IBM DB2 Cube Views

Business Modeling Scenarios

Version 8

SC18-7803-00

Note

Note: Before using this information and the product it supports, read the information in "Notices" on page 31.

Contents

About this book

This book provides supplemental information to the *DB2 Cube Views Setup and User's Guide*, about how to model common real-world business scenarios using DB2 Cube Views metadata.

Who should read this book

Read this book if you are a database administrator (DBA) who works with OLAP metadata and DB2 Universal Database™ (DB2). You should be familiar with:

- OLAP concepts, such as star schemas
- DB2 Cube Views metadata objects, such as cube models, facts objects, dimensions, joins, measures, and attributes

Chapter 1. Calculating the flow and value of the stock in a warehouse over time

A retail business, XYZ Retail, keeps its stock in a warehouse before the stock is sent to a particular store to be sold. XYZ Retail maintains data about the state of the stock in the warehouse over time, and wants to analyze this data. In particular, the company wants to examine two aspects of its warehouse:

- The flow of merchandise in to and out of the warehouse
- The value of the merchandise in the warehouse at a given time

The first aspect, the flow of the merchandise involves looking at data over time. The second aspect, the value of the merchandise takes a snapshot of the warehouse at a particular moment in time.

Details of scenario

XYZ Retail has a fact table with the following warehouse-related columns: QUANTITY_IN, QUANTITY_OUT, CURRENT_QUANTITY, PRODUCT_VALUE, PRODUCT_ID, and TIME_ID. This data is entered in the table on a weekly basis. The database also has a Product table and Time table. For example, a set of sample fact table data is shown in Table 1.

Table 1.

PRODUCT _ID	TIME_ID	QUANTITY _ID	QUANTITY _OUT	CURRENT _QUANTITY	PRODUCT _VALUE
1234	1	5	0	5	5
1234	2	20	10	15	5
1234	3	10	20	5	5

The PRODUCT_ID value for each of the three sample data entries is the same because one product type can move in and out of the warehouse multiple times.

The DBA for XYZ Retail must create three different measures:

Flow In
: Models the flow of merchandise into the warehouse.

Flow Out
: Models the flow out of the warehouse.

Current Value
Models the value of the merchandise at a given time.

To create the first two measures, Flow In and Flow Out, the DBA creates measures that map to the QUANTITY_IN and QUANTITY_OUT columns respectively, and sum the data across all dimensions. This is known as a fully additive measure because the data is aggregated using only the SUM function across all dimensions. For example, Table 2 shows a set of sample data for the QUANTITY_IN and QUANTITY_OUT columns for three months for the product with a PRODUCT_ID of 1234. The Flow In and Flow Out measures sum these monthly values to calculate the total quantities that came in and out of the warehouse over the quarter.

Table 2. Calculation of the sample data for the Flow In and Flow Out fully additive measures for PRODUCT_ID 1234

	January	February	March	Quarter 1
QUANTITY_IN	5	20	10	35
QUANTITY_OUT	0	10	20	30

Fully additive measures are the simplest and most common measures to create and are often used as the building blocks for more complex measures. For measures based on numeric source data, the OLAP Center creates a fully additive measure by default.

To create the third measure, Current Value, the DBA creates a calculated measure that computes the value by multiplying PRODUCT_VALUE by CURRENT_QUANTITY. For example, if the value of the product with PRODUCT_ID=1234 is 5, then the Current Value measure for the sample data is shown in Table 3.

Table 3. Calculation of the sample data for the Current Value measure for PRODUCT_ID 1234

	January	February	March
CURRENT_QUANTITY	5	10	20
Current Value	25	50	100

This data must then be aggregated across the dimensions. However, because this measure is calculating the value at a specific point in time, it does not make sense to sum across the time dimension. Instead, the aggregation will sum the data across the Product dimension, and find the average of the data over time. This is known as a semi-additive measure because only part of the aggregation involves the SUM function.

Measures that calculate snapshot data, data that represents a particular moment in time such as month inventory data, are often semi-additive measures because it does not make sense to add the months into quarters. If a product remains in the warehouse for an entire quarter, that product is included in the CURRENT_QUANTITY snapshot data of the warehouse inventory each of the three months of the quarter. If the CURRENT_QUANTITY data is summed over time, the product that sat in the warehouse for three months is counted three times. As shown in Table 4, the value 25 for Quarter 1 has no significance to the warehouse's activities. The table shows that the warehouse never had 25 products in the warehouse, so calculating the value of 25 products has no meaning.

Table 4. Calculation of the sample data for the CURRENT_QUANTITY column using the SUM function for the time dimension for PRODUCT_ID 1234

	January	**February**	**March**	**Quarter 1**
SUM(CURRENT_ QUANTITY)	5	15	5	25

Instead of using the SUM function across all dimensions, you can perform other aggregation functions such as AVG, MIN, and MAX for the time dimension. For example, with the same set of sample data for January, February, and March, you can use a second aggregation function for the time dimension as shown in Table 5 to create meaningful values for the Quarter. The Current Value measure can represent the average total value of merchandise stored in the warehouse over the quarter, or the maximum or minimum value at any point in time during the quarter.

Table 5. Calculation of the sample data for the CURRENT_QUANTITY column using the AVG, MAX, and MIN functions for the time dimension for PRODUCT_ID 1234

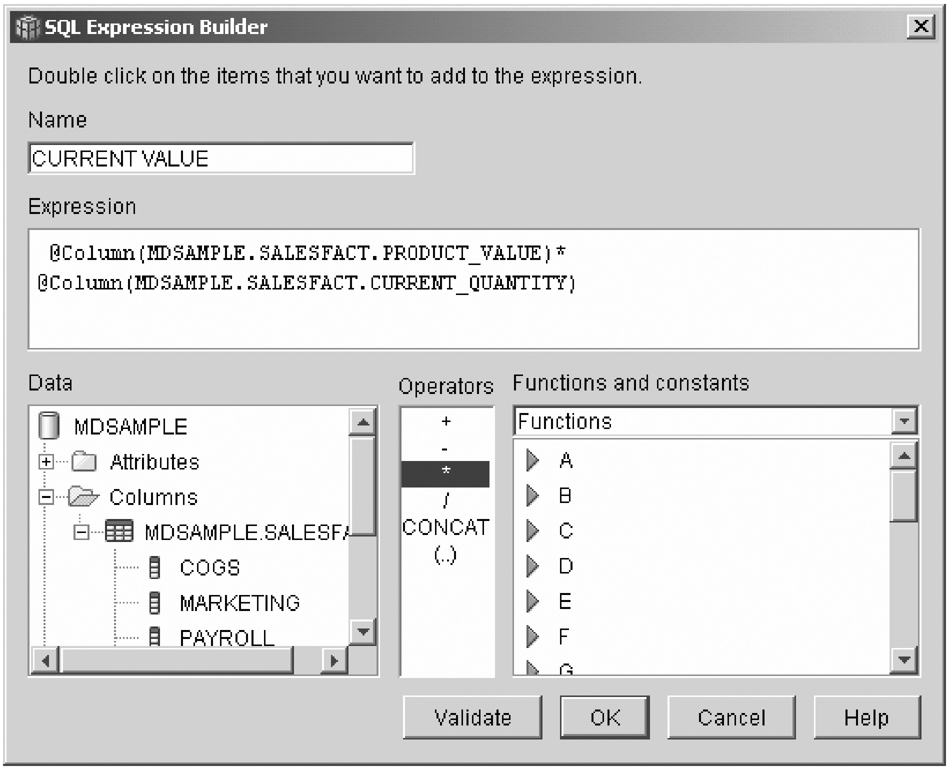
	January	**February**	**March**	**Quarter 1**
AVG(CURRENT_ QUANTITY)	5	15	5	8.3
MAX(CURRENT_ QUANTITY)	5	15	5	15
MIN(CURRENT_ QUANTITY)	5	15	5	5

