

SAP BOXI 3.1 Tutorial to Create Basic Universe



Applies to:

SAP Business Object XI 3.1. For more information, visit the [Business Objects homepage](#).

Summary

This tutorial is to help the beginners to create a basic universe in SAP BusinessObjects 3.1.

Author: Seema Mane

Company: Cognizant Technology Solutions

Created on: 19 July 2011

Author Bio

Seema Mane has been in the IT industry for over six years. She has extensive experience in the data-warehousing field. As part of her assignments, she works as a developer, team lead, on site co-coordinator, Coe in BI Technologies.

Table of Contents

Introduction 3

Create a New Universe and Define its Connection to the Database 3

 Learning Objectives 3

Building the Universe Structure 5

 Learning Objectives 5

Creating Dimension Objects 9

 Learning Objectives 9

Creating Measure Objects 12

 Learning Objectives 12

Related Content 14

Disclaimer and Liability Notice 15

Introduction

This tutorial is to help the beginners to create a basic universe in SAP BusinessObjects 3.1.

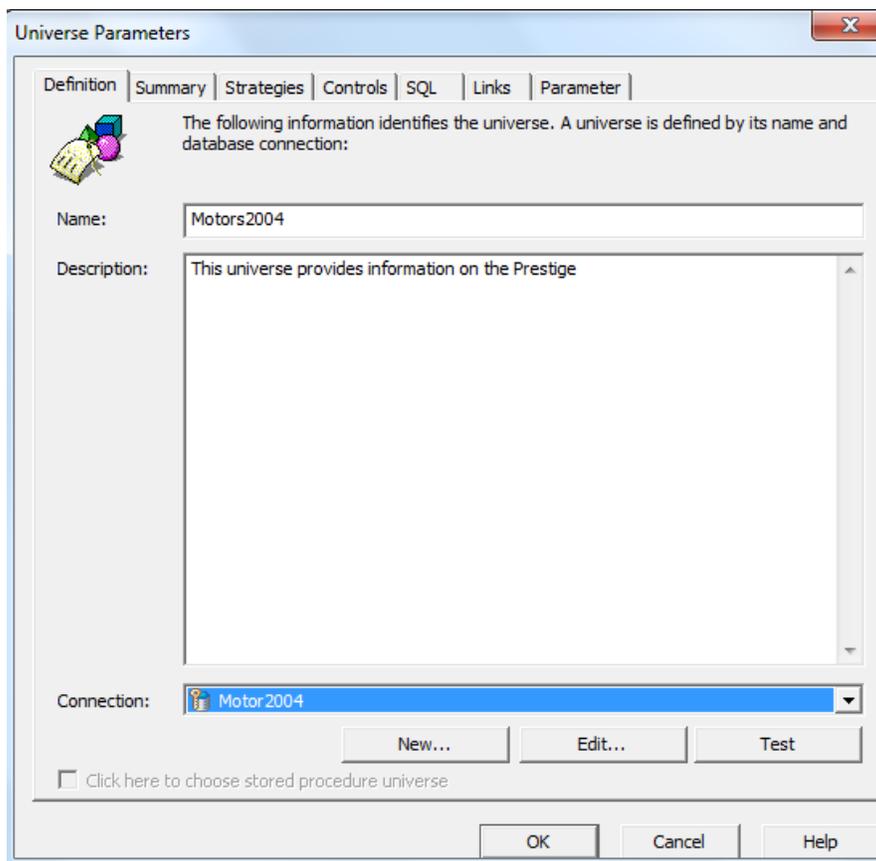
Create a New Universe and Define its Connection to the Database

Learning Objectives

Upon completion of this activity you will be able to create a new universe and define its connection to the database.

