Table of Contents

Introduction 7

Chapter 1: Metric Designer 9
Projects 9
The Project Page 10
Icons 11

Chapter 2: Creating Extracts 13
Security 13
Create a Project 14
Creating Metrics Extracts 15
  Name the Metrics Extract 16
  Specifying an Import Source for the Extract 17
  Create a Metrics Extract for a Cube-based Framework Manager Package 17
  Create a Metrics Extract for a Relational Framework Manager Package 23
  Create a Metrics Extract for an .IQD File 29
Create a Metric Type 35
Filter Categories in a Metrics Extract 38
Combining Metrics from Different Sources on One Scorecard 39
Create an Objects Extract 40
Validate Referenced Objects 42
Check IBM Cognos Connection References 43

Chapter 3: Transferring Extracts and Data to Metric Studio 45
Transfer Extracts and Data to Metric Studio 45
View Metrics in Metric Studio 46

Chapter 4: Managing Projects 47
  Change a Metric Package Reference 47
  Modify an Extract Definition 47
  Copy a Project 47
  Move a Project 48
  Rename a Project 48
  Delete a Project 49
  Upgrade a Project 49

Chapter 5: Tutorial on Creating a Metrics Extract from a PowerCube 51
  Before You Begin 51
  Create a Project 51
  Select an Import Source for Your Extract 52
  Map the Scorecards 52
  Set Time and Currency Mappings 53
  Select Dimensions to Display as Reports 54
  Select a Time Period 54
  Create a Metric Type 54

Licensed Materials – Property of IBM
© Copyright IBM Corp. 2004, 2010.
Map the Metrics 55
Select the Metric Values to Include on the Scorecards 55
Execute the Extract 56
Transfer Data from the Staging Area to the Metric Store 57
View Your Scorecards 57

Chapter 6: Tutorial on Creating a Scorecard Hierarchy Using Multiple Dimensions 59
Before You Begin 59
Create the Project, Metrics Extract, and Package Import Source 59
Map the Scorecards 60
Set Time and Currency Mappings 63
Select Dimensions to Display as Reports 63
Select a Time Period 63
Create a Metric Type 63
Map the Revenue Metric 64
Execute the Extract 64
Transfer Data from the Staging Area to the Metric Store 65
View Your Scorecards 65
View Your Metrics 65
Customize Metric Names 66

Appendix A: Troubleshooting 69
CCLAssert Message Encountered When Running an Extract Against an SAP Data Source 69
Report From OLAP Data Source Is Not Displayed and Error Processing Template Is Encoun-
tered 69
No Rollups Are Generated for Some Calculated Measures in an SAP Info Query Data Source 69
Adding Multiple IQD Files to an Import Source 69
Previewed Scorecard Hierarchy Shows Blanks 70

Appendix B: Using the Expression Editor 71
Searching for Values May Return Unexpected Results 71
Calculation Components 71
Operators 72
Summaries 79
Member Summaries 91
Constants 94
Constructs 96
Business Date/Time Functions 97
Block Functions 101
Macro Functions 102
Common Functions 112
Dimensional Functions 119
DB2 142
Informix 158
MS Access 164
Neterzsa 172
Oracle 178
Red Brick 187
SQL Server 192
Teradata 200
Introduction

This document is intended for use with IBM® Cognos® Metric Designer. Metric Designer is a Microsoft® Windows® program for mapping data and transferring information from data sources into an IBM Cognos Metric Studio metric store. It supports relational data sources, such as published IBM Cognos Framework Manager packages and IBM Cognos Impromptu® Query Definition files, and dimensional data sources, such as cubes.

This document includes information about using Metric Designer to create scorecards and metrics.

Audience
To use this guide, you should be familiar with
- your source data
- Metric Studio
- scorecards, metric types, and metrics

Finding information
To find IBM® Cognos® product documentation on the web, including all translated documentation, access one of the IBM Cognos Information Centers at http://publib.boulder.ibm.com/infocenter/cogic/v1r0m0/index.jsp. Updates to Release Notes are published directly to Information Centers.

You can also read PDF versions of the product release notes and installation guides directly from IBM Cognos product disks.

Samples disclaimer
The Great Outdoors Company, GO Sales, any variation of the Great Outdoors name, and Planning Sample depict fictitious business operations with sample data used to develop sample applications for IBM and IBM customers. These fictitious records include sample data for sales transactions, product distribution, finance, and human resources. Any resemblance to actual names, addresses, contact numbers, or transaction values, is coincidental. Other sample files may contain fictional data manually or machine generated, factual data compiled from academic or public sources, or data used with permission of the copyright holder, for use as sample data to develop sample applications. Product names referenced may be the trademarks of their respective owners. Unauthorized duplication is prohibited.

Accessibility features
This product does not currently support accessibility features that help users with a physical disability, such as restricted mobility or limited vision, to use this product.

Forward-looking statements
This documentation describes the current functionality of the product. References to items that are not currently available may be included. No implication of any future availability should be inferred.
Any such references are not a commitment, promise, or legal obligation to deliver any material, code, or functionality. The development, release, and timing of features or functionality remain at the sole discretion of IBM.
Chapter 1: Metric Designer

IBM® Cognos® Metric Designer is a tool for designing and building scorecards from relational or dimensional data sources. It simplifies the creation of scorecards from existing data by mapping and transferring information from relational or dimensional data sources into an IBM Cognos Metric Studio metric store.

Traditionally, users have used IBM Cognos Data Manager or custom database scripts to load their Metric Studio application. Metric Designer enables you to quickly and easily define and load your Metric Studio application using existing IBM Cognos sources including cubes, IBM Cognos Impromptu® Query Definition (.iqd) files, and IBM Cognos BI models. The Metric Studio extracts are defined in a hierarchical structure that includes connection information to a Metric Studio instance and one or more metrics extracts and objects extracts.

Before you start working with Metric Designer, ensure that

- the packages, cubes, or .iqd files are already defined
- if you are using an Oracle Metric Studio database and delivering multilingual information to Metric Studio, the NLS_LANG environment variable in the System dialog box of the Microsoft® Windows® Control Panel is set to your locale
- you know which items you want to map
- you know which filters you want to create
- if you are creating a metrics extract based on a package, you published the package in IBM Cognos Framework Manager

Projects

When you work in IBM® Cognos® Metric Designer, you work in the context of a project. A project is a set of design extracts which describe the data movement from a source to a metric store.

A Metric Designer project includes a reference to the metric store, design extract folders, and an import source folder that lists the published packages and the IBM Cognos Connection data sources used by the extracts.

You can create multiple extracts in one project. For example, you may want to transfer data from both a published cube-based package and a published relational IBM Cognos Framework Manager package into IBM Cognos Metric Studio. You can define both extracts in the same project.

If you use Framework Manager, note that both Metric Designer and Framework Manager projects use the same file extensions. Keep the two types of projects separate. You can look at the code in the model.xml file to determine which product the model was designed in.

These are the contents of a project folder.
<table>
<thead>
<tr>
<th>File name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project name.cpf</td>
<td>The Metric Designer project file, which references the .xsd and .xml files used to define a project.</td>
</tr>
<tr>
<td>model.xml</td>
<td>The actual model data created by Metric Designer.</td>
</tr>
<tr>
<td>Preferences.xml</td>
<td>The preferences for Metric Designer projects.</td>
</tr>
<tr>
<td>customdata.xml</td>
<td>The stored diagram information, such as the diagram layout, notation, font, and color.</td>
</tr>
<tr>
<td>IDLog.xml</td>
<td>This file tracks objects for models that use branching and merging.</td>
</tr>
<tr>
<td>session-log.xml</td>
<td>A list of unsaved transactions in the model. When the project is saved, this list is deleted. View contents of this file using View Transaction History. When Metric Designer is started, the existing session.log.xml file is renamed to session-log-backup.xml.</td>
</tr>
<tr>
<td>session-log-backup.xml</td>
<td>The session-log.xml file from the previous session. Using this file, a modeler can run a script to restore the unsaved model transactions in the event of an unexpected interruption in the current session. The file is deleted each time Metric Designer is started. Ensure you make a copy of this file before exiting the current Metric Designer session if you want to keep a copy.</td>
</tr>
<tr>
<td>log.xml</td>
<td>A list of all the modifications made to the model.</td>
</tr>
<tr>
<td>archive-log.xml</td>
<td>This file contains the portion of the main log file that was archived.</td>
</tr>
</tbody>
</table>

The Project Page

After you create or open a project, the project page appears. The project page is the environment in which you create and publish extracts. This page contains several panes and views that you can use to view and modify the objects in a project.

Project Explorer

The Project Explorer shows the objects in a project in a hierarchical view.

To view the corresponding object in the Object Explorer view, you must have the Object Explorer open.
Object Diagram
The Object Diagram shows the relationships between data sources and IBM® Cognos® Metric Studio.

In the Object Diagram, you can also organize objects.
Tip: From the Diagram menu, click Auto Layout.

Object Explorer
The Object Explorer shows the contents of a project. Objects can be arranged by name, class, or description. If you have many objects in a project, it may be easier to locate them in the Object Explorer.

Properties Pane
The Properties pane shows the properties of the objects that you last selected in either the Object Explorer or the Object Diagram. Object properties are set during import, and some property values can be modified during extract creation. You can use the Properties pane to add, modify, or delete the properties of objects.

You can modify the properties for multiple objects at the same time. If you select more than one object, IBM Cognos Metric Designer shows only the properties that are common to all the objects.

With multiple objects selected, you can
● sort property values by double-clicking the property heading
  An arrow indicates the direction in which values are sorted. You can toggle between ascending and descending order.
● filter property values by clicking the arrow to the right of the property heading
  Tip: You can either click a value, or click Custom to define the criteria for the rows you want to view.
● resize the width of the rows and columns by right-clicking the object name in the property pane

Icons
IBM® Cognos® Metric Designer uses the following icons to represent objects and states.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Represents the project</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Represents the root namespace or any other namespace in the project</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Represents a folder that contains one or more data sources</td>
</tr>
<tr>
<td>Icon</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td><img src="image1.png" alt="Icon" /></td>
<td>Represents a data source</td>
</tr>
<tr>
<td><img src="image2.png" alt="Icon" /></td>
<td>Represents a package</td>
</tr>
<tr>
<td><img src="image3.png" alt="Icon" /></td>
<td>Represents a query subject within a model</td>
</tr>
<tr>
<td><img src="image4.png" alt="Icon" /></td>
<td>Represents a query item</td>
</tr>
<tr>
<td><img src="image5.png" alt="Icon" /></td>
<td>Represents a query item that is located under a shortcut query subject</td>
</tr>
<tr>
<td><img src="image6.png" alt="Icon" /></td>
<td>Represents a measure</td>
</tr>
<tr>
<td><img src="image7.png" alt="Icon" /></td>
<td>Represents a calculation</td>
</tr>
<tr>
<td><img src="image8.png" alt="Icon" /></td>
<td>Represents a filter</td>
</tr>
<tr>
<td><img src="image9.png" alt="Icon" /></td>
<td>Indicates that the object is a dimension (this icon overlays other icons)</td>
</tr>
<tr>
<td><img src="image10.png" alt="Icon" /></td>
<td>Indicates that the object is a shortcut (this icon overlays other icons)</td>
</tr>
<tr>
<td><img src="image11.png" alt="Icon" /></td>
<td>Represents the IBM Cognos Metric Studio connection folder</td>
</tr>
<tr>
<td><img src="image12.png" alt="Icon" /></td>
<td>Represents a new connection</td>
</tr>
<tr>
<td><img src="image13.png" alt="Icon" /></td>
<td>Represents an extracts folder</td>
</tr>
<tr>
<td><img src="image14.png" alt="Icon" /></td>
<td>Represents a metrics extract</td>
</tr>
<tr>
<td><img src="image15.png" alt="Icon" /></td>
<td>Represents an objects extract or a text file extract</td>
</tr>
<tr>
<td><img src="image16.png" alt="Icon" /></td>
<td>Represents a cube data source</td>
</tr>
<tr>
<td><img src="image17.png" alt="Icon" /></td>
<td>Represents an .iqd data source</td>
</tr>
</tbody>
</table>

Chapter 1: Metric Designer
Chapter 2: Creating Extracts

You create metrics extracts and objects extracts for loading information into IBM® Cognos® Metric Studio.

A metrics extract is a set of mappings between an existing IBM Cognos import source and a Metric Studio object or value. For example, a metrics extract can map a cube measure named Revenue to a Metric Studio metric named Revenue Actual Value.

An objects extract defines the metadata for a Metric Studio object, such as a user-defined column, a scorecard, or a data source.

To create an IBM Cognos Metric Designer extract, do the following:

- Create a new project (p. 14) or open an existing project.
- Create an import source (p. 16).
- Do one of the following:
  - Create a metrics extract for a cube-based IBM Cognos Framework Manager package (p. 17).
  - Create a metrics extract for a relational Framework Manager package (p. 23).
  - Create a metrics extract for an .iqd file (p. 29).
  - Create an objects extract (p. 40).

You will be prompted to validate your extract after you finish creating it, but you can verify that it contains no errors at any other time by right-clicking it in the Project Explorer pane and clicking Validate.

You can now work with your metrics in Metric Studio. For more information, see "Transferring Extracts and Data to Metric Studio" (p. 45).

Security

You need specific permissions to perform some IBM® Cognos® Metric Designer tasks.

<table>
<thead>
<tr>
<th>Task</th>
<th>Permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create or modify a metric extract</td>
<td>You must have read, write, execute and traverse permissions for the metrics package and the import source package. When creating a metric extract, you must specify a time mapping. To be able to do this, you must also have Metric Studio secured function capability or the UI will fail.</td>
</tr>
</tbody>
</table>
### Task Permissions

<table>
<thead>
<tr>
<th>Task</th>
<th>Permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create a metric type</td>
<td>You must have write access to the metric types object in the metric store.</td>
</tr>
<tr>
<td></td>
<td>You must also have IBM Cognos Metric Studio secured function capability or the UI will fail.</td>
</tr>
<tr>
<td>Create or modify an objects extract</td>
<td>You must have read, write, execute and traverse permissions for the metrics package and the import source package.</td>
</tr>
<tr>
<td>Transfer data from staging area into metric store</td>
<td>You must have read and execute permission for the task.</td>
</tr>
<tr>
<td></td>
<td>You must also have Load Data administrator permission for the metric store.</td>
</tr>
<tr>
<td>Publish an extract</td>
<td>You must have read, write, execute and traverse permissions for the metrics package and the import source package.</td>
</tr>
<tr>
<td></td>
<td>You must also have Metric Studio Administration capability.</td>
</tr>
<tr>
<td>Execute an extract</td>
<td>You must have read, write, execute and traverse permissions for the metrics package and the import source package.</td>
</tr>
<tr>
<td></td>
<td>You must also have Load Data administrator permission for the metric store.</td>
</tr>
</tbody>
</table>

## Create a Project

A project is a group of extracts. Each extract contains the metadata that is used to populate the IBM® Cognos® Metric Studio data store or to create applications.

Each project must reference a metric package to provide IBM Cognos Metric Designer with server connection information. If the metric package that you want to reference does not yet exist, you can create it from within Metric Designer.

**Tip:** If you want to create a different folder to store your metric package reference, you can cancel the wizard, right-click the Metric Package References folder, and click Create, References Folder. You can then return to the wizard by right-clicking the new folder and clicking Create, Metric Package Reference.

### Steps to Start a Project

1. If Metric Designer is running, from the File menu, click New. Otherwise, start Metric Designer and, in the Welcome page, under Projects, click Create a new project.
2. Specify a name and location for the project.

3. Click OK.

**Steps to Create a Metric Package Reference**

1. On the Create Metric Package Reference page, click a package. If the package that you want to reference is not shown, you can create it by clicking New Metric Package, and following the wizard.

   For information about creating a new metric package, see the IBM Cognos Administration and Security Guide.

2. After you select a metric package, click OK.

**Creating Metrics Extracts**

You create a metrics extract within a project to define the scorecards and metrics that you want to view in IBM® Cognos® Metric Studio. Each metrics extract consists of a set of mappings between an IBM Cognos source, such as a cube, and a Metric Studio object or value, such as the actual value of a metric named Revenue. A metrics extract creates a complete scorecard structure, including scorecards, metrics, and actual, target, tolerance, and user-defined values. For more information, see the Metric Studio User Guide.

You can create the scorecards, scorecard hierarchies, and qualifiers in either Metric Studio or IBM Cognos Metric Designer. Under some circumstances, you may want to create them in Metric Studio first, before you create the metrics extracts in Metric Designer. Examples of when you may want to do this include the following:

- The scorecard structure, the qualifiers, or both may not be present in the IBM Cognos data source.

- The IBM Cognos data source may provide only part of the scorecard tree.

  For example, the source only has data on individual products, not the product type or product line that each product belongs to.

- You want to organize your scorecards differently from the way the data is organized in your data source.

  For example, your cube has a dimension with a structure of Territory->Country->Cost Center. However, in your scorecard structure, you want to relate cost centers to cities. Therefore, you create the City scorecards in Metric Studio before you create a metrics extract that uses the Cost Center level from the cube.

For more information about setting up your Metric Studio environment, see the Metric Studio User Guide.

On certain pages in the wizard, you can preview your data to that point. For example, you can preview scorecard hierarchies, metrics you mapped, and time hierarchies. The results that appear depend on different factors, such as the expressions that you created and the object references that you chose to that point. Data you previously filtered out does not appear. Because you can filter...
data at various stages while you are creating a metrics extract, the data that is shown may vary from one preview page to another.

You can create a complete metrics extract by following the wizard, or you can create it in stages. After you create a metrics extract, you can edit it.

**Recommendation - Use Network Paths for File-based Data Sources**

If you have a distributed installation with several servers, we recommend that you use network paths for all file-based data sources rather than local paths. This ensures that the data sources can be accessed by the services that require them, regardless of which server requires the data.

When you create a connection to a file-based data source, such as a cube, you enter a path and file name. You can use a local path, such as C:\cubes\Great Outdoors Company.mdc, or a network path, such as \servername\cubes\Great Outdoors Company.mdc, to point to the file.

In a distributed installation, where report servers are running on different computers, using a local path requires that the file and path be valid on each computer where a report server is running. Alternatively, if you use a network path to point to a file, each report server points to the same file on the network without having the file available locally. To ensure that the file is always available, we recommend that you store it in a shared directory that can be accessed on your network.

If you installed IBM Cognos BI components on UNIX® servers, we recommend that you also locate the file-based data source on a UNIX server. You should then use a UNIX path, such as /servername/cubes/Great Outdoors Company.mdc to access the file.

If you installed IBM Cognos BI components on Microsoft® Windows® and UNIX computers, you can use both paths to ensure that the report server running the request can access the cube from either Windows or UNIX. In this situation, you would also need to have the cube available to both file systems. This can be accomplished by duplicating the cube in two locations. For example, you can save the cube file to a location available to Windows report servers, and then make a copy and save it to a location that is available to UNIX report servers. In the cube connection string information, you can provide both paths. To ensure that the data returned is the same, the cube file on both the Windows and UNIX environments must be the same.

If you installed all components on a single computer, you can use local paths, but you must ensure that the services requesting the data have the appropriate access to the data files on the computer. For Windows distributed installations, we recommend that you use UNC paths to shared directories for any file based data sources, such as cubes or XML files.

**Name the Metrics Extract**

After you create a metric package reference, the Create Extract - Provide Name page appears.

If the Create Metrics Extract wizard is not running, in the Project Explorer pane, right-click the extract folder in which you want to create a metrics extract, and then click Create, Metrics Extract.

**Steps**

1. In the Create Extract - Provide Name page, type a name and description for the extract.
2. Click Next.
Specifying an Import Source for the Extract

Each extract must be associated with one of the following import sources:

- dimensional IBM Cognos BI packages that are based on cubes
- relational IBM Cognos Framework Manager packages that were published
- IBM Cognos Impromptu Query Definition (.iqd) files

Not all dimensional packages, Framework Manager packages, or .iqd files make good data sources. You must be able to map the structure from one or more import sources to your scorecard structure in Metric Studio. For cubes, this means that you must have at least one dimension that represents the scorecards. For .iqd files and relational Framework Manager packages, at least one source table (.iqd) or query subject (Framework Manager) should have a structure that represents the scorecards.