Steps to create measures

The following steps explain how you could use the OLAP Center Facts Properties window to create the Flow In, Flow Out, and Current Value measures in an existing facts object:

1. To open the Facts Properties window, right-click the facts object in the OLAP Center object tree, and click **Edit Measures**. The Facts Properties window opens.
2. Create the Flow In measure:
   a. On the Measures page, click **Create Calculated Measure** to create the Flow In measure. The SQL Expression Builder window opens.
   b. In the SQL Expression Builder window, type `FLOW IN` in the **Name** field.
   c. To create the flow in expression, complete the following steps:
      - Expand the **Columns** folder and the fact table in the Data list.
      - Double-click the **QUANTITY_IN** column to add it to the expression.
      - Click **OK** to close the SQL Expression Builder window. You do not need to change the default aggregation function, SUM, on the Aggregations page. The SUM function is the default for the Flow In measure because the data source is numeric and the measure refers to a column, not to only existing measures.
3. Create the Flow Out measure:
   a. On the Measures page, click **Create Calculated Measure** to create the Flow Out measure. The SQL Expression Builder window opens.
   b. In the SQL Expression Builder window, type `FLOW OUT` in the **Name** field.
   c. To create the flow out expression, complete the following steps:
      - Expand the **Columns** folder and the fact table in the **Data** list.
      - Double-click the **QUANTITY_OUT** column.
   d. Click **OK** to close the SQL Expression Builder window. You do not need to change the default aggregation function, SUM, on the Aggregations page. The SUM function is the default for the Flow Out measure because the data source is numeric and the measure refers to a column, not to only existing measures.
4. Create the Current Value measure:
   a. On the Measures page, click **Create Calculated Measure** to create the Current Value measure. The SQL Expression Builder window opens.
   b. In the SQL Expression Builder window, type `CURRENT VALUE` in the **Name** field.
   c. To create the Current Value expression, complete the following steps:
      - Expand the **Columns** folder and the fact table in the **Data** list.
      - Double-click the **PRODUCT_VALUE** column in the **Data** list.
      - Double-click the * operator in the **Operators** list.
      - Double-click the **CURRENT_QUANTITY** column in the **Data** list.

Figure 1 shows the Current Value expression that you can create in the SQL Expression Builder window.



Name: CURRENT VALUE

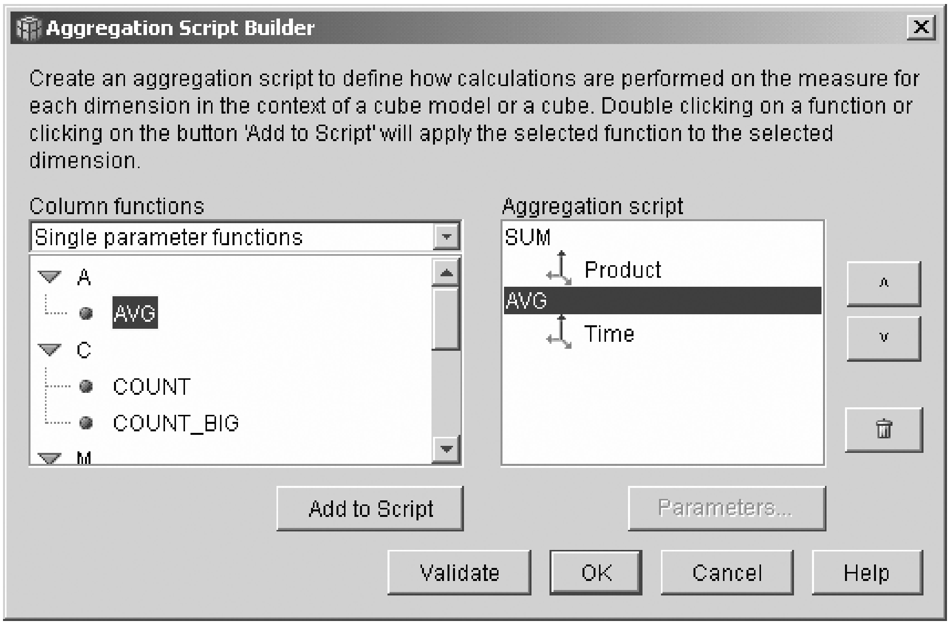
```
 @Column(MDSAMPLE.SALESFACT.PRODUCT_VALUE)*
@Column(MDSAMPLE.SALESFACT.CURRENT_QUANTITY)
```

Figure 1. Complete Current Value expression in the SQL Expression Builder window

d. Click **OK** to close the SQL Expression Builder window.

e. On the Aggregations page, click the aggregation for the Current Value measure, and click **Aggregation script** from the list. The Aggregation Script Builder window opens. The default aggregation script has the SUM function used for all dimensions.

f. If necessary, move the Time dimension down by selecting **Time** and clicking the push button, so that it is the last dimension listed in the script. Then with the Time dimension selected, double-click the **AVG** function in the **Column functions** list. The aggregation script, as shown in Figure 2 on page 6, sums the data across all of the dimensions except Time, over which the data is averaged.


Aggregation Script Builder

Create an aggregation script to define how calculations are performed on the measure for each dimension in the context of a cube model or a cube. Double clicking on a function or clicking on the button 'Add to Script' will apply the selected function to the selected dimension.

Column functions: Single parameter functions

Aggregation script:

```
SUM
    Product
AVG
    Time
```

Figure 2. Aggregation script for the Current Value measure

g. In the Aggregation Script Builder window, click **Validate** to verify that the aggregation script is valid. Click **OK** to save the aggregation script and close the window.

5. Click **OK** to save the changes to the facts object and to close the Facts Properties window.

You now have three calculated measures for the stock in the warehouse. You can use these measures to analyze the patterns of product flow in and out of your warehouse.

Chapter 2. Correlating advertising costs to sales

A car dealership is considering increasing its advertising spending. To make an educated decision, the dealership first wants to analyze the historical relationship between advertising spending and sales. The dealership is interested in determining if varying levels of advertising have affected sales, and in particular if increased advertising is closely associated with increased sales.

Details of scenario

The dealership's database has a fact table with Sales and Ad Costs columns. The database also has several other dimension tables. The DBA can create a measure that uses the DB2 CORRELATION function to perform correlation calculations between the costs and the sales. The CORRELATION function is a multiparameter function that requires two input parameters. In this case, the DBA will use the Sales and Ad Costs columns as the two input parameters.

