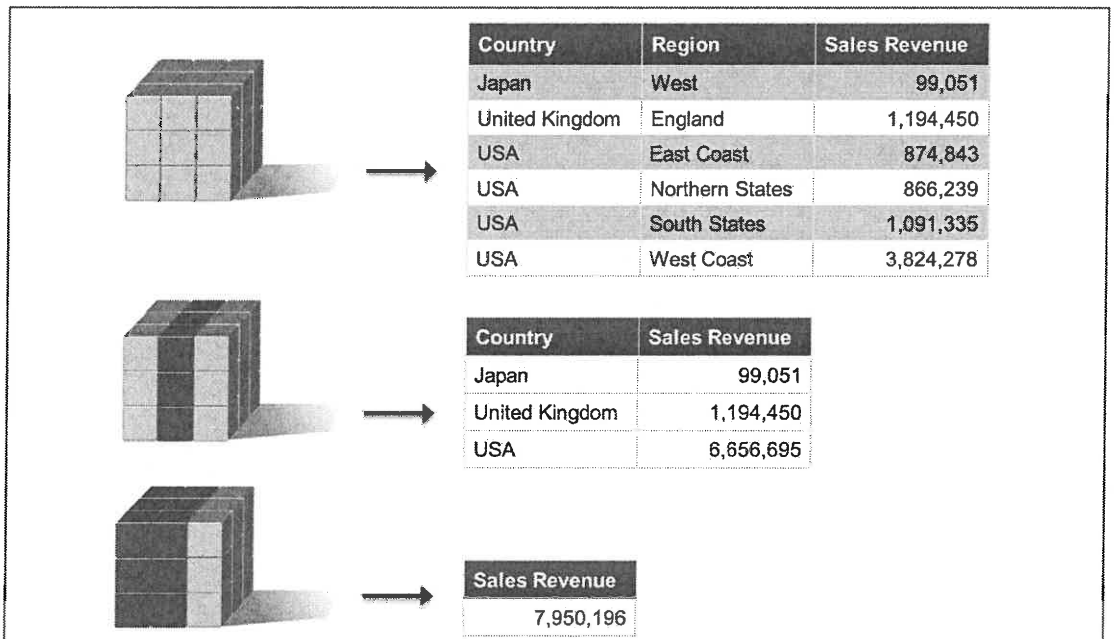
When projecting all variables in the microcube, no projection aggregation takes place.

When projecting only some variables from the microcube, projection aggregation occurs.

Projection aggregation is separate from SELECT aggregation and depends on how you define the measure object properties when you create the measure object using Universe Designer.

Setting selection and projection aggregates

Statistically, only certain SELECT and projection aggregates are compatible.



Country	Region	Sales Revenue
Japan	West	99,051
United Kingdom	England	1,194,450
USA	East Coast	874,843
USA	Northern States	866,239
USA	South States	1,091,335
USA	West Coast	3,824,278

Country	Sales Revenue
Japan	99,051
United Kingdom	1,194,450
USA	6,656,695

Sales Revenue
7,950,196

Figure 19: Recommended Projection Aggregate

For the reports to present statistically correct results for a measure object both at the query and projection levels, the SELECT and projection aggregates must complement each other.

However, as a universe designer, if you configure a measure differently, the Business Objects end-user querying tools will aggregate as you have indicated when projecting the data from the microcube.

The projection aggregation is set in the measure object properties:

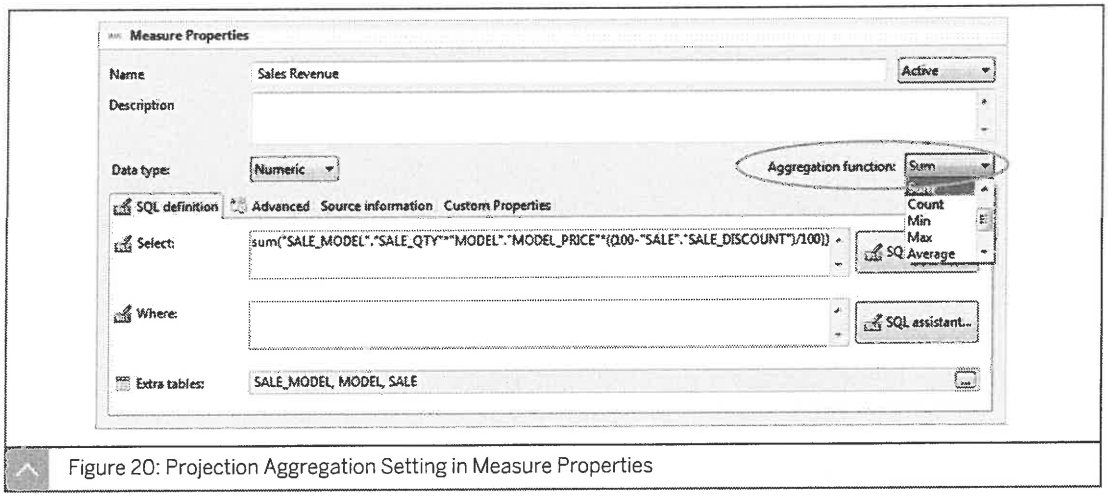


Figure 20: Projection Aggregation Setting in Measure Properties

Measure Object Testing

When you test a measure, be far more rigorous in your checks than with a dimension or an attribute. This rigor is necessary because you must confirm that the values of the measure aggregate correctly both at the SELECT and projections levels.

Remember, the three steps to testing a dimension or a attribute object are:

1. Check that the objects exist.
2. Check the inferred SQL.
3. Check the query results.

For measure objects, the additional steps are:

- Make a query with a minimum of two dimensions and a measure.
- To validate the accuracy of the measure aggregate, we recommend that you test it with at least three separate queries.

At SELECT Level

To test the inferred SELECT statements for a measure object, make at least two separate queries using different dimension objects to produce different levels of aggregation. Three or more queries are preferable.

In each instance, check the following:

- The inferred SQL of the query. In particular, check that the GROUP BY clause has been inferred correctly.

> Hint:
> If it has not been inferred at all, it is likely that you have set a calculation and not an aggregate in the Select field of the measure object properties.

- The results of the query. Check that the query produces the correct results.

Testing measures at projection level

To test for projected aggregation, build a query in an SAP BusinessObjects reporting tool such as Web Intelligence containing at least two dimension objects, as well as the measure

object you are testing. This query allows you to project from other than the lowest level of data held in the microcube and therefore test projection aggregation.

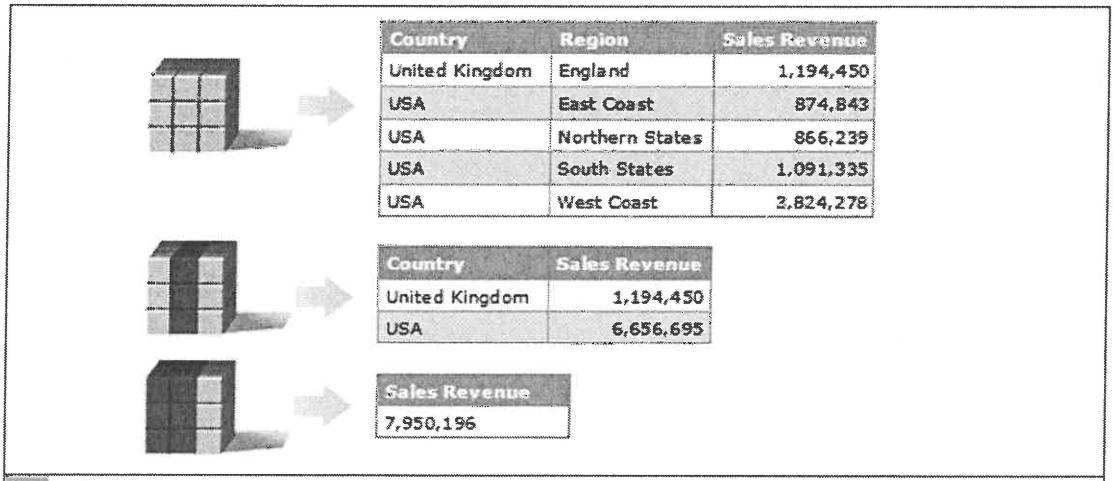


Figure 21: Projected Aggregation at Different Levels

Projection Recommendations

Database Aggregate	Projection Function
Sum	Sum
Minimum	Minimum
Maximum	Maximum
Count	Sum
Average	Delegated

Unit 4
Exercise 9

Create and Test Measure Objects

Business Example

Add measure objects to the Business Layer so that the business users can analyze numeric data in their queries.

Note:
To complete the exercises, replace the value *xx* with the group number the instructor provided you with at the beginning of the course.

1. In the `Sales` folder in the Business Layer, create the subfolder `Sales Figures`.
2. In the new `Sales Figures` subfolder, using the method of your choice, create measure objects as identified in the following tables.

   Objects for the `Sales Figures` folder are listed in the following table:

Object Name	SELECT Statement	Object Description
Sales Revenue	sum(SALE_MODEL.SALE_QTY * MODEL.MODEL_PRICE * ((100 – SALE.SALE_DISCOUNT) / 100))	Total Sale Invoice Revenue
Cost of Car Sale	sum(SALE_MODEL.SALE_QTY * MODEL.MODEL_COST)	Total Cost of Car Sale
Number of Cars Sold	sum(SALE_MODEL.SALE_QTY)	Total Number of Cars Sold

   Object for the `Sale Prices` folder (subfolder of Car) are listed in the following table:

Object Name	SELECT Statement	Object Description
Lowest Priced Value	min(MODEL.MODEL_PRICE)	The lowest priced value based on manufacturers recommended retail price
Highest Priced Value	max(MODEL.MODEL_PRICE)	The highest priced value based on manufacturers recommended retail price

   Objects for the `Client` folder are listed in the following table:

Object Name	SELECT Statement	Object Description
Number of Clients	count(CLIENT.CLIENT_ID)	Total number of clients

Hint:
In the SQL Assistant, under the appropriate table, double click the column name. Your syntax should look like SALE_MODEL.SALE_QTY.

3. By creating and running the following query, test the measure objects in the *Query Drawer*: Country (from the Client folder) and Sales Revenue (from the Sales Figures subfolder)
The results should look as follows:

Country	Sales Revenue
Japan	99, 051
United Kingdom	1, 194, 450
USA	6, 656, 695

Unit 4
Solution 9

Create and Test Measure Objects

Business Example

Add measure objects to the Business Layer so that the business users can analyze numeric data in their queries.

> Note:
> To complete the exercises, replace the value *xx* with the group number the instructor provided you with at the beginning of the course.

1. In the `Sales` folder in the Business Layer, create the subfolder `Sales Figures`.
   a) Right-click the *Sales* folder.
   b) At the top of the *Business Layer* pane, choose the *Insert object* icon and choose the *Insert Folder* icon.
   c) In the *Folder Properties* pane, and in the *Name* field, enter **`Sales Figures`**.
   d) Save your work.
2. In the new `Sales Figures` subfolder, using the method of your choice, create measure objects as identified in the following tables.

   Objects for the `Sales Figures` folder are listed in the following table:

Object Name	SELECT Statement	Object Description
Sales Revenue	sum(SALE_MODEL.SALE_QTY * MODEL.MODEL_PRICE * ((100 – SALE.SALE_DISCOUNT) / 100))	Total Sale Invoice Revenue
Cost of Car Sale	sum(SALE_MODEL.SALE_QTY * MODEL.MODEL_COST)	Total Cost of Car Sale
Number of Cars Sold	sum(SALE_MODEL.SALE_QTY)	Total Number of Cars Sold

   Object for the `Sale Prices` folder (subfolder of Car) are listed in the following table:

Object Name	SELECT Statement	Object Description
Lowest Priced Value	min(MODEL.MODEL_PRICE)	The lowest priced value based on manufacturers recommended retail price

Object Name	SELECT Statement	Object Description
Highest Priced Value	max(MODEL.MOD-EL_PRICE)	The highest priced value based on manufacturers recommended retail price

Objects for the `Client` folder are listed in the following table:

Object Name	SELECT Statement	Object Description
Number of Clients	count(CLIENT.CLIENT_ID)	Total number of clients

Hint:
In the SQL Assistant, under the appropriate table, double click the column name. Your syntax should look like SALE_MODEL.SALE_QTY.

a) In the *Business Layer* pane, right-click the Sales Figures folder and from the context menu, choose *New* → *Measure*.

b) In the editing pane, change the name to **`Sales Revenue`**.

c) Enter the description **`Total Sale Invoice Revenue`**.

d) Choose *SQL Assistant*.

e) In the SQL expression editor in the *New Measure* dialog, choose *Database functions function group*.

f) Choose *Numeric function group* and choose the SUM function.

g) By typing and selecting the columns from the *Tables* pane, replace $1 with the following expression:

```
SALE_MODEL.SALE_QTY * MODEL.MODEL_PRICE * ( - SALE.SALE_DISCOUNT) /
100)
```

h) Choose *OK*.

i) Repeat Steps a to h for the Cost of Cars sales and Number of Cars sold.

3. By creating and running the following query, test the measure objects in the *Query Drawer*: Country (from the Client folder) and Sales Revenue (from the Sales Figures subfolder)
The results should look as follows:

Country	Sales Revenue
Japan	99, 051
United Kingdom	1, 194, 450
USA	6, 656, 695

a) In the business layer, choose the Query drawer.

b) From the `Client` folder, drag the Country object to the *Result Objects* pane.

c) From the `Sales Figures` subfolder, drag the Sales Revenue object to the *Result Objects* pane.

d) Check the SQL, noting the GROUP BY clause.
It should GROUP BY Country.

e) In the *Data Preview* pane, choose *Refresh*.

Delegated Measure Objects

Information Design Tool allows designers to create measures whose calculation is delegated to the database. These measures are called "delegated smart measures".

A delegated measure is a measure that delegates its aggregation calculation to the database. The Information Design Tool defines a delegated measure in the universe to make it available to report users for on demand reporting or for viewing reports built on that universe.

A universe designer can create a delegated measure when conventional aggregation does not provide an accurate result for the measure, such as for:

- Complex averages, such as Weighted averages (an average of a percentage)
- Ratios
- Other measures that do not aggregate along all the dimensions in a report
- OLAP sources where measure aggregations are already available in the OLAP cube

The delegated measure is then available to report designers.

Delegated measures are available for all relational and OLAP data sources.

Delegated Measure Benefits

The delegated measure represents an extension of reporting tool calculations by supporting aggregation within the database and makes nonadditive measures available within the universe.

You benefit from delegated measures because they:

- Increase querying efficiency.
- Make nonadditive measures available within the universe.
- Use database-specific syntax to improve performance and provide optimization on the internal architecture of all vendors.
- Extend support of calculations for all BI 4 reporting tools.

Delegate Measure Calculations

By default, Web Intelligence and Crystal reports calculate measures based on the objects used within the query. The dimensions used in the query are called a grouping set. The delegated measure calculates the aggregation of the measure for any subset of dimensions required in the report.

For example, a query that retrieves Country, Region, Maker, Year, Number of Cars Sold and Average Sales Total dimensions would, by default, calculate the Number of Cars Sold and Average Sales Total for that grouping set.

However, when you format your report, if you want to present the Number of Cars Sold and Average Sales Total for Region and Maker, Web Intelligence cannot calculate the correct Average Sales Total for these smaller entities.

The delegated measure, on the other hand, calculates the measures for all subsets of the dimensions in the report. Thus, it would calculate Number of Cars Sold and Average Sales Total per:

- Country

- Country and Year
- Country, Region, and Year
- Country, Region, Maker, and Year

Delegated Measures and Report Changes

When the report changes, the delegated measure updates the grouping set according to when users save their report, it removes unused grouping sets.

Region	Number of Cars Sold	Average Sales Total
East Coast	13	81,924.38
England	46	31,060.4
Northern States	13	78,749
South States	15	99,679.23
West	1	99,051
West Coast	68	69,742.05
	Average:	**61,155.35**

Figure 22: Car Sales Report by Region and Number of Cars Sold

Delegated Measure as Weighted Average

A report designer wants to calculate the average sales total per region, and also for all regions.

The universe designer has created a measure called Average Sales Total that calculates the average of the sales totals.

When the user runs this measure in a query, together with Region, the average sales total per each region is calculated.

In the report, the user then wants to calculate the average sales total level for all regions, and adds an average calculation on the Average Sales Total column.

Region	Average Sales Total
East Coast	72,903.58
England	29,861.25
Northern States	78,749
South States	90,944.58
West	99,051
West Coast	70,819.96
Average:	**73,721.56**

Figure 23: Default Calculation on Measure Average Sales Total (1)

A new row appears at the bottom of the table and this row displays the average sales total for all regions.

This average appears accurate, but it is not. The reporting tool is adding up the values shown in the Average Sales Total column and then dividing them by six, since there are six regions available. The values in the Average Sales Total column are already calculated averages, so the reporting tool is actually calculating the average of averages.

What is required here is a weighted average, as some regions have more car sales than others. The sales total value returned by those regions with more sales must count more heavily than those regions with fewer cars sold.

When adding the Number of Cars sold figure to the query, you see here that the number of cars sold per region varies considerably. Sixty-eight cars were sold in the West Coast, while only 13 in the East Coast.

Region	Number of Cars Sold	Average Sales Total
East Coast	13	81,924.38
England	46	31,060.4
Northern States	13	78,749
South States	15	99,679.23
West	1	99,051
West Coast	68	69,742.05
Average:		**76,701.01**

Figure 24: Calculation on Measure weighted Average Sales Total (2)

When the reporting tool calculates the overall average sales total, it cannot take into account the fact that the West Coast should have more weight in the calculation of the average. The reporting tool does not have access to the detailed data, just to the regional average value, returned by the Average Sales Total measure.

To calculate a weighted average, create a delegated measure in the universe, which will delegate this calculation to the database.

To do create a delegated measure in the universe, modify the universe using Information Design Tool. In the properties of the Average Sale Total measure, setting the function to delegated delegates this calculation to the database and effectively calculates a weighted average.

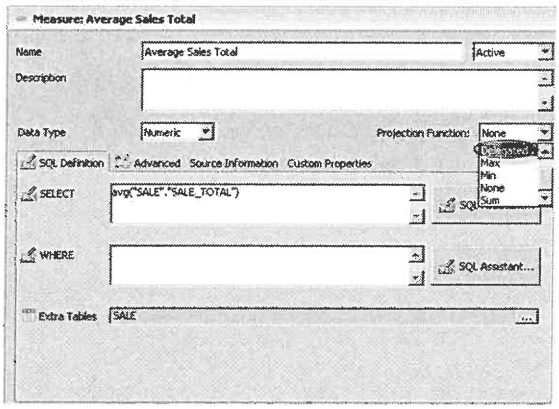


Figure 25: Set Projection Function to Database Delegated

Running the same query using the Average Sales Total measure with the Aggregation function set to Delegated, results function set to **Delegated**, results in the correct average total:

Region	Number of Cars Sold	Average Sales Total
East Coast	13	81,924.38
England	46	31,060.4
Northern States	13	78,749
South States	15	99,679.23
West	1	99,051
West Coast	68	69,742.05
	Average:	**61,155.35**

Figure 26: Result of Database Delegated on Average Sales Total

Delegated Measures Best Practices

Keep in mind the following best practices for effective use of delegated measures:

- Use a delegated measure when report users manipulate the measures in reports. When they only must refresh, view, and print their reports, the calculation may be simpler to perform in the report.
- Use a delegated measure to replace multiple query aggregates.
- Use a delegated measure on calculations that could give inaccurate results when calculated in the report (such as a complex average).
- Use a delegated measure for measures requiring division.
- When you create the delegated measure in the business layer, make sure that in the Measure properties dialog box for the measure, on the Definition tab, you enter identifying text in the description field. This text ensures that report designers can quickly recognize a delegated measure when they glide their mouse over it in a query.

Inappropriateness of Delegated Measures

Do not use delegated measures when you apply a filter to an aggregated value from your query and further aggregate it within the report.

Web Intelligence cannot determine how the filter affects the calculation of the delegated measure and, instead, returns the cell error code #UNAVAILABLE.

For example, do not use delegated measures when there is a:

- Report filter on a dimension that is not in the dimensional context.
- Report filter on the formula.
- Formula in the dimensional context of the measure.

However, for drill filters and for simple filters combined with AND at the first level of drilling, the context takes the related dimension into account.

Do not use a delegated measure when a standard measure will work.

Unit 4
Exercise 10

Create a Delegated Measure

Business Example

Include delegated measure objects to the Business Layer so that the business users can accurately aggregate numeric data in their reports.

1. In the `Sales Figures` folder, create a new measure called Average Sales Total with the following syntax:

   ```
   avg (SALE.SALE_TOTAL)
   ```

2. Save your business layer as in previous exercises.

Unit 4
Solution 10

Create a Delegated Measure

Business Example

Include delegated measure objects to the Business Layer so that the business users can accurately aggregate numeric data in their reports.

1. In the `Sales Figures` folder, create a new measure called Average Sales Total with the following syntax:

   ```
   avg (SALE.SALE_TOTAL)
   ```

   a) In the *Business Layer* pane, right-click on the `Sales Figures` folder.

   b) Choose *Insert* → *Measure*.

   c) In the editing pane. change the name to **`Average Sales Total`**.

   d) Choose *SQL Assistant*.

   e) To define the SQL for the object `avg (SALE.SALE_TOTAL)`, from the *Tables* pane, enter and select the column SALE. SALE_TOTAL.

   f) Choose *OK*.

   g) For the measure's properties, change the project function drop-down to *Delegated*.

2. Save your business layer as in previous exercises.

   a) From the toolbar, choose *Save*.

LESSON SUMMARY

You should now be able to:

- Create a measure object
- Create a delegated measure object

Unit 4
Lesson 5

Creating Shortcut Joins

LESSON OVERVIEW
Shortcut joins are commonly used to link a shared lookup table to another table further along a join path. This lesson describes how to create shortcut joins.

LESSON OBJECTIVES
After completing this lesson, you will be able to:

- Create shortcut joins

Shortcut Joins

A common use of shortcut joins is to link a shared lookup table to another table further along a join path. The join path is comprised of several different tables in the same context.

A shortcut join is a join that provides an alternate path between two tables. It improves the performance of a query by not taking into account intermediate tables, and shortening a longer join path.

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. Therefore, the same value exists at all the levels being joined.

LESSON SUMMARY
You should now be able to:

- Create shortcut joins

Learning Assessment

1. What is a business layer?

2. How are business layers published to the Central Management Server?

   Choose the correct answer.

   - ☐ A As a .txt file
   - ☐ B As a .xls file
   - ☐ C As a .unx file
   - ☐ D As a .pdf file

3. Which of the following is a business object type?

   Choose the correct answers.

   - ☐ A Dimension
   - ☐ B Rule
   - ☐ C Map
   - ☐ D Report

4. The business layer tree view does not allow users to search for an object.

   Determine whether this statement is true or false.

   - ☐ True
   - ☐ False

5. Why are objects grouped into folders in a universe?

6. What is an LOV?

   Choose the correct answers.

   - [ ] A Layers of Values
   - [ ] B Links on Values
   - [ ] C Levels of Values
   - [ ] D List of Values

7. When you are creating a similar time object to a previous one, it can be more efficient to edit a copy of the previous object, rather than creating a new one.

   Determine whether this statement is true or false.

   - [ ] True
   - [ ] False

8. How do you add attributes to objects?

   Choose the correct answers.

   - [ ] A Right-click the object, then click *New* on the popup menu and select *Attribute*.
   - [ ] B Double-click the object, then click *New* on the popup menu, and select *Attribute*.
   - [ ] C Right-click the object, then click *Properties* on the popup menu. Navigate to the *Attributes* tab on the *Properties* screen.
   - [ ] D Right-click the object, then select the *Attributes* popup menu option and the *New* suboption.

9. What do you use to detect errors in the SQL syntaxes that are used in the created objects?

   Choose the correct answers.

   - [ ] A Check Cardinalities
   - [ ] B Object Check
   - [ ] C Information Design Tool
   - [ ] D Check Integrity

10. What is the SQL Rule for inference of the GROUP BY clause?

11. Arrange the three steps for testing a dimension or attribute in the correct order.

Arrange these steps into the correct sequence.

- ☐ Check the inferred SQL
- ☐ Check that the object exists
- ☐ Check the query results

12. Delegated Smart Measures make nonadditive measures available within the universe.

Determine whether this statement is true or false.

- ☐ True
- ☐ False

13. What are dimensions used in the query called?

Choose the correct answer.

- ☐ A A grouping set
- ☐ B A dimensional set
- ☐ C A query dimension
- ☐ D A query set

14. What is a shortcut join?

Learning Assessment - Answers

1. What is a business layer?

   A business layer is a collection of metadata objects that map to SQL or MDX definitions in a database.

2. How are business layers published to the Central Management Server?

   Choose the correct answer.