IBM Cognos Metric Designer does not support Framework Manager import sources with missing connection or credential information. Metric Designer also does not support Framework Manager import sources that contain multiple import sources.

Create a Metrics Extract for a Cube-based Framework Manager Package

If you want to use a cube-based Framework Manager package as an import source, continue with the following instructions. To use a relational Framework Manager package as an import source, see "Create a Metrics Extract for a Relational Framework Manager Package" (p. 23). To use an .iqd file as an import source, see "Create a Metrics Extract for an .IQD File" (p. 29).

Steps to Specify an Import Source

1. In the Create Extract - Select Import Source page, click Create Package Import Source.
   
   If the Wizard is not running, in the Project Explorer, click Import Sources and, from the Actions menu, click Create Package Import Source.

2. In the Create Import Source - Name page, type a name and description for the new import source, and click Next.

3. In the Create Import Source - Select Package page, select a cube package or click Create IBM Cognos PowerCube Package and select a package.
   
   If the data source does not yet exist, click New Data Source, and follow the wizard to create the data source. For more information about creating the data source, see the IBM Cognos Administration and Security Guide.

4. In the Create Extract - Select Import Source page, select the import source that you want to use and click Next.

   The Create Extract - Scorecard Mapping page appears.

Steps to Map the Scorecard

1. In the Create Extract - Scorecard Mapping page, if you want to change the parent scorecard, do one of the following:
**Chapter 2: Creating Extracts**

1. Click the scorecard icon next to the **Parent scorecard** box, and then click the scorecard that you want.

2. Click the arrow next to the scorecard icon, and then click **Metric Package Pick List**.

3. Click the arrow next to the scorecard icon, click **Expression Editor**, and then create an expression to identify the parent scorecard.

2. **In the Available objects** pane, click the items that you want and drag them to the **Levels** pane. The order in which you drag the items sets the order of the levels for the scorecard.

   **Tip:** To create a new level, under **New scorecard levels**, click **Create**.

   By default, Metric Designer uses the value of the item for both the scorecard name and the scorecard ID.

3. If you want to use a different item for either the name or ID, drag the item that you want into the value field for that attribute.

   **Tip:** By clicking the ellipse (...) next to an attribute, you can use the expression editor to modify attributes. For example, modify the name of a scorecard for the city of London as follows:

   - `[city] + '_SC' creates London_SC
   - `[city] + 'Sales Office' creates a scorecard named London Sales Office
   - `left ([city],3)` creates a scorecard named Lon

4. If you want to combine metrics from different data sources on the same scorecard, do the following:

   - In the **Level attributes** pane, click the ellipse (...) next to **Identification Code** and **Name** and change the information to match the scorecard where you want this metric to appear (p. 39).

   - Click the ellipse (...) next to **Description**, **Language code**, and **Owner** and enter the information that you want.

   - Next to **Append to scorecard ID**, in the list, click **Yes**.

     If you click **No**, and Metric Designer finds duplicate IDs, either create an expression to make the ID unique or choose a different query item that will result in a unique ID.

   **Tip:** To check your scorecard hierarchy for duplicate IDs, click **Preview scorecard tree**.

5. If you want to filter to limit the items, under the **Level filters** pane, do the following:

   - Click **Create**.

   - In the expression editor page, in the **Available components** pane, drag the items that you want to the **Expression definition** pane.

     If you create two filters, Metric Designer considers them joined by the **AND** operator.

   - Click the **Functions** tab, click the functions that you want, and drag them to the **Expression definition** pane.
Click OK.

**Note:** In the expression editor, if you want to create a filter that uses a Member Unique Name (MUN) query item, you must use the `roleValue` expression for the member you are filtering against. For example, if you are creating scorecards for products using the Great Outdoor Company sample, and you want to filter out the Golf Equipment product category using the Member Unique Name query item, you must use the `roleValue` function for the second part of the expression as shown in the following expression:

```
[great_outdoors].[Products].[Products].[Products].[Member Unique Name] <> roleValue ( '_memberUniqueName', [great_outdoors].[Products].[Products].[Product line]->: [PC].[Products (Root)].[5~236] )
```

The `roleValue` function is located in the Functions tab of the expression editor under Common Functions, R-Z, Advanced.

6. Repeat steps 3 to 5 for each level you that want to use.

7. We recommend that you click **Preview Scorecard Tree** to view the scorecards that you chose, any duplicate IDs, and optionally the scorecard IDs.

8. We recommend that you click **Preview Level Data** to view the data that will be transferred to the Metric Studio metric store if you do not filter it out before you publish or execute the extract.

   **Tip:** If you want to see a different number of rows of data, change the number, and click **Test sample**.

9. Click **Close** and then click **Next** or **Finish**.

   If you click **Finish**, you exit the wizard. You can continue to create the metrics extract later.

   **Tip:** In the main page, in the **Project Explorer** pane, double-click the metrics extract that you want to continue to create.

**Steps to Map Time and Currency**

1. If the **Time and Currency Mappings** page appears, click the business calendar level and the business calendar member that you want.

   When entering date items, ensure that the items are in the default date format for the Metric Studio database platform.

2. Under **Data Mapping**, next to the business calendar member that you chose, click the ellipsis (...).

3. Click the time period that contains the data to which you want to map the business calendar level, and click **OK**.

   Metric Designer automatically populates all the levels under it and the **Time and Currency Mappings** page reappears.

4. Repeat this process for each business level to which you want to map data.
5. If you want, select the **Automatically update the time mappings when new mappings are added to the cube** box.

If you select this option and choose to publish your extract to IBM Cognos Connection, when IBM Cognos Connection runs the job, Metric Designer updates the metric package with all the new data that was added to the cube, based on the last date in the metric package.

6. Click **Next**.

In the list of currencies that are defined in the cube, click the currency that you want.

7. Under **Data Mapping**, click the country code that you want to associate with the currency that you chose.

   You can make multiple selections.

   If the cube does not contain any country codes, you can choose only one currency. The currency that you choose becomes the default.

8. Click **Next**.

   A list of dimensions and the reports available in the metric package appears.

9. Select the reports that you want to make available in Metric Studio.

   You can change the name of the report.

10. Click **Finish**.

   The **Create Extract - Time Periods Filtering** page appears.

### Steps to Select a Time Period

1. If the **Create Extract - Time Periods Filtering** page appears, click the time period that you want from the following options:
   - **Use all time periods** provides Metric Studio with values for all the time periods mapped between the import source for the extract and the Metric Studio metric store.
   - **Use the current time period** provides values to Metric Studio only for the current time period as defined in Metric Studio.
   - **Use the last completed period** provides values to Metric Studio for the latest period completed, which is the current period minus one. For example, if the current period in Metric Studio is set to month and the date is August 8, Metric Designer provides values for July.
   - **Use the selected time periods** means that you must select the granularity and the time periods for the values provided to Metric Studio.

2. Click **Next** or **Finish**.

   If you click **Finish**, you exit the wizard. You can continue to create the metrics extract later.
Tip: In the main page, in the Projects explorer pane, double-click the metrics extract that you want to continue to create.

Steps to Map Metrics

1. If the Create Extract - Metrics Mapping page appears, in the Available objects pane do the following:
   - Click the measure that you want to use to create the metric for the actual value and drag it to the Metric mappings pane.
     
     You must create at least one metric type before you map a measure to a metric. For more information, see "Create a Metric Type" (p. 35).
   - Click one measure each for the Tolerance and Target attributes shown in the Metric attributes pane and drag the measure to the appropriate row under the Value column.

   If you typed a constant value under Metrics Mapping for actual, tolerance, target, or user-defined columns, Metric Designer considers each value as a constant and does not sum it.

2. For the Qualifier identification and Qualifier name, either type a value or click the ellipsis (...) and do the following:
   - In the Expression Editor page, in the Available Components pane, drag the items that you want to the Expression Definition pane.
   - Click the Functions tab, click the functions that you want, and drag them to the Expression Definition pane.
   - Click OK.

   For qualifiers whose mapped query items/members are based on the same source dimension as that specified in at least one of the scorecards levels in the scorecards mapping, the source level of the query item/member in the qualifier must be at the same level as or lower than the lowest source level in the scorecards mapping.

   The number of rows shown depends on the metric type that you selected. You must already have created the metric type in Metric Studio. For more information, see the Metric Studio User Guide.

3. For the Rollup Aggregate function, click a rule.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatic</td>
<td>The application automatically chooses to apply either Summarize, Calculated, or None.</td>
</tr>
<tr>
<td>Total</td>
<td>Sum the metric items.</td>
</tr>
<tr>
<td>Count</td>
<td>Count the metric items.</td>
</tr>
<tr>
<td>Average</td>
<td>Average the metric items.</td>
</tr>
<tr>
<td>Rule</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Minimum</td>
<td>The minimum metric value.</td>
</tr>
<tr>
<td>Maximum</td>
<td>The maximum metric value.</td>
</tr>
<tr>
<td>Calculated</td>
<td>This can also be thought of as aggregate-then-calculate. That is, all the</td>
</tr>
<tr>
<td></td>
<td>terms with a data item’s expression are aggregated according to their own</td>
</tr>
<tr>
<td></td>
<td>rollup rules and the results of those aggregations are then computed</td>
</tr>
<tr>
<td></td>
<td>within the overall expression.</td>
</tr>
<tr>
<td>Summarize</td>
<td>Metric values are aggregated in the most obvious, or simplistic, manner,</td>
</tr>
<tr>
<td></td>
<td>based on either model or data type information.</td>
</tr>
<tr>
<td>Count Distinct</td>
<td>A distinct count of metric values.</td>
</tr>
</tbody>
</table>

The Rollup Aggregate function allows a report author to specify how detail values are summarized for the scorecards.

4. If you want to create a new metric, do the following:
   - Click Create.
   - In the New Metric dialog box, type a name for the metric and select a metric type.
   - You can create a new metric type for the metric by clicking Create Metric Type. For more information, see "Create a Metric Type" (p. 35).
   - Click OK.

5. If you want to select a metric to filter on, under the Metric filters pane, do the following:
   - Click Create.
   - In the Expression Editor page, in the Available components pane, drag the items that you want to the Expression definition pane.
     - If you create two filters, Metric Designer considers them joined by the AND operator.
   - Click the Functions tab, click the functions that you want, and drag them to the Expression definition pane.
   - Click OK.

6. Click Next or Finish.
   - If you click Finish, you exit the wizard. You can continue to create the metrics extract later.
Tip: In the main page, in the Projects explorer pane, double-click the metrics extract that you want to continue to create.

**Steps to Apply Filters**
1. If the Create Extract - Filter Data page appears, in the Scorecard levels pane, select the levels that you want.
2. If you want, select the Show scorecard IDs check box.
3. Click Refresh to show the data.
4. Expand the list and click the box to the left of the items that you want to filter out.
   You can filter the values for the categories within the levels either to exclude them completely or to hide them but still include their values in the totals (p. 38).
5. Click Finish.
6. Save your project.

Your extract is now complete. For information on using the extract in Metric Studio, see "Transferring Extracts and Data to Metric Studio" (p. 45).

**Create a Metrics Extract for a Relational Framework Manager Package**
If you want to use a relational Framework Manager package as an import source, continue with the following instructions. To use a cube-based package as an import source, see "Create a Metrics Extract for a Cube-based Framework Manager Package" (p. 17). To use an .iqd file as an import source, see "Create a Metrics Extract for an .IQD File" (p. 29).

**Steps to Specify an Import Source**
1. In the Create Extract - Select Import Source page, click Create Package Import Source.
   If the wizard is not running, from the Project Explorer, click Import Sources and, from the Actions menu, click Create Package Import Source.
2. In the Create Import Source - Name page, type a name and description for the new import source, and click Next.
3. In the Create Import Source - Select Package page, select a relational package and click Finish.
4. In the Create Extract - Select Import Source page, select the import source that you want to use and click Next.
   The Create Extract - Scorecard Mapping page appears.

**Steps to Map the Scorecard**
1. In the Create Extract - Scorecard Mapping page, if you want to change the parent scorecard, do one of the following:
   ● Click the scorecard icon next to the Parent Scorecard box, and then click the scorecard that you want.
Chapter 2: Creating Extracts

- Click the arrow next to the scorecard icon, and then click **Metric Package Pick List**.
- Click the arrow next to the scorecard icon, click **Expression Editor**, and then create an expression to identify the parent scorecard.

2. In the **Available objects** pane, click the items that you want and drag them to the **Levels** pane. The order in which you drag the items sets the order of the levels for the scorecard.

**Tip:** To create a new level, under **New scorecard levels**, click **Create**.

By default, Metric Designer uses the value of the item for both the scorecard name and the scorecard ID.

3. If you want to use a different item for either the name or ID, drag the item that you want into the value field for that attribute.

**Tip:** By clicking the ellipsis (...) next to an attribute, you can use the expression editor to modify attributes. For example, modify the name of a scorecard for the city of London as follows:

- 
  
  - `[city] + '_SC' creates London_SC
  - `[city] + 'Sales Office' creates a scorecard named London Sales Office
  - `left ([city],3)` creates a scorecard named Lon

4. If you want to combine metrics from different data sources on the same scorecard, do the following:

- In the **Level attributes** pane, click the ellipsis (...) next to **Identification Code** and **Name** and change the information to match the scorecard where you want this metric to appear (p. 39).
- Click the ellipsis (...) next to **Description**, **Language code**, and **Owner** and enter the information that you want.

- Next to **Append to scorecard ID**, in the list, click **Yes**.

  If you click **No**, and Metric Designer finds duplicate IDs, either create an expression to make the ID unique or choose a different query item that will result in a unique ID.

**Tip:** To check your scorecard hierarchy for duplicate IDs, click **Preview scorecard tree**.

5. If you want to filter to limit the items, under the **Level filters** pane, do the following:

- Click **Create**.

- In the **Expression Editor**, in the **Available components** pane, drag the items that you want to the **Expression definition** pane.

  If you create two filters, Metric Designer considers them joined by the **AND** operator.

- Click the **Functions** tab, click the functions that you want, and drag them to the **Expression definition** pane.

- Click **OK**.
Note: In the expression editor, if you want to create a filter that uses a Member Unique Name (MUN) query item, you must use the roleValue expression for the member you are filtering against. For example, if you are creating scorecards for products using the Great Outdoor Company sample, and you want to filter out the Golf Equipment product category using the Member Unique Name query item, you must use the roleValue function for the second part of the expression as shown in the following expression:

\[
\text{[great_outdoors]._[Products]._[Products]._[Member Unique Name]} \neq \text{roleValue ('_memberUniqueName', [great_outdoors]._[Products]._[Products].}
\text{[Product line]->:[PC]._[Products (Root)].[5~236])}
\]

The roleValue function is located in the Functions tab of the expression editor under Common Functions, R-Z, Advanced.

6. Repeat steps 3 to 5 for each level you want to use.

7. Click Preview Scorecard Tree to view the scorecards that you chose, any duplicate IDs, and optionally the scorecard IDs.

8. Click Preview Level Data to view the data that will be transferred to the Metric Studio metric store if you do not filter it out before you publish or execute the extract.

   Tip: If you want to see a different number of rows of data, change the number, and click Test sample.

9. Click Close, and then click Next or Finish.

   If you click Next, the Create Extract - Time Hierarchy Attributes page appears.

   If you click Finish, you exit the wizard. You can continue to create the metrics extract later.

   Tip: In the main page, in the Project Explorer pane, double-click the metrics extract that you want to continue to create.

Steps to Select a Time Period

1. In the Create Extract - Time Hierarchy Attributes page, in the Available objects pane, click the date that you want to use for the year and drag it to the level of granularity that you want in the Time hierarchy attributes pane.

   Metric Designer delivers metric values for each time level mapped in the Time hierarchy attributes pane. You can map one or many levels. Metric Designer delivers summarized values only for the levels mapped.

   If you use a standard Gregorian calendar to represent your fiscal calendar and your fiscal calendar does not begin on January 1, you must edit the Time expressions for Quarter and Week to reflect your fiscal calendar.

   You can use the same date for more than one level.

   Metric Studio needs time stamp expressions for the mapping attributes shown on this page.

   For Year, Quarter, and Month, Metric Designer needs the date values only for the first dates of each level. For example, for the quarters in 2005, these dates would be January 1, 2005, April 1, 2005, July 1, 2005, and October 1, 2005. If your dates include values at a lower
Chapter 2: Creating Extracts

granularity, Metric Designer automatically creates a time stamp expression to remove them. For example, if the date you used for Quarter includes values at the month level for 2005, the expression removes all values except those for January 1, 2005, April 1, 2005, July 1, 2005, and October 1, 2005. The expression appears under Value.

You can instead create your own time stamp expression.

When entering date items, ensure that the items are in the default date format for the Metric Studio database platform.

2. If you want to filter to limit the items, under the Level filters pane, do the following:
   - Click Create.
   - In the Expression Editor page, in the Available components pane, drag the items that you want to the Expression definition pane.
     If you create two filters, Metric Designer considers them joined by the AND operator.
   - Click the Functions tab, click the functions that you want, and drag them to the Expression definition pane.
   - Click OK.

3. Click Preview time period data to view the data that will be transferred to the Metric Studio metric store if you do not filter it out before you publish or execute the extract.

   If you typed a constant value under Metrics Mapping for actual, tolerance, target, or user-defined columns, Metric Designer sums the values.

   Tip: If you want to see a different number of rows of data, change the number, and click Test sample.

   Click Close, and then click Next or Finish.

   If you click Finish, you exit the wizard. You can continue to create the metrics extract later.

   Tip: In the main page, in the Project Explorer pane, double-click the metrics extract that you want to continue to create.

Steps to Map Metrics

1. If the Create Extract - Metrics Mapping page appears, in the Available objects pane, do the following:
   - Click the measure that you want to use to create the metric for the actual value and drag it to the Metric mappings pane.
     You must create at least one metric type before you map a measure to a metric. For more information, see "Create a Metric Type" (p. 35).
   - Click one measure each for the Tolerance and Target attributes shown in the Metric attributes pane and drag the measure to the appropriate row under the Value column.

   If you typed a constant value under Metrics Mapping for actual, tolerance, target, or user-defined columns, Metric Designer considers each value as a constant and does not sum it.
2. For the **Qualifier identification code** and **Qualifier name**, either type a value or click the ellipsis (...) and do the following:

   - In the **Expression Editor** page, in the **Available components** pane, drag the items that you want to the **Expression definition** pane.
   - Click the **Functions** tab, click the functions that you want, and drag them to the **Expression definition** pane.
   - Click **OK**.

   The number of rows shown depends on the metric type that you selected. You must already have created the metric type in Metric Studio. For more information, see the Metric Studio **User Guide**.

3. For the **Aggregate function** and **Rollup aggregate function**, click a rule.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatic</td>
<td>The application automatically chooses to apply either <strong>Summarize</strong>, <strong>Calculated</strong>, or <strong>None</strong>.</td>
</tr>
<tr>
<td>Total</td>
<td>Sum the metric items.</td>
</tr>
<tr>
<td>Count</td>
<td>Count the metric items.</td>
</tr>
<tr>
<td>Average</td>
<td>Average the metric items.</td>
</tr>
<tr>
<td>Minimum</td>
<td>The minimum metric value.</td>
</tr>
<tr>
<td>Maximum</td>
<td>The maximum metric value.</td>
</tr>
<tr>
<td>Calculated</td>
<td>This can also be thought of as aggregate-then-calculate. That is, all the terms with a data item’s expression are aggregated according to their own rollup rules and the results of those aggregations are then computed within the overall expression.</td>
</tr>
<tr>
<td>Summarize</td>
<td>Metric values are aggregated in the most obvious, or simplistic, manner, based on either model or data type information.</td>
</tr>
<tr>
<td>Count Distinct</td>
<td>A distinct count of metric values.</td>
</tr>
</tbody>
</table>

The **Aggregate function** describes how detail, transaction data from a relational data source, is summarized prior to the application of any other operations.