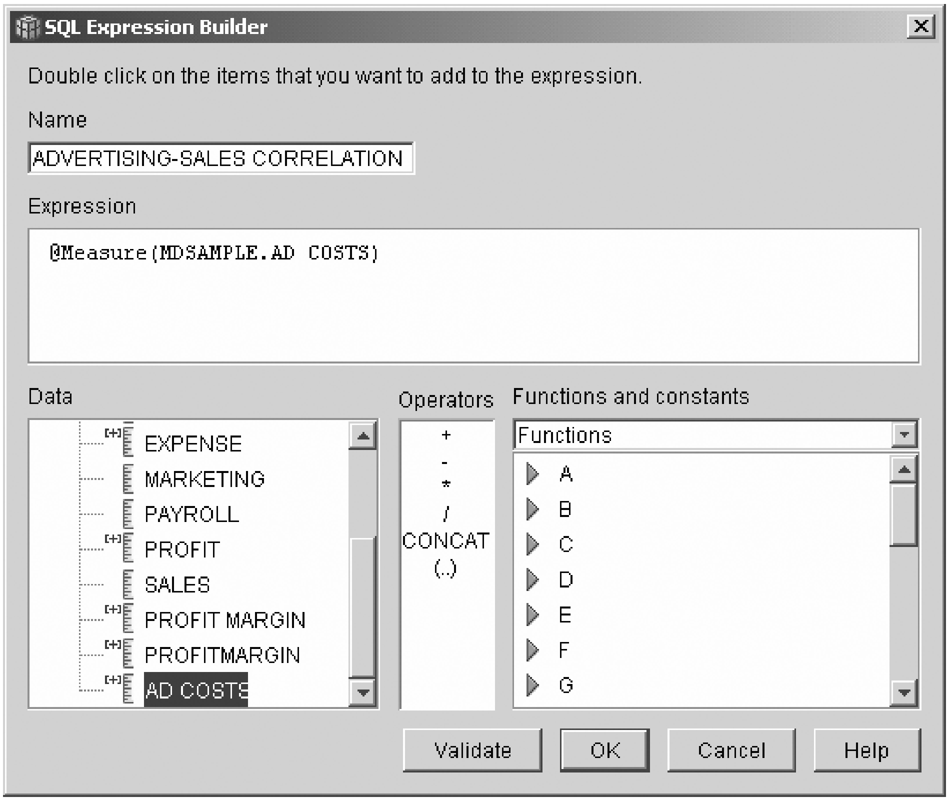
The DBA must apply the multiparameter aggregation function first in the aggregation script. The multiparameter function can be applied across all of the dimensions, or it can be applied first to all dimensions except for the Time dimension, and a second function, such as the MAX function, can be applied to the Time function. The DBA defines the SQL expression for the measure so that it maps directly to the Ad Costs column. The SQL expression is the first of the two parameters used in the multiparameter function. The DBA defines the second parameter as an SQL expression that maps directly to the Sales column. The CORRELATION function is defined as the only aggregation function so that the measure can calculate the statistical correlation between advertising costs and sales results across all dimensions.

Steps to create measure

The following steps explain how you could use the OLAP Center Facts Properties window to create the Advertising-Sales Correlation measure in an existing facts object:

1. Open the Facts Properties window by right-clicking the facts object in the OLAP Center object tree, and clicking **Edit Measures**.
2. Click the **Create Calculated Measure** push button. The SQL Expression Builder window opens.
3. In the SQL Expression Builder window, type `ADVERTISING-SALES CORRELATION` in the **Name** field.

4. Define the measure's expression that will also be used as the first parameter of the multiparameter CORRELATION function in the aggregation script. To define the expression, expand the **Measures** folder in the **Data** list and double-click the **AD COSTS** measure to add it to the **Expression** list. Figure 3 shows the expression that you create in the SQL Expression Builder window.



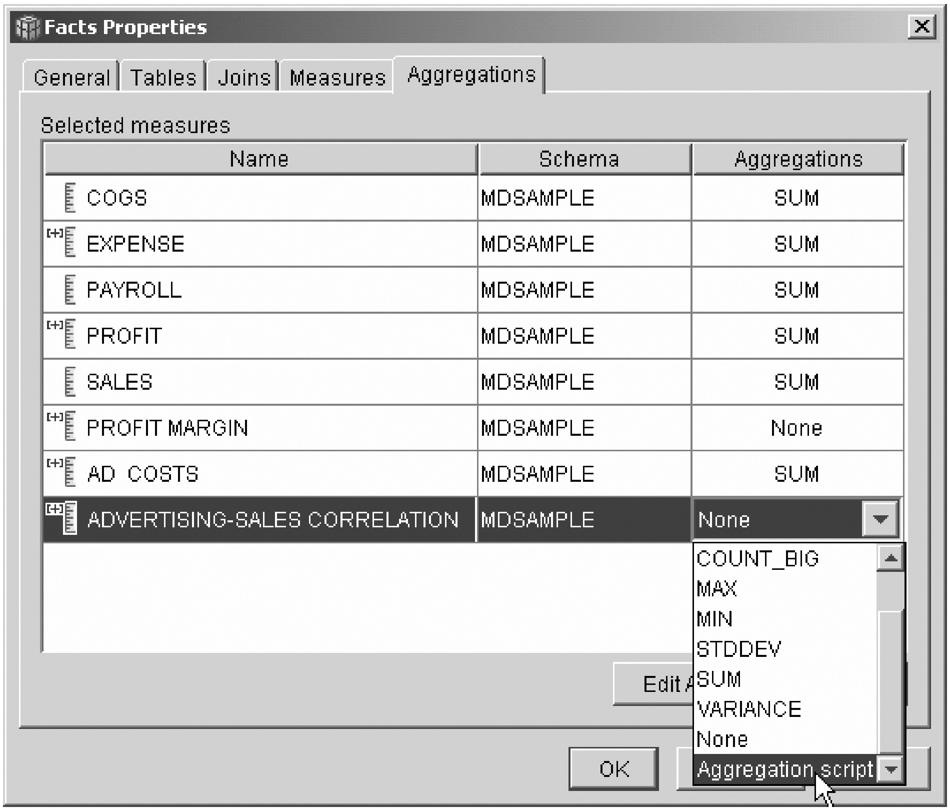
Name: ADVERTISING-SALES CORRELATION

Expression:

```
@Measure(MDSAMPLE.AD COSTS)
```

Figure 3. Complete advertising–sales correlation expression in the SQL Expression Builder window