Here are the steps:

1. Motors2004.mdb is the access DB: Motor Cars Database for Showrooms, Models sold, Rental and Sales Business used for this tutorial you can used any DB.
2. Create a DSN called MotorsDSN.
3. Start a Designer session.
4. Log into the Enterprise server.
5. Create a new universe and define the following parameters:
Name = Motors2004
Description = this universe provides information on the Prestige
6. Connection = Motors2004, **Tip:** Use the New Connection Wizard to create the Motors2004 connection against the Motors2004.mdb.
7. Save the new universe in a file called Motors2004.unv.



8. Create another new universe and define the following parameters:
 - a. Name = Staff2004
 - b. Description = this universe provides information on the personnel of Prestige Cars.
 - c. Connection: Motors2004
9. Save the second universe in a file called Staff.unv.

Universe Parameters

Definition | Summary | Strategies | Controls | SQL | Links | Parameter

The following information identifies the universe. A universe is defined by its name and database connection:

Name: Staff2004

Description: This universe provides information on the personnel of Prestige Cars.

Connection: Motor2004

New... Edit... Test

Click here to choose stored procedure universe

OK Cancel Help

Building the Universe Structure

Learning Objectives

Upon completion of this activity you will be able to:

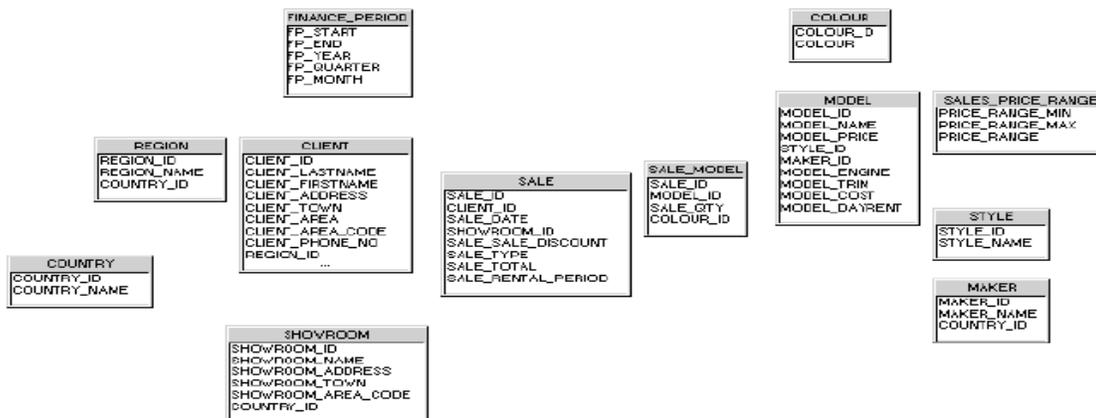
- Insert tables into the Motors2004 universe.
- Insert joins between tables in the Motors2004 universe and set cardinalities.

Here are the steps:

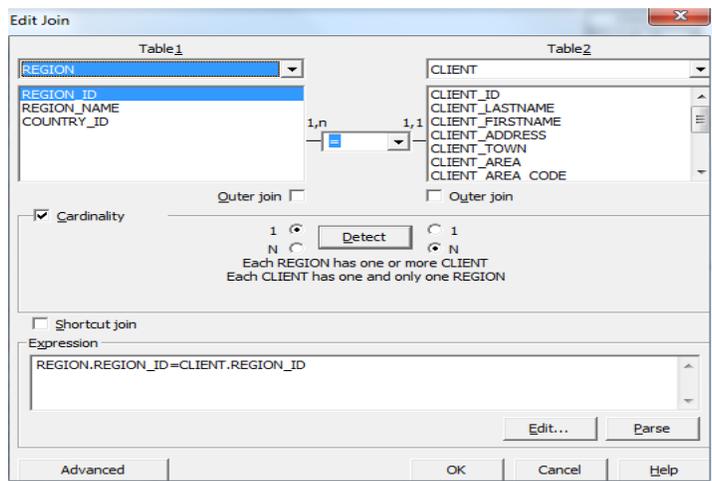
1. Insert the tables listed below into the Motors2004 universe.