For example, a relational fact table may contain transaction records at the granularity of store (geography) and SKU (product), but a metric extract author may only be interested in looking at metric values for city, and product type scorecards. The aggregate attribute allows the author to specify what aggregation rule is to be used to perform this operation.
Note: The aggregate attribute is ignored for all OLAP data sources because all data in such data sources is already summarized.

A measure has associated with it a regularAggregate value that is specific in the Framework Manager model. The regularAggregate or Automatic value is the default unless it is explicitly overridden by the aggregate attribute of a query data item.

The Rollup Aggregate function allows the report author to specify how detail values are summarized for the scorecards. This rollup rule can be different from the rollup rule defined for the aggregate attribute.

For example, a star schema data warehouse contains sales information at a cash register level of detail. The metric author wants the metric values for store level scorecards using the sum of all cash register sales in each store. The specification of Aggregate = Total on the measure controls this summarization of data. The metric extract itself may then average these sales across all sales scorecards. The specification of Rollup Aggregate = Average controls this summarization of data in the scorecard structure.

4. If you want to create a new metric, do the following:
   - Click Create.
   - In the New Metric dialog box, type a name for the metric and select the metric type.
   - You can create a new metric type for the metric by clicking Create Metric Type. For more information, see "Create a Metric Type" (p. 35).
   - Click OK.

5. If you want to select a metric to filter on, under the Metric filters pane, do the following:
   - Click Create.
   - In the Expression Editor page, in the Available components pane, drag the items that you want to the Expression definition pane.
     - If you create two filters, Metric Designer considers them joined by the AND operator.
   - Click the Functions tab, click the functions that you want, and drag them to the Expression definition pane.
   - Click OK.

6. Click Preview metric data to view the data that will be transferred to the Metric Studio metric store if you do not filter it out before you publish or execute the extract.

   Tip: If you want to see a different number of rows of data, change the number, and click Test sample.

7. Click Close and then click Next or Finish.
   - If you click Finish, you exit the wizard. You can continue to create the metrics extract later.
Tip: In the main page, in the **Project Explorer** pane, double-click the metrics extract that you want to continue to create.

**Steps to Apply Filters**

1. If the **Create Extract - Filter Data** page appears, in the **Scorecard levels** pane, select the levels that you want.

2. If you want, select the **Show scorecard IDs** check box.

3. Click **Refresh** to show the data.

4. Expand the list and click the box to the left of the items that you want to filter out.

   You can filter the values for the categories within the levels either to exclude them completely or to hide them but still include their values in the totals (p. 38).

5. Click **Finish**.

6. Save your project.

Your extract is now complete. For information on using the extract in Metric Studio, see "Transferring Extracts and Data to Metric Studio" (p. 45).

**Create a Metrics Extract for an .IQD File**

If you want to use an .iqd file as an import source, continue with the following instructions. To use a cube-based package as an import source, see "Create a Metrics Extract for a Cube-based Framework Manager Package" (p. 17). To use a relational Framework Manager package as an import source, see "Create a Metrics Extract for a Relational Framework Manager Package" (p. 23).

**Steps to Specify an Import Source**

1. In the **Create Extract - Select Import Source** page, click **Create Package Import Source**.

   If the wizard is not running, from the **Project Explorer**, click **Import Sources** and, from the **Actions** menu, click **Create IQD Import Source**.

2. In the **Create Import Source - Name** page, type a name and description for the new import source, and click **Next**.

3. In the **Create Import Source - Select Package** page, select an IQD source and click **Finish**, or do the following:

   - Click **Create .IQD Import Source**.

   - In the **Create Import Source - Name** page, type a name and, if you want, a description of the import source, and click **Next**.

   - In the **Select IQD Sources** page, click **Add**.

   - Browse to the .iqd source that you want, select it, and click **Open**.

   The **Select IQD Sources** page reappears.
Use caution when adding multiple .iqd files to the iqd source because Metric Designer generates cross-product results.

- Under Specify connection, click the database that you want and click the ellipsis (...) next to it.
- In the Select data source connection page, either click the data source that you want, click OK, and click Finish, or click New Data Source, follow the instructions, and click Finish.

For more information about creating the data source, see the IBM Cognos Administration and Security Guide.

**Steps to Map the Scorecard**

1. In the Create Extract - Scorecard Mapping page, if you want to change the parent scorecard, do one of the following:
   - Click the scorecard icon next to the Parent Scorecard box, and then click the scorecard that you want.
   - Click the arrow next to the scorecard icon, and then click Metric Package Pick List.
   - Click the arrow next to the scorecard icon, click Expression Editor, and then create an expression to identify the parent scorecard.

2. In the Available objects pane, click the items that you want and drag them to the Levels pane. The order in which you drag the items sets the order of the levels for the scorecard.

   **Tip:** To create a new level, under New scorecard levels, click Create.

   By default, IBM Cognos Metric Designer uses the value of the item for both the scorecard name and the scorecard ID.

3. If you want to use a different item for either the name or ID, drag the item that you want into the value field for that attribute.

   **Tip:** You can also use the expression editor to modify these attributes. For example, modify the name of a scorecard for the city of London as follows:
   - `[city] + '_SC' creates London.SC
   - `[city] + 'Sales Office' creates a scorecard named London Sales Office
   - `left ([city],3) creates a scorecard named Lon`

4. If you want to combine metrics from different data sources on the same scorecard, do the following:
   - In the Level attributes pane, click the ellipsis (...) next to Identification Code and Name and change the information to match the scorecard where you want this metric to appear (p. 39).
   - Click the ellipsis (...) next to Description, Language code, and Owner and enter the information that you want.
In the list, next to **Append to scorecard ID**, click **Yes**.

If you click **No**, and Metric Designer finds duplicate IDs, either create an expression to make the ID unique or choose a different query item that will result in a unique ID.

**Tip:** To check your scorecard hierarchy for duplicate IDs, click **Preview scorecard tree**.

5. If you want to filter to limit the items, under the **Level filters** pane, do the following:
   - Click **Create**.
   - In the expression editor page, in the **Available components** pane, drag the items that you want to the **Expression definition** pane.
     
     If you create two filters, Metric Designer considers them joined by the **AND** operator.
   - Click the **Functions** tab, click the functions that you want, and drag them to the **Expression definition** pane.
   - Click **OK**.

**Note:** In the expression editor, if you want to create a filter that uses a Member Unique Name (MUN) query item, you must use the roleValue expression for the member you are filtering against. For example, if you are creating scorecards for products using the Great Outdoor Company sample, and you want to filter out the Golf Equipment product category using the Member Unique Name query item, you must use the roleValue function for the second part of the expression as shown in the following expression:

```
[great_outdoors].[Products].[Products].[Products].[Member Unique Name] <> roleValue ( '_memberUniqueName',[great_outdoors].[Products].[Products].[Product line]->:[PC].[Products (Root)].[5~236] )
```

The roleValue function is located in the Functions tab of the expression editor under Common Functions, R-Z, Advanced.

6. Click **Next** or **Finish**.

   If you click **Next**, the **Create Extract - Time Hierarchy Attributes** page appears.

   If you click **Finish**, you exit the wizard. You can continue to create the metrics extract later.

   **Tip:** In the main page, in the **Projects explorer** pane, double-click the metrics extract that you want to continue to create.

**Steps to Select a Time Period**

1. In the **Create Extract - Time Hierarchy Attributes** page, in the **Available objects** pane, click the date that you want to use for the year and drag it to the level of granularity that you want in the **Time hierarchy attributes** pane.

   You can use the same date for more than one level.

   Metric Studio needs time stamp expressions for the mapping attributes shown on this page.

   For Year, Quarter, and Month, Metric Designer needs the date values only for the first dates of each level. For example, for the quarters in 2005, these dates would be January 1, 2005, April 1, 2005, July 1, 2005, and October 1, 2005. If your dates include values at a lower
granularity, Metric Designer automatically creates a time stamp expression to remove them. For example, if the date you used for Quarter includes values at the month level for 2005, the expression removes all values except those for January 1, 2005, April 1, 2005, July 1, 2005, and October 1, 2005. The expression appears under Value.

You can instead create your own time stamp expression.

When entering date items, ensure that the items are in the default date format for the Metric Studio database platform.

2. If you want to filter to limit the items, under the **Level filters** pane, do the following:
   - Click Create.
   - In the expression editor page, in the **Available components** pane, drag the items that you want to the **Expression definition** pane.
     - If you create two filters, Metric Designer considers them joined by the **AND** operator.
   - Click the **Functions** tab, click the functions that you want, and drag them to the **Expression definition** pane.
   - Click **OK**.

3. Click **Next**.

   The **Create Extract - Metrics Mapping** page appears, where you map measures to metric types.

**Steps to Map Metrics**

1. If the **Create Extract - Metrics Mapping** page appears, in the **Available Objects** pane, do the following:
   - Click the measure that you want to use to create the metric for the actual value and drag it to the **Metric mappings** pane.
     - You must create at least one metric type before you map a measure to a metric. For more information, see "Create a Metric Type" (p. 35).
   - Click one measure each for the **Tolerance** and **Target** attributes shown in the **Metric attributes** pane and drag the measure to the appropriate row under the **Value** column.
     - If you typed a constant value under Metrics Mapping for actual, tolerance, target, or user-defined columns, Metric Designer considers each value as a constant and does not sum it.

2. For the **Qualifier identification** and **Qualifier name**, either type a value or click the ellipsis (...) and do the following:
   - In the expression editor, in the **Available components** pane, drag the items that you want to the **Expression definition** pane.
   - Click the **Functions** tab, click the functions that you want, and drag them to the **Expression definition** pane.
   - Click **OK**.
The number of rows shown depends on the metric type that you selected. You must already have created the metric type in Metric Studio. For more information, see the Metric Studio User Guide.

3. For the **Aggregate function** and **Rollup aggregate function**, click a rule.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatic</td>
<td>The application automatically chooses to apply either <strong>Summarize</strong>, <strong>Calculated</strong>, or None.</td>
</tr>
<tr>
<td>Total</td>
<td>Sum the metric items.</td>
</tr>
<tr>
<td>Count</td>
<td>Count the metric items.</td>
</tr>
<tr>
<td>Average</td>
<td>Average the metric items.</td>
</tr>
<tr>
<td>Minimum</td>
<td>The minimum metric value.</td>
</tr>
<tr>
<td>Maximum</td>
<td>The maximum metric value.</td>
</tr>
<tr>
<td>Calculated</td>
<td>This can also be thought of as aggregate-then-calculate. That is, all the terms with a data item’s expression are aggregated according to their own rollup rules and the results of those aggregations are then computed within the overall expression.</td>
</tr>
<tr>
<td>Summarize</td>
<td>Metric values are aggregated in the most obvious, or simplistic, manner, based on either model or data type information.</td>
</tr>
<tr>
<td>Count Distinct</td>
<td>A distinct count of metric values.</td>
</tr>
</tbody>
</table>

The **Aggregate function** describes how detail, transaction data from a relational data source, is summarized prior to the application of any other operations.

For example, a relational fact table may contain transaction records at the granularity of store (geography) and SKU (product), but a metric extract author may only be interested in looking at metric values for city, and product type scorecards. The aggregate attribute allows the author to specify what aggregation rule is to be used to perform this operation.

**Note:** The aggregate attribute is ignored for all OLAP data sources because all data in such data sources is already summarized.

A measure has associated with it a regularAggregate value that is specific in the Framework Manager model. The regularAggregate or **Automatic** value is the default unless it is explicitly overridden by the aggregate attribute of a query data item.

The **Rollup Aggregate function** allows the report author to specify how detail values are summarized for the scorecards. This rollup rule can be different from the rollup rule defined for the aggregate attribute.
For example, a star schema data warehouse contains sales information at a cash register level of detail. The metric author wants the metric values for store level scorecards using the sum of all cash register sales in each store. The specification of \textit{Aggregate} = \textit{Total} on the measure controls this summarization of data. The metric extract itself may then average these sales across all sales scorecards. The specification of \textit{Rollup Aggregate} = \textit{Average} controls this summarization of data in the scorecard structure.

4. If you want to create a new metric, do the following:
   - Under \textit{Metric}, click \textit{Create}.
   - In the \textit{New Metric} dialog box, type a name for the metric and select the metric type.
   - You can create a new metric type for the metric by clicking \textit{Create Metric Type}. For more information, see "Create a Metric Type" (p. 35).
   - Click \textit{OK}.

5. If you want to select a metric to filter on, under the \textit{Metric filters} pane, do the following:
   - Click \textit{Create}.
   - In the expression editor, in the \textit{Available Components} pane, drag the items that you want to the \textit{Expression definition} pane.
     
     If you create two filters, Metric Designer considers them joined by the \textit{AND} operator.
   - Click the \textit{Functions} tab, click the functions that you want, and drag them to the \textit{Expression definition} pane.
   - Click \textit{OK}.

6. Click \textit{Preview metric data} to view the data that will be transferred to the Metric Studio metric store if you do not filter it out before you publish or execute the extract.

   \textbf{Tip:} If you want to see a different number of rows of data, change the number, and click \textit{Test sample}.

7. Click \textit{Close} and then click \textit{Next} or \textit{Finish}.

   If you click \textit{Finish}, you exit the wizard. You can continue to create the metrics extract later.

   \textbf{Tip:} In the main page, in the \textit{Project Explorer} pane, double-click the metrics extract that you want to continue to create.

\textbf{Steps to Apply Filters}

1. If the \textit{Create Extract - Filter Data} page appears, in the \textit{Scorecard levels} pane, select the levels that you want.

2. If you want, select the \textit{Show scorecard IDs} check box.

3. Click \textit{Refresh} to show the data.

4. Expand the list and click the box to the left of the items that you want to filter out.
You can filter the values for the categories within the levels either to exclude them completely or to hide them but still include their values in the totals (p. 38).

5. Click **Finish**.

6. Save your project.

Your extract is now complete. For information on using the extract in Metric Studio, see "Transferring Extracts and Data to Metric Studio" (p. 45).

**Create a Metric Type**

You create metric types to define attributes and calculations for a collection of related metrics. For more information about metric types, see the IBM® Cognos® Metric Studio User Guide.

You must create at least one metric type before you map a measure to a metric.

**Steps for General Settings**

1. Do one of the following:
   - Complete the steps in the Create Extract wizard until you reach the Create Extract - Metrics Mapping page. Note that this applies only if you are creating a metrics extract, not an objects extracts.
   - In the Project Explorer pane of the Project page, right-click the metric extract, click **Edit Definition**, and click the Metrics Mapping tab.

2. Click **Create Metric Type**.

3. Click the **General** tab.

4. Under **Language**, select the language that you want.

5. In the **Name** box, type a descriptive name for the metric type.

6. If you want, type a description and a technical description of the metric type.

7. If you want to change the owner, do the following:
   - Click **Change Owner**.
   - On the Navigate tab, click the owner that you want.
     **Tip:** Alternatively, you can click the Search tab, type all or part of the name of the owner, and then click the string criterion, the Scope item, Search, and the owner that you want.

8. If you want, in the **Identification code** box, type a code for this metric type.
   If you leave this box blank, Metric Studio automatically generates an identification code.

9. Click the **Default group view** value that you want.
   For more information, see the Metric Studio User Guide.

10. Under **Calendar Details**, do the following:
Chapter 2: Creating Extracts

- Click the **Business calendar level** value that you want.
  
The level that you choose sets the lowest level at which Metric Studio stores data. For example, if you select quarterly, Metric Studio does not load monthly values.

If you add a lower calendar period to your calendar, such as adding days to a calendar that contained years and months, you must reload your metric data for the data to be allocated to the new level.

- Click the **Business calendar level for most recent values view** level that you want.
  
The level that you choose sets the level of data that Metric Studio shows in the latest data view. For example, if you choose monthly, Metric Studio shows data for the latest month for which there is a score. If there is no score for the latest month, Metric Studio shows the latest month with data. If there is no score or data, Metric Studio shows the message No data.

11. Under **Number Format**, do the following:
   - Under **Unit**, click the unit of measurement for the metric type.
   - Under **Unit Symbol**, click **Display unit symbol** or **Do not display unit symbol**.
   - Under **Decimal Places**, click the number of decimal places to show for the metric type.

12. Under **History Chart Properties**, do the following:
   - Type the value for **Minimum value**.
   - Type the value for **Maximum value**.
   - If you want, select the **Show zero value** check box.

**Steps to Specify How Metric Values Are Calculated**

1. Click the **Value Types** tab for the metric type that you are creating.

2. Under **Metric type default calculation**, click **No calculation - this value will be loaded or entered** or **Define calculation**.
   
   For more information, see the Metric Studio *User Guide*.

3. If you select **Define calculation**, do the following:
   - Click **Edit**.
   - Define the expression for the calculation.
   - Click **OK**.

4. Under **Actual** and **Target**, do the following:
   - Click the **Business calendar rollup calculation** to use.

   Metric Studio rejects data that is loaded and entered at a level other than the one you specify unless you select **Rollup is supplied by client** and the data entry levels are equal to or greater than the business calendar level for the metric type.
If you define a calculation for actual, target, tolerance, or user-defined columns, the values for business calendar rollup calculations change to reflect the after rollup and before rollup behaviors of the calculated metric types. The rollup is supplied by client value is not available.

- Click the Business calendar level for loading and entering data level that you want.
- Click the type of Value calculation to use.
  If you select Define Calculation, see step 3.

5. Under Tolerance, do the following:
   - Select the Tolerance type to use.
   - Click the Business calendar rollup calculation to use.
     For more information, see step 4.
   - Click the Business calendar level for loading and entering data level that you want.
   - Click the type of Value calculation to use.
     If you select Define Calculation, see step 3.

6. Under User Defined Columns, do the following:
   - Click the Business calendar rollup calculation to use.
     For more information, see step 4.
   - Click the Business calendar level for loading and entering data level that you want.
   - Click the columns to which these settings apply.

Steps to Specify Performance Behavior
1. Click the Status Indicator tab for the metric type that you are creating.
2. Click the Performance pattern value that you want.
   For information about performance patterns, see the Metric Studio User Guide.
3. Under Score Settings, do the following:
   - Click Use default score calculation or Set target boundaries with user defined columns.
     If you select Set target boundaries with user defined columns, you must set the target thresholds that define the status of a metric.
     The available thresholds are either user-defined columns or targets.
     In a 3-state environment, from the list of available thresholds, click the criteria for changing the status from red to yellow and from yellow to green.
In a 5-state environment, from the list of available thresholds, click the criteria for changing the status from green to partially green, from partially green to yellow, from yellow to partially red, and from partially red to red.

**Steps to Set Metric Type Security**

1. Click the **Permissions** tab for the metric type that you are creating.
2. To change or add users, click **Edit**.

   For more information, see the IBM Cognos Metric Designer *User Guide*.

**Filter Categories in a Metrics Extract**

After you create your metrics extract, you can view the levels for the hierarchies that you included in your metrics extract. You can filter the values for the categories within the levels to exclude them or to hide them but still include their values in totals.

You can also create filters while you are creating a metrics extract to filter on time periods, categories, measures, levels, and data.

For a cube, you filter on the member data. The expression you create to filter the data must be True or False. For example, you can create an expression that filters on the Products level to include only camping equipment.

For measures, you create an expression to filter on a threshold. For example, you can create an expression that filters on the measure Revenue to include only amounts above $50,000.

When filtering a parent-child hierarchy on a dimensional data source, child members are not included in the filter.