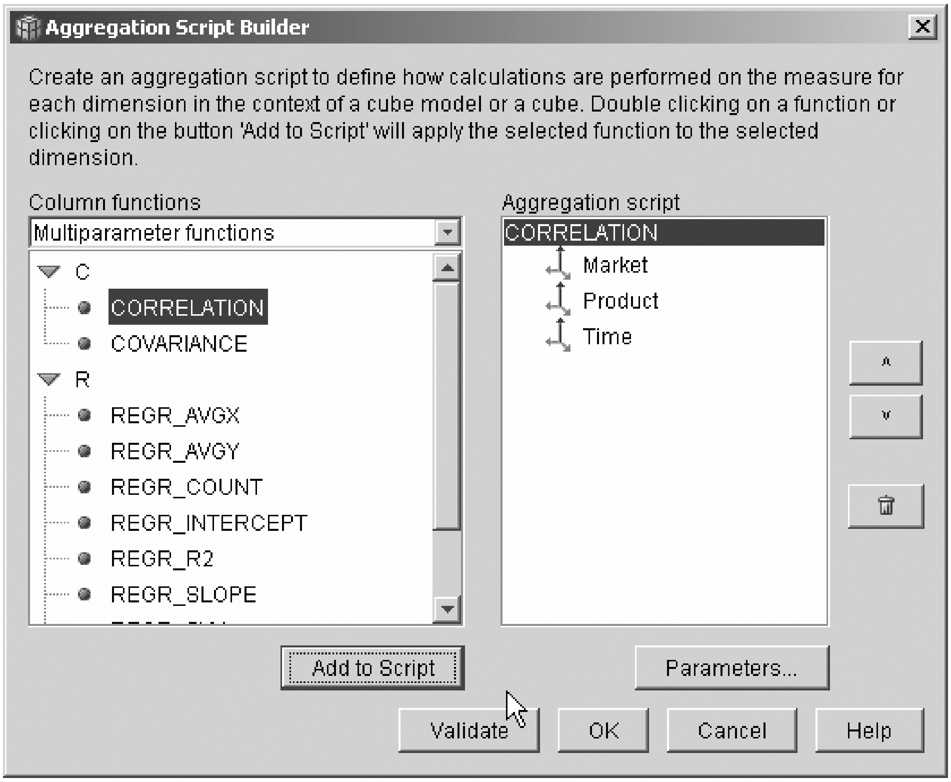
5. On the Aggregations page, click the aggregation function for the **ADVERTISING-SALES CORRELATION** measure and select **Aggregation script**, as shown in Figure 4 on page 9. The Aggregation Script Builder window opens.



Name	Schema	Aggregations
COGS	MDSAMPLE	SUM
EXPENSE	MDSAMPLE	SUM
PAYROLL	MDSAMPLE	SUM
PROFIT	MDSAMPLE	SUM
SALES	MDSAMPLE	SUM
PROFIT MARGIN	MDSAMPLE	None
AD COSTS	MDSAMPLE	SUM
ADVERTISING-SALES CORRELATION	MDSAMPLE	None

Figure 4. Aggregations page of the Facts Properties window

6. In the **Column functions** field, select **Multiparameter functions**. In the list of multiparameter functions, select the **CORRELATION** function and click **Add to Script**. Figure 5 on page 10 shows that the CORRELATION function is at the top of the list of dimensions in the script.



Create an aggregation script to define how calculations are performed on the measure for each dimension in the context of a cube model or a cube. Double clicking on a function or clicking on the button 'Add to Script' will apply the selected function to the selected dimension.

Column functions: Multiparameter functions

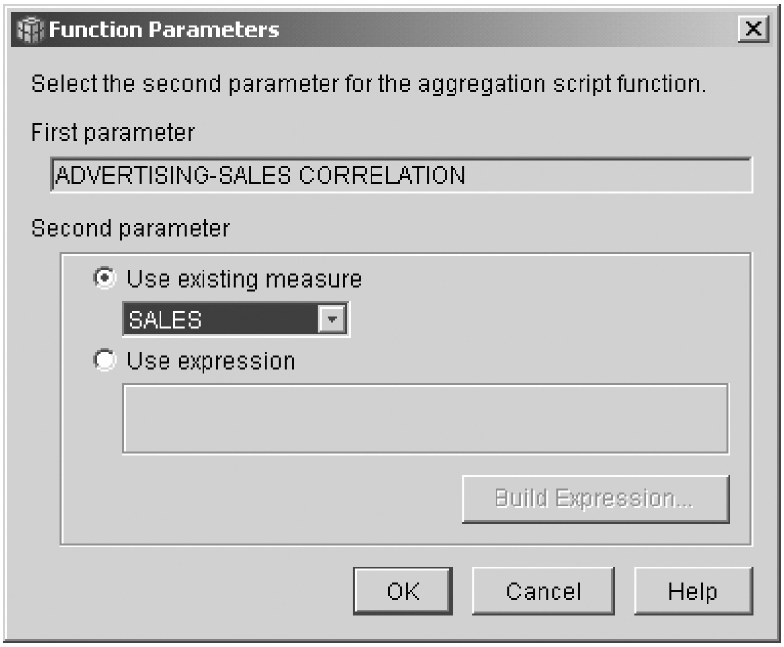
- C
  - CORRELATION
  - COVARIANCE
- R
  - REGR_AVGX
  - REGR_AVGY
  - REGR_COUNT
  - REGR_INTERCEPT
  - REGR_R2
  - REGR_SLOPE

Aggregation script:

- CORRELATION
  - Market
  - Product
  - Time

Figure 5. Aggregation script for the Advertising-Sales Correlation measure

7. Click the **Parameters** push button to specify the second parameter for the CORRELATION function. The Function Parameters window opens. Select **Using existing measure** and select **SALES**. Figure 6 on page 11 shows the Function Parameters window with the Sales measure specified as the second parameter. Click **OK** to save your selection and close the Function Parameters window.



Select the second parameter for the aggregation script function.

First parameter

ADVERTISING-SALES CORRELATION

Second parameter

[✓ Use existing measure]

SALES

Use expression

Figure 6. Sales measure specified as the second parameter in the Function Parameters window

8. In the Aggregation Script Builder window, click the **Validate** push button to verify that the aggregation script is valid. Click **OK** to save the aggregation script and close the window.
9. Click **OK** to save the changes to the facts object and to close the Facts Properties window.

You now have a measure that correlates two types of data in your database. You can use this measure to make decisions on future advertising spending based on historical trends in results.

Chapter 3. Calculating the profit and profit margin of a store

The general manager of a toy store wants to be able to analyze how various factors, such as time of year and type of product , affect the profit and profit margin. Before the more advanced analysis can be completed, the DBA for the toy store must first create profit and profit margin measures. Then the DBA can create additional measures that correlate and compare different factors to the profit and profit margin measures.

Details of scenario

The toy store's database has a fact table with Sales, Costs of Goods Sold (COGS), and Expense columns, in addition to corresponding foreign key columns for each of the several dimension tables. The DBA already created Sales, COGS, and Expense measures that map to the Sales, COGS, and Expense columns, respectively. The Profit and Profit Margin measures can be created entirely from these existing measures.

To create the Profit measure, the DBA creates a measure that calculates SALES-(COGS+EXPENSE) in the SQL Expression, and sums the calculated data across all dimensions. The Profit measure can be created by referencing existing measures, or columns, or a combination of both.