   - [ ] A As a .txt file
   - [ ] B As a .xls file
   - [x] C As a .unx file
   - [ ] D As a .pdf file

3. Which of the following is a business object type?

   Choose the correct answers.

   - [x] A Dimension
   - [ ] B Rule
   - [ ] C Map
   - [ ] D Report

4. The business layer tree view does not allow users to search for an object.

   Determine whether this statement is true or false.

   - [ ] True
   - [x] False

5. Why are objects grouped into folders in a universe?

   Grouping in folders is done to provide a structure for the universe and makes it easier for users to locate particular objects.

6. What is an LOV?

   Choose the correct answers.

   - [ ] A Layers of Values
   - [ ] B Links on Values
   - [ ] C Levels of Values
   - [x] D List of Values

7. When you are creating a similar time object to a previous one, it can be more efficient to edit a copy of the previous object, rather than creating a new one.

   Determine whether this statement is true or false.

   - [x] True
   - [ ] False

8. How do you add attributes to objects?

   Choose the correct answers.

   - [x] A Right-click the object, then click *New* on the popup menu and select *Attribute*.
   - [ ] B Double-click the object, then click *New* on the popup menu, and select *Attribute*.
   - [ ] C Right-click the object, then click *Properties* on the popup menu. Navigate to the *Attributes* tab on the *Properties* screen.
   - [ ] D Right-click the object, then select the *Attributes* popup menu option and the *New* suboption.

9. What do you use to detect errors in the SQL syntaxes that are used in the created objects?

   Choose the correct answers.

   - [ ] A Check Cardinalities
   - [ ] B Object Check
   - [ ] C Information Design Tool
   - [x] D Check Integrity

10. What is the SQL Rule for inference of the GROUP BY clause?

    When the SELECT clause line contains an aggregate, everything outside of that aggregate in the clause must also appear in the GROUP BY clause.

11. Arrange the three steps for testing a dimension or attribute in the correct order.

Arrange these steps into the correct sequence.

[2] Check the inferred SQL

[1] Check that the object exists

[3] Check the query results

12. Delegated Smart Measures make nonadditive measures available within the universe.

Determine whether this statement is true or false.

[X] True

[ ] False

13. What are dimensions used in the query called?

Choose the correct answer.

[X] A A grouping set

[ ] B A dimensional set

[ ] C A query dimension

[ ] D A query set

14. What is a shortcut join?

A shortcut join is a join that provides an alternate path between two tables. It improves the performance of a query by not taking into account intermediate tables, and shortening a longer join path.

UNIT 5 Loops in a Data Foundation

UNIT OBJECTIVES

- Identify loops
- Resolve loops using aliases
- Resolve loops using contexts

- Detect contexts
- Edit contexts
- Test contexts
- Resolve recursive loops

Unit 5
Lesson 1

Resolving Loops with Joined Tables

LESSON OVERVIEW

This lesson describes loops, a particular type of join issue that can arise as you create joins between tables in your schema.

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Identify loops

Loops

A loop is a join path issue that arises from the way that tables are related in a relational database. Loops can produce instances where a query returns too few rows of data.

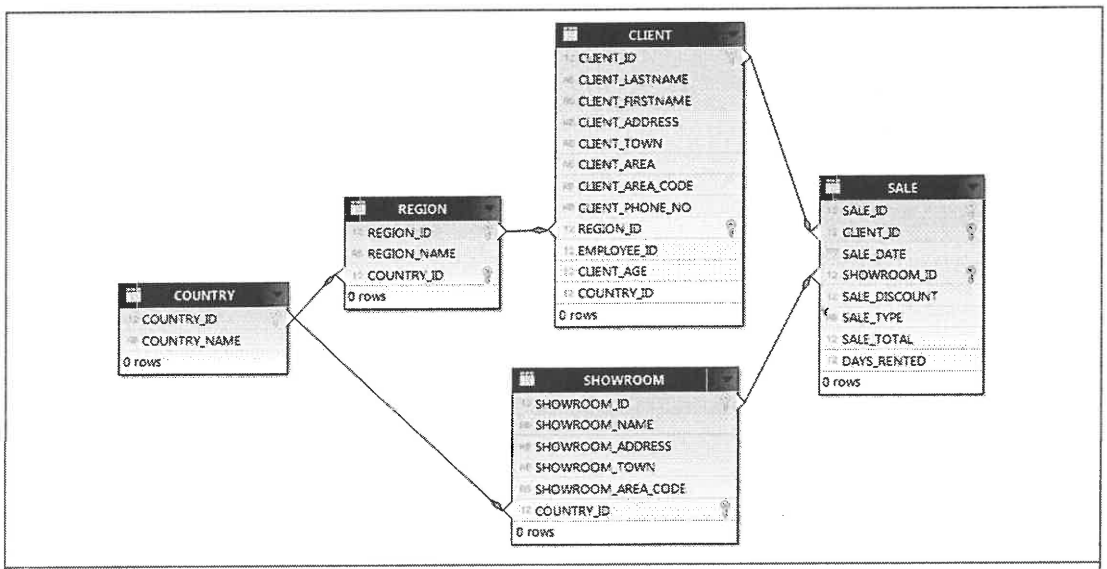


Figure 27: Understanding Loops

A loop exists when the joins between tables form a closed path.

For example, in the table layout above, the designer has added joins between the tables Showroom and Country to create two linked sets of information:

- One set links the sale details, the client, the client's region, and the client's country of residence.
- The other set links the sale details, the showroom, and the country where the showroom is located.
- Together, these joins form a loop.

Loop Problems

Suppose that users of the Motors Business Layer want to produce reports showing the revenue generated by sales, including the showroom location where the products are sold and the address of the clients. The designer adds the tables that provide this information, and creates the joins as shown in the previous example. The designer has also created objects for the Showroom Country, Client Country, and Sales Revenue.

If the loop was allowed to remain and a query was run using the Showroom Country, Client Country and Sales Revenue objects, the report results would be incorrect. The report would suggest that only clients from the US bought products in the US showrooms, and only clients from the UK bought products in the UK showrooms. However, the report would not show any clients from any other countries. When you know that there are clients from other countries, this result indicates that there is a problem with the report.

Loops exist in a Data Foundation schema and not in the database. In a database, multiple paths between tables can 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, a schema that you design in the Data Foundation often allows queries that include more than one path, which a relational database may not be designed to handle. As a result, the information returned can be incorrect. The rows that are returned are an intersection of the results for the path, so fewer rows are returned than expected. It can be difficult to determine the problem when you examine the results.

Joins restrict the data that is returned by the query. In a loop, the joins apply more restrictions than the designer intended, and incorrect data is returned.

Note:
If a loop exists in the Data Foundation schema, all objects that are created from the tables that are involved in the loop are incompatible when used in a query. It is absolutely essential to solve loops.

Loop Detection

The Data Foundation module has automatic tools that detect loops and suggest how to resolve the loop for you. You can use the Detect Loops toolbar icon to detect and indicate the tables causing loops in the active Data Foundation automatically.

Note:
Before using Detect Loops, verify that all the tables in the schema are joined, and that all cardinalities are set.

LESSON SUMMARY
You should now be able to:

- Identify loops

Unit 5
Lesson 2

Resolving Loops Using Aliases

LESSON OVERVIEW
This lesson explains how you can detect and resolve loops using aliases to ensure that the join paths taken by queries run on the universe return correct results.

LESSON OBJECTIVES
After completing this lesson, you will be able to:

- Resolve loops using aliases

Aliases Used to 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 but with a different name. The data in the alias is exactly the same as the original table, but the different name tricks SQL into using the same database table for two different purposes. The Country table has already been identified as a shared lookup table. It was identified because it is serving two purposes in the query you are trying to run (providing data for the Client Country and also for the Showroom Country).

Note:
Another way of spotting the problem table in a loop is that it only has the one end of the one-to-many joins going into it. Check the other tables in the loop. If you find no others with only one-end joins, the loop can be resolved using an alias, assuming there are no other tables joined to country.

To resolve the loop, use the same table (the Country table) twice in the same query when it is being used for different purposes. However, you cannot resolve a loop this way in SQL unless you create an Alias table.

Detect Aliases proposes candidate tables that you can edit and insert in the schema. Manually inserting an alias table, however, is preferred.

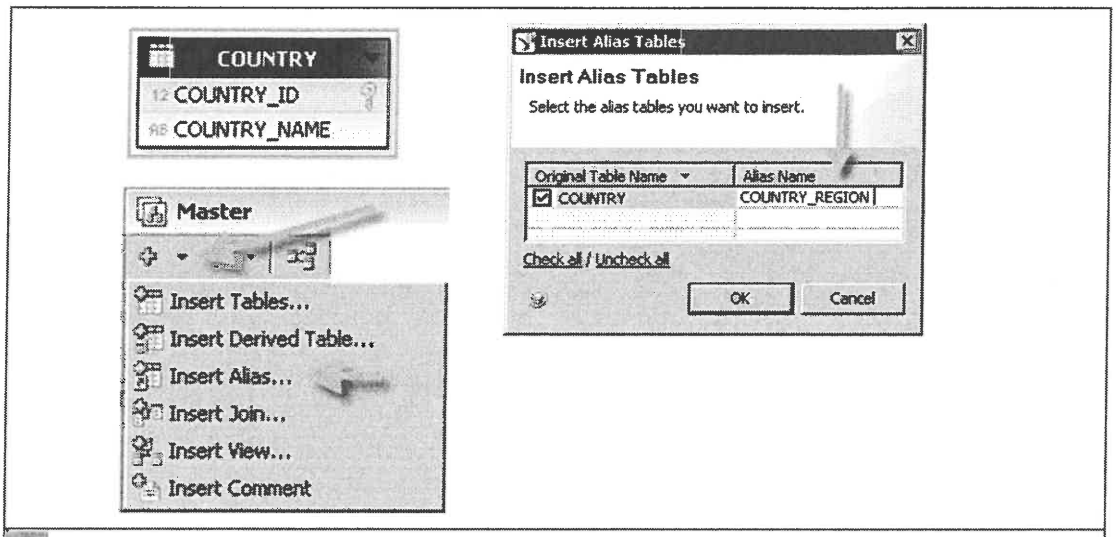


Figure 28: Insert Aliases Manually

In the Master view of your Data Foundation you can click on your potential alias table. After that, you can check the green plus icon in your Master view of your Data Foundation.

Select *Insert Alias* and the *Insert Alias Tables* window appears. Enter an alias name. Although not necessary, the name should reference the table name it is joined to.

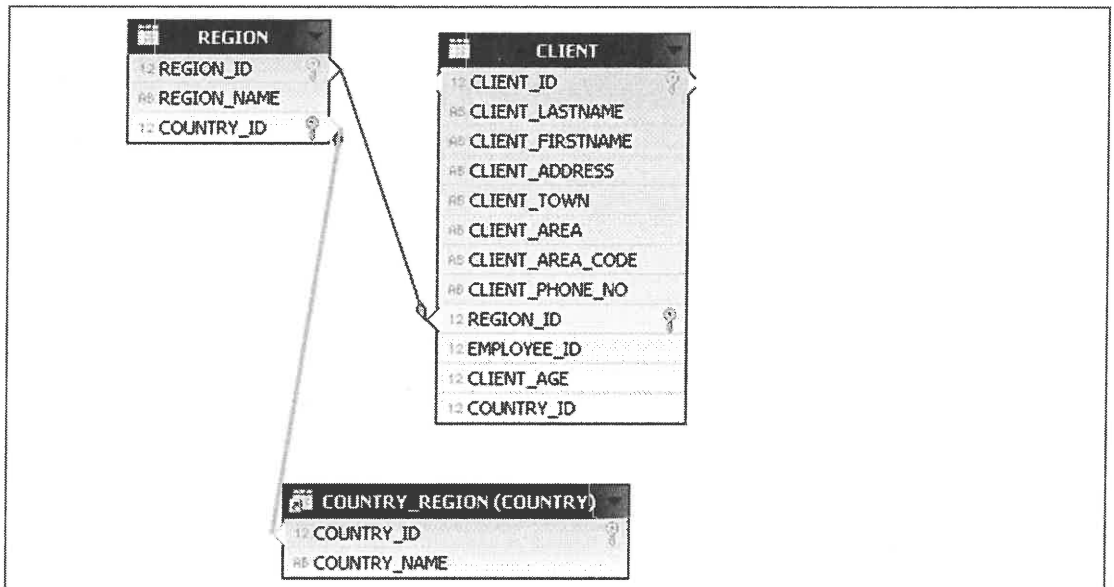


Figure 29: Alias created for COUNTRY table

Join the inserted alias table to the destination table and define cardinalities. The original table name is shown in brackets. Repeat for all tables that act as lookup tables for multiple purposes.

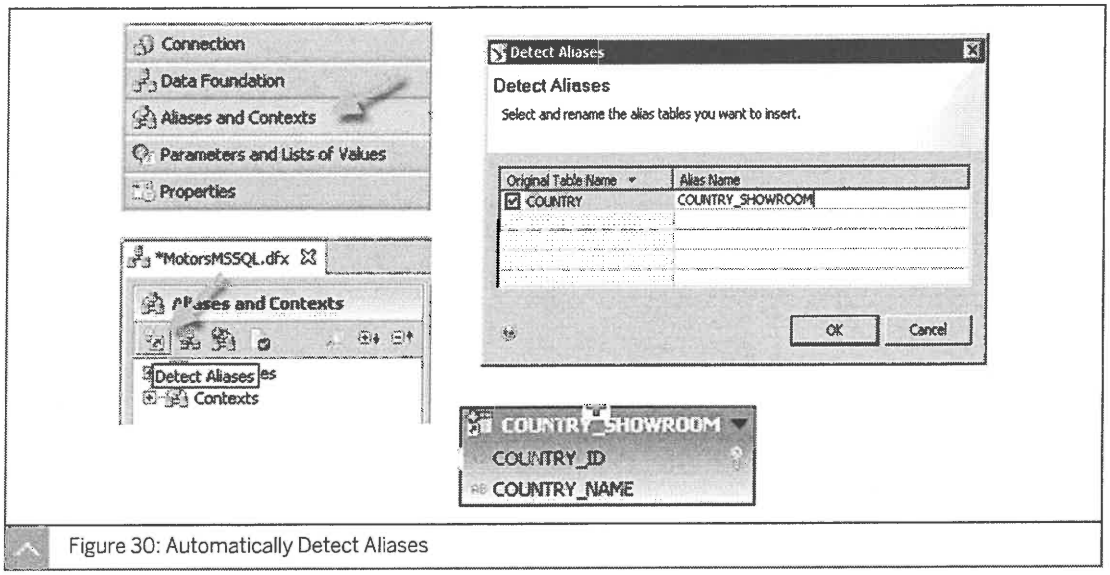


Figure 30: Automatically Detect Aliases

To detect aliases automatically, use the navigation pane of your Data Foundation and select *Aliases* and *Contexts*. Then select *Detect Aliases* from the Data Foundation menu. The *Detect Aliases* window appears. Select the check box of your original tables and enter a name for the alias table if necessary. Click *OK* and check the cardinality of the join.

Note:
When you create an alias table, check that any existing objects that are defined from the original table still refer to the correct table. To obtain the correct results, these references are defined from the alias table to infer the correct SQL.

Unit 5
Exercise 11

Use Aliases to Resolve Loops

Business Example

The business users must report on the countries where clients live, the countries where the showrooms are located, and the countries where the cars are made. Currently, the data foundation of the universe provides reporting only on the countries where the clients live. Update the data foundation to accommodate the users' reporting needs.

1. On your data foundation, insert the following join and set its cardinality to the following:
   `COUNTRY.COUNTRY_ID = SHOWROOM.COUNTRY_ID`
2. Resolve the loop created by this new join.
3. Create the following additional alias table:
   - COUNTRY_MAKER
4. Insert the following join and set the cardinality for all joins to 1,n:
   `COUNTRY_MAKER.COUNTRY_ID = MAKER.COUNTRY_ID`
5. In the appropriate folders, create Country objects as described in the following table::

Object Name	SELECT Statement	Object Description
Showroom Country	COUNTRY_SHOWROOM. COUNTRY_NAME	Country in which showroom exists
Maker Country	COUNTRY_MAKER. COUNTRY_NAME	Country of Manufacturer

6. Check the integrity of your universe.
7. Save your universe.
8. Test your changes by building the following queries in the *Query* drawer:
   - Showroom Country, Showroom, and Sales Revenue
   - Client Country, Client Name, and Sales Revenue
   - Maker Country, Maker, and Model

Unit 5
Solution 11

Use Aliases to Resolve Loops

Business Example

The business users must report on the countries where clients live, the countries where the showrooms are located, and the countries where the cars are made. Currently, the data foundation of the universe provides reporting only on the countries where the clients live. Update the data foundation to accommodate the users' reporting needs.

1. On your data foundation, insert the following join and set its cardinality to the following:
   `COUNTRY.COUNTRY_ID = SHOWROOM.COUNTRY_ID`
   a) In your local project, open the Data Foundation.
   b) In the white area where the tables are located, right-click and from the menu choose *Insert* → *Join*.
   c) In the edit join window, on the left select the COUNTRY table and COUNTRY_ID and on the right SHOWROOM table and COUNTRY_ID.
   Make sure that the operator is "=".
   d) To set the cardinality, choose *Detect* and then choose *OK*.
2. Resolve the loop created by this new join.
   a) From the main menu, choose *Actions* → *Detect Loops*.
   b) In the *Loops* pane, choose *Refresh*.
   c) From the *Aliases and Contexts* pane, choose *Detect Aliases*.
   d) Select the COUNTRY table and choose *OK*.
3. Create the following additional alias table:
   - COUNTRY_MAKER
   a) In your data foundation, right-click the COUNTRY table.
   b) Choose *Insert* → *Alias Table* . . .
   c) Choose *OK*.
   d) Double-click the alias table and change the name to COUNTRY_MAKER.
   e) Choose *OK*.
4. Insert the following join and set the cardinality for all joins to 1,n:
   `COUNTRY_MAKER.COUNTRY_ID = MAKER.COUNTRY_ID`
   a) In the white area where the tables are located, right-click and choose *Insert* → *Join*.

b) In the edit join window, on the left, choose the COUNTRY_MAKER table and COUNTRY_ID and on the right MAKER table and COUNTRY_ID.

Make sure that the operator is "=".

c) To set the cardinality, choose *Detect* and choose *OK*.

d) Save the data foundation.

5. In the appropriate folders, create Country objects as described in the following table::

Object Name	SELECT Statement	Object Description
Showroom Country	COUNTRY_SHOWROOM. COUNTRY_NAME	Country in which showroom exists
Maker Country	COUNTRY_MAKER. COUNTRY_NAME	Country of Manufacturer

a) Open the Business Layer.

b) From the COUNTRY_SHOWROOM table, move the COUNTRY_NAME field to the Showroom folder using drag and drop, and rename the object to Showroom Country.

c) From the COUNTRY_MAKER table, move the COUNTRY_NAME field to the Car folder using drag and drop, and rename the object to Maker Country.

6. Check the integrity of your universe.

a) In the *Local Project* pane, right-click and choose *Check Integrity* . . .

b) Choose *Check All*.

c) Choose *Check Integrity*.

d) Choose *OK*.

7. Save your universe.

a) Choose *File* → *Save All*.

8. Test your changes by building the following queries in the *Query* drawer:

- Showroom Country, Showroom, and Sales Revenue
- Client Country, Client Name, and Sales Revenue
- Maker Country, Maker, and Model

a) In the business layer, choose the *Query* drawer.

b) So they appear in the *Result Objects* pane, double-click the Showroom Country, Showroom, and Sales Revenue objects.

c) Choose *View Script* and notice that the alias table is being used for the query.

The only time the original table is used is when the Client Country object is used.

d) Choose *Refresh*.

e) Edit the query and continue testing the other objects.

LESSON SUMMARY

You should now be able to:

- Resolve loops using aliases

Unit 5
Lesson 3

Resolving Loops Using Contexts

LESSON OVERVIEW
A context is a list of joins that define a path for a query. The tables involved in the joins are included in the context. The context is used to resolve problems in the query results. This lesson explains that another way to solve a loop in the Data Foundation structure is to create contexts.

LESSON OBJECTIVES
After completing this lesson, you will be able to:

- Resolve loops using contexts

Contexts

Another way to solve a loop in the Data Foundation structure is to create contexts.

A context resolves a loop by defining a set of joins that identify one specific path through tables in a loop. It ensures that joins are not included from different paths within the same SQL query.

You use contexts in a schema that contains multiple fact tables that share lookup tables. An example of this situation is the Sale table in the Motors Data Foundation. The Sale table 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). Without the restriction, the result set of the query would produce rows where the Sale_Type column is equal to either 'S' or 'R'.

Previously, you defined this restriction to 'S', so that any object based on the table or joins passing through that table would produce query results covering only car sales. To retrieve data concerning rental sales as well, create an alias of the Sale table called Rental, set the restriction to 'R,' and create an alias table of the Sale_Model table called Rental_Model. Creating the aliases tables, however, creates a loop. The query does not know which table to go through to move from the Client table to the Model table--the Sale or the alias Rental table.

You can solve this type of loop by creating two contexts that define the correct route through the universe structure. These routes link tables together in the structure.

Any objects derived from tables included in a context are compatible with each other. When a query is made with objects related to separate contexts, more than one SELECT statement is inferred and run. The results of the queries are then merged in the micro cube. This method avoids incorrect results that might arise due to a loop or other situation with alternative routes.

Alternative routes can exist without a loop in the universe structure.

Unit 5
Exercise 12

Create a Loop that Needs to Be Resolved by a Context

Business Example

You need to create a loop that can be resolved by a context.

1. In the data foundation, create the following aliased tables:
   `RENTAL (alias of SALE) RENTAL_MODEL (alias of SALE_MODEL)`
2. Insert the RENTAL_PRICE_RANGE table.
3. Insert the following joins and set their cardinality:

Join	Cardinality
CLIENT.CLIENT_ID = RENTAL.CLIENT_ID	1:N
RENTAL.SALE_ID = RENTAL_MODEL.SALE_ID	1:N
SHOWROOM.SHOWROOM_ID = RENTAL.SHOWROOM_ID	1:N
RENTAL.SALE_TYPE = 'R'	1:1
RENTAL.SALE_DATE between FINANCE_PERIOD.FP_START and FINANCE_PERIOD.FP_END	N:1
RENTAL_MODEL.MODEL_ID = MODEL.MODEL_ID	N:1
MODEL.MODEL_DAYRENT between RENTAL_PRICE_RANGE.PRICE_RANGE_MIN and RENTAL_PRICE_RANGE.PRICE_RANGE_MAX	N:1
RENTAL_MODEL.COLOR_ID = COLOR.COLOR_ID	N:1

Unit 5
Solution 12

Create a Loop that Needs to Be Resolved by a Context

Business Example

You need to create a loop that can be resolved by a context.

1. In the data foundation, create the following aliased tables:
   `RENTAL (alias of SALE) RENTAL_MODEL (alias of SALE_MODEL)`
   a) In your data foundation, right-click the SALE table.
   b) Choose *Insert* → *Alias Table*.
   c) Choose *OK*.
   d) In the Insert *Alias Tables* pop-up, double-click the default alias table name and change it to **`Rental`**.
   e) Repeat steps a - d, aliasing the SALE_MODEL table and changing the aliased table name to RENTAL_MODEL.
2. Insert the RENTAL_PRICE_RANGE table.
   a) In the data foundation, choose the *Connections* tab.
   b) To insert the RENTAL_PRICE_RANGE table, either in the insert toolbar, choose Insert tables, or from the menu bar, choose *Insert* → *Tables*.
3. Insert the following joins and set their cardinality:

Join	Cardinality
CLIENT.CLIENT_ID = RENTAL.CLIENT_ID	1:N
RENTAL.SALE_ID = RENTAL_MODEL.SALE_ID	1:N
SHOWROOM.SHOWROOM_ID = RENTAL.SHOWROOM_ID	1:N
RENTAL.SALE_TYPE = 'R'	1:1
RENTAL.SALE_DATE between FINANCE_PERIOD.FP_START and FINANCE_PERIOD.FP_END	N:1
RENTAL_MODEL.MODEL_ID = MODEL.MODEL_ID	N:1

Join	Cardinality
MODEL.MODEL_DAYRENT between RENTAL_PRICE_RANGE.PRICE_RANGE_MIN and RENTAL_PRICE_RANGE.PRICE_RANGE_MAX	N:1
RENTAL_MODEL.COLOR_ID = COLOR.COLOR_ID	N:1

a) In the data foundation view of Motors_xx data foundation, from the *Insert* menu choose *Insert Join*.
The *Edit Join* dialog box appears.

b) In the *Edit Join* dialog box, define the join properties by clicking on the table and column of the first table, and the column of the second table.
You can use this method for all the joins that have a Table1.Column = Table2.Column syntax.

c) Choose *Detect* and confirm that the suggested cardinality is correct.

d) Choose *OK*.