**Steps**

1. In the main page, in the **Project Explorer** pane, click the metrics extract that you want.
2. From the **Actions** menu, click **Edit definition**. You can create three types of filters when defining a metrics extract:
   - level filters
   - metric filters
   - scoping filters
3. In the **Edit extract** page, on the **Scorecard Mapping** tab, you can define a level filter to control the structure of the scorecard tree to be created in IBM® Cognos® Metric Studio. This filter is used to select a subset of the elements of a given level that should appear in the scorecard hierarchy. If the filter happens at an intermediate level then any of the items excluded at the level and their descendants will be omitted from the results. This also has an impact on the values that are generated for the metrics. The values associated with the omitted items will not be aggregated up to higher levels.
For example, you could have a scorecard hierarchy that represents the product categories that you want to monitor. If you do not want the Personal Accessories Product Line to be included in the scorecard application, a filter can be defined to exclude it. Once Personal Accessories has been excluded, all of its descendant members from the Product Type and Product levels will also be excluded. In the expression editor, this would look as follows:

\[[\text{great outdoors}].[\text{Products}].[\text{Products}].[\text{Product Line}] \text{ not in } (\{\text{great outdoors}].[\text{Products}].[\text{Products}].[\text{Product Line}]->:[\text{PC}].[\text{MEMBER}].[\text{Personal Accessories}] )\]

Also if you do not want Golf Accessories represented in the hierarchy, a filter can be created on the Product Type level to exclude those products from the scorecard hierarchy. The impact of this filter will be that no child members of Golf Accessories at the Product level will be included in the scorecard tree and the aggregated metric value calculated for the Golf Equipment scorecard at the Product Line level will not include the value for Golf Accessories. In the expression editor, this would look as follows:

\[[\text{great outdoors}].[\text{Products}].[\text{Products}].[\text{Product Type}] \text{ not in } (\{\text{great outdoors}].[\text{Products}].[\text{Products}].[\text{Product Type}]->:[\text{PC}].[\text{MEMBER}].[\text{Golf Accessories}] )\]

4. On the Metrics Mapping tab, you can define a metric filter to further constrain what data is to be included in the extract. This can be a useful place to put global constraints that are independent of the scorecard hierarchy. In the given example, you could also decide that this particular extract will only have product data related to Canada. In the expression editor, this would look as follows:

\[[\text{great outdoors}].[\text{Locations}].[\text{Locations}].[\text{Country}] \text{ in } (\{\text{great outdoors}].[\text{Locations}].[\text{Locations}].[\text{Country}]->:[\text{PC}].[\text{MEMBER}].[\text{Canada}] )\]

5. On the Metrics Scope tab, you can define a scoping filter to control which metrics are created. For each metric, you can specify whether it is created or not. If the metric is not to be created, you can choose to include or exclude its value in the aggregation. For example, rather than eliminate Golf Accessories completely, you can choose to include the values for Golf Accessories in the aggregation of Golf Equipment.

**Note:** You must remove the level filter that excluded Golf Equipment on the Scorecard Mapping tab. This means that scorecards will be created for Golf Accessories, however no metrics will be generated for those scorecards.

### Combining Metrics from Different Sources on One Scorecard

By default, IBM® Cognos® Metric Designer creates new scorecards based on the scorecard structure that you define in each extract. If you want Metric Designer to add metrics to an existing scorecard or a scorecard created by other extracts, you must ensure that the structure and IDs are the same.

If you are creating a metrics extract based on a package or an .iqd file, you must use the expression editor to change the scorecard name and ID to match the name and ID of the existing scorecard. If your metrics extract is based on a cube data source, Metric Designer creates the scorecard name and ID based on the category ID but does not support expressions for names or IDs. If you want to combine metrics from both cube and relational (package or .iqd file) data sources in one scorecard,
you must change the scorecard name and ID on the extract that is based on the relational data source to match the scorecard name and ID on the extract that is based on the cube data source. If you are combining extracts from multiple cubes, you may have to modify the Transformer models so that the category IDs are the same.

When you publish or run the extract in IBM Cognos Metric Studio, the metrics with the same scorecard name and ID appear on the same scorecard.

Do not create scorecards that reference both relational and dimensional data sources in the same package. The results will be unpredictable. If you want to use both relational and dimensional data sources, create a different extract for each source from separate IBM Cognos BI packages. You can then publish them under the same scorecard hierarchy to simulate a mixed scorecard structure.

### Create an Objects Extract

An objects extract defines the metadata for an IBM® Cognos® Metric Studio object, such as the name and description of a scorecard. Each objects extract consists of a set of mappings between an existing IBM Cognos source, such as a published IBM Cognos Framework Manager package, and the attributes of a predefined list of Metric Studio objects, such as scorecards, user-defined columns, and data sources.

You can create an objects extract within a project.

You can also create an objects extract for a metric, but we recommend that you use the metrics extract wizard (p. 15).

You must already have a project.

#### Steps

1. In the main page, in the Project Explorer pane, click the extract folder that you want to create an objects extract under.

2. From the Actions menu, click Create, Objects extract.

3. In the Create Extract - Provide Name page, type a name and, if you want, a description of the objects extract.

4. Click Next.

   The Create Extract - Select Import Source page appears with a list of the import sources that you can use.

5. If you want to use an existing source, click a source, and then click Next.

6. If you want to import a new .iqd import source, do the following:
   - Click Create .IQD Import Source.
   - In the Create Import Source - Name page, type a name and, if you want, a description of the import source.
   - Click Next.
   - In the Select IQD Sources page, click Add.
• Browse to the .iqd source that you want, select it, and click Open.
  The Select .IQD Sources page reappears.

• Under Specify connection, click the database that you want and click the ellipsis (...) next to it.

• In the Select Data Source Connection page, either click the data source that you want, click OK and click Finish, or click New Data Source, follow the instructions to create the data source and click Finish.

  For more information about creating the data source, see the IBM Cognos Administration and Security Guide.

• When the Create Extract - Select Import Source page reappears, click the source that you want, and click Next.

7. If you want to import a new package import source, do the following:
  • Click Create Package Import Source.
  • Type a name and, if you want, a description of the import source.
  • Click Next.
  • Click the package that you want and click Finish; or click Create IBM Cognos PowerCube Package, click the cube data source that you want, click OK and click Finish; or click New Data Source, follow the instructions to create the data source and click Finish.

  For more information about creating the data source, see the IBM Cognos BI Administration and Security Guide.

  When the Create Extract - Select Import Source page reappears, click the source that you want, and click Next.

8. When the Create Objects Extract - Select Object Type page appears, click the object type that you want, and click Next.

9. In the Create Object Extract - Map Object Attributes page, click the item that you want and drag it to the appropriate row under the Values column in the Object Attributes pane.

10. If you want to create an expression, do the following:
  • Click the ellipsis (...) next to a value.
    The Expression Editor page appears, showing the attribute with its hierarchy in the Expression definition pane.
  • In the Available components pane, drag the items that you want to the Expression definition pane.
  • Click the Functions tab, click the functions that you want, and drag them to the Expression definition pane.
  • Click OK.
11. If you want to filter an expression, drag a filter expression from a model or, in the Create Objects Extract - Map Object Attributes page, in the Filter expression pane, do the following:
   - Click Create.
   - In the Expression Editor page, in the Available components pane, drag the items that you want to the Expression definition pane.
     If you create two filters, Metric Designer considers them joined by the AND operator.
   - Click the Functions tab, click the functions that you want, and drag them to the Expression definition pane.
   - Click OK.
     Tip: Click Results to see the data for the attributes that you chose. Click Close to return to the Create object extract - map object attributes page.

12. Click Finish.

13. Save your project.

You can now publish or run your extract. For more information, see "Transferring Extracts and Data to Metric Studio" (p. 45).

### Validate Referenced Objects

You can check the objects that you referenced in IBM® Cognos® Metric Designer to ensure that they are valid. To do this, Metric Designer looks at the IBM Cognos Metric Studio metric package that you are using and compares the objects that you referenced to the objects in the package. Metric Designer can validate at the metric package level and the metrics extract level.

You may want to validate the referenced objects after you create the extract.

#### Steps

1. In the main page, in the Projects Explorer pane, click the package or metrics extract that you want.

2. From the Actions menu, click Show all referenced objects.
   
   The Metrics object page appears showing the objects that you referenced in Metric Designer and whether they exist in the Metric Studio metric package.

3. For each object that is not in the metric package, do the following:
   - Click the metrics extract that contains the object.
   - From the Actions menu, click Edit definition.
   - Click the Metrics mapping tab and correct the problem.
Check IBM Cognos Connection References

IBM® Cognos® Metric Designer automatically checks your references to IBM Cognos Connection whenever you open a model.

You can also check your references manually.

You may want to do this if you clicked Cancel without retargeting your references or if you deleted objects in IBM Cognos Connection and want to check your Metric Designer model to ensure that it is still valid.

Steps

1. In the main page, from the Project menu, click Verify IBM Cognos Connection references.

   If your references are valid, Metric Designer confirms this. Otherwise, Metric Designer shows each invalid reference with an explanation of the problem.

2. Click the arrow to the right of each invalid reference.

   Metric Designer lists valid IBM Cognos Connection objects to which you can retarget the current object.

3. Under Public Folders, click a new reference, and click OK.

   For more information, see the IBM Cognos Administration and Security Guide.
Chapter 3: Transferring Extracts and Data to Metric Studio

After you create your metrics extracts and objects extracts, you must transfer them and the data that they reference to IBM® Cognos® Metric Studio. You can either run or publish an extract. The data that is loaded into Metric Studio depends on the metadata that you mapped in the metrics extract.

Language Settings for PowerCubes that Use Double-byte Characters
To successfully publish a metrics extract created from an IBM Cognos PowerCube that uses double-byte characters, such as Japanese characters, you must use the appropriate language settings on your computer. If double-byte characters in the file name or path for the extract are not supported in the language settings for your computer, you will receive an error message when you attempt to publish the extract. In the regional and language options for your computer, ensure that the language for non-Unicode programs is set to the double-byte language used in the PowerCube.

Transfer Extracts and Data to Metric Studio
You can run an extract in IBM® Cognos® Metric Designer to either write the extract metadata directly to the staging area or write it to files that you must then import into the staging area. We recommend that you use the first method.

Alternatively, you can publish an extract as a metric import task to the content store. In IBM Cognos Connection, you can run a task, or schedule a task or a job, to import the extract into the staging area.

When you publish an extract, Metric Designer

- checks all prompts
- ensures that all references are valid
- checks that the person who is signed in has the required permissions for the objects and actions
- checks the import source object references
- checks the metric store object references, including the metric types and the levels within the time hierarchy
- checks the root scorecard
Whichever method you use, after the data is loaded into the staging area, you must transfer it to the IBM Cognos Metric Studio metric store. For more information, see the IBM Cognos Administration and Security Guide and the Metric Studio User Guide.

**Steps to Run an Extract**
1. In the main page, in the **Project Explorer** pane, click the metrics extract that you want to run.
2. From the **Actions** menu, click **Execute**.
3. In the **Execute extract** page, choose the action that you want:
   - To write the extract metadata to the staging area, click **Write to staging area**, and click **OK**.
   - To write the extract metadata to files first, click **Write to files**, click **OK**, and then open IBM Cognos Connection and click **Import data from files into staging area**.
4. In IBM Cognos Connection, click **Transfer data from staging area into metric store**.

**Steps to Publish an Extract**
1. In the main page, in the **Project Explorer** pane, click the metrics extract that you want to publish.
   You can publish an extracts folder, including all the extracts folders and metrics extracts under it. IBM Cognos Connection replicates the folder hierarchy up to the metric package level and creates a task for each extract. If you choose an extracts folder, Metric Designer prompts you to confirm that you want to replicate the entire hierarchy under it or to select the extracts that you want individually.
2. From the **Actions** menu, click **Publish to content store**.
3. Choose the action that you want and click **OK**.
   If you do not choose **Launch IBM Cognos Connection after successful publish**, you must open IBM Cognos Connection to run the task or job that you created in Metric Designer.
4. After the data is loaded into the staging area, open IBM Cognos Connection and run **Transfer data from staging area into metric store**.
   For more information, see the IBM Cognos Administration and Security Guide.

**View Metrics in Metric Studio**
After you publish your metrics extract and load the resulting staging files into IBM® Cognos® Metric Studio, you can view the metrics in IBM Cognos Metric Designer. You can use the full functionality of Metric Studio to view and work with your metrics.
To view metrics, in the main page, on the toolbar, click the Metric Studio icon.
Metric Studio opens and you can choose the metrics that you want to see. For more information, see the Metric Studio User Guide.
Chapter 4: Managing Projects

In IBM® Cognos® Metric Designer, you can copy (p. 47), move (p. 48), rename (p. 48), and delete (p. 48) projects. You can also perform these tasks from Microsoft® Windows Explorer or the file system.

You can also change a metric package reference (p. 47) and modify an extract definition (p. 47). Projects can also be upgraded from an older version of Metric Designer to a newer version (p. 49).

Change a Metric Package Reference

You can change the metric package reference.

Steps
1. In the main page, in the Project explorer pane, click the metric package reference that you want to change.
2. From the Actions menu, click Edit definition.
3. In the Create Metric Package Reference page, choose an existing or a new package:
   - To choose an existing package, click the package that you want.
   - To choose a new package, click New metric package, and follow the wizard for a new metric package. For more information, see the IBM® Cognos® Administration and Security Guide.

Modify an Extract Definition

You can finish defining an extract any time after you properly exit an extracts wizard. You can also change the settings and values of a completed extract definition.

Steps
1. In the main page, click the extract that you want to complete or change.
2. From the Actions menu, click Edit definition.
3. Click the tab that contains the items that you want to change.
4. Make the changes.

Copy a Project

You may want to create a new project that is based on an existing one. You can copy the existing one, change it as needed, and save it as a new project.
When you copy a project, you create a replica of that project in another location. All files in the project folder, including subfolders, are copied to the new location. When you make changes to the project in one folder, these changes are not reflected in copies of the project in other folders. You cannot create a copy of a project in the same folder as the original.

**Note:** We recommend that you do not use the **Save As** command from the **File** menu to create a copy of a project. If you do, ensure that you specify a target directory that does not contain an IBM® Cognos® Metric Designer project; otherwise, you will overwrite the project in the target location.

You can also move a project (p. 48).

Before you can copy a project, the project must be closed in Metric Designer.

**Steps**
1. From the **File** menu, click **Manage Projects, Copy**.
2. In the **From** text box, enter the .cpf file for the project that you want to copy.
   The project folder name appears in the text box.
3. In the **To** text box, enter the new project location.
   If the target location is a new folder that does not exist, it is created for you.

**Move a Project**

You may decide to move a project if your folder becomes so full that it is difficult to locate particular projects. When you move a project, you are actually copying it to a new folder and deleting it from the current folder. All files in the project folder, including subfolders, are moved to the new location.

You can also copy a project (p. 47).

Before you can move a project, the project must be closed in IBM® Cognos® Metric Designer.

**Steps**
1. From the **File** menu, click **Manage Projects, Move**.
2. In the **From** text box, enter the .cpf file for the project that you want to move.
   The project folder name appears in the text box.
3. In the **To** text box, enter the new project location.
   If the target location is a new folder that does not exist, it is created for you.

**Rename a Project**

When you rename a project, IBM® Cognos® Metric Designer provides a new name for both the .cpf file and the folder that contains it. Secondary project files and log files keep their original name.
If a project that appears in the recent projects list in the Metric Designer Welcome page is renamed, you cannot open the project by clicking the link. You must open the project using the Open command from the File menu.

Before you can rename a project, the project must be closed in Metric Designer.

**Steps**
1. From the File menu, click **Manage Projects, Rename**.
2. In the From text box, enter the .cpf file for the project that you want to rename.
   The project folder name appears in the text box.
3. In the To text box, type the new name for the project.

If the original project folder and .cpf file have the same name, both the folder and .cpf file are renamed. If the names are different, only the project folder is renamed.

**Delete a Project**

When you delete a project, the project folder and all its contents, including any user files, are deleted from the file system and sent to the recycle bin.

Before you delete a project, ensure that the project is closed. If you delete an open project, the open project can no longer be saved. If you accidentally delete a project, you can restore it from the recycle bin.

**Steps**
1. From the File menu, click **Manage Projects, Delete**.
2. In the Project folder text box, enter the .cpf file for the project that you want to delete.
   The project folder name appears in the text box.

The project folder and all its contents are deleted. However, deleted projects are still listed in the most recently used list from the File menu.

**Upgrade a Project**

You can upgrade the metadata in an earlier version of an IBM® Cognos® Metric Designer project by opening the project in the current version of Metric Designer. A message appears indicating that the project was created using an earlier version of Metric Designer and asking if you want to specify a location for the backup file.

Before upgrading a project with a model that is located on a LAN, you must copy the model to a folder on a local machine. From your local machine, upgrade the model, and then copy the project back to the original LAN location.

**Steps**
1. Start Metric Designer.
2. In the **Start** page, click **Open a Project**.

3. Locate the project folder you want to open, and click the .cpf file.

4. Click **Open**.

5. If you are prompted for a location to store the backup, choose a location.
   
   To save the backup model in the default location, click **No**.
   
   To specify a different location, click **Yes** and browse to the location.
   
   You are prompted to back up the project if the model schema version is older than the currently supported version. If you click **Cancel**, Metric Designer will not open the project.

6. If the metric package or data source connection associated with the project no longer exists, you are prompted to choose an existing metric package or data source connection to be used.

7. Click **OK**.
Chapter 5: Tutorial on Creating a Metrics Extract from a PowerCube

In this tutorial, you will generate scorecards by creating a metrics extract from an IBM® Cognos® PowerCube and transferring data into a metric store. You will then view the scorecards in IBM Cognos Metric Studio.

Your goal is to create scorecards to measure the performance of various order methods. You have set a target of $1,000,000 in monthly revenues for each of the order methods of e-mail, fax, sales visit, telephone, and Web. You will create scorecards to measure the performance of each order method against the revenue target you have set.

Before You Begin

Before you start the tutorial, ensure that
- IBM® Cognos® Metric Designer is installed and configured on your computer
- you have access to a metric store
- the metric store is initialized with Standard Calendar, Years, Quarters, Months, Jan. 1, 2004, for 5 years
- you have access to IBM Cognos Metric Studio

Create a Project

You must first create a project and a metric package reference. You could create a reference to an existing metric package, but in this tutorial you will create a new package to reference.

Steps

1. Start IBM® Cognos® Metric Designer and, on the Welcome page, click Create a New Project.
2. Name the project Tutorial, leave the location set to the default, and click OK.
3. In the Create Metric Package Reference dialog box, click New Metric Package.
4. Name the package Tutorial_Package and click Next.
5. Select a metric store and click Next.
6. Review the summary of your metric package reference and click Finish.
7. In the Create Metric Package Reference dialog box, select Tutorial_Package, and click OK twice.
8. In the message box, click Yes to open the Metrics Extract wizard.
Select an Import Source for Your Extract

You will now name the metrics extract and specify an IBM® Cognos® PowerCube as the import source.

Steps

1. Name the extract Tutorial_Extract and click Next.

2. At the bottom of the Create Extract - Select Import Source page, click Create Package Import Source.

3. Name the import source Tutorial_Cube and click Next.

4. At the bottom of the Create Import Source - Select Package page, click Create IBM Cognos PowerCube Package.

5. In the Select Data Source dialog box, click New Data Source. The New Data Source wizard opens. Click Next.

6. Name the data source Tutorial_Cube and click Next.

7. In the Type box, select IBM Cognos PowerCube and click Next.

8. Leave the Read cache size (MB) box blank.

   When you leave this box blank, IBM Cognos Connection uses the default value in the ppds_cfg.xml file in the configuration folder.

9. Locate the Sales and Marketing.mdc file and paste or type its path and filename into either the Windows location box or UNIX or Linux location box.

   This file is located in the installation folder for IBM Cognos BI:
   Cognos\c10\webcontent\samples\datasources\cubes\PowerCubes\En\n
10. At the bottom of the page, click Test the Connection. After you have confirmed a successful connection, click Finish and close the wizard.

11. In the Select Data Source dialog box, select Tutorial_Cube and click OK.

12. In the Create Import Source - Select Package dialog box, click Finish.

13. In the Create Extract - Select Import Source dialog box, click Next.

Map the Scorecards

You will now specify the order items for which you want to generate scorecards.

Steps

1. In the Available Objects pane of the Create Extract - Scorecard Mapping page, click Order Method and drag it to the New Scorecard Levels pane.
2. Click **Preview Scorecard Tree** to generate the following preview.

3. Close the preview page and click **Next**.

### Set Time and Currency Mappings

You will now map the time periods and currency from the data in the cube to the metric store. You need to map each of the business calendar levels of year, quarter, and month.

If you have already used this data source to create a metrics extract, the Specify Business Calendar Mappings page will not appear. You can access the page by clicking **Edit Metric Studio Time Mappings** at the bottom of the **Time Periods Filtering** page.