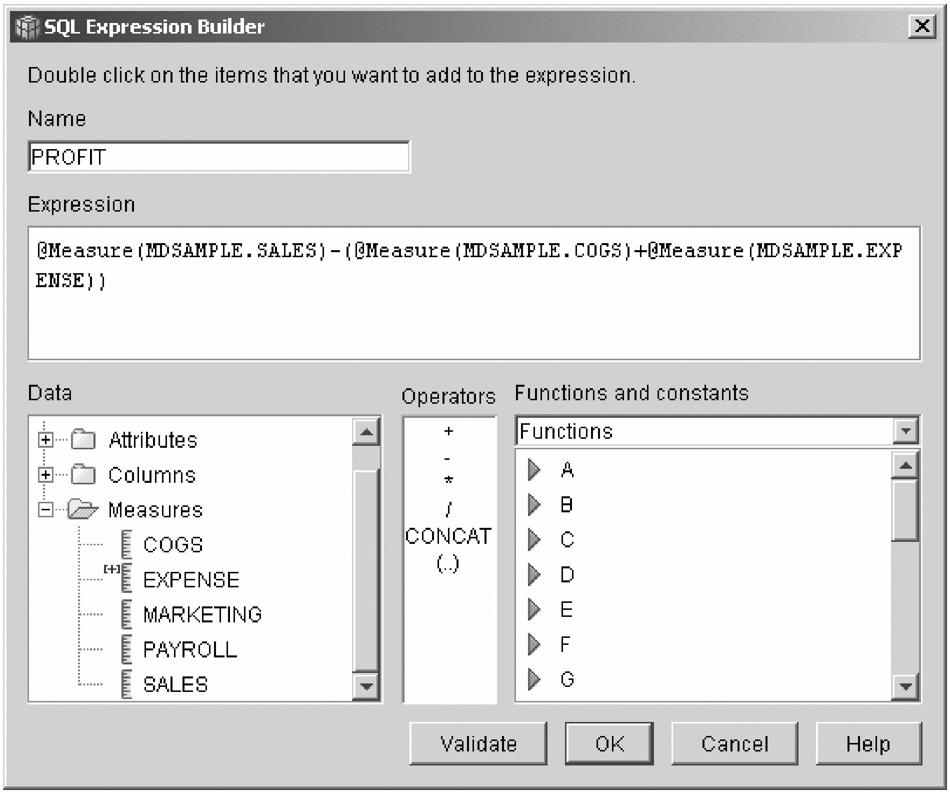
After the Profit measure is created, the DBA can create the Profit Margin measure. The Profit Margin measure is a ratio of two existing measures expressed as a percentage, (Profit / Sales)*100, and does not require its own aggregation function. An aggregation function is not required because the measure refers only to other measures whose data is already aggregated. If the DBA uses a composite measure, a measure that only references other measures, to calculate a ratio, the DBA does not need to define an additional aggregation. Most aggregation functions, such as SUM, do not make sense with ratios. For example, if the toy store has profit margins of 40%, 32%, 28%, and 37% for four consecutive quarters, summing the ratios over time, would result in a profit margin of 137% for the year, which does not make sense.

Steps to create the measures

The following steps explain how you could use the OLAP Center Facts Properties window to create the Profit and Profit Margin measures in an existing facts object:

1. To open the Facts Properties window, right-click the facts object in the OLAP Center object tree, and click **Edit Measures**. The Facts Properties window opens.
2. Create the Profit measure:
   a. On the Measures page, click the **Create Calculated Measure** push button. The SQL Expression Builder window opens.
   b. In the SQL Expression Builder window, type `PROFIT` in the **Name** field.
   c. To create the Profit expression, expand the **Measures** folder in the **Data** list and complete the following steps:
      - Double-click the **SALES** measure in the **Data** list to add it to the expression.
      - Double-click the **–** operator in the **Operators** list.
      - Double-click the **COGS** measure in the **Data** list.
      - Double-click the **+** operator in the **Operators** list.
      - Double-click the **EXPENSE** measure in the **Data** list.
      - In the **Expression** field, highlight the part of the expression that reads: `@Measure(MDSAMPLE.COGS)+@Measure(MDSAMPLE.EXPENSE)` and double-click the **(..)** operator from the **Operators** list to enclose the selected part of the expression in parenthesis.

   Figure 7 on page 15 shows the profit expression that you can create in the SQL Expression Builder window.


SQL Expression Builder
Double click on the items that you want to add to the expression.
Name

PROFIT

```
@Measure(MDSAMPLE.SALES)-(@Measure(MDSAMPLE.COGS)+@Measure(MDSAMPLE.EXP
ENSE))
```

Figure 7. Complete profit expression in the SQL Expression Builder window

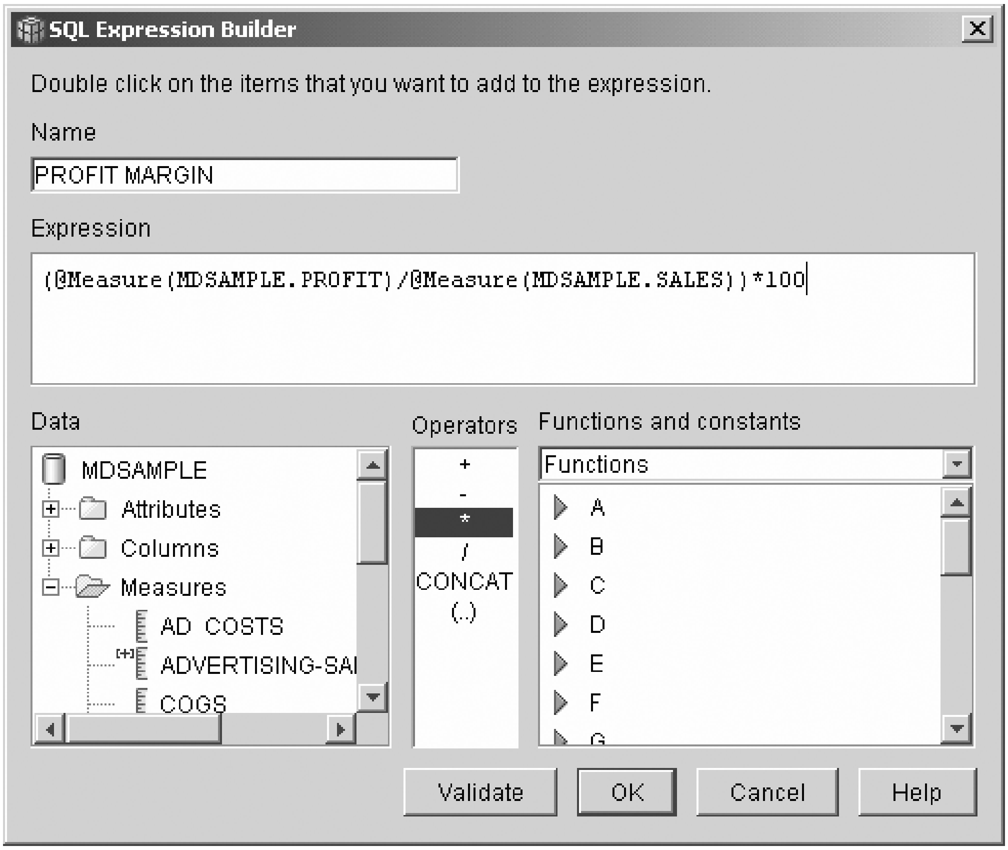
d. Click **OK** to create the Profit measure and close the SQL Expression Builder window.

e. On the Aggregations page, click the aggregation for the Profit measure, and select the SUM function. The Profit measure is complete.