LESSON SUMMARY
You should now be able to:

- Resolve loops using contexts

Unit 5
Lesson 4

Detecting Contexts

LESSON OVERVIEW
This lesson determines how to detect contexts.

LESSON OBJECTIVES
After completing this lesson, you will be able to:

- Detect contexts

Context Detection

The Data Foundation Designer detects contexts by identifying tables that have only the many ends of joins attached.

No joins flowing back from one-to-many are included. To help to see the flow of contexts within a structure, you can arrange the tables so that all the joins flow as many-to-one from left to right across the structure.

Included in the context are all the tables that can be reached by following the flow from many-to-one (N -1). Tables that can only be reached by flowing back from one-to-many (1-N) are not included in the context.

In the Sales and Rental example, you can follow two different paths from the Client table to the Model table.

- By way of Rental and Rental_Model:

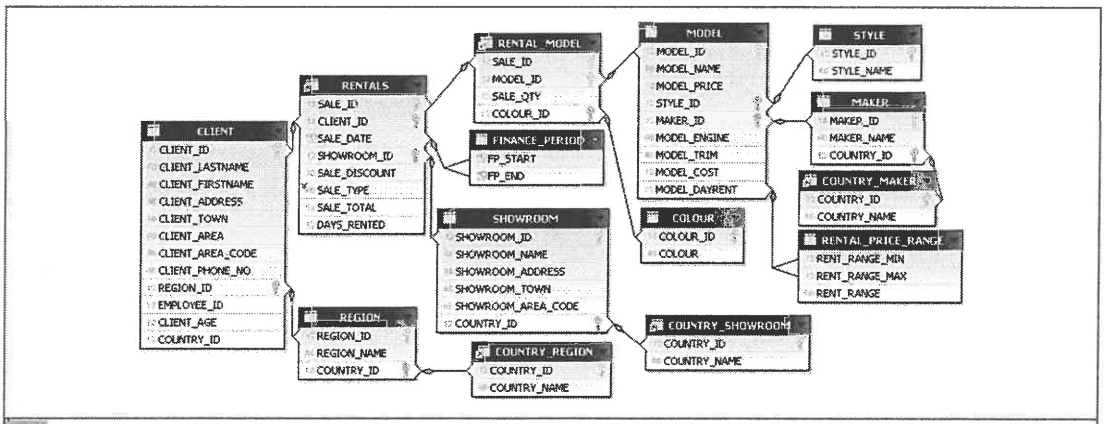


Figure 31: Rental Context Path

- By way of Sale and Sale_Model:

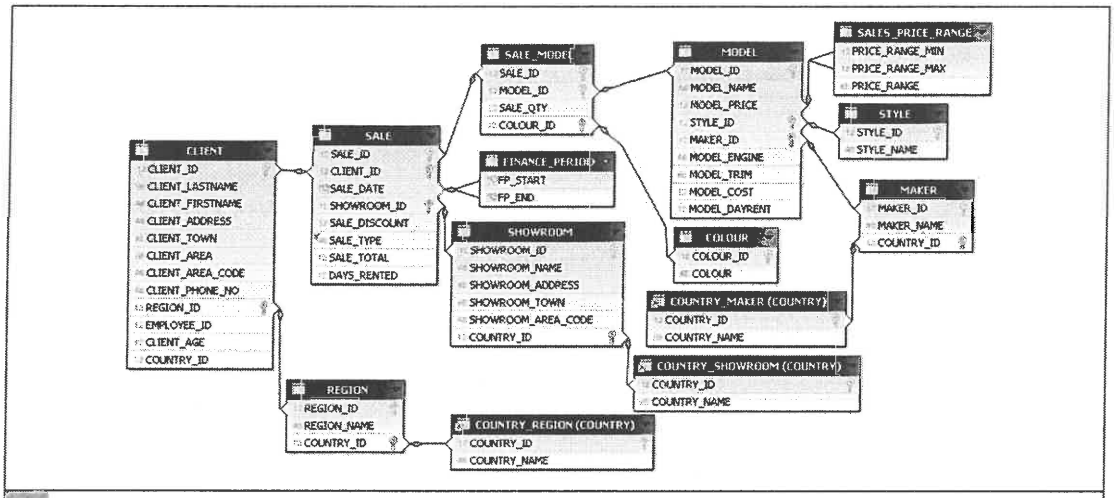


Figure 32: Sales Context Path

Each context represents what may be inferred in a single SELECT statement. Any query that infers some SQL code exclusive to one context and some exclusive to the other infers two separate SELECT statements.

> Note:
> The name of the context is defined by the table with only the many (N) end of joins attached to it.

You then create different sets of objects from the tables in the different contexts. As a result, users can run sales or rentals queries, depending on the objects they select.

> Note:
> Every join (except shortcut joins) must exist in at least one context.

Two methods are possible to detect contexts:

1. Use the main menu of the *Information Design Tool* → *Actions* → *Detect* → *Contexts*.

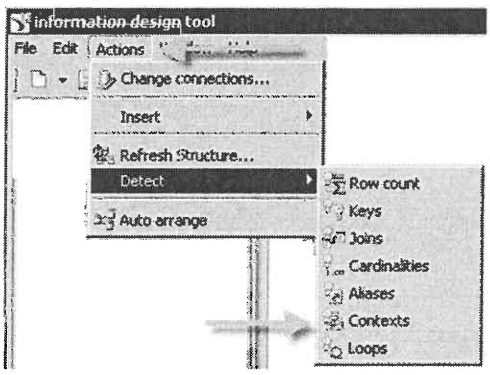


Figure 33: Detect Context from Information Design Tool

2. Use the navigation tabs of your Data Foundation. Select *Aliases and Contexts* on the top of your menu.

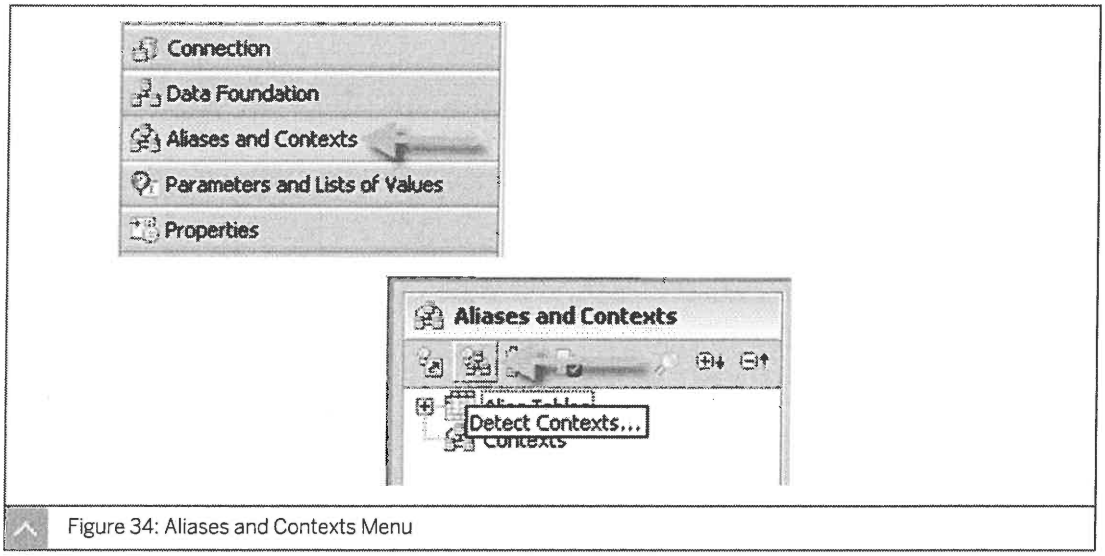


Figure 34: Aliases and Contexts Menu

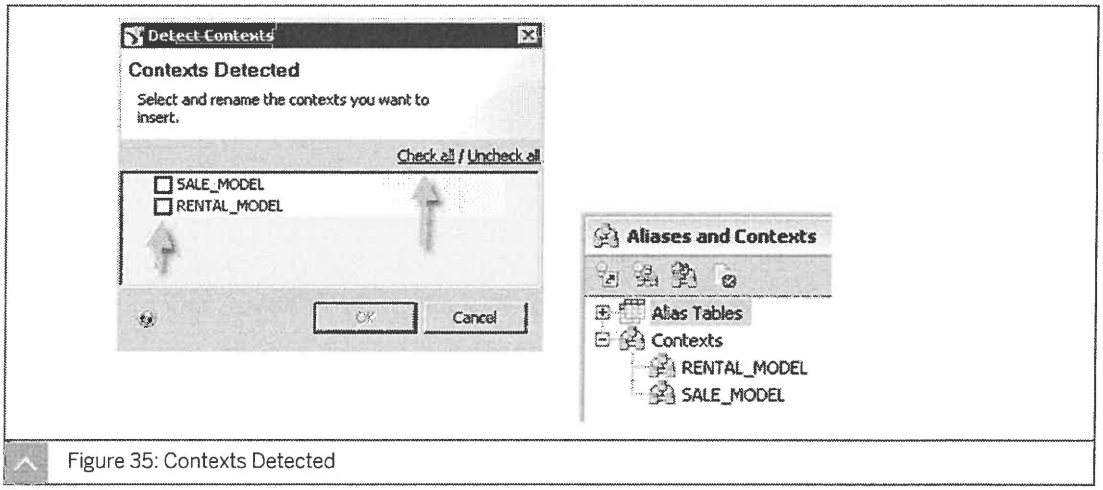


Figure 35: Contexts Detected

The *Detect Contexts* window appears. Select the contexts and click *OK*. The accepted contexts are displayed in the Contexts table of your Data Foundation.

A single click on the context name displays the tables that are involved into the context. The green plus icon in the visualization indicates the tables involved in the select context, and a red minus icon indicates those tables in the opposite context.

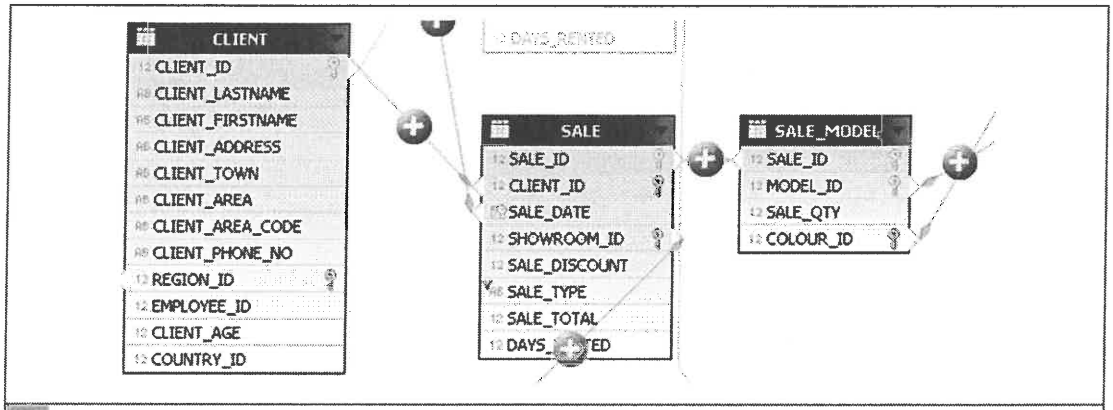


Figure 36: Context Visualization

Unit 5
Exercise 13

Detect Contexts

Business Example

You need to resolve a previously created loop.

1. Detect loops using the main menu.
2. Resolve the loops by using context detection to determine and accept contexts.

Unit 5
Solution 13

Detect Contexts

Business Example

You need to resolve a previously created loop.

1. Detect loops using the main menu.
   a) From the main menu, choose *Actions* → *Detect Loops*.
   b) In the *Loops* pane, choose *Refresh*.
   The pane tells you the loop can be resolved by contexts.
2. Resolve the loops by using context detection to determine and accept contexts.
   a) From the *Aliases and Contexts* pane, choose *Detect Contexts*.
   b) Confirm both contexts and choose *OK*.

LESSON SUMMARY

You should now be able to:

- Detect contexts

Unit 5
Lesson 5

Editing Contexts

LESSON OVERVIEW
You can edit the context definition for each context. This lesson explains how to edit contexts.

LESSON OBJECTIVES
After completing this lesson, you will be able to:

- Edit contexts

Context Editing

By double-clicking on the context name, the *Edit Contexts* window appears. Now you are able to edit the context definition for each context. Enter a meaningful description to the context because it might be prompted to the end-user.

If you double-click one of the join expressions, you can change the status of the join. It can be one of the following:

- Included - the join is specifically included in the context
- Excluded - the join is specifically excluded from the context
- Neutral - the join applies to multiple contexts

  With this option you are able to define the individual context.

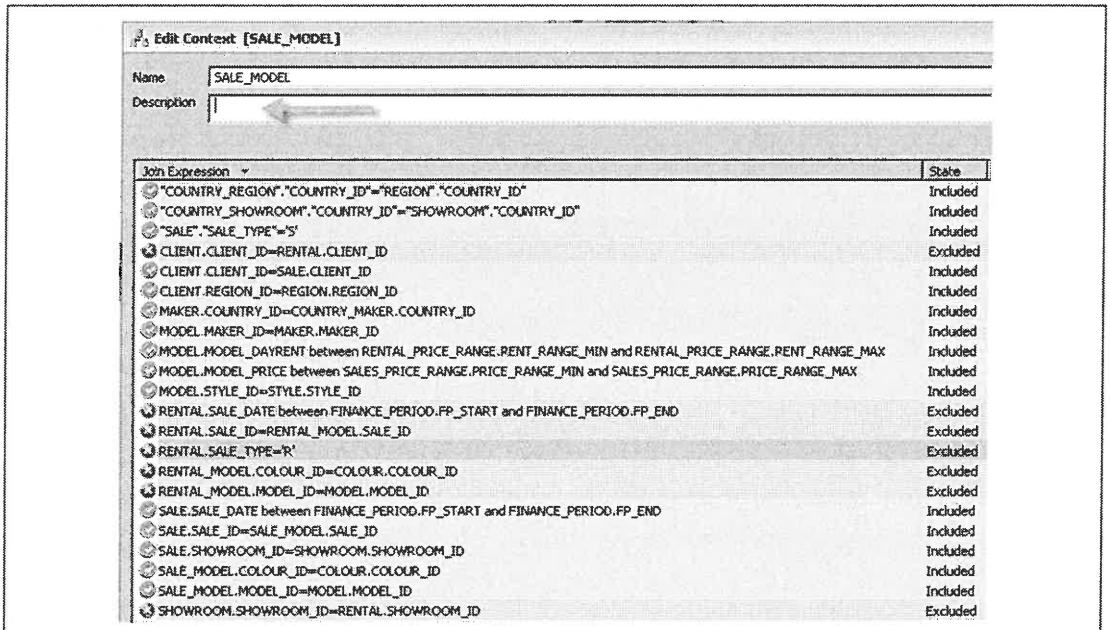


Figure 37: Editing a Context

Unit 5
Exercise 14

Edit Contexts

Business Example

Edit each context to make it user friendly as well as include all appropriate joins.

1. Edit each context to make it user friendly and include all appropriate joins.

Hint:
Any joins referencing Sales, Sales_Model, and the Sales_Price_Range tables are excluded from the Rentals context. The same is true for the Sales context -- exclude any join referring to the Rental, Rental_Model, and the Rental_Price_Range tables. Any join that applies to both contexts (for example, the join between the REGION and the CLIENT tables) is qualified as Included in each context. While these joins could be qualified as Neutral, to avoid any future problems with the data foundation and business layer, all joins on the data foundation are qualified as either Included or Excluded.

Context	Field	Value
RENTAL_MODEL	Name	Change to Rentals (RENTAL_MODEL)
	Description	Provides information on rental transactions.
	List of joins	• Exclude MODEL.MODEL_PRICE between SALES_PRICE_RANGE.PRICE_RANGE_MIN and SALES_PRICE_RANGE.PRICE_RANGE_MAX join • Include RENTAL.SALE_TYPE = 'R' join • Include MODEL.MODEL_DAYRENT BETWEEN RENTAL_PRICE_RANGE.RENT_RANGE_MIN AND RENT-

Context	Field	Value
		AL_PRICE_RANGE.RENT _RANGE_MAX • Include MOD-EL.STYLE_ID=STYLE.ST YLE_ID • Exclude SALE.SALE_TYPE = 'S' join • Include all joins between the following tables: - COUNTRY - REGION - CLIENT - SHOWROOM_COUN-TRY - SHOWROOM - COUNTRY_MAKER - MAKER - The shortcut join be-tween COUNTRY and CLIENT, if you have it.
SALE_MODEL	Name	Sales (SALE_MODEL)
	Description	Provides information on sales transactions.
	List of joins	• Exclude MODEL.MOD-EL_PRICE between RENT-AL_PRICE_RANGE.PRIC E_RANGE_MIN and RENT-

Context	Field	Value
		AL_PRICE_RANGE.PRICE_RANGE_MAX join • Include SALE.SALE_TYPE = 'S' join • Exclude RENTAL.SALE_TYPE = 'R' join • Include MODEL.MODEL_PRICE BETWEEN SALES_PRICE_RANGE.PRICE_RANGE_MIN AND SALES_PRICE_RANGE.PRICE_RANGE_MAX and • Include MODEL.STYLE_ID=STYLE.STYLE_ID • Include all joins between the following tables: - COUNTRY - REGION - CLIENT - SHOWROOM_COUNTRY - SHOWROOM - COUNTRY_MAKER - MAKER - The shortcut join between COUNTRY and CLIENT, if you have it.

Note:
When new tables are added to your data foundation, new contexts may be created. The Detect Contexts tool detects contexts that you have already accepted but edited. To avoid overwriting existing contexts, as the best practice, add the original table name in brackets after your customized context name. For example, enter RENTALS (RENTAL_MODEL), and SALES (SALE_MODEL). That way, you can easily recognize that customization has been applied to a particular context, so reacceptance of it is not necessary.

Unit 5
Solution 14

Edit Contexts

Business Example

Edit each context to make it user friendly as well as include all appropriate joins.

1. Edit each context to make it user friendly and include all appropriate joins.

Hint:
Any joins referencing Sales, Sales_Model, and the Sales_Price_Range tables are excluded from the Rentals context. The same is true for the Sales context -- exclude any join referring to the Rental, Rental_Model, and the Rental_Price_Range tables. Any join that applies to both contexts (for example, the join between the REGION and the CLIENT tables) is qualified as Included in each context. While these joins could be qualified as Neutral, to avoid any future problems with the data foundation and business layer, all joins on the data foundation are qualified as either Included or Excluded.

Context	Field	Value
RENTAL_MODEL	Name	Change to Rentals (RENTAL_MODEL)
	Description	Provides information on rental transactions.
	List of joins	• Exclude MODEL.MODEL_PRICE between SALES_PRICE_RANGE.PRICE_RANGE_MIN and SALES_PRICE_RANGE.PRICE_RANGE_MAX join • Include RENTAL.SALE_TYPE = 'R' join • Include MODEL.MODEL_DAYRENT BETWEEN RENTAL_PRICE_RANGE.RENT_RANGE_MIN AND RENT-

Context	Field	Value
		AL_PRICE_RANGE.RENT_RANGE_MAX • Include MODEL.STYLE_ID=STYLE.STYLE_ID • Exclude SALE.SALE_TYPE = 'S' join • Include all joins between the following tables: - COUNTRY - REGION - CLIENT - SHOWROOM_COUNTRY - SHOWROOM - COUNTRY_MAKER - MAKER - The shortcut join between COUNTRY and CLIENT, if you have it.
SALE_MODEL	Name	Sales (SALE_MODEL)
	Description	Provides information on sales transactions.
	List of joins	• Exclude MODEL.MODEL_PRICE between RENTAL_PRICE_RANGE.PRICE_RANGE_MIN and RENT-

Context	Field	Value
		AL_PRICE_RANGE.PRICE_RANGE_MAX join • Include SALE.SALE_TYPE = 'S' join • Exclude RENTAL.SALE_TYPE = 'R' join • Include MODEL.MODEL_PRICE BETWEEN SALES_PRICE_RANGE.PRICE_RANGE_MIN AND SALES_PRICE_RANGE.PRICE_RANGE_MAX and • Include MODEL.STYLE_ID=STYLE.STYLE_ID • Include all joins between the following tables: - COUNTRY - REGION - CLIENT - SHOWROOM_COUNTRY - SHOWROOM - COUNTRY_MAKER - MAKER - The shortcut join between COUNTRY and CLIENT, if you have it.

Note:
When new tables are added to your data foundation, new contexts may be created. The Detect Contexts tool detects contexts that you have already accepted but edited. To avoid overwriting existing contexts, as the best practice, add the original table name in brackets after your customized context name. For example, enter RENTALS (RENTAL_MODEL), and SALES (SALE_MODEL). That way, you can easily recognize that customization has been applied to a particular context, so reacceptance of it is not necessary.

a) In the *Aliases and Contexts* pane, expand the *Contexts*.

b) Double-click the RENTAL_MODEL context.

c) Change the name of the context to **`Rentals (RENTAL_MODEL)`**.

d) In the *Description*, enter the following text:
`Provides information on rental transactions.`

e) To mark it as Excluded, in the list of joins for the context, click the MODEL.MODEL_PRICE between SALES_PRICE_RANGE.PRICE_RANGE_MIN and SALES_PRICE_RANGE.PRICE_RANGE_MAX joins until the join state is Excluded.

f) To mark it as Included, below the *Description*, in the list of joins for the context, click the RENTAL.SALE_TYPE = 'R' join until the join state is Excluded.

g) To mark it as Excluded, in the list of joins for the context, click the SALE.SALE_TYPE = 'S' join until the join state is Excluded.

h) For the remaining neutral joins, click each join until the State is Included.

i) Repeat steps a - g with the SALE_MODEL context, naming the context Sales, and entering a similar but sales-related description as well as including all sales-related joins and excluding any rental-related joins.

LESSON SUMMARY
You should now be able to:

- Edit contexts

Unit 5
Lesson 6

Testing Contexts

LESSON OVERVIEW
You can test the context integrity. This lesson explains how to validate updated contexts.

LESSON OBJECTIVES
After completing this lesson, you will be able to:

- Test contexts

Context Testing

You can test the context integrity. Use the Aliases and Context menu of your Data Foundation and select the *Check Context Integrity* button.

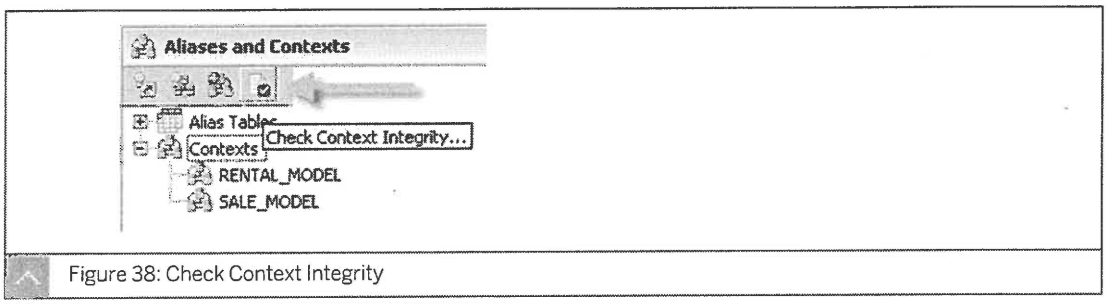


Figure 38: Check Context Integrity

You can also create a context manually by selecting the *Insert Context* icon on your Data Foundation menu. Double-click the new context name and click on each join that you want to include to your context. Rename the context and exclude the joins, which are not part of the context.

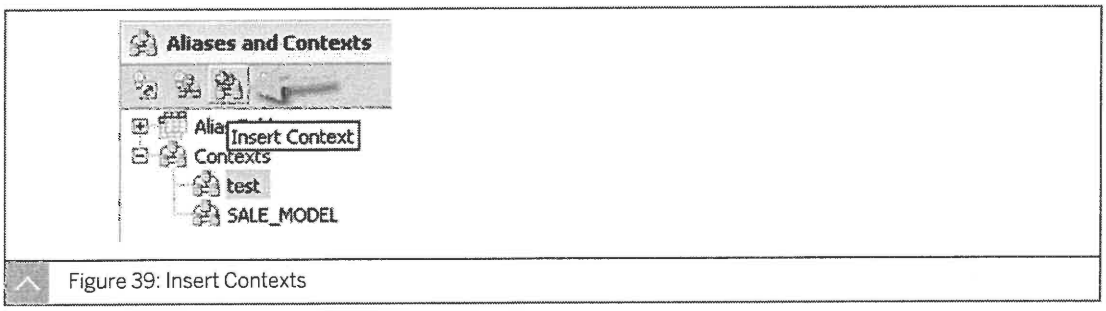


Figure 39: Insert Contexts

Unit 5
Exercise 15

Test Contexts

Business Example

Test that the contexts are working as expected.