**Steps**

1. In the **Specify Business Calendar Mappings** page, under **Business Calendar Level**, click **Year**.
2. In the **Data Mapping** column, click the ellipsis points (...) beside the first period: **2004 (Jan 1 2004 - Dec 31 2004)**.
3. In the **Available Objects** pane, click **2004**, and click **OK**.
The first year and all years following are automatically mapped.

4. Use the same method to map the business calendar levels for both quarters and months.

5. When you finish mapping the business calendar, click **Next**.

6. On the Specify Currency Mappings page, select **United States of America, dollar** and click **Next**.

### Select Dimensions to Display as Reports

You can choose the reports that you want to be automatically generated from the cube.

**Steps**

1. In the **Select Dimensions** page, select each dimension that you want displayed as a report.
   
   You can use the box to the right of each dimension to change the name of a report.

2. Click **Finish**.

### Select a Time Period

You will now specify the time period that you want to include data for.

**Step**

- Select **Use All Time Periods** and click **Next**.

### Create a Metric Type

Before you can map the metrics you need to create at least one metric type.

**Steps**

1. On the **Metrics Mapping** page, click **Create Metric Type**.

2. In the **New Metric Type** page, type the name **Revenue**.
3. Click the **Columns and Calculations** tab.

4. In the **Business Calendar Rollup Calculation** boxes for Actual, Target, and Tolerance, select Rollup is supplied by client.

5. Leave the default settings for all other boxes and click **OK**.

## Map the Metrics

You will now select the measures you want to use as metrics in your scorecards. In this case, you want to select **Revenue** as your metric. You also want to set a revenue target of $1,000,000 for each order method, and you want your scorecards to have a tolerance of 20% relative to the target.

**Steps**

1. On the **Metrics Mapping** page, in the **Available Objects** pane, select **Revenue** and drag it to the **Metric Mappings** pane.

2. In the **Metric Attributes** area, type .2 for **Tolerance**, and 1000000 for **Target**.  
   The target applies to the smallest time period in your calendar. In this case it is month. 
   In this case, you provide an absolute value for the target. If your cube contained a dimension for projected revenue, then you would drag it from the **Available Objects** pane to the target attribute.

3. Click **Next**.

## Select the Metric Values to Include on the Scorecards

You want your scorecards to show values for all order methods except for **Mail** and **Special**.

**Steps**

1. On the **Filter Data** page, click **Refresh**.

2. Exclude the metric values for both **Mail** and **Special**.
3. Click Finish and validate the extract when prompted.

**Execute the Extract**

You have created a metrics extract, and now you will use it to move the data to the metric staging area. You can do this by either publishing or executing the extract.

When you publish an extract, the processing occurs on the server. You can publish a job and then schedule it to be run.

When you execute an extract, the processing occurs on your computer. You can either write directly to the metric staging area, or save your extract as a file to be used later.

In this tutorial, you will use the Execute command to write to the metric staging area.

**Steps**

1. In the Project Explorer pane, right-click Tutorial_Extract, and then click Execute.

2. Select Write to metric staging area and click OK.
Transfer Data from the Staging Area to the Metric Store

You will now open IBM® Cognos® Metric Studio and transfer your data from the metric staging area to the metric store. The scorecards you created can then be viewed.

Steps
1. Open Metric Studio and click Tutorial_Package.
2. From the Tools menu, click Metric maintenance.
3. Click Transfer data from staging area into metric store.
   The transfer may take several minutes.

View Your Scorecards

In IBM® Cognos® Metric Studio you can now view the scorecards you created.

Steps
1. Open Metric Studio, with Tutorial_Package selected.
2. In the left pane, click Scorecards.

3. Click each order method to view the scorecards.
Chapter 6: Tutorial on Creating a Scorecard Hierarchy Using Multiple Dimensions

Building on the skills learned in the tutorial "Tutorial on Creating a Metrics Extract from a PowerCube" (p. 51), you will generate scorecards using multiple dimensions in an IBM® Cognos® PowerCube, resolve duplicate scorecard IDs, and modify the expression used to generate metric names so that metric names are unique.

Your goal is to create scorecards to measure the revenue of product lines across regions.

Before You Begin

Before you start the tutorial, ensure that

- IBM® Cognos® Metric Designer is installed and configured on your computer
- you have access to a metric store
- you have access to IBM Cognos Metric Studio
- you are familiar with the Metric Designer concepts of creating a project, selecting an import source, mapping scorecards, setting time and currency mappings, mapping metrics, and executing extracts
- you are familiar with the Metric Studio concepts of transferring data from the staging area to the metric store and viewing scorecards and metric types
- you create a Metric Studio package called Great Outdoors Company that references an empty metric store called Great Outdoors Company

Create the Project, Metrics Extract, and Package Import Source

A project is a set of design extracts that describe the data movement from a source to a metric store. You select the package import source to identify the source of your scorecard hierarchy and metrics.

Steps

1. Start IBM® Cognos® Metric Designer.
2. Create a new project and name it Product Line by Region. Ensure that it references the Great Outdoors metric package.
3. Click Yes when prompted to open the Metrics Extract wizard.
4. Name the extract Scorecard_Hierarchy_Extract and click Next.
5. At the bottom of the Create Extract - Select Import Source page, click Create Package Import Source.
6. Name the import source **Great_Outdoors_Cube** and click Next.

7. At the bottom of the **Create Import Source - Select Package** page, click Create IBM Cognos PowerCube Package.

8. In the **Select Data Source** dialog box, click New Data Source. The New Data Source wizard opens. Click Next.

9. Name the data source **Great_Outdoors_Cube** and click Next.

10. In the **Type** box, select IBM Cognos PowerCube and click Next.

11. Leave the **Read cache size (MB)** box blank.

   When you leave this box blank, IBM Cognos Connection uses the default value in the ppdsCfg.xml file in the configuration folder.

12. Locate the Sales and Marketing.mdc file and paste or type its path and filename into either the Windows location box or UNIX or Linux location box.

   This file is located in the installation folder for IBM Cognos BI:
   
   Cognos\c10\webcontent\samples\datasources\cubes\PowerCubes\En\n
13. At the bottom of the page, click Test the Connection. After you have confirmed a successful connection, click Finish and close the wizard.

14. In the **Select Data Source** dialog box, select **Great_Outdoors_Cube** and click OK.

15. In the **Select Package** dialog box, click Finish.

16. In the **Create Extract - Select Import Source** dialog box, click Next.

   The Create Extract - Scorecard Mapping window appears.

**Map the Scorecards**

You will now specify the scorecards that you want to generate and their order.

**Steps**

1. In the Available Objects pane of the Create Extract - Scorecard Mapping page, click Region and drag it to the New Scorecard Levels pane.
2. In the Available Objects pane of the Create Extract - Scorecard Mapping page, click Product line and drag it under Region in the New Scorecard Levels pane.

3. Click Preview Scorecard Tree to generate the following preview.

When you expand the scorecard hierarchy in the Scorecard data pane, you see that the scorecards have duplicate IDs. This is also shown in the Duplicate IDs pane.
Duplicate scorecard IDs occur when a scorecard is created under more than one parent. For example, there are many regions so the product line scorecards have duplicate IDs.

You must resolve the duplicate IDs before you continue or you will see unexpected results when viewing the scorecards in IBM® Cognos® Metric Studio.

4. Close the preview page.

5. In the Create Extract - Scorecard Mapping window, click the Product Line scorecard level and select Yes for the Append Scorecard attribute.

When you click on Preview Scorecard Tree, the duplicate IDs are resolved.

6. Click Close.

7. To set time and currency mappings, click Next.
Set Time and Currency Mappings

You will now map the time periods and currency from the data in the cube to the metric store. You must map each of the business calendar levels of year, quarter, and month.

If you have already used this data source to create a metrics extract, the Specify Business Calendar Mappings page will not appear. You can access the page by clicking Edit Metric Studio Time Mappings at the bottom of the Time Periods Filtering page.

Steps
1. In the Specify Business Calendar Mappings page, under Business Calendar Level, click Year.
2. In the Data Mapping column, click the ellipsis points (...) beside the first period: 2004 (Jan 1 2004 - Dec 31 2004).
3. In the Available Objects pane, click 2004, and click OK.
   The first year and all years following are automatically mapped.
4. Use the same method to map the business calendar levels for both quarters and months.
5. When you finish mapping the business calendar, click Next.

Select Dimensions to Display as Reports

You can choose the reports that you want to be automatically generated from the cube.

Steps
1. In the Select Dimensions page, select each dimension that you want displayed as a report.
   You can use the box to the right of each dimension to change the name of a report.
2. Click Finish.

Select a Time Period

You will now specify the time period that you want to include data for.

Step
• Select Use All Time Periods and click Next.

Create a Metric Type

Before you can map the metrics you need to create at least one metric type.

Steps
1. On the Metrics Mapping page, click Create Metric Type.
2. In the New Metric Type page, type the name Revenue.

3. Click the Columns and Calculations tab.

4. In the Business Calendar Rollup Calculation boxes for Actual, Target, and Tolerance, select Rollup is supplied by client.

5. Leave the default settings for all other boxes and click OK.

Map the Revenue Metric

You will now select the measures you want to use as metrics in your scorecards. In this case, you want to select Revenue as your metric. You also want to set a revenue target of $1,000,000 for each order method, and you want your scorecards to have a tolerance of 20% relative to the target.

Steps
1. On the Metrics Mapping page, in the Available Objects pane, select Revenue and drag it to the Metric Mapping pane.

2. In the Metric Attributes area, type .2 for Tolerance, and 1000000 for Target.
   The target applies to the smallest time period in your calendar. In this case it is month.

   In this case, you provide an absolute value for the target. If your cube contained a dimension for projected revenue, then you would drag it from the Available Objects pane to the target attribute.

3. Click Next.

4. In the Create Extract - Filter Data window, ensure that the check boxes next to Region and Product line are selected.

5. Click Finish, and validate the extract when prompted.

Execute the Extract

You have created a metrics extract, and now you will use it to move the data to the metric staging area. You can do this by either publishing or executing the extract.

When you publish an extract, the processing occurs on the server. You can publish a job and then schedule it to be run.

When you execute an extract, the processing occurs on your computer. You can either write directly to the metric staging area, or save your extract as a file to be used later.

In this tutorial, you will use the Execute command to write to the metric staging area.

Steps
1. In the Project Explorer pane, right-click Scorecard_Hierarchy_Extract, and then click Execute.

2. Select Write to metric staging area and click OK.
Transfer Data from the Staging Area to the Metric Store
You will now open IBM® Cognos® Metric Studio and transfer your data from the metric staging area to the metric store. The scorecards you created can then be viewed.

Steps
1. Open Metric Studio and click Great Outdoors Company.
2. From the Tools menu, click Metric maintenance.
3. Click Transfer data from staging area into metric store.
   The transfer may take several minutes.

View Your Scorecards
In IBM® Cognos® Metric Studio you can now view the scorecards you created.

Steps
1. Open Metric Studio with Great Outdoors Company selected.
2. In the left pane, click Scorecards.
3. Click each region to view the scorecards.

View Your Metrics
In IBM® Cognos® Metric Studio you can now view the metrics you created under the Revenue metric type.

Steps
1. In the left pane, click Metric Types.
2. Click Revenue.

Even though you created unique scorecards IDs, there appear to be duplicate metrics. These are not duplicates but rather individual metrics for each region. You must now uniquely name the metrics by customizing the display names.

### Customize Metric Names

You will customize the metric names by adding the long name of the region to the product line name.

**Steps**

1. In IBM® Cognos® Metric Designer, open the Product Line by Region project.
2. Open the Scorecard_Hierarchy_Extract metrics extract.
3. Under New scorecard levels, click Product line.
4. Under the Scorecard Mapping tab, click the ellipsis (...) button next to the attribute Name.
5. From the Available Components pane, drag Region-Long Name to the Expression Definition pane.
6. In the Expression Definition pane, type + '-' between the two dimensions.
7. Click **OK** twice.

8. Execute the extract again and transfer the data from the staging area to the metric store.

9. View the metrics under the metric type **Revenue**.

   Each metric is now uniquely named by region.
Chapter 6: Tutorial on Creating a Scorecard Hierarchy Using Multiple Dimensions
Appendix A: Troubleshooting

This chapter describes some common problems you may encounter and possible ways to solve them.

CCLAssert Message Encountered When Running an Extract Against an SAP Data Source

When an extract is run against an SAP data source, you encounter a CCLAssert message. This error typically occurs if the extract is being run against a ragged hierarchy and a level filter is being used. To resolve this issue, avoid using a filter.

Report From OLAP Data Source Is Not Displayed and Error Processing Template Is Encountered

When you drill down on a metric sourced from an OLAP data source and navigate to the report tab, the report is not displayed. The following error is displayed:

Error processing template.

This error occurs when you execute the extract directly into staging tables from the Metric Designer UI or by running a published extract from IBM Cognos Connection.

Workaround: Execute the extracts to flat files and then load these files using the metrics maintenance task Import and transfer data from files into metric store.

No Rollups Are Generated for Some Calculated Measures in an SAP Info Query Data Source

Metric Designer may not generate any rollups for an extract that references a calculated member of an SAP Info Query data source. The affected measures will appear in the IBM Cognos Framework Manager model as having a Regular Aggregate attribute value of unknown.

There is no workaround.

Adding Multiple IQD Files to an Import Source

When you add multiple IQD files to an import source, Metric Designer creates an outer join between the first pair of non-numeric, non-date columns with matching names. The join approach is designed to work with a single fact IQD file and with multiple dimension IQD files.

Metric Designer does not recognize joins where more than one column is required for a join condition. In this case, enter the IQD files into Metric Designer as separate import sources or combine them into a single IQD using IBM Cognos Impromptu.
**Previewed Scorecard Hierarchy Shows Blanks**

If you use `if () then () else ()` statements in an expression for level attributes, you will see blank entries when you preview the scorecard hierarchy.

The workaround is to change the expression to cast the query item to VARCHAR. For example:

```sql
if (cast([great_outdoors].[Locations].[Locations].[Country].[PPDS_CODE],VARCHAR(1000)) = 'Canada') then ('Craig') else ('George')
if ( cast([great_outdoors].[Locations].[Locations].[Country].[Country] ,VARCHAR(1000)) = 'Canada') then ('Craig') else ('George')
```

To eliminate duplicates for items other than 'Canada', you can add a level filter expression. For example:

```sql
[great_outdoors].[Locations].[Locations].[Country].[Country] = 'Canada' or [great_outdoors].[Locations].[Locations].[Country].[Country] = 'China'
```
Appendix B: Using the Expression Editor

An expression is any combination of operators, constants, functions, and other components that evaluates to a single value. You build expressions to create calculation and filter definitions. A calculation is an expression that you use to create a new value from existing values contained within a data item. A filter is an expression that you use to retrieve a specific subset of records.

Searching for Values May Return Unexpected Results

In the expression editor, when searching for values for a data item, the results you obtain may contain unexpected results if the data item is not a string data type. Because users can edit the expression for a data item, IBM Cognos BI cannot determine with certainty what the data type is. Therefore, IBM Cognos BI guesses the data type of the data item by looking at its aggregate and rollup aggregate set.

Calculation Components

You build calculations, or expressions, in the expression editor using the following components:

- operators (p. 72)
- summaries (p. 79)
- member summaries (p. 91)
- constants (p. 94)
- constructs (p. 96)
- business date/time functions (p. 97)
- block functions (p. 101)
- macro functions (p. 102)
- common functions (p. 112)
- dimensional functions (p. 119)
- DB2® (p. 142)
- Informix (p. 158)
- Microsoft® Access (p. 164)
- Oracle (p. 178)
- Red Brick (p. 187)
- Microsoft SQL Server (p. 192)
Appendix B: Using the Expression Editor

- Teradata (p. 200)
- SAP BW (p. 206)
- Sybase (p. 207)
- Postgres (p. 215)
- Vertica (p. 221)
- Netezza (p. 172)
- Paraccel (p. 227)
- MySQL (p. 229)
- Greenplum (p. 234)
- report functions (p. 239)

Operators

Operators specify what happens to the values on either side of the operator. Operators are similar to functions, in that they manipulate data items and return a result.

( Identifies the beginning of an expression.

**Syntax**

```
(expression)
```

) Identifies the end of an expression.

**Syntax**

```
(expression)
```

* Multiplies two numeric values.

**Syntax**

```
value1 * value2
```

, Separates expression components.

**Syntax**

```
expression (parameter1, parameter2)
```

/ Divides two numeric values.
**Syntax**
\[
\text{value1} / \text{value2}
\]

||

Concatenates, or joins, strings.

**Syntax**
\[
\text{string1} \text{|| string2}
\]

+  

Adds two numeric values.

**Syntax**
\[
\text{value1} + \text{value2}
\]

-  

Subtracts two numeric values or negates a numeric value.

**Syntax**
\[
\text{value1} - \text{value2}
\text{or}
\text{value} - \text{value}
\]

<  

Compares the values that are represented by "value1" against "value2" and retrieves the values that are less than "value2".

**Syntax**
\[
\text{value1} < \text{value2}
\]

<=  

Compares the values that are represented by "value1" against "value2" and retrieves the values that are less than or equal to "value2".

**Syntax**
\[
\text{value1} <= \text{value2}
\]

<>  

Compares the values that are represented by "value1" against "value2" and retrieves the values that are not equal to "value2".

**Syntax**
\[
\text{value1} <> \text{value2}
\]

=  

Compares the values that are represented by "value1" against "value2" and retrieves the values that are equal to "value2".
Syntax
value1 = value2

>  

Compares the values that are represented by "value1" against "value2" and retrieves the values that are greater than "value2".

Syntax
value1 > value2

->

Separates the components in a literal member expression.

Syntax
[namespace].[dimension].[hierarchy].[level]->[L1]

>=

Compares the values that are represented by "value1" against "value2" and retrieves the values that are greater than or equal to "value2".

Syntax
value1 >= value2

and

Returns "true" if the conditions on both sides of the expression are true.

Syntax
argument1 and argument2

auto

Works with summary expressions to define the scope to be adjusted based on the grouping columns in the query. The scope is context-dependent.

Syntax
aggregate_function ( expression AUTO )

between

Determines if a value falls in a given range.

Syntax
expression between valuel1 and valuel2

Example
[Revenue] between 200,000 and 300,000

Result: Returns the number of results with revenues between 200,000 and 300,000.
case

Works with when, then, else, and end. Case identifies the beginning of a specific situation, in which when, then, and else actions are defined.

Syntax

```
    case expression { when expression then expression } [ else expression ] end
```

contains

Determines if "string1" contains "string2".

Syntax

```
    string1 contains string2
```

currentMeasure

Keyword that can be used as the first argument of member summary functions. This function appears in the Total Revenue by Country sample report in the GO Data Warehouse (query) package.

Syntax

```
    aggregate_function ( currentMeasure within set expression )
```

default

Works with the lookup construct.

Syntax

```
    lookup (....) in (....) default (....)
```

distinct

A keyword used in an aggregate expression to include only distinct occurrences of values. See also the function unique.

Syntax

```
    distinct dataItem
```

Example

```
    count ( distinct [OrderDetailQuantity] )
```

Result: 1704
else

Works with the if or case constructs. If the if condition or the case expression are not true, then the else expression is used. This function appears in the Top 10 Retailers for 2005 sample report in the GO Data Warehouse (analysis) package.

Syntax
\[
\text{if } (\text{condition}) \text{ then } \ldots \text{ else } (\text{expression}) \text{, or case } \ldots \text{ else } (\text{expression})\]
end

end

Indicates the end of a case or when construct.

Syntax
\[
\text{case } \ldots \text{ end}
\]
ends with

Determines if "string1" ends with "string2".

Syntax
\[
\text{string1 ends with string2}
\]
for

Works with summary expressions to define the scope of the aggregation in the query.

Syntax
\[
\text{aggregate\_function} (\text{expression for expression} \{, \text{expression} \})
\]
for all

Works with summary expressions to define the scope to be all the specified grouping columns in the query. See also the for clause.

Syntax
\[
\text{aggregate\_function} (\text{expression for ALL expression} \{, \text{expression} \})
\]
for any

Works with summary expressions to define the scope to be adjusted based on a subset of the grouping columns in the query. Equivalent to the for clause.