- COUNTRY
- REGION
- CLIENT
- SALE
- SALE_MODEL
- COLOUR
- MODEL
- MAKER
- STYLE
- SALES_PRICE_RANGE
- SHOWROOM
- FINANCE_PERIOD.

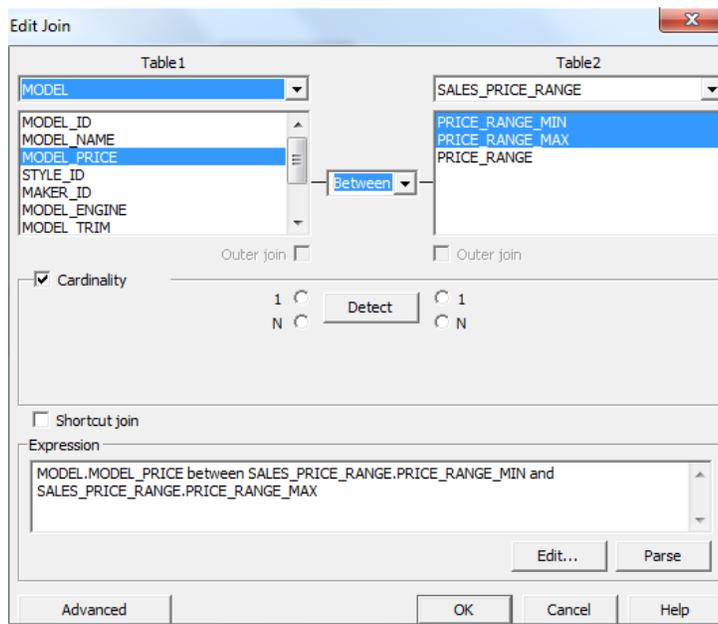
2. Order the tables so that they are laid out in the same way as the illustration below.



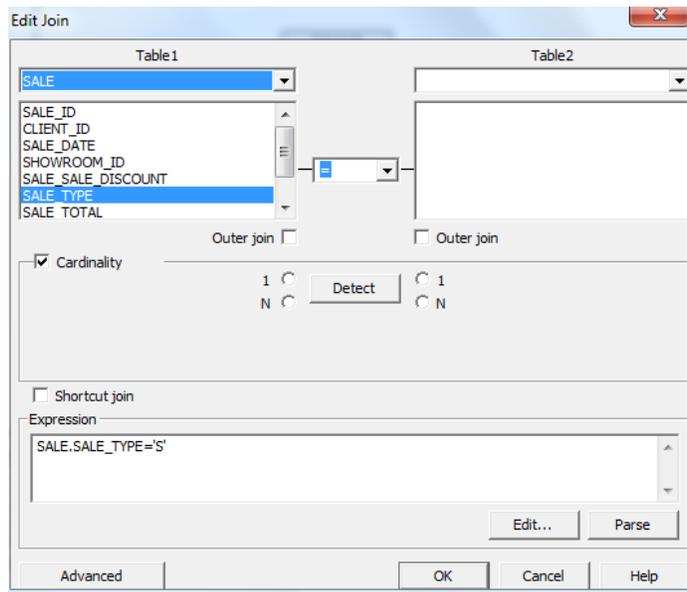
3. Save the changes to the universe.
4. Insert the following equi-join using the drag and drop technique:
 - a. COUNTRY.COUNTRY_ID to REGION.COUNTRY_ID
5. Insert the following equi-join using the Edit Join dialog box.
 - a. REGION.REGION_ID to CLIENT.REGION_ID



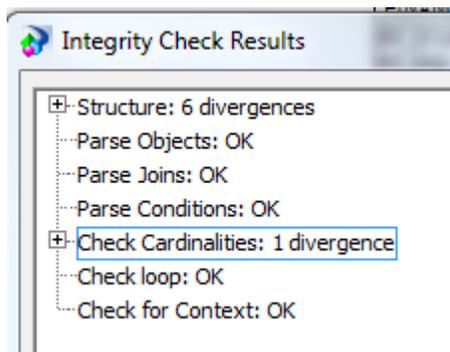
6. Insert the following theta join:
 - a. MODEL.MODEL_PRICE BETWEEN
 - b. SALE_PRICE_RANGE.PRICE_RANGE_MIN AND
 - c. SALE_PRICE_RANGE.PRICE_RANGE_MAX