1. In the business layer, create a folder called Rentals and position it above the Sales folder.
2. In the Rentals folder, create the following subfolders.
   - Rental Details
   - Rental Dates
   - Rental Figures
3. In the appropriate subfolder of the Rentals folder, create the following objects.

Folder	Object Name	SELECT Statement	Object Description
Rental Details	Rental Invoice ID	RENTAL.SALE_ID	Unique Invoice ID Number
Rental Dates	Rental Date	RENTAL.SALE_DATE	First day of Rental
Rental Figures	Rental Revenue	sum(RENTAL.DAYS_RENTED * RENTAL) MODEL.SALE_QTY * MODEL.MODEL_DAY-RENT * ((100-RENTAL.SALE_DISCOUNT)/100))	Total Rental Invoice Value

Hint:
To distinguish the Sales invoice object from the Rental invoice object, under the *Sales Details*, rename the Invoice ID Number as **`Sales Invoice ID Number`**. This allows the end user to distinguish the transaction type of the invoice.

4. In the Car folder, create a subfolder called Day Rental Charges, and then populate the subfolder with the objects in the following table and save your business layer:

Object Name	SELECT Statement	Object Description
Day Rental Range	RENTAL_PRICE_RANGE.RENT_RANGE	Description of Rental Charge banding

Object Name	SELECT Statement	Object Description
Day Rental Charge	MODEL.MODEL_DAYRENT	Standard Day Rental Charge

Unit 5
Solution 15

Test Contexts

Business Example

Test that the contexts are working as expected.

1. In the business layer, create a folder called Rentals and position it above the Sales folder.
   a) In the business layer drawer, right-click and choose *New* → *Folder*.
   b) Name the folder `Rentals` and drag it above the Sales folder.
2. In the Rentals folder, create the following subfolders.
   - Rental Details
   - Rental Dates
   - Rental Figures

   a) In the Rentals folder, create the subfolders as you have done in previous exercises.
3. In the appropriate subfolder of the Rentals folder, create the following objects.

Folder	Object Name	SELECT Statement	Object Description
Rental Details	Rental Invoice ID	RENTAL.SALE_ID	Unique Invoice ID Number
Rental Dates	Rental Date	RENTAL.SALE_DATE	First day of Rental
Rental Figures	Rental Revenue	sum(RENTAL.DAYS_RENTED * RENTAL) MODEL.SALE_QTY * MODEL.MODEL_DAY-RENT * ((100-RENTAL.SALE_DISCOUNT)/ 100))	Total Rental Invoice Value

Hint:
To distinguish the Sales invoice object from the Rental invoice object, under the *Sales Details*, rename the Invoice ID Number as `Sales Invoice ID Number`. This allows the end user to distinguish the transaction type of the invoice.

a) In the appropriate subfolder of the Rentals folder, right-click and select *New* → *Dimension* (or *New* → *Measure*, as appropriate).

The Rentals folder and subfolders should look very similar to the Sales folder and subfolders. The difference is that the objects in the Rentals folder all related to rental transactions.

b) Using the information in the table in step 3, name the object and complete the SELECT statement.

4. In the Car folder, create a subfolder called Day Rental Charges, and then populate the subfolder with the objects in the following table and save your business layer:

Object Name	SELECT Statement	Object Description
Day Rental Range	RENT-AL_PRICE_RANGE.RENT_RANGE	Description of Rental Charge banding
Day Rental Charge	MODEL.MODEL_DAYRENT	Standard Day Rental Charge

a) Using the information in the table in step 4, in the Car folder, create a subfolder called Day Rental Charges, and then populate the subfolder with the objects, just as in steps 3a and 3b.

b) Save your business layer.

LESSON SUMMARY
You should now be able to:

- Test contexts

Unit 5
Lesson 7

Resolving Recursive Loops

LESSON OVERVIEW
This lesson shows how to resolve recursive loops to uncover potential problems.

LESSON OBJECTIVES
After completing this lesson, you will be able to:

- Resolve recursive loops

Recursive Loops

A join does not necessarily involve two different tables; you can join a table to itself, thus creating a self-referencing join. A self-referencing join is a join from one column of a table to another column of the same table. Joining a table to itself can be useful when you want to compare values in a column to other values in a different column in the same table.

A classic example of when such a join is required is when you want to report on the hierarchical structure of an organization using a Personnel database. Typically, all employee records are held in a single table, irrespective of employee status. Therefore, a self-referencing join is required to report on the hierarchical relationship between those employees. This self-referencing join is effectively a recursive loop; the path forms a closed circuit. However, you cannot resolve this loop by using the usual method of detecting the cardinalities and then detecting aliases, because the cardinality detection tool cannot work on a self-referencing join. Moreover, a structure expressed this way does not infer the correct SQL.

LESSON SUMMARY
You should now be able to:

- Resolve recursive loops

Unit 5

Learning Assessment

1. What is a loop?

2. Loops can exist in both a Data Foundation schema and in the database.

   Determine whether this statement is true or false.

   - ☐ True
   - ☐ False

3. How does an alias break a loop?

4. What is a context?

5. Select the correct menu path for detecting contexts.

   Choose the correct answer.

   - ☐ A Information Design Tool > Aliases and Contexts > Detect
   - ☐ B Information Design Tool > Contexts > Detect
   - ☐ C Information Design Tool > Actions > Detect > Contexts
   - ☐ D Information Design Tool > Aliases and Contexts > Detect > Contexts

6. Double click a context name to access the *Edit Contexts* window.

   Determine whether this statement is true or false.

   ☐ True

   ☐ False

7. Why is it a good idea to edit contexts?

   Choose the correct answer.

   ☐ A To provide detail on the context

   ☐ B To make it user friendly and include all appropriate joins

   ☐ C To access joins for editing

   ☐ D To change the schema

8. Use the Test Context Integrity button to check the context integrity.

   Determine whether this statement is true or false.

   ☐ True

   ☐ False

9. What is a recursive loop?

   _____

   _____

   _____

10. Use the Cardinality Detection Tool to resolve recursive loops.

    Determine whether this statement is true or false.

    ☐ True

    ☐ False

Learning Assessment - Answers

1. What is a loop?

   A loop is a join path issue that arises from the way that tables are related in a relational database. A loop exists when the joins between tables form a closed path.

2. Loops can exist in both a Data Foundation schema and in the database.

   Determine whether this statement is true or false.

   - [ ] True
   - [x] False

3. 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.

4. What is a context?

   A context is a list of joins that define a path for a query. A context resolves a loop by defining a set of joins that define one specific path through tables in a loop. It ensures that joins are not included from different paths within the same SQL query.

5. Select the correct menu path for detecting contexts.

   Choose the correct answer.

   - [ ] A Information Design Tool > Aliases and Contexts > Detect
   - [ ] B Information Design Tool > Contexts > Detect
   - [x] C Information Design Tool > Actions > Detect > Contexts
   - [ ] D Information Design Tool > Aliases and Contexts > Detect > Contexts

6. Double click a context name to access the *Edit Contexts* window.

   Determine whether this statement is true or false.

   [X] True

   [ ] False

7. Why is it a good idea to edit contexts?

   Choose the correct answer.

   [ ] A To provide detail on the context

   [X] B To make it user friendly and include all appropriate joins

   [ ] C To access joins for editing

   [ ] D To change the schema

8. Use the Test Context Integrity button to check the context integrity.

   Determine whether this statement is true or false.

   [ ] True

   [X] False

9. What is a recursive loop?

   A recursive loop is a self referencing join, which joins a table to itself. A self-referencing join (or recursive loop) is a join from one column of a table to another column of the same table.

10. Use the Cardinality Detection Tool to resolve recursive loops.

    Determine whether this statement is true or false.

    [ ] True

    [X] False

UNIT 6 Data Restrictions

UNIT OBJECTIVES

- Define data restrictions
- Apply restrictions to objects
- Apply restrictions using an alternative method
- Restrict by inferring matching data between tables
- Restrict data using a filter object

Unit 6
Lesson 1

Defining Data Restrictions

LESSON OVERVIEW
This lesson defines data restrictions, which are conditions in SQL that set criteria to limit the data returned by a query. You define restrictions on objects to limit the data available to users.

LESSON OBJECTIVES
After completing this lesson, you will be able to:

- Define data restrictions

Data Restrictions

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 depend on the data requirements of the target user. A user might not need access to all the values returned by an object. You might also want to restrict user access to certain values for security reasons.

The WHERE clause in an SQL statement restricts the number of rows that are returned by the query. So far in your Data Foundation design work, the restrictions have only been populated by the joins you made between the tables. The joins restrict the result sets, based on equality between tables, and prevent Cartesian products. You can also use the WHERE clause to restrict further the data that is returned in a query when you want to limit certain users to query on a subset of the data.

Drawback to Applying Restrictions to Objects

If two or more similarly restricted objects are included in the same query, the conflict between the WHERE clauses causes no data to be returned. Consider the situation if a user wanted data for UK Clients and US Clients. You might think that including both the UK Clients and US Clients objects would meet that need. However, the inferred SQL for the query would include the following two lines:

- (COUNTRY_REGION.COUNTRY_NAME = 'United Kingdom')
  AND (COUNTRY_REGION.COUNTRY_NAME = 'USA')

Since no country satisfies both these conditions, no data is returned.

Data Restriction Types

The following types of data restrictions can be used:

Types of Data Restrictions

- Data restrictions to objects
- Data restrictions using filters

- Data restrictions to tables

Within the design of a Data Foundation or a Business Layer, you can either:

- Force restrictions, which the end user cannot override: object, table, conditional SELECT, and additional joins
- Provide optional restrictions, which the end user can choose to apply: condition objects.

There are often problems associated with forced restrictions. You are advised only to force restrictions where they are absolutely necessary. Remember that users can apply conditions themselves in their Business Objects querying tools.

When to Use Each Method

You can use the following guidelines to set restrictions in a Business Layer or Data Foundation:

- Avoid using Where clauses in object definitions. If you use a Where clause, 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 folder and objects view of the Business Layer pane.
- Use column filter 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 Data Foundation view.

Note:
Apart from self restricting joins, do 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. Ensure that all joins are visible in the data foundation. This checking ensures that all joins are available to the Data Foundation automatic detection tool.

LESSON SUMMARY
You should now be able to:

- Define data restrictions

Unit 6
Lesson 2

Applying Mandatory Data Restrictions

LESSON OVERVIEW
This lesson explains how to apply mandatory restrictions on objects in the Business Layer.

LESSON OBJECTIVES
After completing this lesson, you will be able to:

- Apply restrictions to objects
- Apply restrictions using an alternative method
- Restrict by inferring matching data between tables

Restriction Application Methods

On the WHERE Clause

To ensure that a restriction is always inferred when a particular object is used in an end-user query, place the restriction in the Where field of the *Dimension Properties* tab and use the SQL assistant. You can do this restriction when you create the object or you can add it later.

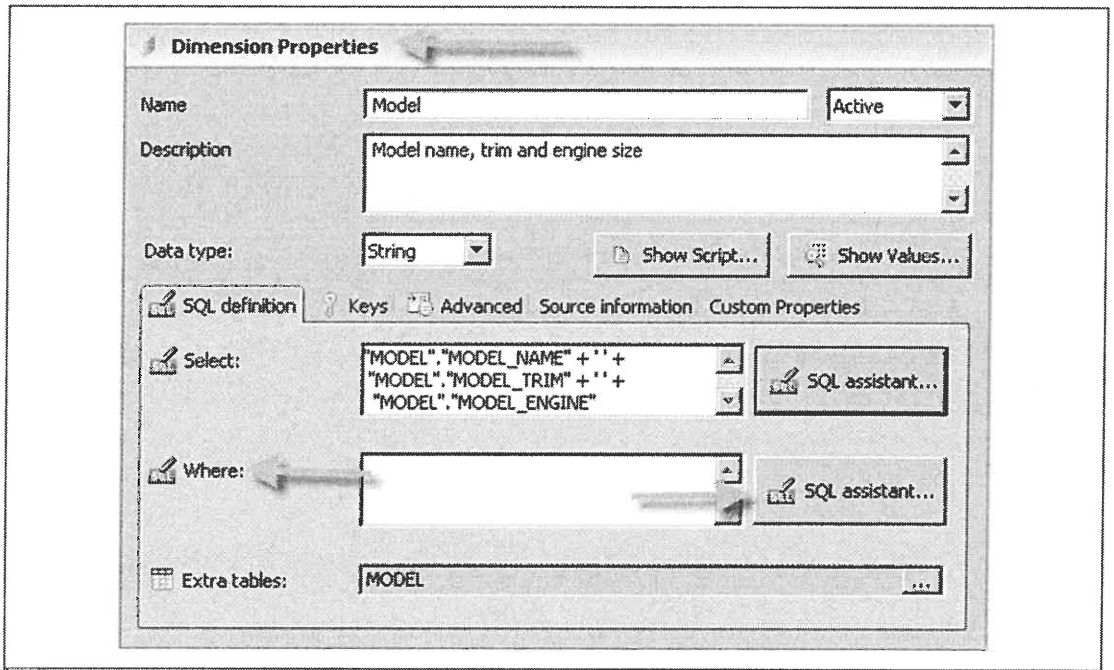


Figure 40: Dimension Properties

For example, consider the situation where users of Motors make queries of only those cars that can be rented. In the Model table in the underlying Motors database, the distinguishing factor between cars that can be rented and those stocked for sale is that the Model_Dayrent

column contains data for rental cars and is null for sale-only cars. To create an object to list cars for rental, the object would have to include the restriction:

MODEL.MODEL_DAYRENT IS NOT NULL

Alternative Restriction Method

There is an alternative to applying restrictions to objects without using WHERE clauses. You have multiple objects, but you avoid the conflicts that affect the return of data in queries.

This method involves using a conditional SELECT clause for the object instead of a WHERE clause.

For example, if you want to force users to select financial results by year, you could create a series of Sales Revenue objects (one for each year). Each object would be edited, starting from the standard sum aggregate used in the basic Sales Revenue SELECT statement:

```
sum(SALE_MODE.SALE_QTY*MODEL.MODEL_PRICE*(100-SALE.SALE_DISCOUNT)/100)
```

You would apply the condition for each year using the database function that applies IF THEN ELSE logic.

For Sales Revenue 2003, the SELECT statement appears as follows:

```
sum(CASE{fn year(SALE.SALE_DATE)} WHEN 2003 THEN
(SALE_MODE.SALE_QTY*MODEL.MODEL_PRICE*((100-SALE.SALE_DISCOUNT)/
100))ELSE 0 END)
```

Note:
The Else value 0 is optional.

For Sales Revenue 2004, the SELECT statement appears as follows:

```
sum(CASE{fn year(SALE.SALE_DATE)} WHEN 2004 THEN
(SALE_MODE.SALE_QTY*MODEL.MODEL_PRICE*((100-SALE.SALE_DISCOUNT)/
100))ELSE 0 END)
```

Note:
Many databases support the CASE function. Consult the documentation provided by your database vendor to see what types of conditional functions are supported.

After you have created or edited the objects, test them individually and together in a single query. When you view the SQL to check whether the inferred SELECT statement includes the conditional SELECT syntaxes, the SQL appears as follows:

```
SELECT DISTINCT
   CLIENT.CLIENT_LASTNAME+','+CLIENT.CLIENT_FIRSTNAME,
   sum(CASE{fn year(SALE.SALE_DATE)} WHEN 2003 THEN
(SALE_MODE.SALE_QTY*MODEL.MODEL_PRICE*((100-SALE.SALE_DISCOUNT)/
100))ELSE 0 END),
   sum(CASE{fn year(SALE.SALE_DATE)} WHEN 2004 THEN
(SALE_MODE.SALE_QTY*MODEL.MODEL_PRICE*((100-SALE.SALE_DISCOUNT)/
100))ELSE 0 END)
FROM
   CLIENT,
   MODEL,
   SALE,
   SALE_MODEL
WHERE
```

```
   (CLIENT.CLIENT_ID=SALE.CLIENT_ID)
   AND (SALE.SALE_ID=SALE_MODEL.SALE_ID)
   AND (SALE_MODEL.MODEL_ID=MODEL.MODEL_ID)
   AND (SALE.SALE_TYPE='S')
GROUP BY
   CLIENT.CLIENT_LASTNAME+','+CLIENT.CLIENT_FIRSTNAME
```

When the query is run, the report looks similar to this example:

Report 1

Client Name	2003 Sales Revenue	2004 Sales Revenue
Abaunza, Howard	0	$13,250.00
Answersen, Pamela	0	$22,800.00
Barkley, John	0	$45,210.00
Barry, John	$105,444.00	0
Benson, Jack	0	$24,300.00
Blacksen, Karen	$22,455.00	0
Bloomhead, Peter	$71,605.00	0
Blumenhein, Joe	0	$47,745.00
Bonnehammer, John	0	$155,005.00
Brent, Paul	$145,005.00	$170,959.50

Figure 41: Report Result from Conditional SELECT Statement

The conditional SELECT statements remove the problem of the conflicting WHERE clauses. The data correctly shows the 2003 and 2004 Sales Revenue for each client.

Column Filters in the Data Foundation

If a table in your database has a flag that is used to switch between two or more domains, you can use this flag to apply restrictions at the table level.

For example, the Sale table in the Motors database has a column called Sale_Type, which is used to distinguish between sales transactions and rentals transactions. The flag is set to S for sales or R for rentals.

If you do not apply any restriction to this table, users running queries for sales (with appropriate objects) receive a resulting report that includes data on rentals as well as sales. Therefore, the results are wrong.

With this restriction in place, the data returned is restricted to sales data, no matter where the table is used in the inferred SQL. For example, if the Sale table appears only in the FROM clause of the SQL, the restriction is still applied in the WHERE clause.

This is the main advantage of applying restrictions at the table level.

Inference Method

One specific mandatory filter is the Inference Method. 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.

In our example, the measure Number of Cars sold is based on the table "SALE_MODEL". The table "SALE_MODEL" is not filtered to "SALE_TYPE" ='S'.

If you do not select the table SALES where the column filter "SALE" ."SALE_TYPE" ='S' applies, then users running queries for sales (with appropriate objects) receive a resulting report that includes all data. Therefore, the results are wrong.

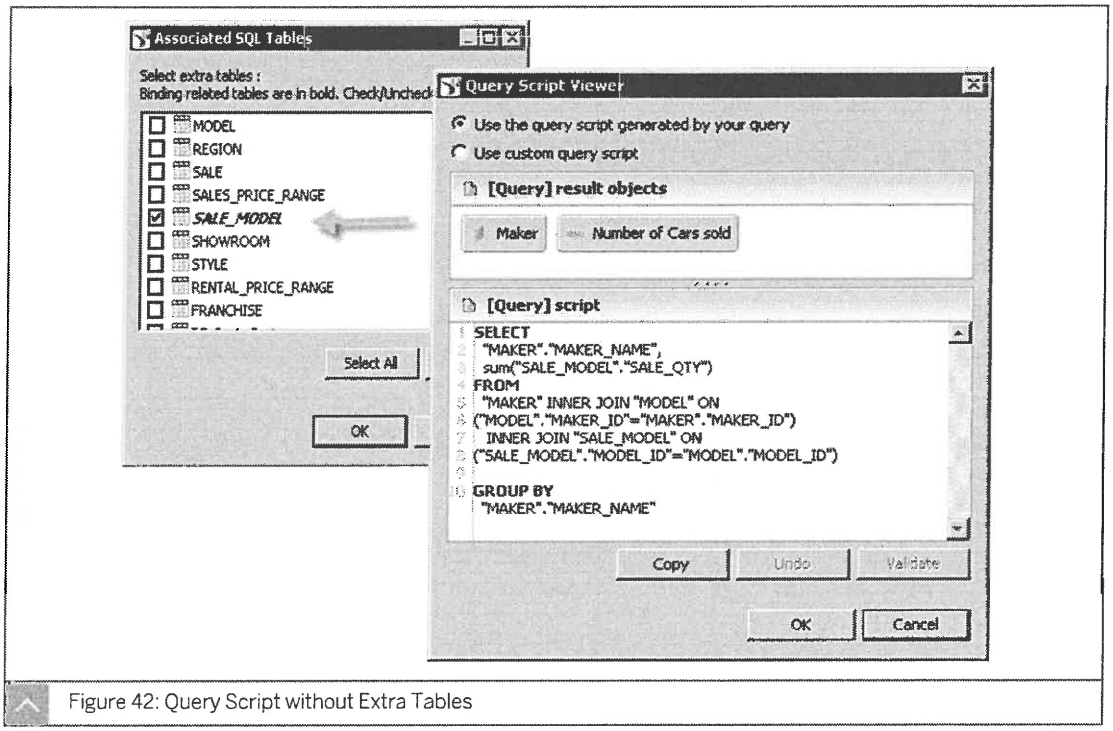


Figure 42: Query Script without Extra Tables

The SALES_MODEL table is marked because the object Select referenced this table. If you run a query with the objects Maker and Number of Cars sold, the following Query script is generated:

```
SELECT "MAKER"."MAKER_NAME", sum("SALE_MODEL"."SALE_QTY")
FROM "MAKER" INNER JOIN "MODEL" ON
"MODEL"."MAKER_ID"="MAKER"."MAKER_ID")
INNER JOIN "SALE_MODEL" ON ("SALE_MODEL"."MODEL_ID"="MODEL"."MODEL_ID")
GROUP BY "MAKER"."MAKER_NAME"
```

The restriction on the SALE table: SALE.SALE_TYP=S is missing; this table will return wrong values for the query.

LESSON SUMMARY

You should now be able to:

- Apply restrictions to objects
- Apply restrictions using an alternative method
- Restrict by inferring matching data between tables

Unit 6
Lesson 3

Applying Optional Data Restrictions

LESSON OVERVIEW
This lessons shows you how to apply optional data restrictions to objects by filtering.

LESSON OBJECTIVES
After completing this lesson, you will be able to:

- Restrict data using a filter object

Filter Object Definition

A filter object is a predefined Where clause that can be inserted into the Select statement inferred by objects in the Query pane. Filter objects are stored in the Business Layer. Each folder defined in the Business Layer can contain a filter object.

Using filter objects has the following advantages:

Filter Object Advantages

- Useful for complex or frequently used conditions.
- Gives users the choice of applying the condition.
- No need for multiple objects.

The only disadvantage for using filter 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 can make the filter object mandatory, but you then risk conflicting WHERE clauses again.

Using filter objects does not inherently solve the problem of conflicting WHERE clauses returning an empty data set. If a user runs a query that includes two filter objects that access the same data, the two conditions are still combined with the AND operator, so the two conditions are not met, and no data is returned. However, this problem is easily solved in the reporting tools simply by changing the word AND to OR..

Mandatory Filters

There are two types of mandatory filters:

- Universe: A universe mandatory filter has no dependency on the folder to which it belongs. A universe mandatory filter is included in the query independently of the objects (dimensions, measures, and attributes) 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.
- Folder: Folder mandatory filters appear only if an object of the folder used in the query. A folder mandatory filter is triggered when users:

- Add an object (dimension, measure, or attribute) 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 folder has been selected in the Result pane.
- Create a filter with an object (dimension, measure, or attribute) that belongs to a folder 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 end user reporting tool. By default, all mandatory filters are joined in the query with the AND operator. All subfolder inherit the mandatory filters from the parent folder.

Note, however:

- An object (dimension, measure, attribute) that references another object with the @Select function does not inherit the folder mandatory filter of the referenced object
- A WHERE clause of an object that references another object's Where clause with the @Where function does not inherit the folder mandatory filter of the referenced object
- A pre-defined filter that references another pre-defined filter or an object's Where clause with the @Where function does not inherit the folder mandatory filter of the referenced object

Unit 6
Exercise 16

Apply Data Restrictions

Business Example

The Motors_xx Universe will contain restrictions on objects, filter objects, and additional tables on an object to infer a join.

1. In the business layer, in the Car folder, create a Luxury Model dimension object.
   It should have the same SELECT as the Model object and the Where clause should designate the Price Range for luxury cars only.
2. In the Day Rental Charges subfolder, create a Model for the Rental dimension object.
   The Where clause is `MODEL.MODEL_DAYRENT IS NOT NULL.`
3. In the business layer, use the data in the following table to create two objects in the Rental Figures folder.
   Test the new objects by running a few queries against them in the *Query* drawer.

New Object Name	SELECT Statement
Rental Revenue 2003	`sum(CASE {fn year(RENTAL.SALE_DATE)} WHEN 2003 THEN (RENTAL.DAYS_RENTED*RENTAL_MODEL.SALE_QTY*MODEL.MODEL_DAYRENT*((100 - RENTAL.SALE_DISCOUNT)/100)) ELSE 0 END)`
Rental Revenue 2004	`sum(CASE {fn year(RENTAL.SALE_DATE)} WHEN 2004 THEN (RENTAL.DAYS_RENTED*RENTAL_MODEL.SALE_QTY*MODEL.MODEL_DAYRENT*((100 - RENTAL.SALE_DISCOUNT)/100)) ELSE 0 END)`

Hint:
You can copy the Rental Revenue object in the Rental Figures folder and paste it in the same folder. Then copy and paste the new Rental Revenue 2003 object and edit it to create the Rental Revenue 2004 object.