Syntax
\[
\text{aggregate\_function} (\text{expression for ANY expression} \{, \text{expression} \})
\]
for report

Works with summary expressions to set the scope to be the whole query. See also the for clause. This function appears in the Customer Returns and Satisfaction sample report in the GO Data Warehouse (analysis) package.
Syntax
aggregate_function ( expression for report )

if

Works with the then and else constructs. If defines a condition; when the if condition is true, the then expression is used. When the if condition is not true, the else expression is used. This function appears in the Top 10 Retailers for 2005 sample report in the GO Data Warehouse (analysis) package.

Syntax
if ( condition ) then ( expression ) else ( expression )

in

Determines if "expression1" exists in a given list of expressions.

Syntax
expression1 in ( expression_list )

in_range

Determines if "expression1" exists in a given list of constant values or ranges.

Syntax
expression1 in_range { constant : constant [ , constant : constant ] }

Example 1
[code] in_range { 5 }
Result: This is equivalent to [code] = 5.

Example 2
[code] in_range { 5: }
Result: This is equivalent to [code] >= 5.

Example 3
[code] in_range { :5 }
Result: This is equivalent to [code] <= 5.

Example 4
[code] in_range { 5:10 }
Result: This is equivalent to ( [code] >= 5 and [code] <= 10 ).

Example 5
[code] in_range { :5,10,20: }
Result: This is equivalent to ( [code] <= 5 or [code] = 10 or [code] >= 20 ).
Appendix B: Using the Expression Editor

**is missing**
Determined if "value" is undefined in the data.

**Syntax**
value is missing

**is null**
Determined if "value" is undefined in the data.

**Syntax**
value is null

**is not missing**
Determined if "value" is defined in the data.

**Syntax**
value is not missing

**is not null**
Determined if "value" is defined in the data.

**Syntax**
value is not null

**like**
Determines if "string1" matches the pattern of "string2".

**Syntax**
string1 LIKE string2

**lookup**
Finds and replaces data with a value you specify. It is preferable to use the case construct.

**Syntax**
lookup ( name ) in ( value1 --> value2 ) default ( expression )

**Example**
lookup ( [Country]) in ( 'Canada'--> ( [List Price] * 0.60), 'Australia'--> ( [List Price] * 0.80 ) ) default ( [List Price] )

**not**
Returns TRUE if "argument" is false or returns FALSE if "argument" is true.

**Syntax**
NOT argument
or

Returns TRUE if either of "argument1" or "argument2" are true.

Syntax

argument1 or argument2

prefilter

Performs a summary calculation before applying the summary filter.

Syntax

summary ([expression] prefilter)

rows

Counts the number of rows output by the query. Use with Count ()

Syntax

count ( ROWS )

starts with

Determines if "string1" starts with "string2".

Syntax

string1 starts with string2

then

Works with the if or case constructs. When the if condition or the when expression are true, the then expression is used. This function appears in the Top 10 Retailers for 2005 sample report in the GO Data Warehouse (analysis) package.

Syntax

if ( condition ) then ..., or case expression when expression then .... end

when

Works with the case construct. You can define conditions to occur when the when expression is true.

Syntax

case [expression] when ... end

Summaries

This list contains predefined functions that return either a single summary value for a group of related values or a different summary value for each instance of a group of related values.
aggregate

Returns a calculated value using the appropriate aggregation function, based on the aggregation type of the expression. This function appears in the Budget vs. Actual sample report in the GO Data Warehouse (analysis) package.

**Syntax**

\[
\text{aggregate ( expression [ auto ] )}
\]
\[
\text{aggregate ( expression for [ all|any ] expression { , expression } )}
\]
\[
\text{aggregate ( expression for report )}
\]

average

Returns the average value of selected data items. Distinct is an alternative expression that is compatible with earlier versions of the product.

**Syntax**

\[
\text{average ( [ distinct ] expression [ auto ] )}
\]
\[
\text{average ( [ distinct ] expression for [ all|any ] expression { , expression } )}
\]
\[
\text{average ( [ distinct ] expression for report )}
\]

**Example**

\[
\text{average ( Sales )}
\]

Result: Returns the average of all Sales values.

count

Returns the number of selected data items excluding null values. Distinct is an alternative expression that is compatible with earlier versions of the product.

**Syntax**

\[
\text{count ( [ distinct ] expression [ auto ] )}
\]
\[
\text{count ( [ distinct ] expression for [ all|any ] expression { , expression } )}
\]
\[
\text{count ( [ distinct ] expression for report )}
\]

**Example**

\[
\text{count ( Sales )}
\]

Result: Returns the total number of entries under Sales.

maximum

Returns the maximum value of selected data items. Distinct is an alternative expression that is compatible with earlier versions of the product.

**Syntax**

\[
\text{maximum ( [ distinct ] expression [ auto ] )}
\]
\[
\text{maximum ( [ distinct ] expression for [ all|any ] expression { , expression } )}
\]
\[
\text{maximum ( [ distinct ] expression for report )}
\]

**Example**

\[
\text{maximum ( Sales )}
\]

Result: Returns the maximum value out of all Sales values.
**median**

Returns the median value of selected data items.

**Syntax**

```
median ( expression [ auto ] )
median ( expression for [ all|any ] expression { , expression } )
median ( expression for report )
```  

**minimum**

Returns the minimum value of selected data items. Distinct is an alternative expression that is compatible with earlier versions of the product.

**Syntax**

```
minimum ( [ distinct ] expression [ auto ] )
minimum ( [ distinct ] expression for [ all|any ] expression { , expression } )
minimum ( [ distinct ] expression for report )
```  

**Example**

```
minimum ( Sales )
```  

Result: Returns the minimum value out of all Sales values.

**moving-average**

Returns a moving average by row for a specified set of values of over a specified number of rows. The "<for-option>" defines the scope of the function. The "at" option defines the level of aggregation and can be used only in the context of relational datasources.

**Syntax**

```
moving-average ( numeric_expression , numeric_expression [ at expression { , expression } ] [ <for-option> ] [ prefilter ] )
moving-average ( numeric_expression , numeric_expression [ <for-option> ] [ prefilter ] )
<for-option> ::= for expression { , expression } | for report | auto
```  

**Example**

```
moving-average ( Qty , 3 )
```  

Result: For each row, returns the quantity and a moving average of the current row and the preceding two rows.

<table>
<thead>
<tr>
<th>Qty</th>
<th>Moving-Average (Qty, 3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>700</td>
<td>450</td>
</tr>
<tr>
<td>400</td>
<td>433.3333</td>
</tr>
<tr>
<td>200</td>
<td>433.3333</td>
</tr>
<tr>
<td>200</td>
<td>266.6667</td>
</tr>
<tr>
<td>500</td>
<td>300.0000</td>
</tr>
</tbody>
</table>
moving-total

Returns a moving total by row for a specified set of values over a specified number of rows. The "<for-option>" defines the scope of the function. The "at" option defines the level of aggregation and can be used only in the context of relational datasources.

Syntax

```
moving-total ( numeric_expression , numeric_expression [ at expression { , expression } ] [ <for-option> ] [ prefilter ] )
moving-total ( numeric_expression , numeric_expression [ <for-option> ] [ prefilter ] )
<for-option> ::= for expression { , expression }|for report|auto
```

Example

```
moving-total ( Qty , 3 )
```

Result: For each row, returns the quantity and a moving total of the current row and the preceding two rows.

<table>
<thead>
<tr>
<th>Qty</th>
<th>Moving-Total (Qty, 3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>700</td>
<td>900</td>
</tr>
<tr>
<td>400</td>
<td>1300</td>
</tr>
<tr>
<td>200</td>
<td>1300</td>
</tr>
<tr>
<td>200</td>
<td>800</td>
</tr>
<tr>
<td>500</td>
<td>900</td>
</tr>
</tbody>
</table>

percentage

Returns the percent of the total value for selected data items. The "<for-option>" defines the scope of the function. The "at" option defines the level of aggregation and can be used only in the context of relational datasources. This function appears in the Percentage Calculation (by year) interactive sample report.

Syntax

```
percentage ( numeric_expression [ at expression { , expression } ] [ <for-option> ] [ prefilter ] )
percentage ( numeric_expression [ <for-option> ] [ prefilter ] )
<for-option> ::= for expression { , expression }|for report|auto
```

Example

```
percentage ( Sales 98 )
```

Result: Returns the percentage of the total sales for 1998 that is attributed to each sales representative.
Sales Rep     Sales 98     Percentage
----------     --------     ----------
Gibbons       60646       7.11%
Flertjan      62523       7.35%
Cornel        22396       2.63%

**percentile**

Returns a value, on a scale of one hundred, that indicates the percent of a distribution that is equal to or below the selected data items. The "<for-option>" defines the scope of the function. The "at" option defines the level of aggregation and can be used only in the context of relational datasources.

**Syntax**

\[
\text{percentile} \ ( \text{numeric_expression} \ [ \text{at} \ expression \ [ , \ expression \ ] ] \ [ \text{<for-option>} \ ] \ [ \text{prefilter} \ ] )
\]

\[
\text{percentile} \ ( \text{numeric_expression} \ [ \text{<for-option>} \ ] \ [ \text{prefilter} \ ] )
\]

\[
\text{<for-option>} \ ::= \text{for} \ expression \ [ , \ expression \ ] | \text{for report} | \text{auto}
\]

**Example**

percentile ( Sales 98 )

Result: For each row, returns the percentage of rows that are equal to or less than the quantity value of that row.

<table>
<thead>
<tr>
<th>Qty</th>
<th>Percentile (Qty)</th>
</tr>
</thead>
<tbody>
<tr>
<td>800</td>
<td>1</td>
</tr>
<tr>
<td>700</td>
<td>0.875</td>
</tr>
<tr>
<td>600</td>
<td>0.75</td>
</tr>
<tr>
<td>500</td>
<td>0.625</td>
</tr>
<tr>
<td>400</td>
<td>0.5</td>
</tr>
<tr>
<td>400</td>
<td>0.5</td>
</tr>
<tr>
<td>200</td>
<td>0.25</td>
</tr>
</tbody>
</table>

**quantile**

Returns the rank of a value within a range that you specify. It returns integers to represent any range of ranks, such as 1 (highest) to 100 (lowest). The "<for-option>" defines the scope of the function. The "at" option defines the level of aggregation and can be used only in the context of relational datasources.

**Syntax**

\[
\text{quantile} \ ( \text{numeric_expression} , \text{numeric_expression} \ [ \text{at} \ expression \ [ , \ expression \ ] ] \ [ \text{<for-option>} \ ] \ [ \text{prefilter} \ ] )
\]

\[
\text{quantile} \ ( \text{numeric_expression} , \text{numeric_expression} \ [ \text{<for-option>} \ ] \ [ \text{prefilter} \ ] )
\]

User Guide 83
Example

quantile ( Qty , 4 )

Result: Returns the quantity, the rank of the quantity value, and the quantity values broken down into 4 quantile groups (quartiles).

<table>
<thead>
<tr>
<th>Qty</th>
<th>Rank (Qty)</th>
<th>Quantile (Qty, 4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>800</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>700</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>600</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>500</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>400</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>400</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>200</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>200</td>
<td>7</td>
<td>4</td>
</tr>
</tbody>
</table>

quartile

Returns the rank of a value, represented as integers from 1 (highest) to 4 (lowest), relative to a group of values. The "<for-option>" defines the scope of the function. The "at" option defines the level of aggregation and can be used only in the context of relational datasources.

Syntax

quartile ( numeric_expression [ at expression { , expression } ] [ <for-option> ] [ prefilter ] )
quartile ( numeric_expression [ <for-option> ] [ prefilter ] )

Example

quantile ( Qty )

Result: Returns the quantity and the quartile of the quantity value represented as integers from 1 (highest) to 4 (lowest).
rank

Returns the rank value of selected data items. The sort order is optional; descending order (DESC) is assumed by default. If two or more rows tie, then there is a gap in the sequence of ranked values (also known as Olympic ranking). The "<for-option>" defines the scope of the function. The "at" option defines the level of aggregation and can be used only in the context of relational datasources. Distinct is an alternative expression that is compatible with earlier versions of the product. Null values are ranked last. This function appears in the Top 10 Retailers for 2005 sample report in the GO Data Warehouse (analysis) package.

Syntax

\[
\text{rank ( expression [ ASC|DESC ] { , expression [ ASC|DESC ] } [ at expression }
\{ , expression } ] [ <for-option> ] [ prefilter ] )
\text{rank ( [ distinct ] expression [ ASC|DESC ] { , expression [ ASC|DESC ] } }
\{ <for-option> ] [ prefilter ] )
\text{<for-option> ::= for expression [ , expression ]|for report|auto}
\]

Example

\[
\text{rank ( Sales 98 )}
\]

Result: For each row, returns the rank value of sales for 1998 that is attributed to each sales representative. Some numbers are skipped when a tie between rows occurs.

<table>
<thead>
<tr>
<th>Sales Rep</th>
<th>Sales 98</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gibbons</td>
<td>60000</td>
<td>1</td>
</tr>
<tr>
<td>Flertjan</td>
<td>50000</td>
<td>2</td>
</tr>
<tr>
<td>Cornel</td>
<td>50000</td>
<td>2</td>
</tr>
<tr>
<td>Smith</td>
<td>48000</td>
<td>4</td>
</tr>
</tbody>
</table>
running-average

Returns the running average by row (including the current row) for a set of values. The "<for-option>" defines the scope of the function. The "at" option defines the level of aggregation and can be used only in the context of relational datasources.

Syntax

running-average ( numeric_expression [ at expression { , expression } ] [ <for-option> ] [ prefilter ] )
running-average ( numeric_expression [ <for-option> ] [ prefilter ] )
<for-option> ::= for expression { , expression } | for report | auto

Example

running-average ( Qty )

Result: For each row, returns the quantity and a running average of the current and the previous rows.

<table>
<thead>
<tr>
<th>Name</th>
<th>Qty</th>
<th>Avg</th>
<th>Running-Average for nameAvg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>7</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>Smith</td>
<td>3</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Smith</td>
<td>6</td>
<td>5</td>
<td>5.33</td>
</tr>
<tr>
<td>Smith</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Wong</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Wong</td>
<td>5</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

running-count

Returns the running count by row (including the current row) for a set of values. The "<for-option>" defines the scope of the function. The "at" option defines the level of aggregation and can be used only in the context of relational datasources.

Syntax

running-count ( numeric_expression [ at expression { , expression } ] [ <for-option> ] [ prefilter ] )
running-count ( numeric_expression [ <for-option> ] [ prefilter ] )
<for-option> ::= for expression { , expression } | for report | auto

Example

running-count ( Qty )

Result: For each row, returns the quantity and a running count of the position of the current row.
### running-difference

Returns a running difference by row, calculated as the difference between the value for the current row and the preceding row, (including the current row) for a set of values. The "<for-option>" defines the scope of the function. The "at" option defines the level of aggregation and can be used only in the context of relational datasources.

**Syntax**

```
running-difference ( numeric_expression [ at expression { , expression } ] [ <for-option> ] [ prefilter ] )
```

```
<for-option> ::= for expression { , expression }|for report|auto
```

**Example**

```
running-difference ( Qty )
```

Result: For each row, returns the quantity and a running difference between the value for the current row and the preceding row.

<table>
<thead>
<tr>
<th>Name</th>
<th>Qty</th>
<th>Running-Difference for name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>7</td>
<td>NULL</td>
</tr>
<tr>
<td>Smith</td>
<td>3</td>
<td>-4</td>
</tr>
<tr>
<td>Smith</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>Smith</td>
<td>4</td>
<td>-2</td>
</tr>
<tr>
<td>Wong</td>
<td>3</td>
<td>-1</td>
</tr>
<tr>
<td>Wong</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

### running-maximum

Returns the running maximum by row (including the current row) for a set of values. The "<for-option>" defines the scope of the function. The "at" option defines the level of aggregation and can be used only in the context of relational datasources.
Syntax
running-maximum ( numeric_expression [ at expression [ , expression ] ] [ <for-option> ] [ prefilter ] )
running-maximum ( numeric_expression [ <for-option> ] [ prefilter ] )
<for-option> ::= for expression [ , expression ]|for report|auto

Example
running-maximum ( Qty )
Result: For each row, returns the quantity and a running maximum of the current and previous rows.

<table>
<thead>
<tr>
<th>Name</th>
<th>Qty</th>
<th>Max</th>
<th>Running-Maximum (Qty) for name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>2</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>Smith</td>
<td>3</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>Smith</td>
<td>6</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>Smith</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Wong</td>
<td>3</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Wong</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

running-minimum
Returns the running minimum by row (including the current row) for a set of values. The "<for-option>" defines the scope of the function. The "at" option defines the level of aggregation and can be used only in the context of relational datasources.

Syntax
running-minimum ( numeric_expression [ at expression [ , expression ] ] [ <for-option> ] [ prefilter ] )
running-minimum ( numeric_expression [ <for-option> ] [ prefilter ] )
<for-option> ::= for expression [ , expression ]|for report|auto

Example
running-minimum ( Qty )
Result: For each row, returns the quantity and a running minimum of the current and previous rows.
## running-total

Returns a running total by row (including the current row) for a set of values. The "<for-option>" defines the scope of the function. The "at" option defines the level of aggregation and can be used only in the context of relational datasources.

### Syntax

```plaintext
running-total ( numeric_expression [ at expression { , expression } ]

<for-option> ::= for expression { , expression }|for report|auto
```

### Example

```plaintext
running-total ( Qty )
```

Result: For each row, returns the quantity and a running total of the current and previous rows.

<table>
<thead>
<tr>
<th>Name</th>
<th>Qty</th>
<th>Total</th>
<th>Running-Total (Qty) for name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>2</td>
<td>18</td>
<td>2</td>
</tr>
<tr>
<td>Smith</td>
<td>3</td>
<td>18</td>
<td>5</td>
</tr>
<tr>
<td>Smith</td>
<td>6</td>
<td>18</td>
<td>11</td>
</tr>
<tr>
<td>Smith</td>
<td>7</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>Wong</td>
<td>3</td>
<td>12</td>
<td>3</td>
</tr>
<tr>
<td>Wong</td>
<td>5</td>
<td>12</td>
<td>8</td>
</tr>
</tbody>
</table>

## standard-deviation

Returns the standard deviation of selected data items.

### Syntax

```plaintext
standard-deviation ( expression [ auto ] )

standard-deviation ( expression for [ all|any ] expression { , expression } )

standard-deviation ( expression for report )
```

### Example

```plaintext
standard-deviation ( ProductCost )
```
Result: Returns a value indicating the deviation between product costs and the average product cost.

**standard-deviation-pop**

Computes the population standard deviation and returns the square root of the population variance.

**Syntax**

\[
\text{standard-deviation-pop ( expression [ auto ] )}
\]
\[
\text{standard-deviation-pop ( expression for [ all|any ] expression [ , expression ] )}
\]
\[
\text{standard-deviation-pop ( expression for report )}
\]

**Example**

\[
\text{standard-deviation-pop ( ProductCost )}
\]

Result: Returns a value of the square root of the population variance.

**total**

Returns the total value of selected data items. Distinct is an alternative expression that is compatible with earlier versions of the product. This function appears in the Budget vs. Actual sample report in the GO Data Warehouse (analysis) package.

**Syntax**

\[
\text{total ( [ distinct ] expression [ auto ] )}
\]
\[
\text{total ( [ distinct ] expression for [ all|any ] expression [ , expression ] )}
\]
\[
\text{total ( [ distinct ] expression for report )}
\]

**Example**

\[
\text{total ( Sales )}
\]

Result: Returns the total value of all Sales values.

**variance**

Returns the variance of selected data items.

**Syntax**

\[
\text{variance ( expression [ auto ] )}
\]
\[
\text{variance ( expression for [ all|any ] expression [ , expression ] )}
\]
\[
\text{variance ( expression for report )}
\]

**Example**

\[
\text{variance ( Product Cost )}
\]

Result: Returns a value indicating how widely product costs vary from the average product cost.

**variance-pop**

Returns the population variance of a set of numbers after discarding the nulls in this set.