3. Create the Profit Margin measure:
   a. On the Measures page, click **Create Calculated Measure**. The SQL Expression Builder window opens.
   b. In the SQL Expression Builder window, type `PROFIT MARGIN` in the **Name** field.
   c. To create the Profit Margin expression, expand the **Measures** folder in the **Data** list and complete the following steps:
      - Double-click the **PROFIT** measure in the **Data** list to add it to the expression.
      - Double-click the / operator from the **Operators** list.
      - Double-click the **SALES** measure in the **Data** list.

- Enclose the entire expression in parentheses by typing in the **Expression** field.
- Position the cursor at the end of the expression and double-click the * operator from the **Operators** list.
- Type `100` at the end of the expression in the **Expression** field.

Figure 8 shows the profit margin expression that you can create in the SQL Expression Builder window.



Name: PROFIT MARGIN

Expression:

```
(@Measure(MDSAMPLE.PROFIT)/@Measure(MDSAMPLE.SALES))*100
```

Figure 8. Complete profit margin expression in the SQL Expression Builder window

d. Click **OK** to create the Profit Margin measure and close the SQL Expression Builder window.

   On the Aggregations page, the OLAP Center sets the aggregation function to NONE by default for composite measures, so you do not need to change the aggregation function.

4. Click **OK** to close the Facts Properties window and save the two new measures that you added to the facts object.

After the DBA creates these two measures, additional analysis can be completed with respect to these important measures.

Chapter 4. Counting the number of Internet orders

A retail company expanded its business by adding Internet sales a few years ago. Now the company wants to analyze the impact of the Internet sales. One of the first calculations that the company needs is the number of orders completed over the internet.

Details of the scenario

The company's database has a fact table for Internet orders with ORDER_ID, PRODUCT_ID, QUANTITY, and TIME_ID columns. The PRODUCT_ID column includes each product sold in a corresponding order, and the QUANTITY column stores the quantity of the product purchased in the order. Orders with more than one product have as many row entries as the number of unique products sold in the order. For example, Table 6 shows three orders, where Order 1 included three Product As, one Product O and one Product G.

Table 6. Partial fact table contents

ORDER_ID	PRODUCT_ID	QUANTITY
1	A	3
1	O	1
1	G	1
2	L	1
2	Q	2
3	P	5

The DBA can create an Order Count measure that counts each unique entry in the ORDER_ID column. The Order Count measure is defined using the DISTINCT keyword in the SQL expression, and the COUNT function for the aggregation across all of the dimensions. The measure's SQL expression will create a list of distinct orders, which are counted during the aggregation. Because the measure does not involve any summing, it is called a non-additive measure.

Non-additive measures are also useful when you have character data or other data that you want to count. For example, you might use non-additive measures to count the number of postal codes that you shipped products to.

In this example, the DBA decided to define an Order ID measure that maps directly to the ORDER_ID column. However, you can choose to use the

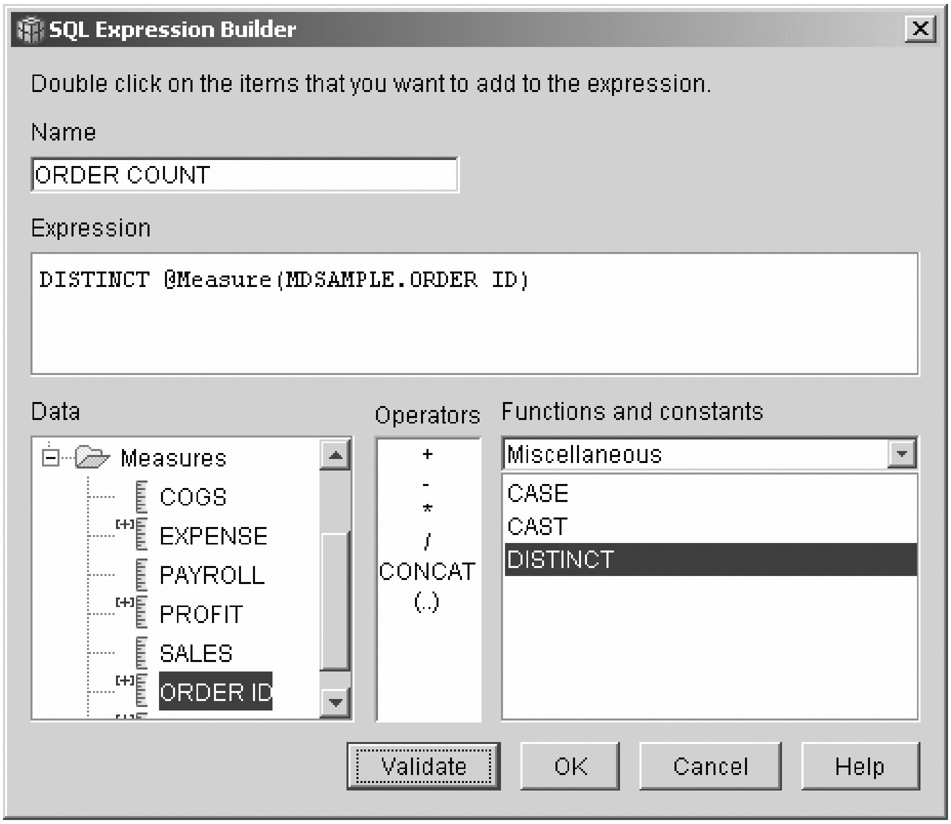
ORDER_ID column in the same way. The default aggregation is different based on if a column or a measure is used in the SQL expression, but in either case, you need to change the default aggregation to the COUNT function, as described in “Steps to create the measure”.

Steps to create the measure

The following steps explain how you could use the OLAP Center Facts Properties window to create the Order Count measure in an existing facts object:

1. To open the Facts Properties window, right-click the facts object in the OLAP Center object tree, and click **Edit Measures**. The Facts Properties window opens.
2. On the Measures page, click the **Create Calculated Measure** push button. The SQL Expression Builder window opens.
3. In the SQL Expression Builder window, type `ORDER COUNT` in the **Name** field.
4. To create the order count expression, expand the **Measures** folder in the **Data** list and complete the following steps:
   - In the **Functions and constants** filed, select **Miscellaneous**. In the list of miscellaneous functions and constants, double-click the **DISTINCT** keyword.
   - Double-click the **ORDER ID** measure in the **Data** list.

   Figure 9 on page 21 shows the order count expression that you can create in the SQL Expression Builder window.


SQL Expression Builder

Double click on the items that you want to add to the expression.

Name: ORDER COUNT

Expression:

```
DISTINCT @Measure(MDSAMPLE.ORDER ID)
```

Data: Measures — COGS, EXPENSE, PAYROLL, PROFIT, SALES, ORDER ID

Operators: + - * / CONCAT (..)

Functions and constants: Miscellaneous — CASE, CAST, DISTINCT

Figure 9. Complete order count expression in the SQL Expression Builder window

5. Click **OK** to close the SQL Expression Builder window.
6. On the Aggregations page, click the aggregation for the Order Count measure, and select the **COUNT** function.
7. Click **OK** to save the changes to the facts object and to close the Facts Properties window.