Note:
You cannot have 2 objects with the same name in the same folder. We change the new objects' names later.

4. In the Client folder named US Clients, create a filter.
5. Save your business layer.
6. By selecting the appropriate table using the *Tables* button, edit your business layer for the Showroom Country, Client Country, and Maker Country objects. Save your business layer.
7. Using the *Query* drawer, test the changes.

   View the script and note the results and choose *Refresh* to view the results. Note the following:

   - In the script there are additional joins in the FROM clause, restricting the data results to only those countries where clients live.
   - After choosing *Refresh*, there are only 5 countries listed, indicating only the countries where the clients live.

Unit 6
Solution 16

Apply Data Restrictions

Business Example

The Motors_xx Universe will contain restrictions on objects, filter objects, and additional tables on an object to infer a join.

1. In the business layer, in the Car folder, create a Luxury Model dimension object.
   It should have the same SELECT as the Model object and the Where clause should designate the Price Range for luxury cars only.
   a) Open the Car folder and copy the Model object.
   b) Paste the Model object in the Car folder and rename it to Luxury Models.
   c) Create the following statement in the WHERE clause:
   ```
   SALES_PRICE_RANGE.PRICE_RANGE = 'Luxury (<= 100,000)'
   ```


   Note:
   Luxury (<= 100,000)' is a string and must be typed in exact form. There is a space between "Luxury" and "<=" and "100,000)". While in the *SQL Assistant*, you can also expand the SALES_PRICE_RANGE table, select the PRICE_RANGE column and from the drop-down list, choose "Luxury (<=100,000)".

   d) Choose *OK*.
   e) Save your business layer.
2. In the Day Rental Charges subfolder, create a Model for the Rental dimension object.
   The Where clause is `MODEL.MODEL_DAYRENT IS NOT NULL.`
   a) Copy the Model object into the Day Rental Charges subfolder.
   b) Choose the copied Model and rename it to Model for Rental.
   c) Use the SQL assistant to enter the WHERE clause `MODEL.MODEL_DAYRENT IS NOT NULL.`
   d) Choose *OK*.
   e) Save your business layer.
3. In the business layer, use the data in the following table to create two objects in the Rental Figures folder.
   Test the new objects by running a few queries against them in the *Query* drawer.

New Object Name	SELECT Statement
Rental Revenue 2003	sum(CASE {fn year(RENTAL.SALE_DATE)} WHEN 2003 THEN (RENTAL.DAYS_RENTED*RENTAL_MODEL.SALE_QTY*MODEL.MODEL_DAYRENT*((100 - RENTAL.SALE_DISCOUNT)/100)) ELSE 0 END)
Rental Revenue 2004	sum(CASE {fn year(RENTAL.SALE_DATE)} WHEN 2004 THEN (RENTAL.DAYS_RENTED*RENTAL_MODEL.SALE_QTY*MODEL.MODEL_DAYRENT*((100 - RENTAL.SALE_DISCOUNT)/100)) ELSE 0 END)

Hint:
You can copy the Rental Revenue object in the Rental Figures folder and paste it in the same folder. Then copy and paste the new Rental Revenue 2003 object and edit it to create the Rental Revenue 2004 object.

Note:
You cannot have 2 objects with the same name in the same folder. We change the new objects' names later.

a) In the business layer, copy the Rental Revenue object in the Rental Figures folder and paste it in the same folder.

b) Highlight the object to edit its properties.

c) Edit the SELECT statement to contain the statement for Rental Revenue 2003 in the table in step 3.

Note:
The Else value 0 is optional.

d) Change the object name to **`Rental Revenue 2003`**.

e) Copy the Rental Revenue 2003 object and paste is in the same folder.

f) Edit the SELECT statement to contain the statement for Rental Revenue 2004 in the table in step 3.

g) Change the object name to **`Rental Revenue 2004`**.

h) Test the new objects by running a few queries using them and other objects in the Query drawer.

Be sure to use both the objects in the same query and note that there is not conflict in the WHERE clause because there is no WHERE-clause restriction based on the year.

4. In the Client folder named US Clients, create a filter.
   a) Choose the Client folder.
   b) Choose *Insert* → *Filter*.
   c) Name the filter US Clients.
   d) Use the SQL assistant on the Where clause to enter
      `COUNTRY_REGION.COUNTRY_NAME = 'USA'`
   e) Save your business layer.
5. Save your business layer.
6. By selecting the appropriate table using the *Tables* button, edit your business layer for the Showroom Country, Client Country, and Maker Country objects. Save your business layer.
   a) In the Information Design Tool, and in the business layer, open the Showroom folder and choose the Showroom Country object.
   b) In the *Tables* area on the *SQL definition* tab, choose the ellipsis button and place a check next to the Showroom Table.
   c) For the Client Country object located in the Client folder, repeat steps a - b, choosing the Client table.
   d) For the Maker Country object located in the Cars folder, repeat steps a - b, choosing the Maker table.
7. Using the *Query* drawer, test the changes.
   View the script and note the results and choose *Refresh* to view the results. Note the following:
   - In the script there are additional joins in the FROM clause, restricting the data results to only those countries where clients live.
   - After choosing *Refresh*, there are only 5 countries listed, indicating only the countries where the clients live.

   a) In the business layer, choose the *Query* drawer.
   b) From the Client folder, drag the Client Country object to the *Result Objects* pane.
   c) Choose *View Script* and note that there are additional joins in the FROM clause, restricting the data results to only those countries where clients live.
   d) Choose *Refresh*.
   e) Note that there are only 5 countries listed, indicating only the countries where the clients live.

LESSON SUMMARY

You should now be able to:

- Restrict data using a filter object

Learning Assessment

1. What is a restriction?

2. Which restriction is optional and applicable by end users?

   Choose the correct answer.

   - ☐ A Condition objects
   - ☐ B Object
   - ☐ C Table
   - ☐ D Conditional SELECT

3. What is a filter object?

4. What logic is applied when using the conditional SELECT restriction?

   Choose the correct answer.

   - ☐ A IF THEN SELECT
   - ☐ B IF THEN ELSE
   - ☐ C IF THEN OR
   - ☐ D IF THEN AND

5. What does the inference method do?

6. Which filter lets you restrict the values returned whenever the table is used in a query?

Choose the correct answer.

- ☐ A Table filter
- ☐ B Column filter
- ☐ C Row filter
- ☐ D Function filter

Learning Assessment - Answers

1. What is a restriction?

   A restriction is a condition in SQL that sets criteria to limit the data returned by a query.

2. Which restriction is optional and applicable by end users?

   Choose the correct answer.

   - [X] A Condition objects
   - [ ] B Object
   - [ ] C Table
   - [ ] D Conditional SELECT

3. What is a filter object?

   A filter object is a predefined Where clause that can be inserted into the Select statement inferred by objects in the Query pane.

4. What logic is applied when using the conditional SELECT restriction?

   Choose the correct answer.

   - [ ] A IF THEN SELECT
   - [X] B IF THEN ELSE
   - [ ] C IF THEN OR
   - [ ] D IF THEN AND

5. What does the inference method do?

   Inference limits the data returned for an object to values from the table inferred by the object that also match values in another table.

6. Which filter lets you restrict the values returned whenever the table is used in a query?

Choose the correct answer.

- [ ] A Table filter
- [x] B Column filter
- [ ] C Row filter
- [ ] D Function filter

UNIT 7 Lists of Values (LOV)

Lesson 1

UNIT OBJECTIVES

- Define a list of values
- Use the list of values editor
- Create static lists of values
- Create lists of values based on business layer objects
- Associate a list of values with a business object

Unit 7
Lesson 1

Providing a List of Values

LESSON OVERVIEW

This lesson explains how you can add, modify, or remove a list of values (LOV) for an object. It also introduces how to create a cascading list of values in the Business Layer.

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Define a list of values
- Use the list of values editor
- Create static lists of values
- Create lists of values based on business layer objects
- Associate a list of values with a business object

List of Values

A list of values (LOV) is a list that contains the data values associated with an object. A LOV allows users to choose values when creating a query filter or responding to a prompted query filter.

LOV is an independent component in the Business Layer or Data Foundation and is available to all business objects in the Business Layer. A LOV can be associated with an object at any time.

> Note:
> LOVs inserted in the Data Foundation are inherited by any Business Layer based on the Data Foundation. These LOVs cannot be edited in the Business Layer. Edit them in the Data Foundation.

List of Values Types

You can define the following types of lists of values:

Table 14: Types of Lists of Values

Type of LOV	Description
List of values based on business layer objects (available only in the business layer)	The LOV is based on either a query or a custom hierarchy that includes objects in the business layer. The list is based on the values returned by the query or the hierarchy values.

Type of LOV	Description
Static list of values	The LOV is based on a list of specified values entered manually or imported from a file.
List of values based on custom SQL	The LOV is based on the values returned by a specified SQL expression.

List of Values Options

The Options tab in list of values (LOV) properties lets you set user and query constraints on LOVs.

The following options are available:

Table 15: List of Values Options

Option	Description
Allow users to edit list of values	If selected, the LOV can be edited and personalized by users other than the designer.
Automatic refresh before use	If selected, the LOV is automatically refreshed each time the LOV is called. This refresh can effect performance each time the LOV is refreshed. If the LOV returns many values, disable this option.
Force users to filter values before use	If selected, the user running a query using this LOV is required to enter search criteria before receiving filtered values for the LOV. Only the values that match the search criteria are returned in the LOV. Characters used to define the matching criteria are: • * - Matches any number of characters, even zero characters. • ? - Matches exactly one character. • \ - Escapes the next character allowing you to search for a wildcard character.
Allow users to search values in the database	If selected, the user running a query using this LOV can search for a LOV value in the database. This option is useful when the user performs a search on partial LOV results.
Query Execution time out	If selected, limits the time in seconds that the LOV query runs.
Max number of rows	If selected, you can enter the maximum number of rows to be returned by the LOV query.

List of Values Column Properties

The list of values column properties consists of the following:

Table 16: List of Values Column Properties

Property	Description
Column Name	Lets you edit the column name.
Key Column	Lets you select a column to be the index-aware key.
Data Type	Lets you select the data type for the column.
Hidden	When selected, the column will not be displayed to the user. For example, you can hide a column that is only used as a key for another column.

List of Values Insertion Method

The list of values editor can be started from the Business Layer or Data Foundation editor tabs.

Note:
Lists of values (LOV) inserted in the Data Foundation are inherited by any Business Layer based on the Data Foundation. These LOVs cannot be edited in the Business Layer. You edit them in the Data Foundation.

1. Click the Parameters and Lists of Values tab in the browsing pane of the Data foundation or Business Layer editor.
2. Do one of the following:
   - To insert a LOV, click the *Insert List of Values* icon at the top of the Lists of values pane and select the type of LOV.
   - To edit a LOV, click the LOV name in the list. The properties for the LOV appear in the editor to the right of the Lists of values pane.
3. Edit properties and query options as required. The properties vary depending on the type of LOV.

Static Lists of Values

The LOV is based on a list of specified values entered manually or imported from a file.

Business Layer Object Lists of Values

There are two options for creating a list of values based on an object from the business layer:

- List of values based on the query panel.

- List of values based on a custom hierarchy.

To base the LOV on business layer objects:

1. In the *Definition* tab, select *List of values based on the query panel*.
2. Click *Edit Query*.
3. In the *Query Panel*, select objects and define query filters to define the query that returns the list of values required.
4. Select *Refresh* in the *Data Preview Panel* and see the result.
5. Click *OK*.

Unit 7
Exercise 17

Create a List of Values

Business Example

1. Choose the data foundation and create a list of values based on custom SQL.

   Name: Client Name and ID

   Build the SQL with the data in the following table:

Name	Build SQL with
Client Name and ID	• CLIENT.CLIENT_ID, CLIENT.FIRST_NAME, • CLIENT.LAST_NAME

   Preview your selection and validate the SQL expression. Save the data foundation.

2. To retrieve the names of the car manufacturers, create a list of values in the data foundation based on custom SQL.
   - Name: Car Makers

3. In the Business Layer, create a list of values base on a custom hierarchy based on the geography of the Client Name.
   - Name: Client Hierarchy
   - Use the list of values based on custom hierarchy option. Organize the dimensions by Country, Region, Town, Client Name
   - Preview your selection and save the Business Layer.

4. In the business layer, create a static list of values for European Countries..
   - Name: European Countries
   - Use the *Properties* tab and name the column Country
   - Add the following row values: Germany, Italy, Sweden, and the United Kingdom
   - Associate the European Countries as the list of values for the Maker Country object

Unit 7
Solution 17

Create a List of Values

Business Example

1. Choose the data foundation and create a list of values based on custom SQL.

   Name: Client Name and ID

   Build the SQL with the data in the following table:

Name	Build SQL with
Client Name and ID	• CLIENT.CLIENT_ID, CLIENT.FIRST_NAME, • CLIENT.LAST_NAME

   Preview your selection and validate the SQL expression. Save the data foundation.

   a) Choose *Data Foundation* and the *Parameter and List of Values* pane.

   b) Choose *Insert* → *List of values based on custom SQL*.

   c) Choose *Edit SQL* → *SQL Builder*, and expand the Client Table.

   d) Double-click Client ID, Client First Name, and Client Last name.

   e) Preview the selection and validate the SQL expression.

   f) Save the data foundation.

2. To retrieve the names of the car manufacturers, create a list of values in the data foundation based on custom SQL.

   • Name: Car Makers

   a) Choose *Data Foundation* and the *Parameter and List of Values* pane.

   b) Choose *Insert* → *List of values based on custom SQL*.

   c) Choose *Edit SQL* → *SQL Builder*, and expand the MAKER Table.

   d) Replace the * by double-clicking MAKER_NAME.

   e) After the FROM, double-click the MAKER table.

   f) Validate the SQL expression and preview the selection.

   g) Save the data foundation.

3. In the Business Layer, create a list of values base on a custom hierarchy based on the geography of the Client Name.

- Name: Client Hierarchy
- Use the list of values based on custom hierarchy option. Organize the dimensions by Country, Region, Town, Client Name
- Preview your selection and save the Business Layer.

a) Choose *Business Layer* and the *Parameter and List of Values* pane.

b) From the *List of Values* pane, choose *Insert* → *Use the list of values based on business layer objects.*

c) Name the list of values **`Client Hierarchy`** and choose *List of values based on custom hierarchy.*

d) Choose *Add Dimension* and double click Country, Region, Town, Client Name.

e) Choose *Preview* and close.

f) Save the business layer.

4. In the business layer, create a static list of values for European Countries..
   - Name: European Countries
   - Use the *Properties* tab and name the column Country
   - Add the following row values: Germany, Italy, Sweden, and the United Kingdom
   - Associate the European Countries as the list of values for the Maker Country object

   a) From the data foundation, choose the *Parameter and List of Values* pane.

   b) Choose *Insert* → *Static list of values*.

   c) In the *Name* field, enter **`European Countries`**

   d) Use the *Properties* tab and change the Column name to **`Country`**.

   e) To add the following row values, On the right side of the *Definition* tab, choose the green plus sign button:
      - Germany
      - Italy
      - Sweden
      - United Kingdom

   f) Save the data foundation.

Business Object Associated List of Values

The list of values (LOV) must be available in the Business Layer. The LOV is on the list in the Parameters and Lists of Values tab of the Business Layer editor. Associate a LOV to a business object to restrict possible input values when the object is used as a filter in the Query Panel.

1. Open the Business Layer in the editor by double-clicking the Business Layer name in the Local Projects View.
2. Click the *Business Layer* tab to open the Business Layer pane.
3. Click the Business Layer object in the Business Layer pane.
4. Click the *Advanced* tab in the editing pane.
5. Select the *Associate list of values* check box.
6. Click the *browse* icon, select the LOV from the list, and click *OK*.
7. Save the Business Layer.

Unit 7
Exercise 18

Associate a List of Values with a Business Object

Business Example

You need to associate a list of values with the business object.

1. In the business layer, associate the list of values of the *Client ID* object in the *Client* folder.
2. In the business layer, associate the list of values of the *Maker* object in the *Car* folder.
3. Associate the list of values for the *Client Name* to the *Client Hierarchy*.
4. In the business layer, associate the *European Countries* list of values to the *Maker Country* object in the *Car* folder.

Unit 7
Solution 18

Associate a List of Values with a Business Object

Business Example

You need to associate a list of values with the business object.

1. In the business layer, associate the list of values of the *Client ID* object in the *Client* folder.
   a) Expand the *Client* folder, and choose the *Client ID* object.
   b) Choose the *Advanced* tab, choose the ellipsis button, and from the *List of Values* choose Client Name and ID as the list of values for the *Client ID* object.
2. In the business layer, associate the list of values of the *Maker* object in the *Car* folder.
   a) Expand the Car folder, and choose the *Maker* object.
   b) Choose the *Advanced* tab, choose the ellipsis button, and from the *List of Values*, choose *Car Makers*.
3. Associate the list of values for the *Client Name* to the *Client Hierarchy*.
   a) Expand the *Client* folder, and choose the *Client Country* object.
   b) Choose the *Advanced* tab, choose the ellipsis button, and from the *List of Values,* choose *Client Hierarchy*.
4. In the business layer, associate the *European Countries* list of values to the *Maker Country* object in the *Car* folder.
   a) Expand the *Car* folder, and choose the *Maker Country* object.
   b) Choose the *Advanced* tab, choose the ellipsis button, and from the *List of Values* choose *European Countries* as the list of values for the *Maker Country* object.

LESSON SUMMARY

You should now be able to:

- Define a list of values
- Use the list of values editor
- Create static lists of values
- Create lists of values based on business layer objects
- Associate a list of values with a business object

Unit 7

Learning Assessment

1. A LOV can only be edited in the Business Layer.

   Determine whether this statement is true or false.

   - ☐ True
   - ☐ False

2. Choose the correct method for inserting a new LOV.

   Choose the correct answer.

   - ☐ A Business Layer Editor > Parameters and Lists of Values tab > Lists of Values pane > Insert Lists of Values button
   - ☐ B Business Layer Editor > Lists of Values pane > Actions >Insert > Lists of Values
   - ☐ C Business Layer Editor > Lists of Values pane > Insert Lists of Values button
   - ☐ D Business Layer Editor > Parameters and Lists of Values tab> Insert Lists of Values icon

3. A static list of values cannot be based on values that are imported from a file.

   Determine whether this statement is true or false.

   - ☐ True
   - ☐ False

4. Select the correct options for creating a list of values based on an object from the business layer.

   Choose the correct answers.

   - ☐ A LOV based on a query panel
   - ☐ B LOV based on imported file
   - ☐ C LOV based on a custom hierarchy
   - ☐ D LOV based on a restricted data set

5. Why do you associate a LOV to a business object?

Learning Assessment - Answers

1. A LOV can only be edited in the Business Layer.

   Determine whether this statement is true or false.

   - [ ] True
   - [x] False

2. Choose the correct method for inserting a new LOV.

   Choose the correct answer.

   - [ ] A Business Layer Editor > Parameters and Lists of Values tab > Lists of Values pane > Insert Lists of Values button
   - [ ] B Business Layer Editor > Lists of Values pane > Actions >Insert > Lists of Values
   - [ ] C Business Layer Editor > Lists of Values pane > Insert Lists of Values button
   - [x] D Business Layer Editor > Parameters and Lists of Values tab> Insert Lists of Values icon

3. A static list of values cannot be based on values that are imported from a file.

   Determine whether this statement is true or false.

   - [ ] True
   - [x] False

4. Select the correct options for creating a list of values based on an object from the business layer.

   Choose the correct answers.

   - [x] A LOV based on a query panel
   - [ ] B LOV based on imported file
   - [x] C LOV based on a custom hierarchy
   - [ ] D LOV based on a restricted data set

5. Why do you associate a LOV to a business object?

Associate a LOV to a business object to restrict possible input values when the object is used as a filter in the Query Panel.

UNIT 8 Parameters

Lesson 1

UNIT OBJECTIVES

- Use parameters to restrict data

Unit 8
Lesson 1

Illustrating Runtime Parameters

LESSON OVERVIEW
As a universe designer, you can use parameters in the Data Foundation and the Business Layer to restrict data at runtime. This lesson explains how to define parameters and parameter properties.

LESSON OBJECTIVES
After completing this lesson, you will be able to:

- Use parameters to restrict data

Parameters

A parameter is a variable in the Business Layer or the Data Foundation that requires a value at runtime. A parameter can have two input types:

- User input as a response to a prompt. The prompt is a question or directive that requires a user to set one or more values to restrict a result set.
- Predefined input that specifies a fixed value for the parameter at runtime.

Parameters are defined as individual components in a business layer or Data Foundation, and are available to all objects in the Business Layer.

You use parameter objects in the SQL or MDX definition of an object to prompt a user response or to implement a fixed value response to a query.

Note:
Parameters inserted in the Data Foundation are inherited by any Business Layer based on the data foundation. These parameters cannot be edited in the Business Layer. You edit the parameters in the data foundation.

The following properties are available for parameters:

Table 17: Parameter Properties

Property	Description
Prompt to users	If checked, the user is prompted to enter a value at runtime. If cleared, a predefined value is entered at runtime for the parameter.
Prompt Text	The text for the prompt question or directive if Prompt to users is checked.
Set values	Available when the Prompt to users check box is unselected. Lets you enter one or more values to be used for the parameter at the runtime.

Data Type	The data type required for the answer to the prompt.
Allow multiple values	If checked, lets the user select multiple values from the list of values.
Keep last values	If checked, the last value chosen by the user is kept when the prompt is rerun.
Index aware prompt	If checked, the key column is included in the prompt to restrict the values in a list. The key column is not visible to the user.
Associated list of values	A list of values to provide values for the prompt.
Select only from list	If checked, the user is forced to select a member in the list.
Set default value	Lets you select values to be used as default.

Unit 8
Exercise 19

Insert a Data Foundation Parameter

Business Example

Insert a data foundation parameter.

1. In the data foundation, create a parameter for Car Makers, eventually to use as a filter in the Car folder.
   - Name: Car Makers
   - Prompt text: Select a Car Manufacturer
   - Select the Car Makers list of values from the list of values in the data foundation.

Unit 8
Solution 19

Insert a Data Foundation Parameter

Business Example

Insert a data foundation parameter.

1. In the data foundation, create a parameter for Car Makers, eventually to use as a filter in the Car folder.
   - Name: Car Makers
   - Prompt text: Select a Car Manufacturer
   - Select the Car Makers list of values from the list of values in the data foundation.

   a) On the data foundation, choose the *Parameter and List of Values* pane.

   b) In the *Parameters* area, choose *Insert Prompt*.

   c) In the *Name* field, enter **`Car Makers`**.

   d) In the prompt text field, enter **`Select a Car Manufacturer`**.

   e) In the *List of Values* area, associate a list of values by choosing the ellipses button and choosing the Car Makers list of values.

Unit 8
Exercise 20

Use Parameters

Business Example

After you have defined a parameter in either the data foundation or the business layer, you can use it in an SQL statement.

1. Add to the WHERE clause for the Model for Rental Object in the Cars folder to use the parameter.

   The WHERE clause should be the following statement:

   ```
   MODEL.MODEL_DAYRENT IS NOT NULL AND MAKER.MAKER_NAME IN @prompt(Car Makers)
   ```

2. Create a filter called *Car Maker* in the Car folder.

Unit 8
Solution 20

Use Parameters

Business Example

After you have defined a parameter in either the data foundation or the business layer, you can use it in an SQL statement.

1. Add to the WHERE clause for the Model for Rental Object in the Cars folder to use the parameter.

   The WHERE clause should be the following statement:

   ```
   MODEL.MODEL_DAYRENT IS NOT NULL AND MAKER.MAKER_NAME IN @prompt(Car Makers)
   ```

   a) In the business layer, in the Car folder choose the Model for Rental object.

   b) Using the *SQL Assistant* to change the WHERE clause, place the cursor after `MODEL.MODEL_DAYRENT IS NOT NULL` and type the word `AND`.

   c) Still in the *SQL Assistant*, in the Maker table, double-click the Maker Name.

   d) Add the `IN` operator.

   e) In the *Parameters*, double-click the Maker Prompt.

   f) Save the business layer.

2. Create a filter called *Car Maker* in the Car folder.

   a) Right-click the Car folder and choose *New* → *Filter*.

   b) Name the filter *Car Maker*.

   c) Click the SQL Assistant for the WHERE clause.

   d) Expand the MAKER table and double-click the MAKER_NAME column.

   e) Type `IN` after MAKER.MAKER_NAME.

   f) Double-click the *Car Makers* parameter. The finished WHERE clause should look similar to this: MAKER.MAKER_NAME IN @Prompt(Car Makers)

   g) Click *OK*.