**Syntax**

\[
\text{variance-pop ( expression [ auto ] )}
\]
\[
\text{variance-pop ( expression for [ all|any ] expression [ , expression ] )}
\]
\[
\text{variance-pop ( expression for report )}
\]
**Example**

variance-pop ( Qty )

Result: For each row, returns the population variance of a set of numbers after discarding the nulls in this set.

**Member Summaries**

This list contains predefined functions that return either a single summary value for a set of members or a different summary value for each member of a set of members.

**aggregate**

Returns a calculated value using the appropriate aggregation function based on the aggregation type of the expression.

**Syntax**

aggregate ( < currentMeasure|numeric_expression > within set set_expression )
aggregate ( < currentMeasure|numeric_expression > within < detail|aggregate > expression )

**average**

Returns the average value of the selected data items.

**Syntax**

average ( < currentMeasure|numeric_expression > within set set_expression )
average ( < currentMeasure|numeric_expression > within < detail|aggregate > expression )

**Example**

average ( Sales )

Result: Returns the average of all Sales values.

**count**

Returns the number of selected data items excluding null values.

**Syntax**

count ( < currentMeasure|numeric_expression > within set set_expression )
count ( < currentMeasure|numeric_expression > within < detail|aggregate > expression )

**Example**

count ( Sales )

Result: Returns the total number of entries under Sales.

**maximum**

Returns the maximum value of selected data items.
Appendix B: Using the Expression Editor

**Syntax**

\[
\text{maximum} \{ \langle \text{currentMeasure}\mid\text{numeric\_expression} \rangle \ \text{within set set\_expression} \} \\
\text{maximum} \{ \langle \text{currentMeasure}\mid\text{numeric\_expression} \rangle \ \text{within} \ \langle \text{detail}\mid\text{aggregate} \rangle \ \text{expression} \}
\]

**Example**

\[
\text{maximum} \{ \text{Sales} \}
\]

Result: Returns the maximum value out of all Sales values.

**median**

Returns the median value of selected data items.

**Syntax**

\[
\text{median} \{ \langle \text{currentMeasure}\mid\text{numeric\_expression} \rangle \ \text{within set set\_expression} \} \\
\text{median} \{ \langle \text{currentMeasure}\mid\text{numeric\_expression} \rangle \ \text{within} \ \langle \text{detail}\mid\text{aggregate} \rangle \ \text{expression} \}
\]

**minimum**

Returns the minimum value of selected data items.

**Syntax**

\[
\text{minimum} \{ \langle \text{currentMeasure}\mid\text{numeric\_expression} \rangle \ \text{within set set\_expression} \} \\
\text{minimum} \{ \langle \text{currentMeasure}\mid\text{numeric\_expression} \rangle \ \text{within} \ \langle \text{detail}\mid\text{aggregate} \rangle \ \text{expression} \}
\]

**Example**

\[
\text{minimum} \{ \text{Sales} \}
\]

Result: Returns the minimum value out of all Sales values.

**percentage**

Returns the percent of the total value for the selected data items.

**Syntax**

\[
\text{percentage} \{ \text{numeric\_expression} \ \text{[ tuple member\_expression \{ , member\_expression \} ] within set set\_expression } \}
\]

**Example**

\[
\text{percentage} \{ [\text{gosales}].[\text{sales measures}].[\text{quantity}] \ \text{tuple} \ [\text{gosales}].[\text{Staff}].[] . \ [\text{department}] \ \rightarrow \ [\text{West}] \ \text{within set children} \ ( [\text{gosales}].[\text{Staff}].[].[\text{Staff}]) \}
\]

**percentile**

Returns a value, on a scale from 0 to 100, that indicates the percent of a distribution that is equal to or below the selected data items.

**Syntax**

\[
\text{percentile} \{ \text{numeric\_expression} \ \text{[ tuple member\_expression \{ , member\_expression \} ] within set set\_expression } \}
\]
quantile

Returns the rank of a value for the specified range. It returns integers to represent any range of ranks, such as 1 (highest) to 100 (lowest).

**Syntax**

```
quantile ( numeric_expression , numeric_expression [ tuple member_expression { , member_expression } ] within set set_expression )
```

quartile

Returns the rank of a value, represented as integers from 1 (highest) to 4 (lowest), relative to a group of values.

**Syntax**

```
quartile ( numeric_expression [ tuple member_expression { , member_expression } ] within set set_expression )
```

rank

Returns the rank value of the selected data items. The type of ranking returned (Olympic, dense, or serial) is data source dependent. The sort order is optional; DESC is assumed by default.

**Syntax**

```
rank ( numeric_expression [ ASC|DESC ] [ tuple member_expression { , member_expression } ] within set set_expression )
```

**Example**

```
rank ( [gosales].[sales measures].[quantity] tuple [gosales].[Staff].[].[department] -> [West] within set children ( [gosales].[Staff].[].[Staff] ) )
```

standard-deviation

Returns the standard deviation of the selected data items.

**Syntax**

```
standard-deviation ( < currentMeasure|numeric_expression > within set set_expression )
standard-deviation ( < currentMeasure|numeric_expression > within < detail|aggregate > expression )
```

standard-deviation-pop

Returns the standard deviation population of the selected data items.

**Syntax**

```
standard-deviation-pop ( < currentMeasure|numeric_expression > within set set_expression )
standard-deviation-pop ( < currentMeasure|numeric_expression > within < detail|aggregate > expression )
```

total

Returns the total value of the selected data items.
Syntax

total ( < currentMeasure|numeric_expression > within set set_expression )
total ( < currentMeasure|numeric_expression > within < detail|aggregate > expression )

variance

Returns the variance of the selected data items.

Syntax

variance ( < currentMeasure|numeric_expression > within set set_expression )
variance ( < currentMeasure|numeric_expression > within < detail|aggregate > expression )

variance-pop

Returns the variance population of the selected data items.

Syntax

variance-pop ( < currentMeasure|numeric_expression> within set set_expression )
variance-pop ( < currentMeasure|numeric_expression > within < detail|aggregate > expression )

Constants

A constant is a fixed value that you can use in an expression.

date

Inserts the current system date.

date-time

Inserts the current system date and time.

time with time zone

Inserts a zero time with time zone.

timestamp with time zone

Inserts an example of a timestamp with time zone.

interval

Inserts a zero interval: 000 00:00:00.000.

interval year

Inserts a zero year interval: 0 year.

interval month

Inserts a zero month interval: 0 month.
interval year to month
   Inserts a zero year to month interval: 0000-00 year to month.

interval day
   Inserts a zero day interval: 0 day.

interval hour
   Inserts a zero hour interval: 0 hour.

interval minute
   Inserts a zero minute interval: 0 minute.

interval second
   Inserts a zero second interval: 0 second.

interval day to hour
   Inserts a zero day to hour interval: 0 00 day to hour.

interval day to minute
   Inserts a zero day to minute interval: 0 00:00 day to minute.

interval day to second
   Inserts a zero day to second interval: 0 00:00:00.000000000 day to second.

interval hour to minute
   Inserts a zero hour to minute interval: 00:00 hour to minute.

interval hour to second
   Inserts a zero hour to second interval: 00:00:00.000000000 hour to second.

interval minute to second
   Inserts a zero minute to second interval: 00:00.000000000 minute to second.

null
   Inserts "null" if the expression conditions are not met.

number
   Inserts the number 0, which can be replaced with a new numeric value.

string
   Inserts an empty string as two single quotation marks between which you can type a string.

time
   Inserts the current system time.
Constructs

This list contains constructs and templates that can be used to create an expression. Templates combine multiple functions into a group. For example, the search case template includes the case, when, else, and end functions.

if then else

This construct is the template for an if...then...else statement. This construct appears in the Top 10 Retailers for 2005 sample report in the GO Data Warehouse (analysis) package.

Syntax
IF ([Country] = 'Canada') THEN ([List Price] * 0.60) ELSE ([List Price])

in_range

This is the template for an in_range expression.

Syntax
[code] IN_RANGE { :30 , 40, 50, 999: }

Example 1
[code] IN_RANGE { 5 }
Result: This is equivalent to [code] = 5.

Example 2
[code] IN_RANGE { 5: }
Result: This is equivalent to [code] >= 5.

Example 3
[code] IN_RANGE { :5 }
Result: This is equivalent to [code] <= 5.

Example 4
[code] IN_RANGE { 5:10 }
Result: This is equivalent to ( [code] >= 5 and [code] <= 10 ).

Example 5
[code] IN_RANGE { :5,10,20: }
Result: This is equivalent to ( [code] <= 5 or [code] = 10 or [code] >= 20 ).

search case

This construct is the template for a search case, including the case, when, else, and end functions.

Syntax
CASE WHEN [Country] = 'Canada' THEN ([List Price] * 0.60) WHEN [CountryCode] > 100 THEN [List Price] * 0.80 ELSE [List Price] END
**simple case**

This construct is the template for a simple case, including the case, when, else, and end functions.

**Syntax**

```
CASE [Country] WHEN 'Canada' THEN ([List Price] * 0.60) WHEN 'Australia' THEN ([List Price] * 0.80) ELSE [List Price] END
```

**Business Date/Time Functions**

This list contains business functions for performing date and time calculations.

**_add_days**

Returns the date or datetime, depending on the format of "date_expression", that results from adding "integer_expression" days to "date_expression".

**Syntax**

```
_add_days ( date_expression, integer_expression )
```

**Example 1**

```
_add_days ( 2002-04-30 , 1 )
```

Result: 2002-05-01

**Example 2**

```
_add_days ( 2002-04-30 12:10:10.000, 1 )
```

Result: 2002-05-01 12:10:10.000

**Example 3**

```
_add_days ( 2002-04-30 00:00:00.000, 1/24 )
```

Note that the second argument is not a whole number. This is supported by some database technologies and increments the time portion.

Result: 2002-04-30 01:00:00.000

**_add_months**

Returns the date or datetime, depending on the format of "date_expression", that results from the addition of "integer_expression" months to "date_expression".

**Syntax**

```
_add_months ( date_expression, integer_expression )
```

**Example 1**

```
_add_months ( 2002-04-30 , 1 )
```

Result: 2002-05-30

**Example 2**

```
_add_months ( 2002-04-30 12:10:10.000, 1 )
```

Result: 2002-05-30 12:10:10.000
Appendix B: Using the Expression Editor

_add_years

Returns the date or datetime, depending on the format of "date_expression", that results from the addition of "integer_expression" years to "date_expression".

Syntax
_add_years ( date_expression, integer_expression )

Example 1
_add_years ( 2002-04-30 , 1 )
Result: 2003-04-30

Example 2
_add_years ( 2002-04-30 12:10:10.000 , 1 )
Result: 2003-04-30 12:10:10.000

_age

Returns a number that is obtained from subtracting "date_expression" from today's date. The returned value has the form YYYYMMDD, where YYYY represents the number of years, MM represents the number of months, and DD represents the number of days.

Syntax
_age ( date_expression )

Example
_age ( 1990-04-30 ) (if today's date is 2003-02-05)
Result: 120906, meaning 12 years, 9 months, and 6 days.

_day_of_week

Returns the day of week (1 to 7), where 1 is the first day of the week as indicated by the second parameter (1 to 7, 1 being Monday and 7 being Sunday). Note that in ISO 8601 standard, a week begins with Monday being day 1.

Syntax
_day_of_week ( date_expression, integer )

Example
_day_of_week ( 2003-01-01 , 1 )
Result: 3

_day_of_year

Returns the day of year (1 to 366) in "date_expression". Also known as Julian day.

Syntax
_day_of_year ( date_expression )
Example
_day_of_year ( 2003-03-01 )
Result: 61

days_between

Returns a positive or negative number representing the number of days between "date_expression1" and "date_expression2". If "date_expression1" < "date_expression2", then the result will be a negative number.

Syntax
days_between ( date_expression1 , date_expression2 )

Example
days_between ( 2002-04-30 , 2002-06-21 )
Result: -52

days_to_end_of_month

Returns a number representing the number of days remaining in the month represented by "date_expression".

Syntax
days_to_end_of_month ( date_expression )

Example
days_to_end_of_month ( 2002-04-20 14:30:22.123 )
Result: 10

first_of_month

Returns a date or datetime, depending on the argument, by converting "date_expression" to a date with the same year and month but with the day set to 1.

Syntax
first_of_month ( date_expression )

Example 1
first_of_month ( 2002-04-20 )
Result: 2002-04-01

Example 2
first_of_month ( 2002-04-20 12:10:10.000 )
Result: 2002-04-01 12:10:10.000

last_of_month

Returns a date or datetime, depending on the argument, that is the last day of the month represented by "date_expression".
Syntax
_last_of_month ( date_expression )

Example 1
_last_of_month ( 2002-01-14 )
Result: 2002-01-31

Example 2
_last_of_month ( 2002-01-14 12:10:10.000 )
Result: 2002-01-31 12:10:10.000

_make_timestamp
Returns a timestamp constructed from "integer_expression1" (the year), "integer_expression2" (the month), and "integer_expression3" (the day). The time portion defaults to 00:00:00.000.

Syntax
_make_timestamp ( integer_expression1, integer_expression2, integer_expression3 )

Example
_make_timestamp ( 2002, 01, 14 )
Result: 2002-01-14 00:00:00.000

_months_between
Returns a positive or negative integer number representing the number of months between "date_expression1" and "date_expression2". If "date_expression1" is earlier than "date_expression2", then a negative number is returned.

Syntax
_months_between ( date_expression1, date_expression2 )

Example
_months_between ( 2002-04-03, 2002-01-30 )
Result: 2

_week_of_year
Returns the number of the week of the year of "date_expression" according to the ISO 8601 standard. Week 1 of the year is the first week of the year to contain a Thursday, which is equivalent to the first week containing January 4th. A week starts on Monday (day 1) and ends on Sunday (day 7).

Syntax
_week_of_year ( date_expression )

Example
_week_of_year ( 2003-01-01 )
Result: 1

_years_between

Returns a positive or negative integer number representing the number of years between "date_expression1" and "date_expression2". If "date_expression1" < "date_expression2" then a negative value is returned.

Syntax
_years_between ( date_expression1, date_expression2 )

Example
_years_between ( 2003-01-30 , 2001-04-03 )
Result: 1

_ymdint_between

Returns a number representing the difference between "date_expression1" and "date_expression2". The returned value has the form YYYYMMDD, where YYYY represents the number of years, MM represents the number of months, and DD represents the number of days.

Syntax
_ymdint_between ( date_expression1 , date_expression2 )

Example
_ymdint_between ( 1990-04-30 , 2003-02-05 )
Result: 120906, meaning 12 years, 9 months and 6 days.

Block Functions

This list contains functions used to access members of a set, usually in the context of Analysis Studio.

_firstFromSet

Returns the first members found in the set up to "numeric_expression_maximum" + "numeric_expression_overflow". If "numeric_expression_maximum" + "numeric_expression_overflow" is exceeded, then only the maximum number of members are returned. For a set that has only a few members more than the specified numeric_expression_maximum, the numeric_expression_overflow allows the small set of extra members to be included. If the set has more members than the overflow allows, then only the numeric_expression_maximum members will be returned.

Syntax
_firstFromSet ( set_expression , numeric_expression_maximum , numeric_expression_overflow )

Example 1
_firstFromSet ( [great_outdoors_company].[Products].[Products].[Product line] , 2 , 8 )

Result: Returns the five members in the Product line set. The first two members are returned within the maximum and the following three members are returned as the overflow.
Appendix B: Using the Expression Editor

Camping Equipment
Golf Equipment
Mountaineering Equipment
Outdoor Protection
Personal Accessories

**Example 2**

```plaintext
_firstFromSet { [great_outdoors_company].[Products].[Products].[Product line] , 2 , 2 }
```

Result: Camping Equipment, Golf Equipment

**_remainderSet**

Returns the set containing "member_expression" when the size of "set_expression" is greater than "numeric_expression"; i.e., a new member will be generated if the number of members in "set_expression" is larger than the specified "numeric_expression".

**Syntax**

```plaintext
_remainderSet { member_expression, set_expression , numeric_expression }
```

**Example**

```plaintext
_remainderSet { member ( aggregate ( currentMeasure within set [great_outdoors_company].[Products].[Products].[Product line] ) , 'Product Aggregate' , 'Product Aggregate' , [great_outdoors_company].[Products].[Products]) , [great_outdoors_company].[Products].[Products].[Product line] , 1 )
```

Result: Quantity sold for Product Aggregate

**Macro Functions**

This list contains functions that can be used within a macro. A macro may contain one or more macro functions. A macro is delimited by a number sign (#) at the beginning and at the end. Everything between the number signs is treated as a macro expression and is executed at run time. For macro functions that accept expressions of datatype timestamp with time zone as arguments, the accepted format is 'yyyy-mm-dd hh:mm:ss[.ff]+hh:mm' where fractional seconds are optional and can be represented by 1 to 9 digits. In lieu of a space separating the date portion to the time portion, the character 'T' is also accepted. Also, in lieu of the time zone '+hh:mm', the character 'Z' is accepted and will be processed internally as '+00:00'. The macro functions that return expressions of datatype timestamp with time zone return 9 digits by default for their fractional seconds. The macro function timestampMask () can be used to trim the output if required.

**+**

Concatenates two strings.

**Syntax**

```plaintext
value1 + value2
```

**Example**

```plaintext
# '{ ' + $runLocale + ' }' #
```

Result: {en-us}
_add_days

Returns the timestamp with time zone (as a string) that results from adding "integer_expression" number of days to "string_expression", where "string_expression" represents a timestamp with time zone.

Syntax

_add_days ( string_expression , integer_expression )

Example 1

# _add_days ( '2005-11-01 12:00:00.000-05:00' , -1 ) #
Result: 2005-10-31 12:00:00.000000000-05:00

Example 2

# _add_days ( $current_timestamp , 1 ) #
Result: 2005-11-02 12:00:00.000000000-05:00

Example 3

# timestampMask ( _add_days ( $current_timestamp , 1 ) , 'yyyy-mm-dd' ) #
Result: 2005-11-02

_add_months

Returns the timestamp with time zone (as a string) that results from adding "integer_expression" number of months to "string_expression", where "string_expression" represents a timestamp with time zone.

Syntax

_add_months ( string_expression , integer_expression )

Example 1

# _add_months ( '2005-11-01 12:00:00.000-05:00' , -1 ) #
Result: 2005-10-01 12:00:00.000000000-05:00

Example 2

# _add_months ( $current_timestamp , 1 ) #
Result: 2005-12-01 12:00:00.000000000-05:00

Example 3

# timestampMask ( _add_months ( $current_timestamp , 1 ) , 'yyyy-mm-dd' ) #
Result: 2005-12-01

_add_years

Returns the timestamp with time zone (as a string) that results from adding "integer_expression" number of years to "string_expression", where "string_expression" represents a timestamp with time zone.
Appendix B: Using the Expression Editor

Syntax

_add_years ( string_expression , integer_expression )

Example 1

# _add_years ( '2005-11-01 12:00:00.000-05:00' , -1 ) #

Result: 2004-11-01 12:00:00.000000000-05:00

Example 2

# _add_years ( $current_timestamp , 1 ) #

Result: 2006-11-01 12:00:00.000000000-05:00

Example 3

# timestampMask ( _add_years ( $current_timestamp , 1 ) , 'yyyy-mm-dd' ) #

Result: 2006-11-01

array

Constructs an array out of the list of parameters.

Syntax

array ( string_expression|array_expression { , string_expression|array_expression } )

Example

# csv ( array ( 'a1' , array ( 'x1' , 'x2' ) , 'a2' ) ) #

Result: 'a1', 'x1', 'x2', 'a2'

csv

Constructs a string from the elements of the array where the values are separated by commas. Optionally, the separator and quote strings can be specified. The default separator is a comma (,) and the default quote character is a single quote (').

Syntax

csv ( array_expression [ , separator_string [ , quote_string ] ] )

Example

# csv ( array ( 'a1' , 'a2' ) ) #

Result: 'a1', 'a2'

dq

Surrounds "string_expression" with double quotes.