7. Insert the following self-restricting join.
 - a. SALE.SALE_TYPE='S'



8. Manually set the cardinality for the following join:
 - a. REGION.REGION_ID to CLIENT.REGION_ID
9. Detect the cardinality for the following join.
 - a. COUNTRY.COUNTRY_ID to REGION.COUNTRY_ID
10. Check integrity for:
 - a. Universe structure and joins
11. Remedy the divergences found in the integrity check of the Motors2004 universe using manual techniques. **Tip:** The remaining joins required involve inserting equi and theta joins only.



12. Check integrity to ensure no divergences in the Motors2004 universe remain.
13. Check that the universe structure includes all of the joins in the table below:

Joins	Type
COUNTRY.COUNTRY_ID = REGION.COUNTRY_ID	Equi
REGION.REGION_ID = CLIENT.REGION_ID	Equi
MODEL.MODEL_PRICE BETWEEN SALE_PRICE_RANGE.PRICE_RANGE_MIN AND SALE_PRICE_RANGE.PRICE_RANGE_MAX	Theta
SALE.SALE_TYPE='S'	Self
CLIENT.CLIENT_ID=SALE.CLIENT_ID	Equi
SHOWROOM.SHOWROOM_id = SALE.SHOWROOM_ID	Equi
SALE_MODEL.MODEL_ID = MODEL.MODEL_ID	Equi
MODEL.STYLE_ID = STYLE.STYLE_ID	Equi
MODEL.MAKER_ID = MAKER.MAKER_ID	Equi
SALE_MODEL.COLOUR_ID=COLOUR.COLOUR_ID	Equi
SALE.SALE_DATE BETWEEN FINANCE_PERIOD.FP_START AND FINANCE_PERIOD.FP_END	Theta
SALE.SALE_ID = SALE_MODEL.SALE_ID	Equi

14. Save the changes to the universe.

Joins	Type	Cardinality
COUNTRY.COUNTRY_ID = REGION.COUNTRY_ID	Equi	1:N
REGION.REGION_ID = CLIENT.REGION_ID	Equi	1:N
MODEL.MODEL_PRICE BETWEEN SALE_PRICE_RANGE.PRICE_RANGE_MIN AND SALE_PRICE_RANGE.PRICE_RANGE_MAX	Theta	N:1
SALE.SALE_TYPE='S'	Self	
CLIENT.CLIENT_ID=SALE.CLIENT_ID	Equi	1:N
SHOWROOM.SHOWROOM_id = SALE.SHOWROOM_ID	Equi	1:N
SALE_MODEL.MODEL_ID = MODEL.MODEL_ID	Equi	N:1
MODEL.STYLE_ID = STYLE.STYLE_ID	Equi	N:1
MODEL.MAKER_ID = MAKER.MAKER_ID	Equi	N:1
SALE_MODEL.COLOUR_ID=COLOUR.COLOUR_ID	Equi	N:1
SALE.SALE_DATE BETWEEN FINANCE_PERIOD.FP_START AND FINANCE_PERIOD.FP_END	Theta	N:1
SALE.SALE_ID = SALE_MODEL.SALE_ID	Equi	1:N

Creating Dimension Objects

Learning Objectives

Upon completion of this activity you will be able to create and test classes and objects.