LESSON SUMMARY

You should now be able to:

- Use parameters to restrict data

Unit 8

Learning Assessment

1. What is a parameter?

2. A parameter can have two input types. What are the two input types?

   Choose the correct answer.

   - ☐ A Defined input and object input
   - ☐ B Folder input and object input
   - ☐ C User input and predefined input
   - ☐ D User input and folder input

Learning Assessment - Answers

1. What is a parameter?

   A parameter is a variable in the Business Layer or the Data Foundation that requires a value at runtime.

2. A parameter can have two input types. What are the two input types?

   Choose the correct answer.

   - [ ] A Defined input and object input
   - [ ] B Folder input and object input
   - [x] C User input and predefined input
   - [ ] D User input and folder input

UNIT 9 Object @functions

UNIT OBJECTIVES

- Identify object @functions
- Set up aggregate awareness
- Use the select function
- Use the where function
- Use the variable function
- Use the execute function

Unit 9
Lesson 1

Using Object @functions in Queries

LESSON OVERVIEW
This lesson gives you a overview of the Object @function and its specific use to create more powerful queries.

LESSON OBJECTIVES
After completing this lesson, you will be able to:

- Identify object @functions

@functions

@Functions are special functions that provide more flexible methods for specifying the query script for an object. The Functions box in the SQL and MDX Expression Editor lists the allowed @functions for the expression you are defining.

Information Design Tool Supported @functions

The following is the list of functions supported in the Information Design Tool:

Table 18: Information Design Tool Supported @functions

@Function	Description	
@Aggregate _Aware	Directs an object to query first of all the aggregate tables listed as parameters in the @Aggregate_Aware function.	
@DerivedTable	References a derived table. 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.	
@Execute	References a list of values.	
@Prompt	Prompts the user to enter a value each time the object using the @Prompt function is included in a query. Note: The existing syntax of the @Prompt function is supported in the information design tool. You can also define a named parameter for the prompt and reference the parameter in the query script using the @Prompt function.	
@Select	Enables you to use the SELECT statement of another object.	
@Variable	@Function reference	
	Variable	**Description**
	BOUSER	User's login name
	DBUSER	Name used for authorization when connecting to the data source
	UNVID	ID of the universe

	UNVNAME	Name of the universe
	PREFFERED_VIEWING_LOCALE	The user's preferred locale for viewing report and query objects in an application
	DOMINANT_PREFERED_VIEWING_LOCALE	A predefined fallback locale that is used when no fallback locale is defined for the resource
	DPTYPE	Data provider type
	DPNAME	Data provider name
	DOCNAME	Document name
@Where	Enables you to use the WHERE clause of another object.	

Query Expression Supported @functions

The following table shows which @functions are supported in the different query expressions:

Table 19: The @functions Supported in Query Expressions

@Function	Joins	Calculated Columns	Derived Tables	BusinessObjects
@Derived Table	Not allowed	Not allowed	Allowed Note: In database specific SQL (multi source enabled data foundations), all arguments must reference tables or columns from the same connection.	Not allowed
@Execute	Allowed	Allowed	Not allowed	Allowed
@Prompt	Allowed	Allowed Note: Not allowed in database specific SQL in multi-source-enabled data foundations.	Allowed	Allowed
@Select	Not allowed	Not allowed	Not allowed	Allowed
@Variable	Allowed	Allowed Note: In database specific SQL (multi source enabled data foundations), all arguments must reference tables or columns from the same connection.	Allowed	Allowed
@Where	Not allowed	Not allowed	Not allowed	Allowed (SQL only)

LESSON SUMMARY
You should now be able to:

- Identify object @functions

Unit 9
Lesson 2

Applying the Aggregate Awareness Optimization Method

LESSON OVERVIEW

Aggregate awareness is a term that describes the ability of a Business Layer to make use of aggregate tables in a database. Using aggregate tables speeds up the execution of queries, improving the performance of SQL transactions. The reliability and usefulness of aggregate awareness in a Business Layer depends on the accuracy of the aggregate tables. They must be refreshed at the same time as all fact tables.

LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Set up aggregate awareness

Aggregate Awareness Function

Table 20: Aggregate Awareness function

@Function	Description	Usually used in object
@Aggregate_Aware	Incorporates columns containing aggregated and dimensions data into objects	SELECT statement

Some databases contain summary tables. These tables are created by the Database Administrator (DBA) and contain figures such as revenue aggregated to a high level (year, for example), rather than to the fact or event level. The summary tables tend to be populated and updated regularly by an automated program that runs SQL against the fact or event data at transaction level.

This method of population means that there are two methods that you can use to return aggregated data:

- Run a SELECT statement for the fact or event data.
- Run a SELECT statement for the summary data.

Where possible, choose the latter method, as the statement processes quicker. In Information Design Tool, you can use a function called @aggregate_aware in the SELECT statement for an object, so that both methods are referenced. This function directs a query to run against aggregate tables whenever possible. If the data in the aggregate table is not calculated at the level of granularity required to run the query, the object directs the query to run against the tables containing the nonaggregated data.

A Business Layer 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. The reliability and usefulness of aggregate awareness in a Business Layer depends on the accuracy of the aggregate tables. These tables are refreshed at the same time as all fact tables.

Aggregate Awareness Function Use

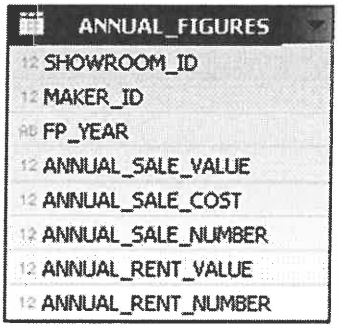


Figure 43: Summary Table

Each row in a summary table is made up of columns containing:

- Aggregated data:

  Numeric event data aggregated to a higher level. In the example table image, columns containing aggregated data are Annual_Sale_Value, Annual_Sale_Cost, Annual_Sale_Number, Annual_Rent_Value, and Annual_Rent_Number.

- Dimension data: Attributes defining the level of the aggregated data.

In this example, there is only one column containing dimension data: FP_Year.

Foreign keys (optional): Joins to other tables. If foreign keys exist, queries can be made using summary table aggregated data based on dimension objects held in other tables of the database, as opposed to just objects contained within the summary table. Foreign keys are used to set joins in the structure of the Data Foundation.

Note:
Aggregate awareness works on all tables, not just fact tables. It can be applied to all object types with all data types, not just numbers and measures.

Where possible, it is prudent to use summary table data, because the processing required to return the aggregated data is far quicker.

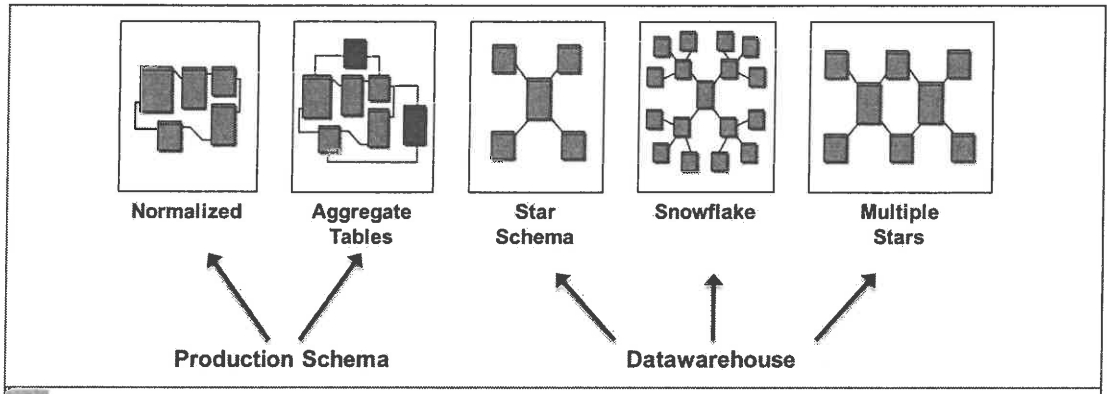


Figure 44: Production Schema / Data Warehouse

As shown in the example diagram, making a query based on event data at a transaction level requires a more complex SELECT statement and the processing of more database rows than one based on summary data.

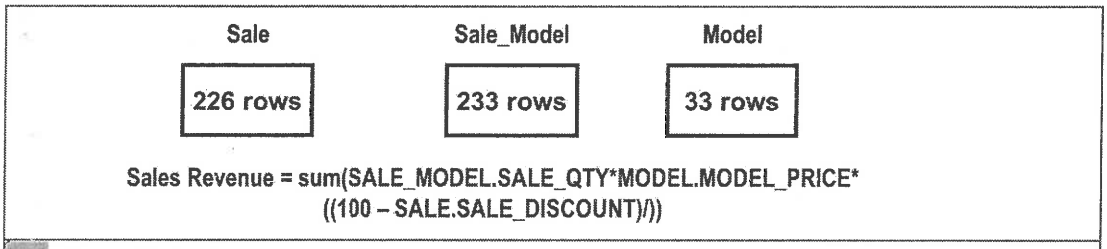


Figure 45: Row Count without Aggregation

Aggregates of a normalized database are based on event or fact level data:

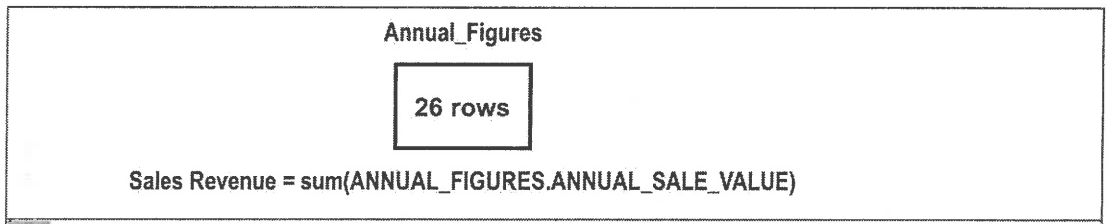


Figure 46: Total Rows Used with Aggregation

Summary tables can be added to a database that hold data at a higher level of aggregate. Using summary table data speeds up response times because:

- There are fewer rows to process.
- Fewer, if any, joins are required.

Aggregate Awareness Process

Applying aggregate awareness to objects in a Business Layer involves a four-step procedure:

Steps in the Aggregate Awareness Procedure

1. Insert one or more summary tables in the universe structure. Set joins and cardinality.
2. Set the contexts.
3. Redefine the objects using @Aggregate_Aware.

4. Define incompatible objects using Aggregate Navigation.

> Note:
> If the summary table does not contain foreign keys, part of step 1 (setting joins and cardinality) and step 2 are not required.

Step 1: Insert a summary table

Step 1 of the aggregate awareness process is to insert one or more summary tables to the Data Foundation structure. The procedure for inserting a summary table in the structure of the Data Foundation is the same as for any other table or view.

1. Insert the required summary tables in the structure using the Connections Browser in your Data Foundation.
2. Position the table where it is convenient to make the joins.
3. Add joins from the summary table to the existing structure. You examine the foreign keys in the summary table to see where the summary table can be joined.
4. Set cardinality for the new joins.

Step 2: Detect contexts

Step 2 of the aggregate awareness process is to detect contexts. After you add the summary table, several new loops may exist. You resolve these using contexts.

1. Click *Detect Contexts* on the toolbar.
2. Accept the new contexts.

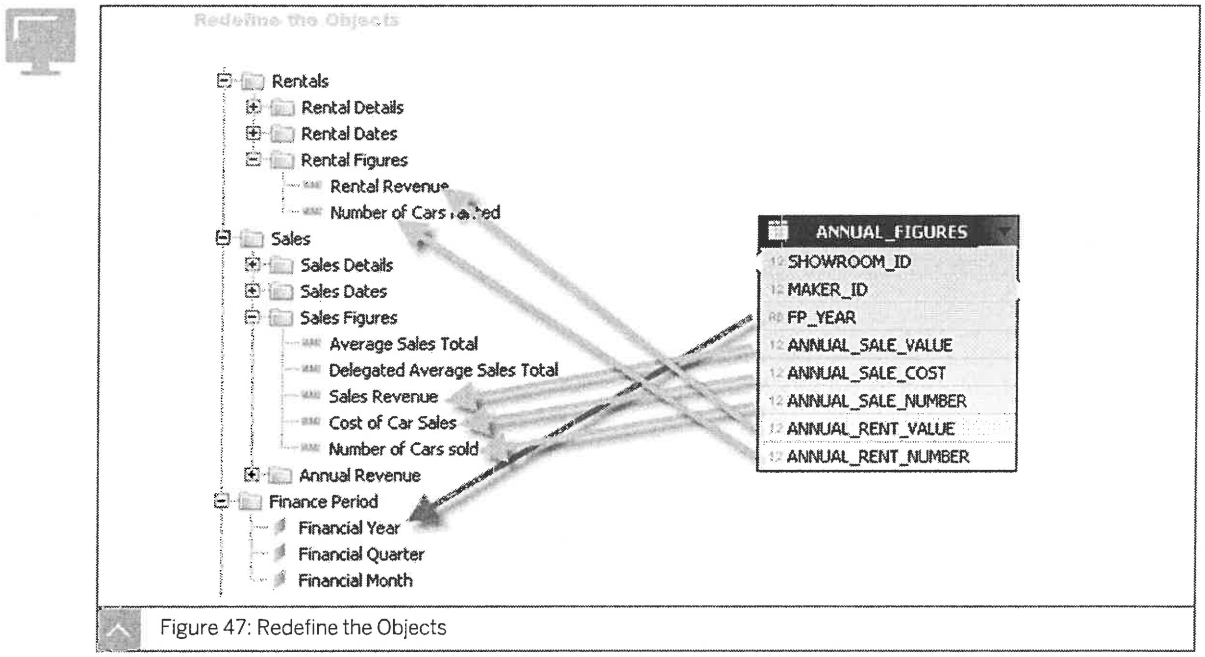


Figure 47: Redefine the Objects

The columns in the summary table containing aggregated and dimension data can be used to define object SELECT properties. In our example, the summary table columns can be used in the SELECT properties of the following objects:

- Financial Year
- Sales Revenue
- Number of Cars sold
- Cost of Cars Sold
- Rental Revenue
- Number of Cars rented

When you have added the summary table, each of these objects has two possible sources for returning data.

Step 3: Applying aggregate awareness to objects

Before editing any objects, confirm exactly what levels of aggregation are available. In our example, we have the basic aggregation calculation and the precalculated **annual data** from the summary table. If you also have summary tables for **quarterly figures** and **monthly figures**, you would have four possible levels altogether. When you apply the @aggregate_aware function, be aware of the available levels, and be clear about the descending order of aggregation, for example, Annual Figures, Quarterly Figures, Monthly Figures, and basic aggregation calculation.

The syntax of the @aggregate_aware function:

@Aggregate_Aware(sum(agg_table_1), ... sum(agg_table_n))

You enter the names of all aggregate tables as arguments. Place the names of the tables from left to right in descending order of aggregation.

agg_table_1: Is the aggregate with the highest level of aggregation.

agg_table_n: Is the aggregate with the lowest level of aggregation

Each aggregation level SELECT statement is separated by a comma, and the entire expression is enclosed in brackets. The final SELECT statement must be valid for all queries. The @aggregate aware function is directing the query engine to use the sum of the measure value taken from the summary table where possible and, where not possible, to use the next segment in the SELECT statement. The last segment in the @aggregate_aware function is always the original SELECT that does not use the summary tables.

For example:

@aggregate_aware(sum(ANNUAL_FIGURES.ANNUAL_SALE_VALUE),sum(SALE_MODEL.SALE_QTY *MODEL.MODEL_PRICE* ((100-SALE.SALE_DISCOUNT) / 100)))

Similarly, for dimension objects in the summary tables, the @aggregate_aware function simply selects the column from the summary tables first (in descending order of aggregation), and then from the normal source.

For example:

@aggregate_aware(ANNUAL_FIGURES.FP_YEAR,FINANCE_PERIOD.FP_YEAR)

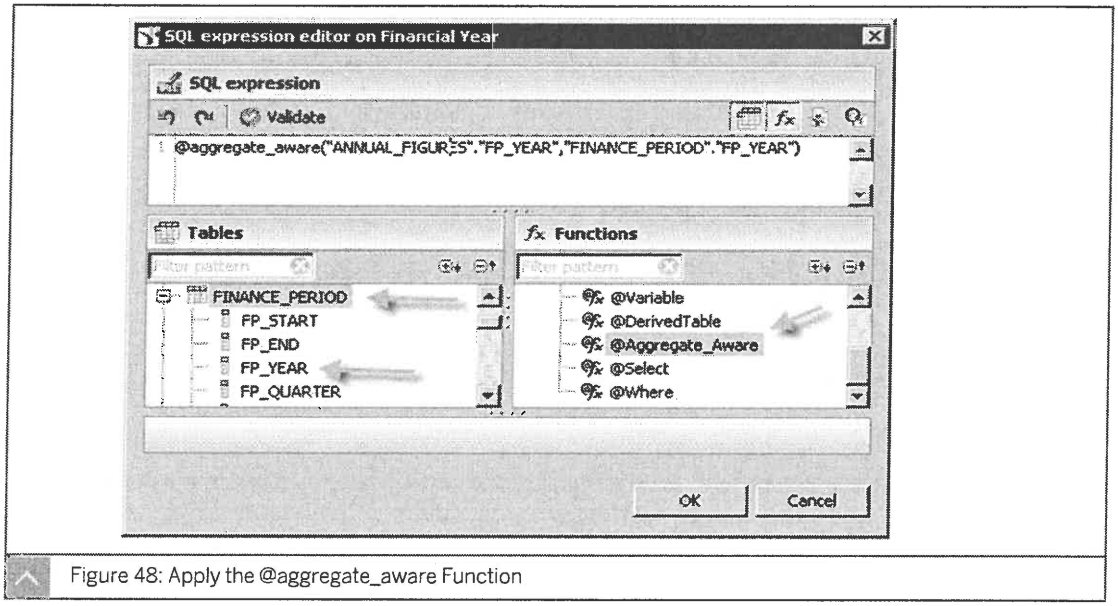


Figure 48: Apply the @aggregate_aware Function

To apply aggregate awareness to an object's SQL definition:

1. Double-click the object whose properties you want to edit, or click Insert Object on the toolbar to create a new object.
2. Click *Select Assistant of the Select* field to open the *Edit Select* dialog box. The current SELECT properties for the object, if any, displays in the top panel of the dialog box.
3. Click at the beginning of the existing statement, if necessary.
4. Double-click the @Aggregate_Aware function in the @functions list dialog box.
5. Insert the aggregate actions within the brackets of the @Aggregate_Aware function in order of highest to lowest level of aggregation data. Separate each action with a comma.
6. Parse the redefined objects.
7. Click *Save* on the main Menu to accept the SELECT statement.
8. Repeat the process for all the appropriate objects.

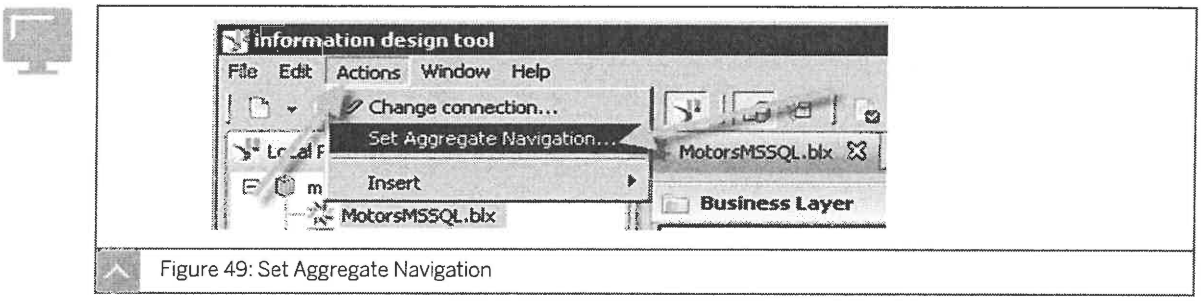


Figure 49: Set Aggregate Navigation

4. Define incompatible objects using Aggregate Navigation

1. Open the Business Layer in the editor by double-clicking the business layer name in the Local Projects View.

2. From the information design tool main menu, select *Actions* → *Set Aggregate Navigation.*

In the *Aggregate Awareness* dialog box, you specify which classes contain objects that are not compatible with aggregate tables.

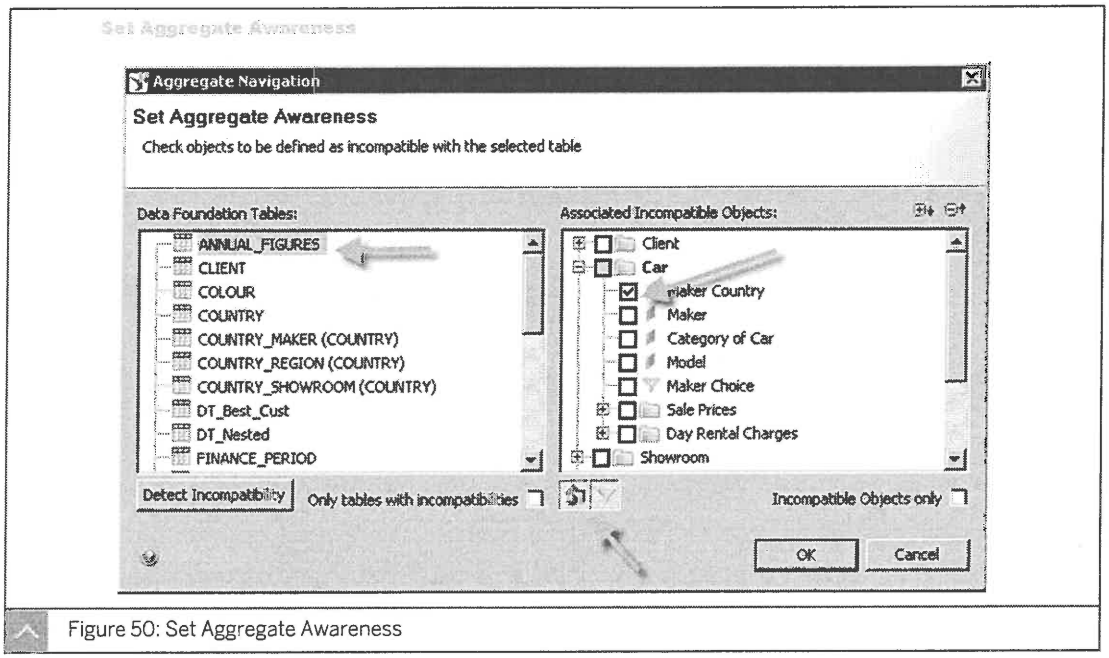


Figure 50: Set Aggregate Awareness

3. Click an aggregate table in the left pane.
4. Click on the *Objects* and *Filter* Icons to see both of them in the right pane.
5. In the right pane, select the check box for each incompatible object.
6. Repeat the previous steps for each aggregate table in the data foundation.
7. Click *OK*, when all incompatible objects for all the tables are specified.

Note:
A hook in the check box on a class or an object or a filter, means that it is incompatible to the aggregation process.

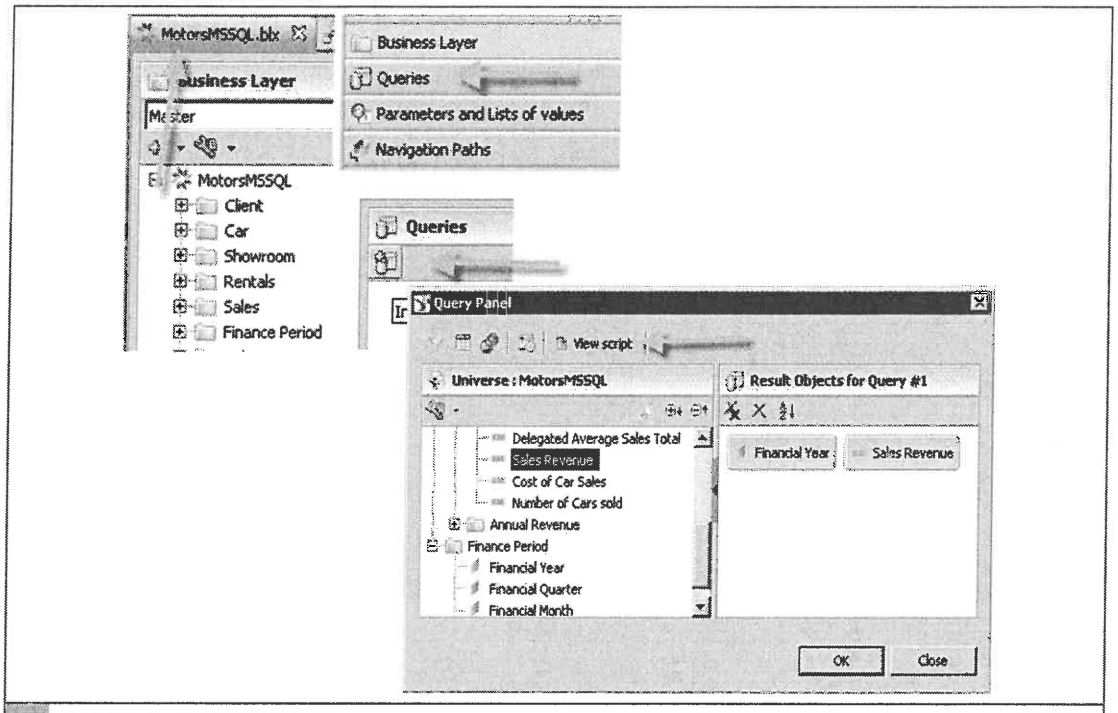


Figure 51: Test Your Work (1)

Open a new query and select Financial Year object and Sales Revenue measure. View the script.