You now have the a measure that counts the number of distinct Order ID row values. You can use this measure in conjunction with other measures to further analyze your data.

Chapter 5. Ranking sales figures

An office supply store chain has expanded rapidly over the last several years. The business executives are considering closing some of the lowest performing stores to cut costs and increase profits. The sales history for a store is an important factor in deciding to close a store. Analysts need to be able to rank the sales figures and drill-down across the dimensions to complete their analysis.

Details of the scenario

The office supply store's database has a fact table with a Sales column in addition to other columns. The database also has several dimension tables. The DBA can create a Sales Rank measure that uses the RANK function, which is an OLAP function provided by DB2 Universal Database (DB2 UDB).

DB2 Cube Views supports the following OLAP functions that are provided by DB2 UDB:

RANK

Orders the rows and assigns a ranking to each row. The rank is defined as 1 plus the number of preceding rows that are distinct with respect to the ordering. If the relative order of two or more rows cannot be determined because they have duplicate row values, the same rank number is assigned. The ranking results might have gaps in the numbers if there are duplicate row values. Table 7 on page 24 shows an example of what the ranking results are from the RANK function for a set of sample row values.

The typical syntax for the RANK function is:

```
RANK ( ) OVER (ORDER BY sort-key-expression expression-order)
```

where *sort-key-expression* is the set of data to be ranked, and *expression-order* is a keyword, **ASC** or **DESC**, that orders the values of the sort-key-expression in ascending or descending order. DB2 Cube Views requires that the *sort-key-expression* be an existing measure, not a column or attribute. Also, DB2 Cube Views does not support the PARTITION BY clause that is provided by DB2 UDB with this function. More information about the RANK function is available in the DB2 UDB Information Center.

DENSERANK

Orders the rows and assigns a ranking to each row. The rank of the row is defined as 1 plus the number of rows that strictly precede the row. Therefore, the ranking results will be sequential and without gaps in the rank numbering. Table 7 shows an example of what the ranking results are from the DENSERANK function for a set of sample row values.

The typical syntax for the DENSERANK function is:

```
DENSERANK ( ) OVER (ORDER BY sort-key-expression expression-order)
```

where *sort-key-expression* is the set of data to be ranked, and *expression-order* is a keyword, **ASC** or **DESC**, that orders the values of the sort-key-expression in ascending or descending order. DB2 Cube Views requires that the *sort-key-expression* be an existing measure, not a column or attribute. Also, DB2 Cube Views does not support the PARTITION BY clause that is provided by DB2 UDB with this function. More information about the DENSERANK function is available in the DB2 UDB Information Center.

ROWNUMBER

Computes the sequential row number of the row by the ordering, starting with 1 for the first row. If the ORDER BY clause is not specified, the row numbers are assigned to the rows in arbitrary order.

The typical syntax for the ROWNUMBER function is:

```
ROWNUMBER ( ) OVER ([ORDER BY sort-key-expression expression-order])
```

where *sort-key-expression* is the set of data to be ranked, and *expression-order* is a keyword, **ASC** or **DESC**, that orders the values of the sort-key-expression in ascending or descending order. DB2 Cube Views requires that an existing measure, not a column or attribute, be used as the data source for this function. Also, DB2 Cube Views does not support the PARTITION BY clause that is provided by DB2 UDB with this function. More information about the ROWNUMBER function is available in the DB2 UDB Information Center.

These OLAP functions are not listed in the SQL Expression Builder Functions and constants list.

Table 7. Ranking results for a sample set of values using the RANK and DENSERANK functions

Row values	Ordering	Ranking results from the RANK function	Ranking results from the DENSERANK function
100	1	1	1

Table 7. Ranking results for a sample set of values using the RANK and DENSERANK functions (continued)

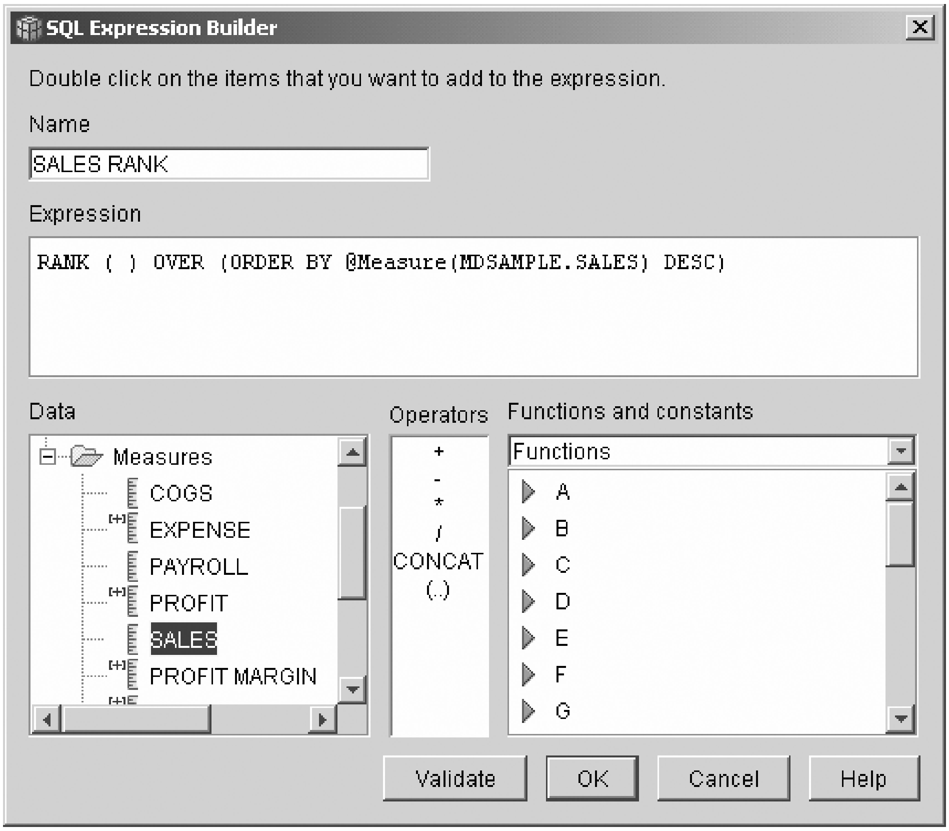
Row values	Ordering	Ranking results from the RANK function	Ranking results from the DENSERANK function
35	2	2	2
23	3	3	3
8	4	4	4
8	4	4	5
6	5	6	6

Steps to create the measure

The following steps explain how you could use the OLAP Center Facts Properties window to create the Sales Rank measure in an existing facts object:

1. To open the Facts Properties window, right-click the facts object in the OLAP Center object tree, and click **Edit Measures**. The Facts Properties window opens.
2. On the Measures page, click **Create Calculated Measure** to create the Sakes Rank measure. The SQL Expression Builder window opens.
3. In the SQL Expression Builder window, type `SALES RANK` in the **Name** field.
4. To create the Sales Rank expression, complete the following steps:
   - Type the following function syntax in the Expression field: `RANK ( ) OVER (ORDER BY measure DESC)`.
   - Expand the **Measures** folder in the **Data** list.
   - Highlight the word `measure` in the function syntax in the **Expression** field and double-click the **SALES** measure to add the SALES measure to the expression.