Here are the steps:

1. Create the following class: Client.
2. Create the following class and subclass:
 - a. Car
 - b. Sale Prices (subclass of Car)
3. Automatically create the following class from its table in the structure: Region
4. Remove the following class: Region.
5. Create a Client Name dimension object manually with the settings:
 - a. Type = Character
 - b. Description = Surname, Forename
 - c. SELECT = CLIENT.CLIENT_LASTNAME + ', ' + CLIENT.CLIENT_FIRSTNAME
 - d. No Associated LOV
6. Create a Client ID detail object automatically with the settings:
 - a. Type = Number
 - b. Description = Unique Client ID Number
 - c. SELECT = CLIENT.CLIENT_ID
 - d. No Associated LOV
7. Check the integrity of the objects.
8. Create the Showroom class.
9. Create objects for each of the classes as identified in the tables below. Some of the properties for each object have been specified for you. However, you will have to determine the data type, qualification, and whether or not a LOV should be associated with each object.

Car class

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, salon, estate)
Model	MODEL.MODEL_NAME & '' & MODEL.MODEL_TRIM & '' & MODEL.MODEL_ENGINE	Model name, trim, and engine size

Sale Prices class (a subclass of Car)

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

Showroom class

Object Name	SELECT Statement	Object Description
Town	SHOWROOM.SHOWROOM_TOWN	Town in which showroom exists
Showroom	SHOWROOM.SHOWROOM_NAME	Name of showroom
Address	SHOWROOM.SHOWROOM_ADDRESS	Address of showroom

Client class

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)
Town	CLIENT.CLIENT_TOWN	Town or city in which client resides
Phone Number	CLIENT.CLIENT_PHONE_NO	Phone number of client
Address	CLIENT.CLIENT_ADDRESS	Address of client

10. Create Area Code (CLIENT.CLIENT_AREA_CODE) object under the client Class.
11. Create Sales Class.

	Object Name	SELECT Statement	Object Description
Class	Sales		
Sub-Class	Sales Details		
	Invoice ID Number	SALE.SALE_ID	Unique Invoice ID Number

12. Create Financial Class.

Financial class

Object Name	SELECT Statement	Object Description
Financial Year	FINANCE_PERIOD.FP_YEAR	For example, FY03-04
Financial Quarter	FINANCE_PERIOD.FP_QUARTER	For example, Q1
Financial Month	FINANCE_PERIOD.FP_MONTH	For example, Month 01

13. Make sure you have defined each object using the appropriate object type.
14. Save the universe.
15. Check the integrity of the objects, and make any alterations required. **Note:** Test the validity of the joins also.
 - a. Save the universe again and export to the Enterprise server.
 - b. In InfoView, create a new Web Intelligence document and select the Motors2004 universe that has been exported.
 - c. Build a query in the Java Report Panel using the objects created and test the results.

Creating Measure Objects

Learning Objectives

Upon completion of this activity you will be able to create and test measure objects.

Here are the steps:

1. Create the following subclass in the Sales class.
 - a. Sales Figures
2. Create a Sales Revenue measure object with the settings:
 - a. Type = Number
 - b. Description = Total Sale Invoice Revenue
 - c. Select = `SUM (SALE_MODEL.SALE_QTY * MODEL.MODEL_PRICE * ((100 - SALE.SALE_SALE_DISCOUNT) / 100))`
 - d. Function = SUM
 - e. No Associated LOV
3. Test the Sales Revenue object as follows:
 - a. Save the universe. Export the universe and log in to InfoView. Create a new Web Intelligence document and select the universe just exported.
 - b. Build a query containing only the Sales Revenue object. Note the value returned.
 - c. Edit the query by adding the Country dimension object. Check the SQL and note the GROUP BY clause. It should contain the SQL for the Country object. Run the query. Apply a SUM calculation to the Sales Revenue column of the projected block. Does the sum match the value of the previous query? Note the value of the Sales Revenue in a row of the block (for example, USA).
 - d. Edit the query by adding the Region dimension object. Check the SQL and note the GROUP BY clause. It should now contain the SQL for the Country and Region objects. Run the query. Apply a break to the Country column and reapply the sum calculation to the Sales Revenue column of the projected block. Does the Country group sum match the value of the noted block row from the previous query (e.g. USA)?
4. Change the projection to the block from the microcube by removing the Region column from the block. Does it aggregate to Country level correctly?
 - a. Edit projection to the block from the microcube by removing the Country column from the block. Does it aggregate to the total Sales Revenue level correctly?
 - b. Create another query using the Showroom and Sales Revenue objects, then another using Maker and Sales Revenue. Note the total values remain the same.
5. In Designer, create the following measure objects in the Sales Figures subclass and test them. The SQL code for the Select properties of each object has been specified. However, determine the appropriate projection function aggregate.