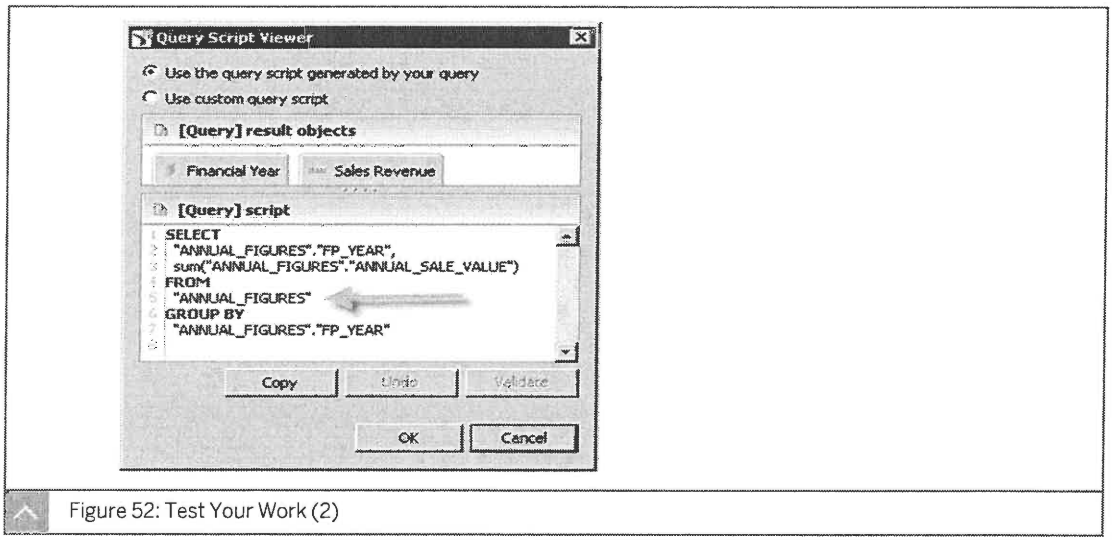


Figure 52: Test Your Work (2)

The SQL script should look like shown in the proceeding Screen. ANUAL_FIGURES table is in the FROM part.

Now let us see what happens when incompatible objects are part of the query?

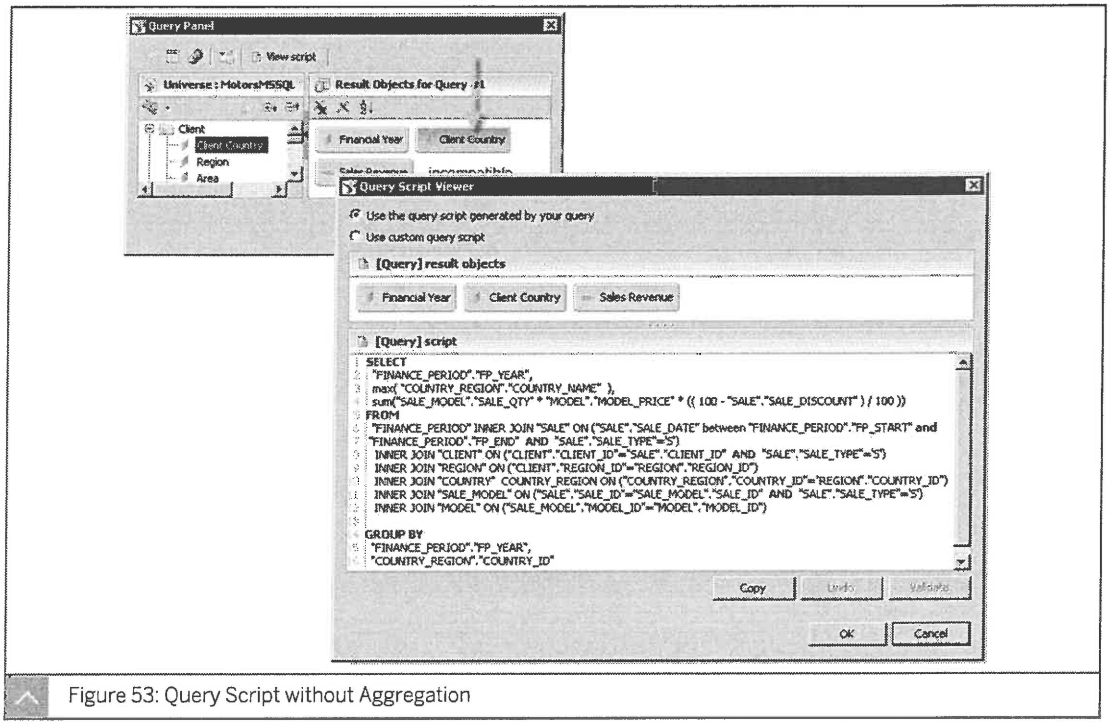


Figure 53: Query Script without Aggregation

Usage of incompatible objects in a query effects the usage of the aggregate table n definition.

Unit 9
Exercise 21

Set up Aggregate Awareness

Business Example

Prestige Motors management is finding that the annual and quarterly reports take a long time to run. They have requested that these reports be produced without any noticeable processing delay. You decide to use the aggregate tables that exist in the Motors database to take advantage of preaggregated data.

Note:
To complete the exercise, replace the value *xx* with the group number the instructor provided you with at the beginning of the course.

1. Add the ANNUAL_FIGURES summary table to the Motors_XX data foundation and join it to the appropriate tables.

   Detect and accept the ANNUAL_FIGURES context.

2. Modify objects to use the ANNUAL_FIGURES table and set the appropriate incompatible objects with the ANNUAL_FIGURES table.

Object	SELECT Statement
Financial Year	@Aggregate_Aware(ANNUAL_FIGURES.FP_YEAR, FINANCE_PERIOD.FP_YEAR)
Sales Revenue	@Aggregate_Aware(sum(ANNUAL_FIGURES.ANNUAL_SALE_VALUE),sum(SALE_MODEL.SALE_QTY * MODEL.MODEL_PRICE * ((100 - SALE.SALE_DISCOUNT) / 100)))
Cost of Car Sales	@Aggregate_Aware(sum(ANNUAL_FIGURES.ANNUAL_SALE_COST), sum(SALE_MODEL.SALE_QTY * MODEL.MODEL_COST))
Number of Cars sold	@Aggregate_Aware(sum(ANNUAL_FIGURES.ANNUAL_SALE_NUMBER), sum(SALE_MODEL.SALE_QTY))

Object	SELECT Statement
Rental Revenue	@Aggregate_Aware(sum(ANNUAL_FIGURES.ANNUAL_RENT_VALUE), sum(RENTAL.DAYS_RENTED * RENTAL_MODEL.SALE_QTY * MODEL.MODEL_DAYRENT * ((100 - RENTAL.SALE_DISCOUNT) / 100)))

3. By making the following queries in the query panel in your business layer, test the annual level of aggregate awareness.

   View the SQL for each query and note the use of the ANNUAL FIGURES table in some of the queries.

   - Showroom, Financial Year, Sales Revenue, and Rental Revenue
   - Model, Financial Year, Sales Revenue, and Rental Revenue
   - Financial Year, Sales Revenue, and Financial Month

Unit 9
Solution 21

Set up Aggregate Awareness

Business Example

Prestige Motors management is finding that the annual and quarterly reports take a long time to run. They have requested that these reports be produced without any noticeable processing delay. You decide to use the aggregate tables that exist in the Motors database to take advantage of preaggregated data.

Note:
To complete the exercise, replace the value *xx* with the group number the instructor provided you with at the beginning of the course.

1. Add the ANNUAL_FIGURES summary table to the Motors_XX data foundation and join it to the appropriate tables.

   Detect and accept the ANNUAL_FIGURES context.

   a) Open the data foundation.

   b) Right click in the white area and from the context menu, choose *Insert* → *Tables...*.

   c) Place a checkmark beside the ANNUAL_FIGURES table and choose *Finish*.

   d) Insert the joins by dragging and dropping the SHOWROOM_ID from the ANNUAL_FIGURES table to the SHOWROOM_ID in the SHOWROOM table.

   e) Double click the join and detect the cardinality.

   f) Do the same for the MAKER_ID field.

   g) Click on the *Aliases and Context* section and choose *Detect Contexts*.

   h) Accept the ANNUAL_FIGURES context and choose *OK*.

   i) Edit the context to make all neutral joins Excluded.

   j) Save the data foundation.

2. Modify objects to use the ANNUAL_FIGURES table and set the appropriate incompatible objects with the ANNUAL_FIGURES table.

Object	SELECT Statement
Financial Year	@Aggregate_Aware(ANNUAL_FIGURES.FP_YEAR, FINANCE_PERIOD.FP_YEAR)

Object	SELECT Statement
Sales Revenue	@Aggregate_Aware(sum(ANNUAL_FIGURES.ANNUAL_SALE_VALUE),sum(SALE_MODEL.SALE_QTY * MODEL.MODEL_PRICE * ((100 - SALE.SALE_DISCOUNT) / 100)))
Cost of Car Sales	@Aggregate_Aware(sum(ANNUAL_FIGURES.ANNUAL_SALE_COST), sum(SALE_MODEL.SALE_QTY * MODEL.MODEL_COST))
Number of Cars sold	@Aggregate_Aware(sum(ANNUAL_FIGURES.ANNUAL_SALE_NUMBER), sum(SALE_MODEL.SALE_QTY))
Rental Revenue	@Aggregate_Aware(sum(ANNUAL_FIGURES.ANNUAL_RENT_VALUE), sum(RENTAL.DAYS_RENTED * RENTAL_MODEL.SALE_QTY * MODEL.MODEL_DAYRENT * ((100 - RENTAL.SALE_DISCOUNT) / 100)))

a) Open your Motors business layer and redefine the SELECT properties of the objects as indicated in the table in step 3.

b) Choose *Actions* → *Set Aggregate Navigation*.

c) In the *Aggregation Navigation* pop-up, on the left side, choose the ANNUAL_FIGURES table.

d) On the right side, in the *Associated Incompatible Objects* area, put a check mark beside the following folders:

- Client Showroom Sale Prices (subfolder of Car)
- Day Rental Charges (subfolder of Car)
- Rental Details (subfolder of Rentals)
- Rental Dates (subfolder of Rentals)
- Sales Details (subfolder of Sales)
- Sale Dates (subfolder of Sales)

e) On the right side of the *Associated Incompatible Objects* area, put a check mark beside the following objects:

- Category of Car (Car folder)
- Model (Car folder)
- Financial Quarter (Finance Period folder)
- Financial Month (Finance Period folder)

f) Save the changes.

3. By making the following queries in the query panel in your business layer, test the annual level of aggregate awareness.

   View the SQL for each query and note the use of the ANNUAL FIGURES table in some of the queries.

   - Showroom, Financial Year, Sales Revenue, and Rental Revenue
   - Model, Financial Year, Sales Revenue, and Rental Revenue
   - Financial Year, Sales Revenue, and Financial Month

   a) In the business layer, at the bottom left, choose the *Queries* tab.

   b) Click on the Insert Query button.

   c) Insert the objects Showroom, Financial Year, Sales Revenue, and Rental Revenue.

   d) To see the results, choose Refresh.

   e) Add the Client Name object and look at the SQL.

LESSON SUMMARY
You should now be able to:

- Set up aggregate awareness

Unit 9
Lesson 3

Using Other Functions

LESSON OVERVIEW
This lesson helps you use the other @functions to provide more flexible methods for specifying the SQL for an object.

LESSON OBJECTIVES
After completing this lesson, you will be able to:

- Use the select function
- Use the where function
- Use the variable function
- Use the execute function

Function @select

The @select function is a pointer to the Select field properties of another object. It is used by placing the @select in the Select field of the Edit Properties dialog box of an object, using the following syntax:

@select (path of existing object)

You specify the path in the form of Folder_Name\Object_Name. The purpose of the @select function is to allow you to reuse existing code, giving the advantage of having to specify SQL code only once. There are two key advantages to specifying SQL only once:

- Maintain only one instance of the SQL code.
- It ensures consistency of the code

For example:

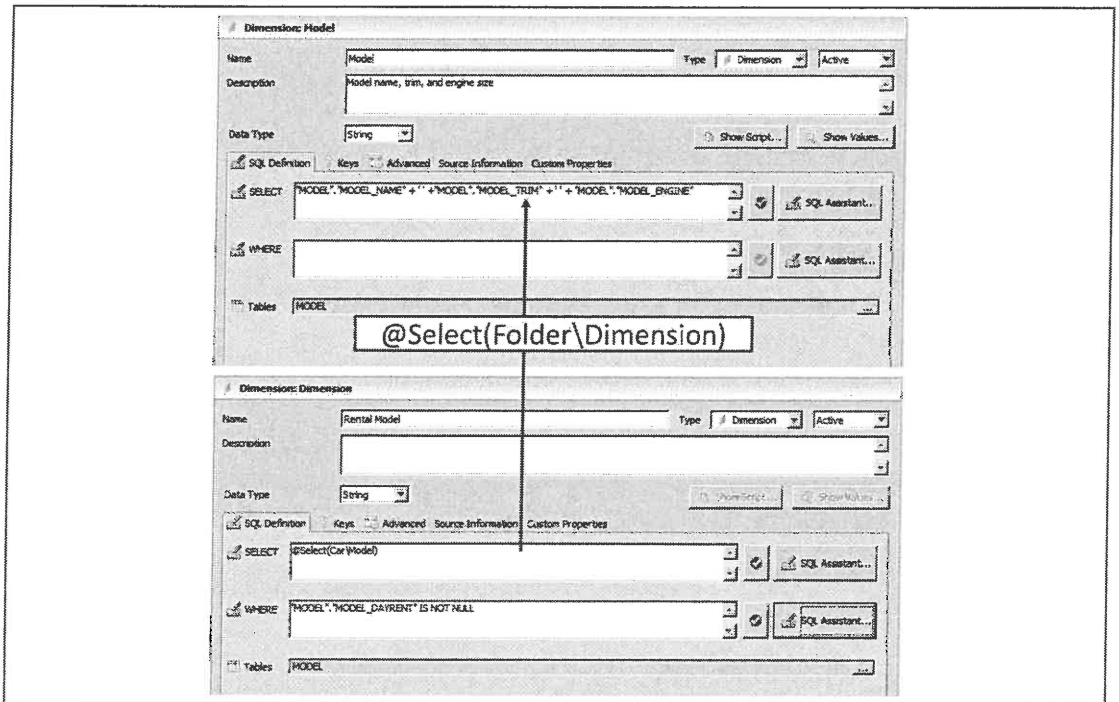


Figure 54: Using @select

This example shows how the @Select works. The code in the SELECT properties of the Model object is:

MODEL.MODEL_NAME+' '+ MODEL.MODEL_TRIM+' '+MODEL.MODEL_ENGINE

To create a new object called Model for Rental with the same code, instead of creating the same code twice, you can refer to the original Model object via the @select function:

@Select(Car\Model)

The benefit is that a dynamic link is created between the objects. When changes occur in the SELECT statement of the original object, the changes are reflected in the SELECT statement of any other objects that refer to it via the @Select function. When you change the code, you only change it once in the original object

Unit 9
Exercise 22

Use the @select Function

Business Example

Rather than creating the same code twice, you can reuse the code in the original Model object by referring to it using the @select function. When changes occur in the SELECT statement of the referenced object, the changes are reflected in the SELECT statement of the new object that refers to it.

1. The SELECT properties of the Model and Model for Rental objects are the same. Use the @select function in the Model for Rental object to point to the SELECT properties of the Model object.

Unit 9
Solution 22

Use the @select Function

Business Example

Rather than creating the same code twice, you can reuse the code in the original Model object by referring to it using the @select function. When changes occur in the SELECT statement of the referenced object, the changes are reflected in the SELECT statement of the new object that refers to it.

1. The SELECT properties of the Model and Model for Rental objects are the same. Use the @select function in the Model for Rental object to point to the SELECT properties of the Model object.
   a) Choose the Model for Rental dimension.
   b) Delete the current SQL and use the SQL Assistant to create the following SELECT statement:
      ```
      @Select(Car\Model)
      ```
   c) Choose *OK*.
   d) Save your business layer.

Function @where

The @Where function is a pointer to the WHERE properties of another object. It is used by placing the @Where in the Where field of the *Edit Properties* dialog box of an object, using the following syntax:

@Where (path of existing object)

For example:

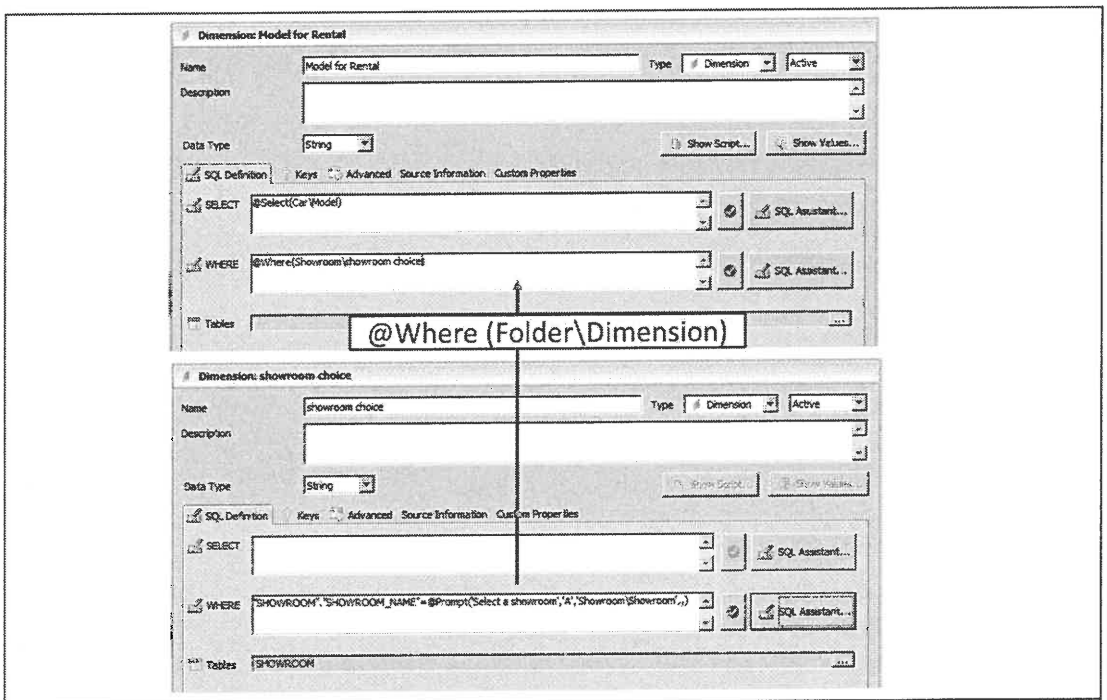


Figure 55: Using @where

This example shows how the @Where works. The code in the WHERE syntax of the Showroom Choice object is:

"SHOWROOM"."SHOWROOM_NAME"= @prompt('Select a Showroom name','A','Showroom\Showroom',Mono,Constrained)

If you wish to create a new object called Model for Rental that contains the same WHERE syntax. Rather than creating the same code twice, you can refer to the original Showroom Choice object via the @Where function:

@Where(Where Restriction Objects\Showroom Choice)

The benefit is that a dynamic link is created between the objects. When changes occur in the WHERE clause of the original object, the changes are reflected in the WHERE clause of any other objects that refer to it via the @Where function. Therefore, to change the syntax, you only change it once in the original object.

As with @Select, its purpose is to allow you to reuse existing code, and it has the same advantages:

- Maintain only one instance of the SQL code.

- It ensures consistency of the code.

If there are a number of objects or condition objects that require the same restrictions be placed upon them, use a WHERE restriction object strategy to make the most efficient use of that restrictions code.

The idea behind the strategy is that you create a new and separate object for every restriction required, in a separate folder to the "normal" object folder. Then, within the original objects, whenever a restriction is required, you point to the appropriate WHERE restriction object using the @where function.

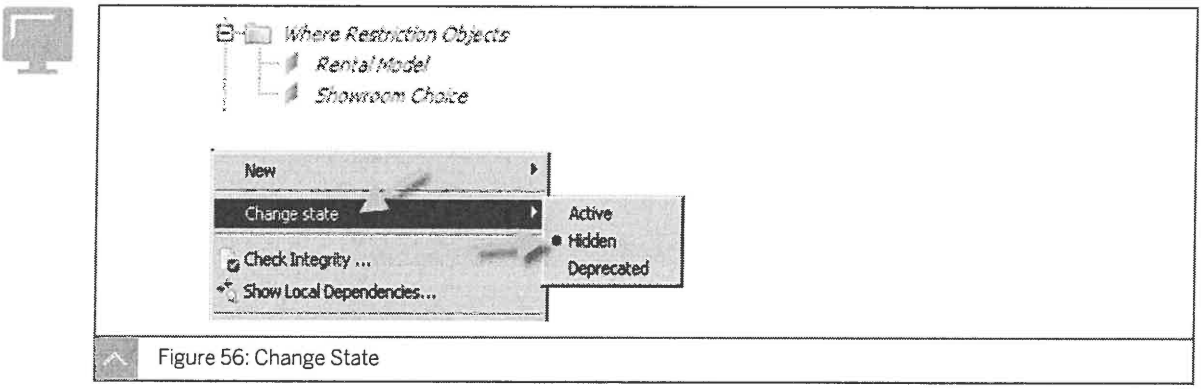


Figure 56: Change State

To hide the folder containing the entire Where clause restriction objects from end users:

1. Click the folder or object you want to hide.
2. Right-click the object or folder and select *Change state* → *Hidden*

Hidden folders and objects appear in italics in the Business Layer pane. They are not shown at all in Business Objects end-user querying tools.

Unit 9
Exercise 23

Use the @where Function

Business Example

You want to use the @where function to point to the WHERE clause restriction objects.

1. Create two new filters in the `Where Restriction Objects` folder as follows:
   - Rental Model containing the restriction:

     ```
     MODEL.MODEL_DAYRENT IS NOT NULL
     ```

   - Choose a Car Manufacturer containing the restriction:

     ```
     MAKER.MAKER_NAME = @prompt('Car Maker')
     ```

2. Hide the `Where Restriction Objects` folder.
3. Edit the following object and filter object so that the WHERE clause of each contains no SQL code, but, instead, uses @where functions to point to the WHERE clause restriction objects.

   Model for Rental object inherits the Rental Model object's WHERE clause and inherits the Showroom Choice objects' WHERE clause.

Unit 9
Solution 23

Use the @where Function

Business Example

You want to use the @where function to point to the WHERE clause restriction objects.

1. Create two new filters in the `Where Restriction Objects` folder as follows:
   - Rental Model containing the restriction:

     ```
     MODEL.MODEL_DAYRENT IS NOT NULL
     ```

   - Choose a Car Manufacturer containing the restriction:

     ```
     MAKER.MAKER_NAME = @prompt('Car Maker')
     ```

   a) In the business layer drawer of the, right-click and choose *New* → *Folder* and name the folder **`Where Restriction Objects`**.

   b) Right-click the `Where Restriction Objects` folder and insert two filters, renaming them to Rental Model and Showroom Choice.

   c) Choose the Rental Model filter, and use the SQL assistant to put the WHERE section in the following clause:

   ```
   MODEL.MODEL_DAYRENT IS NOT NULL
   ```

   d) From the `Car` folder, copy the Select a Car Manufacturer filter and paste it in the `Where Restriction Objects` folder.

2. Hide the `Where Restriction Objects` folder.
   a) Right-click the `Where Restriction Objects` folder and choose *Change State* → *Hidden*.

3. Edit the following object and filter object so that the WHERE clause of each contains no SQL code, but, instead, uses @where functions to point to the WHERE clause restriction objects.

   Model for Rental object inherits the Rental Model object's WHERE clause and inherits the Showroom Choice objects' WHERE clause.

   a) The WHERE clause will be the following:

   ```
   @Where(Where Restriction Objects\ Rental Model) AND @Where(Where
   Restriction Objects\Select a Car Manufacturer)
   ```

   b) The WHERE clause is @Where (Where Restriction Objects\ Rental Model) AND @Where(Where Restriction Objects\Showroom Choice.