   The final expression is shown in Figure 10 on page 26.


SQL Expression Builder

Double click on the items that you want to add to the expression.

Name: SALES RANK

Expression:

```
RANK ( ) OVER (ORDER BY @Measure(MDSAMPLE.SALES) DESC)
```

Figure 10. Complete Sales Rank expression in the SQL Expression Builder window

5. Click **Validate** to ensure that the expression is valid. Click **OK** to close the SQL Expression Builder window.

   You do not need to change the default aggregation, None, on the Aggregations page. The None option is the default for the Sales Rank measure because the data source is numeric and refers only to existing measures.

Using the RANK function in the measure's expression to order the Sales column in descending order, the analysts can drill-down across the other dimensions to determine the store with the worst sales history over the last year, for a particular product line, or with respect to any other dimensional data stored in the database.

Chapter 6. Using time data stored in the fact table to create a Time dimension

A retail business, XYZ Retail, is modeling their sales transaction data in DB2 Cube Views so that they can analyze their data more effectively. However, because of the transactional nature of the data, the only time information that is available is a date that is associated with each transaction. Time information, that is modeled in a time dimension, is needed to add context to many common calculations, such as analyzing sales trends by quarter, and calculating the average inventory value for each week.

Many DBAs avoid storing time data as a date or timestamp for a transaction because if there are no transactions on one day, then there are holes in the data, which can create problems aggregating and displaying the data accurately. Usually, modeling time data in a time table is a better choice. However, the DBA for XYZ Retail is confident that there will be at least one transaction every day and decides to keep the current structure for the data.

Details of the scenario

XYZ Retail has a fact table with measurable data about each transaction including Sales, Costs, Quantity Sold, and Date. Additionally, the database contains a Region dimension table and a Product dimension table. The problem is that time data is included in the fact table rather than being stored in a separate dimension table. The DBA must create a dimension object based on the date data in the facts object.

Creating a time dimension based on a single column of date data in the fact table has two unique requirements:

- Because all dimension objects in a valid cube model must be joined to the facts object, and the time dimension object and the facts object are based on the same fact table, the time dimension object must be joined to the facts object by using a self-join to join the fact table to itself.
- The DBA must build calculated attributes that aggregate the date data into meaningful levels such as Week, Month, Quarter, and Year.

A self-join is a type of join that joins a table to itself, in this case the table is the fact table. The self join should join one or more columns that together can uniquely identify any row in the fact table. The primary key is the best choice. However, if a primary key is not defined, a good primary key candidate is the set of columns that are used to join the fact table with the dimension tables.

To optimize the cube model, a primary key must be defined. The join cardinality must be 1:1, and the join type must be inner.

Figure 11 shows how a facts object, a dimension based on the fact table, and a facts-to-dimension join can map to the same fact table.

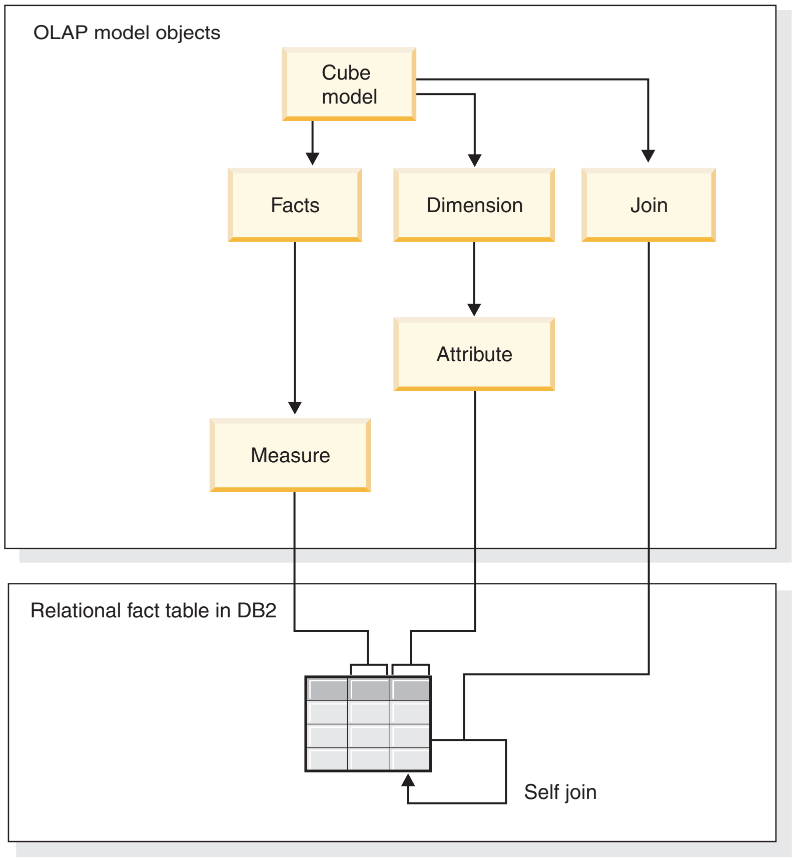


Figure 11. How a self-join joins a table to itself

Steps to create the attributes and dimension

The following steps explain how you could use the OLAP Center Dimension wizard to create the Time dimension and calculated attributes based on the fact table:

1. To open the Dimension wizard, right-click the cube model in the OLAP Center object tree, and click **Create Dimension**. The Dimension wizard opens.
2. On the Name page, type `Time` in the **Name** field. You can optionally change the business name and type a comment. Click **Next**.

3. Select the cube model's fact table. Click **Next**. You do not need to specify dimensional joins because you have only one table in your dimension. On the Dimension Joins page, click **Next**.
4.
5. On the Dimension Attributes page, select the **Timestamp** column.
6. Optional: Create the additional calculated attributes that aggregate the timestamp data into larger chunks such Month, Quarter, and Year. To create calculated attributes, click the **Create Calculated Attribute** push button to open the SQL Expression Builder and define the expression for each attribute that calculates the source timestamp column into months, quarters, and years. After defining each calculated attribute, click the **Validate** push button to ensure that the expression is valid, and then click **OK** to close the SQL Expression Builder and return to the Dimension wizard. Click **Next** after you select and create all of the attributes that you want.
7. On the Dimension Type page, select **Time**. Click **Next**.
8. On the Fact-Dimension Join page, click **Create Join**. In the Join wizard that opens, create the self join. Type a name and click **Next**. Select the column or set of columns that uniquely define any row in the fact table, such as the primary key, for both the left and right attributes. Select one pair at a time and click **Add** to add the attribute pair to the join. Select the inner join type and 1:1 cardinality. After adding the necessary attribute pairs, click **Finish**. The Join wizard closes.
9. On the Fact-Dimension Join page, click **Finish**.

With the Time dimension defined in the cube model, XYZ Retail can add a new level of meaning to its data analysis. They can now perform time-related analyses including inventory.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

Trademarks