Object Name	Select Statement	Object Description
Cost of Car Sales	<code>SUM(SALE_MODEL.SALE_QTY * MODEL.MODEL_COST)</code>	Total Cost of Car Sales
Number of Cars Sold	<code>SUM(SALE_MODEL.SALE_QTY)</code>	Total Number of Cars Sold

Object Name	Select Statement	Object Description
Cost of Car Sales	$SUM(SALE_MODEL.SALE_QTY * MODEL.MODEL_COST)$	Total Cost of Car Sales
Number of Cars Sold	$SUM(SALE_MODEL.SALE_QTY)$	Total Number of Cars Sold

6. Create the measure objects listed below. Place them in the appropriate existing class or subclass and then test them in Web Intelligence. Lowest Priced Value based on Manufacturers recommended retail price.
 - a. Highest Priced Value based on Manufacturers recommended retail price.
 - b. Number of Clients. **Note:** Two of the above objects will produce inconsistent results that are sometimes incorrect. The reason for this is the same in both instances. When testing the objects, this should become apparent. Do not attempt to remedy the issue. Just identify what the problem is and why it is occurring.

(**Resolution:** Click the Tables button in the Edit Properties Definition tab for both the Cost of Car Sales and Number of Cars Sold objects. Select the Sale table to force a join to that table as well. To test, create a query with Showroom and Cost of Car Sales, and then create a query with Maker and Cost of Car Sales.)
7. Save the universe and export it to the Enterprise server.
8. Create an additional measure object as shown below:

Object Name	Select Statement	Object Description
Average Revenue per Car Sold	$SUM(SALE_MODEL.SALE_QTY * MODEL.MODEL_PRICE * ((100 - SALE.SALE_SALE_DISCOUNT) / 100)) / SUM(SALE_MODEL.SALE_QTY)$	Average Revenue per Car Sold, based on Revenue divided by Numbers Sold

- a. Set the Select Projection function to AVERAGE.
9. Save and export the universe. Test the finished objects by building three queries in Web Intelligence:
 - b. Query 1: Country, Average Revenue per Car Sold
 - c. Query 2: Town, Average Revenue per Car Sold
 - d. Query 3: Country, Town, Average Revenue per Car Sold

Which Country and Town spends the most on cars on average?

What happens to the average if Town is sliced away from Query 3?

What conclusion can be drawn from this?

Create another document to demonstrate this.

1. Create a query to return Country, Town, Sales Revenue, and Number of Cars Sold.
2. Create a variable to show the average.
3. Insert a sum on the Sales Revenue and Number of Cars Sold columns
4. Create a second query to return Country and the Average Revenue per Cars Sold.
5. Compare the results.

Web Intelligence has summed the average, which are not accurate results.

Resolution: Modify the Average Revenue object so that the projected aggregate on the Edit Object Properties tab is set to none. Recreate the query and the results are correct or create a variable in the report.

Related Content

For more information, visit the [Business Objects homepage](#).

Disclaimer and Liability Notice

This document may discuss sample coding or other information that does not include SAP official interfaces and therefore is not supported by SAP. Changes made based on this information are not supported and can be overwritten during an upgrade.

SAP will not be held liable for any damages caused by using or misusing the information, code or methods suggested in this document, and anyone using these methods does so at his/her own risk.

SAP offers no guarantees and assumes no responsibility or liability of any type with respect to the content of this technical article or code sample, including any liability resulting from incompatibility between the content within this document and the materials and services offered by SAP. You agree that you will not hold, or seek to hold, SAP responsible or liable with respect to the content of this document.