Function @variable

The @Variable function is used in the WHERE clause to call the value assigned to one of the following variable types:

- BusinessObjects system variables
- Report variables
- Language (Locale) variables
- Operating system variables

Insert the @Variable on the operand side of the condition in the WHERE clause for an object. 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.
- When the same @Variable function is executed several times in a query, the prompt only appears once.
- The @Variable function is equivalent to a single value @Prompt function with the following settings:

  @Prompt('Question','A',,mono,free)

You can merge the @Variable function with the @Prompt function in the same query when the @Prompt function is monovalue

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').

Table 21: @Variable Property Descriptions

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. These variables 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 locale 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 variable prevents the users from having to translate data in all locales (fr_FR, fr_BE, fr_CA, ...). For example, in a situation where translations are available in fr_FR, if the user locale is fr_BE or fr_CA, since those locales share the same dominant locale (fr), they can reuse translations in fr_FR.
Operating system variables	You can enter Windows environment variables to obtain information about your installation.

BusinessObjects System Variable Use

You can use the @Variable function with BusinessObjects system variables. Use this function to restrict data based on the identity of the BusinessObjects user currently logged on.

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.

@Variable Example:

In the Business Layer 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 restriction 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.

Locale Variable Use

Use the locale variables of the @Variable function to define the locale settings so the end user reporting tool 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.

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')

Product Table Example:

The following PRODUCT table has been translated in many languages. The user wants to list the product names in a specific locale.

Table 22: Translated Product Table

Product ID	LOCALE	Product_Name
DC1212	en_GB	Digital camera
DC1212	fr_FR	Apparell photo numèrique
DC1212	de_DE	Digitalkamera
DC1212	es_ES	Cámera 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.

Report Variable Use

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 report variable 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 and WHERE clauses
- Filters
- The Join expression
- The Begin_SQL parameter

Function @execute

The @Execute function allows you to references a list of values in a SQL expression. It is designed to improve performance of SQL queries, especially when the list of values definition is a complex query.

Unit 9
Exercise 24

Use the @execute Function

Business Example

Your task is to provide a simple method for your reporting users to determine which clients have had more than one transaction (either sales or rentals) with us. We will call these customers, Return Customers. We will create a query filter that the reporting users can use in any query to restrict their results to only those clients who have more than one sales ID. We will also use the @Execute function to determine these clients.

1. Create a list of values on the data foundation that returns the number of transactions per client ID that are greater than 1.

> Note:
> We are creating this list of values on the data foundation so that any business layer that uses the data foundation can use it.

2. To restrict query results to only those customers who have more than one sales ID, use that list of values in a new query filter by using the @Execute function.
3. By creating a query that uses it, test the filter.

Unit 9
Solution 24

Use the @execute Function

Business Example

Your task is to provide a simple method for your reporting users to determine which clients have had more than one transaction (either sales or rentals) with us. We will call these customers, Return Customers. We will create a query filter that the reporting users can use in any query to restrict their results to only those clients who have more than one sales ID. We will also use the @Execute function to determine these clients.

1. Create a list of values on the data foundation that returns the number of transactions per client ID that are greater than 1.

> Note:
> We are creating this list of values on the data foundation so that any business layer that uses the data foundation can use it.

   a) On the data foundation, select the *Parameters and Lists of Values* drawer.

   b) In the *Lists of Values* area, choose *Add* → *List of values based on custom SQL*.

   c) Name the LOV **`Return Customers`**.

   d) To define the LOV, use the following SQL:

```
SELECT COUNT(SALE.SALE_ID) AS Number_of_Transactions
FROM SALE
GROUP BY SALE.CLIENT_ID
HAVING
count(SALE.SALE_ID)  >  1
```

   e) Validate the SQL expression and choose *OK*.

2. To restrict query results to only those customers who have more than one sales ID, use that list of values in a new query filter by using the @Execute function.

   a) On the business layer, choose the *Business Layer* drawer.

   b) Right-click on the Client folder and choose *New* → *Filter*.

   c) Name the filter **`Return Customers`**.

   d) To define the WHERE clause, use the following SQL:

```
CLIENT.CLIENT_ID IN
  (SELECT SALE.CLIENT_ID
   FROM SALE
   GROUP BY SALE.CLIENT_ID
   HAVING count(SALE.SALE_ID) IN @Execute(Return Customers))
```

   e) Validate the SQL expression and choose *OK*.

3. By creating a query that uses it, test the filter.

a) On the business layer, choose the *Queries* drawer.

b) Create a new query with Client ID and Client Name.

c) Add the new Return Customer filter to the Query Filters area.

d) To see the results, choose *Preview*.
There should be approximately 20 rows of data returned.

LESSON SUMMARY

You should now be able to:

- Use the select function
- Use the where function
- Use the variable function
- Use the execute function

Unit 9

Learning Assessment

1. What is the purpose of @Functions?

   Choose the correct answer.

   - ☐ A @Functions provide search terms for a query.
   - ☐ B @Functions direct communication on the platform.
   - ☐ C @Functions allow querry mapping for objects.
   - ☐ D @Functions provide flexible methods for specifying the query script for an object.

2. Running a SELECT statement on a summary table is the fastest method of returning aggregated data.

   Determine whether this statement is true or false.

   - ☐ True
   - ☐ False

3. Upon what are aggregates of a normalized database based?

   Choose the correct answer.

   - ☐ A Business objects
   - ☐ B Summary tables
   - ☐ C Query tables
   - ☐ D Event or fact level data

4. What is the @select function?

   _____

   _____

   _____

5. What is the @where function?

   _____

   _____

   _____

6. How do you use the @where function?

   Choose the correct answer.

   - ☐ A *Object Edit Properties* dialog box *Where* field
   - ☐ B *Object Function* tab *@where* check box
   - ☐ C *Object Actions* tab *Functions Add Function* button, select *@where* from functions list
   - ☐ D *Object Edit Properties* dialog box *Functions* tab *@Where* field

7. The @variable function is used in the WHERE clause to call the value assigned to which variable types?

   Choose the correct answers.

   - ☐ A BusinessObjects system variables
   - ☐ B Location variables
   - ☐ C Report variables
   - ☐ D Language variables
   - ☐ E Date variables
   - ☐ F Operating system variables

8. What is the purpose of the @execute function?

   _____

   _____

   _____

Learning Assessment - Answers

1. What is the purpose of @Functions?

   Choose the correct answer.

   - [ ] A @Functions provide search terms for a query.
   - [ ] B @Functions direct communication on the platform.
   - [ ] C @Functions allow querry mapping for objects.
   - [x] D @Functions provide flexible methods for specifying the query script for an object.

2. Running a SELECT statement on a summary table is the fastest method of returning aggregated data.

   Determine whether this statement is true or false.

   - [x] True
   - [ ] False

3. Upon what are aggregates of a normalized database based?

   Choose the correct answer.

   - [ ] A Business objects
   - [ ] B Summary tables
   - [ ] C Query tables
   - [x] D Event or fact level data

4. What is the @select function?

   The @select function is a pointer to the Select field properties of another object.

5. What is the @where function?

   The @Where function is a pointer to the WHERE properties of another object.

6. How do you use the @where function?

   Choose the correct answer.

   - [X] A *Object Edit Properties* dialog box *Where* field
   - [ ] B *Object Function* tab *@where* check box
   - [ ] C *Object Actions* tab *Functions Add Function* button, select *@where* from functions list
   - [ ] D *Object Edit Properties* dialog box *Functions* tab *@Where* field

7. The @variable function is used in the WHERE clause to call the value assigned to which variable types?

   Choose the correct answers.

   - [X] A BusinessObjects system variables
   - [ ] B Location variables
   - [X] C Report variables
   - [X] D Language variables
   - [ ] E Date variables
   - [X] F Operating system variables

8. What is the purpose of the @execute function?

   The @Execute function allows you to reference a list of values in a SQL expression. It is designed to improve performance of SQL queries, especially when the list of values definition is a complex query.

UNIT 10 Relative-Time Objects

Lesson 1

UNIT OBJECTIVES

- Create relative-time objects

Unit 10
Lesson 1

Creating Relative-Time Objects

LESSON OVERVIEW
This lesson explains how to create relative-time objects.

LESSON OBJECTIVES
After completing this lesson, you will be able to:

- Create relative-time objects

Relative-Time Objects

Typically, universes are designed that directly reflect only the contents and structure of the underlying database. When building a report based on this universe, it turns out that many questions cannot be answered using the simple objects available. Working with dates, users must calculate a date from a different date. In other words, the calculated date is relative to another date.

You can extend the universe with additional complex objects using database-specific functions. Each database has a number of date functions that give the base syntaxes for a current date, current year, current month, and so on. By adding or subtracting numbers, you can create previous and future date syntaxes.

Unit 10
Exercise 25

Create Relative-Time Objects

Business Example

Prompt objects are constructed using database date functions, to return an input base for end users to enter a year, month, or a full date value. These prompted values are compared to the actual database date column (or datepart), for example SALE.SALE_DATE.

A CASE statement can be used to compare the prompted period with the actual database column period, and build boolean "reference" objects, or "flags" that can be used to build relative date-time measures objects. If the prompted period input value matches the actual database column period, the CASE statement returns a 1 value.

After creating the boolean flags, the final stage is to build the relative date-time measure objects. For each 1 value of the boolean flag, the relative date-time measure value is returned.

Hint:
Time objects are not created until the last three objects of the activity; the rest of the objects, however, are used by those three objects.

Create a Date Parameter and Change Objects to Numeric Data Types

1. In the Information Design Tool, open your Motors_xx business layer, if it is not already open.
2. Create a new parameter in the business layer called **`Date Prompt for Relative Time`** with the following information:

Field	Value
Name	**`Date Prompt for Relative Time`**
Prompt text	**`Select a date or type a date.`**
List of Values	*Sales Date list of values*
Data Type	Date
Allow Multiple Values	Uncheck the box

3. Change the SELECT statements of the Year and Month objects in the Sales Dates subfolder using the following information:

Object	Field	Value	Type
Sales Year	SELECT syntax	`datepart(yyyy,SALE.SALE_DATE)`	Numeric

Object	Field	Value	Type
Sales Month	SELECT syntax	`datepart(mm,SALE.SALE_DATE)`	Numeric

Create a Date Prompt and Related Objects

1. In the *Sales* folder, create a new **`YTD Date Objects`** subfolder.
2. In the *YTD Date Objects* subfolder, create the following sale date prompt objects as listed in the following tables.

   For the *Sales Date* prompt, use the following information:

Field	Value
SELECT syntax (Using the SQL Assistant, drag the *Date Prompt for Relative Time* parameter into the SELECT statement)	`@Prompt(Date Prompt for Relative Time)` Note: The Sales Date Prompt object contains parameter in the SELECT clause. A parameter is handled in the WHERE clause and the SELECT contains the table.column reference. For the purpose of feeding through the prompted value into other object SELECT clauses using the @select function, the parameter is placed in the Sales Date Prompt object's SELECT clause. The table reference is set using the Tables button. Change the type to Date.
Description	Prompts the user to choose a sales date from the list of values or manually enter a date value.
Type	Date
Tables button	Select SALE as the reference table.

For the *Year In prompt*, use the following table:

Field	Value
SELECT syntax	`datepart(yyyy,@select(YTD Date Objects\Sales Date Prompt))`
Description	Extracts the year from the entered sales date prompt value.

Field	Value
Type	Number

For the *Month In prompt*, use the following table:

Field	Value
SELECT syntax	`datepart(mm,@select(YTD Date Objects\Sales Date Prompt))`
Description	Extracts the month from the entered sales date prompt value.
Type	Number

Create Boolean Objects Comparing Prompted Date Against Database Dates

1. In the *YTD Date Objects* subfolder, create the following indicator or boolean flag objects that are used to create relative date-time measure objects in the remaining activity steps. Use the information in the following tables.

> Note:
> Hide the boolean flag objects. We do not recommend that they be available to end users.

> Hint:
> You can generate the @select statements shown in the table by selecting the objects and functions in the *Edit Select* menu.

For the *Sales Month/Year Match Flag*, use the following information:

SELECT syntax	Description
`Case WHEN @Select(Sales\Sales Dates\Sales Year)= @Select(YTD Objects\Year in Prompt) AND @Select(Sales\Sales Dates\Sales Month)= @Select(YTD Objects\Month in Prompt) THEN 1 ELSE 0 END`	Indicator that returns 1 if both the year and the month of the sales date are equal to the year and month in the entered prompt date. Otherwise, it returns 0.
Type	Number

For the *Sales Month/Prev Year Match Flag*, use the following information:

SELECT syntax	Description
Case WHEN sign (@Select(Sales\Sales Dates \Sales Year)+1 - @Select(YTD Objects\Year in Prompt))=0 AND sign (@Select(Sales\Sales Dates \Sales Month) - @Select(YTD Objects\Month in Prompt))=0 THEN 1 ELSE 0 END	Indicator that returns 1 if the year of the sales date equals the year in the entered prompt date minus one year and the month of the sales date equals the month of the reference period. Otherwise, it returns 0.
Type	Number

For the *Later Months/Year Match Flag*, use the following information:

SELECT syntax	Description
Case WHEN @Select(Sales\Sales Dates \Sales Year) =@Select(YTD Objects\Year in Prompt) AND sign (@Select(Sales \Sales Dates\Sales Month)- (@Select(YTD Objects \Month in Prompt)-1))=1 THEN 1 ELSE 0 END	Indicator that returns 1 if the month of the sales date >= the month in the entered prompt date and the year of the sales date is equal to the year in the entered prompt date. Otherwise it returns 0.
Type	Number

2. Hide the entire *YTD Date Objects* subfolder.

Create and Test Relative Time Measure Objects

1. In the *Sales* folder, create a new **`YTD Measure Objects`** subfolder.
2. In the *YTD Measure Objects* subfolder, create the following relative date-time measure objects:

Hint:
The @select statements shown in the following table can be generated by choosing the objects and functions in the *Edit Select* menu.

Measure Object	SELECT Syntax
Sales Total	sum(SALE.SALE_TOTAL)

Measure Object	SELECT Syntax
Sales Total in Prompted Year/Month Period	`sum(SALE.SALE_TOTAL* @Select(YTD Objects\Sales Month/Year Match Flag))`
Sales Total in Previous Year Period	`sum(SALE.SALE_TOTAL* @Select(YTD Objects\Sales Month/Prev Year Match Flag))`
Sales Total in Current and Later Months of Prompted Year Period	`sum(SALE.SALE_TOTAL* @Select(YTD Objects\Later Months/Year Match Flag))`

3. Create a new query using the *Sales Date*, *Sales Total*, and *Sales Total in Prompted Year/Month Period* objects.

   The Total Sales in Prompted Year/Month Period object returns only sales data for the month and year values that equal the month and year values entered in the prompt, all other rows return 0.

4. Run a second query using the *Sales Date*, *Sales Total*, and *Total in Prompted Year/Month Period* objects.

   The Sales Total in Previous Year Period object returns only sales data for the month and year values that equal the month and year values entered in the prompt minus 1 year. All other rows return 0.

5. Run a new query using the *Sales Date*, *Sales Total*, and *Sales Total in Current and Later Months of Prompted Year Period* objects.

   The Sales Total in Current and Later Months of Prompted Year Period object returns sales data for the month and year values that equal the month and year values entered in the prompt as well as sales data for the months following, as long as months are within the same year. All other rows return 0.

Unit 10
Solution 25

Create Relative-Time Objects

Business Example

Prompt objects are constructed using database date functions, to return an input base for end users to enter a year, month, or a full date value. These prompted values are compared to the actual database date column (or datepart), for example SALE.SALE_DATE.

A CASE statement can be used to compare the prompted period with the actual database column period, and build boolean "reference" objects, or "flags" that can be used to build relative date-time measures objects. If the prompted period input value matches the actual database column period, the CASE statement returns a 1 value.

After creating the boolean flags, the final stage is to build the relative date-time measure objects. For each 1 value of the boolean flag, the relative date-time measure value is returned.

Hint:
Time objects are not created until the last three objects of the activity; the rest of the objects, however, are used by those three objects.

Create a Date Parameter and Change Objects to Numeric Data Types

1. In the Information Design Tool, open your Motors_xx business layer, if it is not already open.
2. Create a new parameter in the business layer called **`Date Prompt for Relative Time`** with the following information:

Field	Value
Name	**`Date Prompt for Relative Time`**
Prompt text	**`Select a date or type a date.`**
List of Values	*Sales Date list of values*
Data Type	Date
Allow Multiple Values	Uncheck the box

   a) On the business layer, choose the *Parameter and List of Values* pane.

   b) In the *Name* field, enter **`Date Prompt for Relative Time`**.

   c) In the *Prompt* text field, enter **`Select a date or type a date.`**

   d) In the List of Values area, associate a list of values by choosing the ellipses button, and choosing the *Sales Date list of values*.

3. Change the SELECT statements of the Year and Month objects in the Sales Dates subfolder using the following information:

Object	Field	Value	Type
Sales Year	SELECT syntax	`datepart(yyyy,SALE.SALE_DATE)`	Numeric
Sales Month	SELECT syntax	`datepart(mm,SALE.SALE_DATE)`	Numeric

   a) In the *Sales Dates* folder, click the *Sales Year* object.

   b) Using the information in the table, redefine the SELECT statement.

   c) Repeat steps a and b for the *Sales Month* object.

Create a Date Prompt and Related Objects

1. In the *Sales* folder, create a new **`YTD Date Objects`** subfolder.
   a) Right-click the *Sales* folder and choose *New* → *Folder*.
   b) Name the folder **`YTD Date Objects`**.

2. In the *YTD Date Objects* subfolder, create the following sale date prompt objects as listed in the following tables.

   For the *Sales Date* prompt, use the following information:

Field	Value
SELECT syntax (Using the SQL Assistant, drag the *Date Prompt for Relative Time* parameter into the SELECT statement)	`@Prompt(Date Prompt for Relative Time)` Note: The Sales Date Prompt object contains parameter in the SELECT clause. A parameter is handled in the WHERE clause and the SELECT contains the table.column reference. For the purpose of feeding through the prompted value into other object SELECT clauses using the @select function, the parameter is placed in the Sales Date Prompt object's SELECT clause. The table reference is set using the Tables button. Change the type to Date.
Description	Prompts the user to choose a sales date from the list of values or manually enter a date value.

Field	Value
Type	Date
Tables button	Select SALE as the reference table.

For the *Year In prompt*, use the following table:

Field	Value
SELECT syntax	`datepart(yyyy,@select(YTD Date Objects\Sales Date Prompt))`
Description	Extracts the year from the entered sales date prompt value.
Type	Number

For the *Month In prompt*, use the following table:

Field	Value
SELECT syntax	`datepart(mm,@select(YTD Date Objects\Sales Date Prompt))`
Description	Extracts the month from the entered sales date prompt value.
Type	Number

a) Right-click the subfolder and choose *New* → *Dimension*.

b) Name the object **`Sales Date Prompt`**.

c) Using the information in the table in step 5, define the SELECT statement.

d) Repeat steps a through c for the remaining objects.

Create Boolean Objects Comparing Prompted Date Against Database Dates

1. In the *YTD Date Objects* subfolder, create the following indicator or boolean flag objects that are used to create relative date-time measure objects in the remaining activity steps. Use the information in the following tables.

Note:
Hide the boolean flag objects. We do not recommend that they be available to end users.

Hint:
You can generate the @select statements shown in the table by selecting the objects and functions in the *Edit Select* menu.

For the *Sales Month/Year Match Flag*, use the following information:

SELECT syntax	Description
`Case WHEN @Select(Sales\Sales Dates \Sales Year)= @Select(YTD Objects\Year in Prompt) AND @Select(Sales\Sales Dates \Sales Month)= @Select(YTD Objects\Month in Prompt) THEN 1 ELSE 0 END`	Indicator that returns 1 if both the year and the month of the sales date are equal to the year and month in the entered prompt date. Otherwise, it returns 0.
Type	Number

For the *Sales Month/Prev Year Match Flag*, use the following information:

SELECT syntax	Description
`Case WHEN sign (@Select(Sales\Sales Dates \Sales Year)+1 - @Select(YTD Objects\Year in Prompt))=0 AND sign (@Select(Sales\Sales Dates \Sales Month) - @Select(YTD Objects\Month in Prompt))=0 THEN 1 ELSE 0 END`	Indicator that returns 1 if the year of the sales date equals the year in the entered prompt date minus one year and the month of the sales date equals the month of the reference period. Otherwise, it returns 0.
Type	Number

For the *Later Months/Year Match Flag*, use the following information:

SELECT syntax	Description
`Case WHEN @Select(Sales\Sales Dates \Sales Year) =@Select(YTD Objects\Year in Prompt) AND sign (@Select(Sales \Sales Dates\Sales Month)-(@Select(YTD Objects \Month in Prompt)-1))=1 THEN 1 ELSE 0 END`	Indicator that returns 1 if the month of the sales date >= the month in the entered prompt date and the year of the sales date is equal to the year in the entered prompt date. Otherwise it returns 0.
Type	Number

a) Right-click the subfolder and choose *New* → *Dimension*.

b) Name the object **`Sales Month/Year Match Flag`**.

c) Using the information in the table in step 6, define the SELECT statement.

d) Repeat steps a through c for the remaining objects.

2. Hide the entire *YTD Date Objects* subfolder.
   a) Right-click the *YTD Date Objects* subfolder and choose *Change State* → *Hidden*.

Create and Test Relative Time Measure Objects

1. In the *Sales* folder, create a new **`YTD Measure Objects`** subfolder.
   a) Right-click the *Sales* folder and choose *New* → *Folder*.
   b) Name the folder **`YTD Measure Objects`**.
2. In the *YTD Measure Objects* subfolder, create the following relative date-time measure objects:

Hint:
The @select statements shown in the following table can be generated by choosing the objects and functions in the *Edit Select* menu.

Measure Object	SELECT Syntax
Sales Total	`sum(SALE.SALE_TOTAL)`
Sales Total in Prompted Year/Month Period	`sum(SALE.SALE_TOTAL* @Select(YTD Objects\Sales Month/Year Match Flag))`
Sales Total in Previous Year Period	`sum(SALE.SALE_TOTAL* @Select(YTD Objects\Sales Month/Prev Year Match Flag))`
Sales Total in Current and Later Months of Prompted Year Period	`sum(SALE.SALE_TOTAL* @Select(YTD Objects\Later Months/Year Match Flag))`

   a) Right-click the subfolder and choose *New* → *Dimension*.
   b) Name the object **`Sales Total`**.
   c) Using the information in the table in step 9, define the SELECT statement.
   d) Repeat steps a through c for the remaining objects.
3. Create a new query using the *Sales Date*, *Sales Total*, and *Sales Total in Prompted Year/Month Period* objects.

   The Total Sales in Prompted Year/Month Period object returns only sales data for the month and year values that equal the month and year values entered in the prompt, all other rows return 0.
   a) In the business layer, choose the *Query* drawer.
   b) Choose *Create Query*.

c) Into the *Refresh Objects* area, drag the *Sales Date*, *Sales Total*, and *Sales Total in Prompted Year/Month Period* objects.

d) Choose *Results*.

e) From the list of date values, choose a date.

f) Note the results.

4. Run a second query using the *Sales Date*, *Sales Total*, and *Total in Prompted Year/Month Period* objects.

   The Sales Total in Previous Year Period object returns only sales data for the month and year values that equal the month and year values entered in the prompt minus 1 year. All other rows return 0.

   a) In the business layer, choose the *Query* drawer.

   b) Choose *Create Query*.

   c) Into the *Refresh Objects* area, drag the *Sales Date*, *Sales Total*, and *Total in Prompted Year/Month Period* objects.

   d) Choose *Refresh*.

   e) From the list of date values, choose a date.

   f) Note the results.

5. Run a new query using the *Sales Date*, *Sales Total*, and *Sales Total in Current and Later Months of Prompted Year Period* objects.

   The Sales Total in Current and Later Months of Prompted Year Period object returns sales data for the month and year values that equal the month and year values entered in the prompt as well as sales data for the months following, as long as months are within the same year. All other rows return 0.

   a) In the business layer, choose the *Query* drawer.

   b) Choose *Create Query*.

   c) Into the *Refresh Objects* area, drag the *Sales Date*, *Sales Total*, and *Sales Total in Current and Later Months of Prompted Year Period* objects.

   d) Choose *Refresh*.

   e) From the list of date values, choose a date.

   f) Note the results.

LESSON SUMMARY
You should now be able to:

- Create relative-time objects

Unit 10

Learning Assessment

1. We recommend that boolean flag objects be available to end users.

   Determine whether this statement is true or false.

   ☐ True

   ☐ False

2. What is the benefit of using database-specific functions?

Learning Assessment - Answers

1. We recommend that boolean flag objects be available to end users.

   Determine whether this statement is true or false.

   ☐ True

   ☒ False

2. What is the benefit of using database-specific functions?

   Database-specific functions extend the universe with additional complex objects.