Syntax

dq ( string_expression )

Example

# dq ( 'zero' ) #
Result: "zero"

**getConfigurationEntry**

Get an entry from the IBM Cognos configuration file. The `force_decode_flag` is optional and must be one of: 'true', '1', 'false', '0', 0. The default is 'true'. When true, the value of the configuration entry will be decrypted into plain text if it is encrypted.

**Syntax**

```java
getConfigurationEntry ( entry_string , force_decode_flag )
```

**Example**

```java
# getConfigurationEntry ( 'serverLocale' ) #
Result: en
```

**grep**

Searches for and returns elements of an array that match the pattern specified in "pattern_string".

**Syntax**

```java
grep ( pattern_string , array_expression )
```

**Example**

```java
# csv ( grep ( 's' , array ( 'as', 'an', 'arts' ) ) ) #
Result: 'as', 'arts'
```

**_first_of_month**

Returns a timestamp with time zone (as a string) by converting the day value in "string_expression" to 1, where "string_expression" is a timestamp with time zone.

**Syntax**

```java
_first_of_month ( string_expression )
```

**Example 1**

```java
# _first_of_month ( '2005-11-11 12:00:00.000-05:00' ) #
Result: 2005-11-01 12:00:00.000000000-05:00
```

**Example 2**

```java
# timestampMask ( _first_of_month ( '2005-11-11 12:00:00.000-05:00' ) , 'yyyymmdd' ) #
Result: 20051101
```

**_last_of_month**

Returns a timestamp with time zone (as a string) that is the last day of the month represented by "string_expression", where "string_expression" is a timestamp with time zone.

**Syntax**

```java
_last_of_month ( string_expression )
```
Example 1

```sh
# _last_of_month ( '2005-11-11 12:00:00.000-05:00' ) #
```
Result: 2005-11-30 12:00:00.000000000-05:00

Example 2

```sh
# timestampMask ( _last_of_month ( '2005-11-11 12:00:00.000-05:00' ) , 'yyyy-mm-dd' ) #
```
Result: 2005-11-30

**join**

Joins the elements of an array using "separator_string".

**Syntax**

`join ( separator_string , array_expression )`

**Example**

```sh
# sq ( join ( ' | | ' , array ( 'as', 'an', 'arts' ) ) ) #
```
Result: 'as | | an | | arts'

**prompt**

Prompts the user for a single value. Only "prompt_name" is required. The datatype defaults to "string" when it is not specified. The prompt is optional when "defaultText" is specified. "Text", when specified, will precede the value. "QueryItem" can be specified to take advantage of the prompt information properties of "queryItem". "Trailing_text", when specified, will be appended to the value.

**Syntax**

`prompt ( prompt_name , datatype , defaultText , text , queryItem , trailing_text )`

**Example 1**

```sh
select . . . where COUNTRY_MULTILINGUAL.COUNTRY_CODE > #prompt ( 'Starting CountryCode' , 'integer' , '10' ) #
```
Result: select . . . where COUNTRY_MULTILINGUAL.COUNTRY_CODE > 10

**Example 2**

```sh
[gosales].[COUNTRY].[COUNTRY] = # prompt ( 'countryPrompt' , 'string' , '''Canada''' ) #
```
Result: [gosales].[COUNTRY].[COUNTRY] = 'Canada'

**Notes**

- The "defaultText" parameter must be specified such that it is literally valid in the context of the macro since no formatting takes place on this value. The default string "'Canada'" in Example 2 is specified as a string using single quotes, in which the embedded single quotes are doubled up, thus 3 quotes. This results in the string being properly displayed within single quotes in the expression. As a general rule for the string datatype, "defaultText" should always
be specified like this, except in the context of a stored procedure parameter. For "defaultText" of types 'date' or 'datetime', a special format should be used in the context of SQL. Examples of these formats are 'DATE "2001-12-25"' and 'DATETIME "2001-12-25 12:00:00"'. In all other contexts, use the date/datetime without the keyword and escaped single quotes (e.g., '2001-12-25').

**promptmany**

Prompts the user for one or more values. Only "prompt_name" is required. The datatype defaults to string when it is not specified. The prompt is optional when "defaultText" is specified. "Text", when specified, will precede the list of values. "QueryItem" can be specified to take advantage of the prompt information properties of "queryItem". "Trailing_text", when specified, will be appended to the list of values.

**Syntax**

```
promptmany ( prompt_name , datatype , defaultText , text , queryItem , trailing_text )
```

**Example 1**

```
select . . . where COUNTRY_MULTILINGUAL.COUNTRY in ( # promptmany
   ( 'CountryName' ) # )
```

Result: select . . . where COUNTRY_MULTILINGUAL.COUNTRY_CODE in ( 'Canada' , 'The Netherlands' , 'Russia' )

**Example 2**

```
select . . . from gosales.gosales.dbo.COUNTRY_MULTILINGUAL COUNTRY_MULTILINGUAL,
gosales.gosales.dbo.COUNTRYXX where COUNTRY_MULTILINGUAL.COUNTRY_CODE = XX.COUNTRY_CODE # promptmany ( 'Selected CountryCodes' , 'integer' , ',' , ',' , ' ) #
```

Result: select . . . from gosales.gosales.dbo.COUNTRY_MULTILINGUAL COUNTRY_MULTILINGUAL,
gosales.gosales.dbo.COUNTRYXX where COUNTRY_MULTILINGUAL.COUNTRY_CODE = XX.COUNTRY_CODE and COUNTRY_MULTILINGUAL.COUNTRY_CODE in ( 'Canada' , 'The Netherlands' , 'Russia' )

**sb**

Surrounds "string_expression" with square brackets.

**Syntax**

```
sb ( string_expression )
```

**Example**

```
# sb ( 'abc' ) #
```

Result: [abc]

**sq**

Surrounds "string_expression" with single quotes.
Appendix B: Using the Expression Editor

Syntax

sq ( string_expression )

Example

# sq ( 'zero' ) #
Result: 'zero'

sort

Sorts the elements of the array in alphabetical order. Duplicates are retained.

Syntax

sort ( array_expression )

Example

# csv ( sort ( array ( 's3', 'a', 'x' ) ) ) #
Result: 'a', 's3', 'x'

split

Splits a string or string elements of the array into separate elements.

Syntax

split ( pattern_string, string_expression|array_expression )

Example 1

# csv ( split ( '::', 'ab=c::de=f::gh=i' ) ) #
Result: 'ab=c', 'de=f', 'gh=i'

Example 2

# csv ( split ( '=' , split ( '::', 'ab=c::de=f::gh=i' ) ) ) #
Result: 'ab', 'c', 'de', 'f', 'gh', 'i'

substitute

Searches for a pattern in a string or in the string elements of an array and substitutes the first occurrence of "pattern_string" with "replacement_string".

Syntax

substitute ( pattern_string, replacement_string, string_expression|array_expression )

Example 1

#sq ( substitute ( '^cn=', '***', 'cn=help' ) )#
Result: '***help'

Example 2

# csv ( substitute ( '^cn=', '***', array ( 'cn=help', 'acn=5' ) ) ) #
Result: '***help', 'acn=5'

**Example 3**

```
# csv ( substitute ( 'cn=', '', array ( 'cn=help' , 'acn=5' ) ) ) #
```

Result: 'help', 'a5'

**timestampMask**

Returns "string_expression1", representing a timestamp with time zone, trimmed to the format specified in "string_expression2". The format in "string_expression2" must be one of the following: 'yyyy', 'mm', 'dd', 'yyyy-mm', 'yyyyym', 'yyyy-mm-dd', 'yyyyymdd', 'yyyy-mm-dd hh:mm:ss', 'yyyy-mm-dd hh:mm:ss+hh:mm', 'yyyy-mm-dd hh:mm:ss+hh:mm', 'yyyy-mm-ddThh:mm:ss', 'yyyy-mm-ddThh:mm:ss+hh:mm', or 'yyyy-mm-ddThh:mm:ss+hh:mm'. The macro functions that return a string representation of a timestamp with time zone show a precision of 9 digits for the fractional part of the seconds by default. The format options allow this to be trimmed down to a precision of 3 or 0.

**Syntax**

```
timestampMask ( string_expression1 , string_expression2 )
```

**Example 1**

```
# timestampMask ( $current_timestamp , 'yyyy-dd-mm' ) #
```

Result: 2005-11-01

**Example 2**

```
# timestampMask ( '2005-11-01 12:00:00.000-05:00' , 'yyyy-mm-dd hh:mm:ss+hh:mm' ) #
```

Result: 2005-11-01 12:00:00-05:00

**Example 3**

```
# timestampMask ( '2005-11-01 12:00:00.123456789-05:00' , 'yyyy-mm-ddThh:mm:ss+hh:mm.ff3+hh:mm' ) #
```

Result: 2005-11-01T12:00:00.123-05:00

**toLocal**

Returns the string representing a timestamp with time zone resulting from adjusting "string_expression" to the time zone of the operating system. Note that the macro function timestampMask () can be used to trim the output.

**Syntax**

```
toLocal ( string_expression )
```

**Example 1**

```
# toLocal ( '2005-11-01 17:00:00.000-00:00' ) # where OS local time zone is -05:00
```

Result: 2005-11-01 12:00:00.000000000-05:00

---

User Guide 109
**Example 2**

```plaintext
# timestampMask ( toLocal ( '2005-11-01 17:00:00.000-00:00' ), 'yyyy-mm-dd hh:mm:ss+hh:mm' ) # where OS local time zone is -05:00

Result: 2005-11-01 12:00:00-05:00
```

**Example 3**

```plaintext
# toLocal ( '2005-11-01 13:30:00.000-03:30' ) # where OS local time zone is -05:00

Result: 2005-11-01 12:00:00.000000000-05:00
```

**toUTC**

Returns the string representing a timestamp with time zone resulting from adjusting "string_expression" to the zero-point reference UTC time zone, also known as GMT time. Note that the macro function timestampMask() can be used to trim the output.

**Syntax**

toUTC ( string_expression )

**Example 1**

```plaintext
# toUTC ( '2005-11-01 12:00:00.000-05:00' )#

Result: 2005-11-01 17:00:00.000000000-00:00
```

**Example 2**

```plaintext
# timestampMask ( toUTC ( '2005-11-01 12:00:00.000-00:00' ), 'yyyy-mm-dd hh:mm:ss+hh:mm' )#

Result: 2005-11-01 17:00:00.000-00:00
```

**Example 3**

```plaintext
# toUTC ( $current_timestamp )#

Result: 2005-11-01 17:00:00.000000000-00:00
```

**unique**

Removes duplicate entries from the array. The order of the elements is retained.

**Syntax**

unique ( array_expression )

**Example**

```plaintext
# csv ( unique ( array ( 's3', 'a', 's3', 'x' ) ) )#

Result: 's3', 'a', 'x'
```

**urlencode**

URL-encodes the passed argument. This function is useful when specifying XML connection strings.

**Syntax**

urlencode ( prompt ( 'userValue' ) )
Example
urlencode ( prompt ( 'some_val' ) )
Result: %27testValue%27

CSVIdentityName
Uses the identity information of the current authenticated user to look up values in the specified parameter map. Each individual piece of the user’s identity (account name, group names, role names) is used as a key into the map. The unique list of values that is retrieved from the parameter map is then returned as a string, where each value is surrounded by single quotes and where multiple values are separated by commas.

Syntax
CSVIdentityName ( %parameter_map_name [ , separator_string ] )

Example
# CSVIdentityName ( %security_clearance_level_map ) #
Result: 'level_500', 'level_501', 'level_700'

CSVIdentityNameList
Returns the pieces of the user’s identity (account name, group names, role names) as a list of strings. The unique list of values is returned as a string, where each value is surrounded by single quotes and where multiple values are separated by commas.

Syntax
CSVIdentityNameList ( [ separator_string ] )

Example
# CSVIdentityNameList ( ) #
Result: 'Everyone', 'Report Administrators', 'Query User'

CAMPassport
Returns the Cognos Access Manager passport.

Syntax
CAMPassport ( )

Example
# CAMPassport ( ) #
Result: 111:98812d62-4fd4-037b-4354-26414cf7ebef:3677162321

CAMIDList
Returns the pieces of the user’s Cognos Access Manager ID (CAMID), such as account name, group names, or role names, as a list of values separated by commas.
Appendix B: Using the Expression Editor

**Syntax**

CAMIDList ( [ separator_string ] )

**Example**

```text
#CAMIDList ( ) #
```

Result: CAMID ( "::Everyone" ) , CAMID ( "::Authors" ) , CAMID ( "::Query Users" ) , CAMID ( "::Consumers" ) , CAMID ( "::Metrics Authors" )

**CAMIDListForType**

Returns an array of the user's Cognos Access Manager IDs (CAMIDs) based on the identity type (account, group, or role). CAMIDListForType can be used with the macro functions csv or join.

**Syntax**

CAMIDListForType ( identity type )

**Example**

```text
[qs].[UserRole] in ( # csv ( CAMIDListForType ( 'role' ) ) # )
```

Result: [qs].[UserRole] in ( 'CAMID ( "::System Administrators" )' , 'CAMID ( "::Authors" )' )

**Common Functions**

**abs**

Returns the absolute value of "numeric_expression". Negative values are returned as positive values.

**Syntax**

```text
abs ( numeric_expression )
```

**Example 1**

```text
abs ( 15 )
```

Result: 15

**Example 2**

```text
abs ( -15 )
```

Result: 15

**cast**

Converts "expression" to a specified data type. Some data types allow for a length and precision to be specified. Make sure that the target is of the appropriate type and size. The following can be used for "datatype_specification": character, varchar, char, numeric, decimal, integer, smallint, real, float, date, time, timestamp, time with time zone, timestamp with time zone, and interval. When type casting to an interval type, one of the following interval qualifiers must be specified: year, month, or year to month for the year-to-month interval datatype; day, hour, minute, second, day to hour, day to minute, day to second, hour to minute, hour to second, or minute to second for the day-to-second interval datatype. Notes: When you convert a value of type timestamp to type date, the time portion of the timestamp value is ignored. When you convert a value of type
timestamp to type time, the date portion of the timestamp is ignored. When you convert a value of
type date to type timestamp, the time components of the timestamp are set to zero. When you
convert a value of type time to type timestamp, the date component is set to the current system
date. It is invalid to convert one interval datatype to the other (for instance because the number of
days in a month is variable). Note that you can specify the number of digits for the leading qualifier
only, i.e. YEAR(4) TO MONTH, DAY(5). Errors will be reported if the target type and size are
not compatible with the source type and size.

**Syntax**

cast ( expression , datatype_specification )

**Example 1**

cast ( '123' , integer )

Result: 123

**Example 2**

cast ( 12345 , varchar ( 10 ) )

Result: a string containing 12345

**ceil**

Returns the smallest integer that is greater than or equal to "numeric_expression".

**Syntax**

ceil ( numeric_expression )

**ceiling**

Returns the smallest integer that is greater than or equal to "numeric_expression".

**Syntax**

ceiling ( numeric_expression )

**Example 1**

ceiling ( 4.22 )

Result: 5

**Example 2**

ceiling ( -1.23 )

Result: -1

**char_length**

Returns the number of logical characters in "string_expression". The number of logical characters
can be distinct from the number of bytes in some East Asian locales.

**Syntax**

char_length ( string_expression )
Appendix B: Using the Expression Editor

**Example**
```
char_length ( 'Canada' )
```
Result: 6

**character_length**

Returns the number of characters in "string_expression".

**Syntax**
```
character_length ( string_expression )
```

**Example**
```
character_length ( 'Canada' )
```
Result: 6

**coalesce**

Returns the first non-null argument (or null if all arguments are null). Requires two or more arguments in "expression_list".

**Syntax**
```
coalesce ( expression_list )
```

**Example**
```
coalesce ( [Unit price], [Unit sale price] )
```
Result: Returns the unit price, or the unit sale price if the unit price is null.

**current_date**

Returns a date value representing the current date of the computer that the database software runs on.

**Syntax**
```
current_date
```

**Example**
```
current_date
```
Result: 2003-03-04

**current_time**

Returns a time with time zone value, representing the current time of the computer that runs the database software if the database supports this function. Otherwise, it represents the current time of the computer that runs IBM Cognos BI software.

**Syntax**
```
current_time
```
**Example**

current_time

Result: 16:33:11+05:00

**current_timestamp**

Returns a datetime with time zone value, representing the current time of the computer that runs the database software if the database supports this function. Otherwise, it represents the current time of the computer that runs IBM Cognos BI software.

**Syntax**

current_timestamp

**Example**

current_timestamp

Result: 2003-03-03 16:40:15.535000+05:00

**exp**

Returns 'e' raised to the power of "numeric_expression". The constant 'e' is the base of the natural logarithm.

**Syntax**

exp ( numeric_expression )

**Example**

exp ( 2 )

Result: 7.389056

**extract**

Returns an integer representing the value of datepart (year, month, day, hour, minute, second) in "datetime_expression".

**Syntax**

extract ( datepart , datetime_expression )

**Example 1**

extract ( year , 2003-03-03 16:40:15.535 )

Result: 2003

**Example 2**

extract ( hour , 2003-03-03 16:40:15.535 )

Result: 16

**floor**

Returns the largest integer that is less than or equal to "numeric_expression".
Appendix B: Using the Expression Editor

**Syntax**

floor ( numeric_expression )

**Example 1**

floor ( 3.22 )

Result: 3

**Example 2**

floor ( -1.23 )

Result: -2

**ln**

Returns the natural logarithm of "numeric_expression".

**Syntax**

ln ( numeric_expression )

**Example**

ln ( 4 )

Result: 1.38629

**localtime**

Returns a time value, representing the current time of the computer that runs the database software.

**Syntax**

localtime

**Example**

localtime

Result: 16:33:11

**localtimestamp**

Returns a datetime value, representing the current timestamp of the computer that runs the database software.

**Syntax**

localtimestamp

**Example**

localtimestamp

Result: 2003-03-03 16:40:15.535000

**lower**

Returns "string_expression" with all uppercase characters shifted to lowercase.
Syntax
lower ( string_expression )

Example
lower ( 'ABCDEF' )
Result: abcdef

mod

Returns the remainder (modulus) of "integer_expression1" divided by "integer_expression2". "Integer_expression2" must not be zero or an exception condition is raised.

Syntax
mod ( integer_expression1, integer_expression2 )

Example
mod ( 20 , 3 )
Result: 2

nullif

Returns null if "expression1" equals "expression2", otherwise returns "expression1".

Syntax
nullif ( expression1, expression2 )

octet_length

Returns the number of bytes in "string_expression".

Syntax
octet_length ( string_expression )

Example 1
octet_length ( 'ABCDEF' )
Result: 6

Example 2
octet_length ( '' )
Result: 0

position

Returns the integer value representing the starting position of "string_expression1" in "string_expression2" or 0 when the "string_expression1" is not found.

Syntax
position ( string_expression1 , string_expression2 )
Example 1
position ( 'C' , 'ABCDEF' )
Result: 3

Example 2
position ( 'H' , 'ABCDEF' )
Result: 0

power

Returns "numeric_expression1" raised to the power "numeric_expression2". If "numeric_expression1" is negative, then "numeric_expression2" must result in an integer value.

Syntax
power ( numeric_expression1 , numeric_expression2 )

Example
power ( 3 , 2 )
Result: 9

_round

Returns "numeric_expression" rounded to "integer_expression" places to the right of the decimal point. Notes: "integer_expression" must be a non-negative integer. Rounding takes place before data formatting is applied.

Syntax
_round ( numeric_expression , integer_expression )

Example
_round ( 1220.42369, 2 )
Result: 1220.42

sqrt

Returns the square root of "numeric_expression". "Numeric_expression" must be non-negative.

Syntax
sqrt ( numeric_expression )

Example
sqrt ( 9 )
Result: 3

substring

Returns the substring of "string_expression" that starts at position "integer_expression1" for "integer_expression2" characters or to the end of "string_expression" if "integer_expression2" is omitted. The first character in "string_expression" is at position 1.
Syntax
substring ( string_expression , integer_expression1 [ , integer_expression2 ] )

Example
substring ( 'abcdefg' , 3 , 2 )
Result: cd

trim

Returns "string_expression" trimmed of leading and trailing blanks or trimmed of a certain character specified in "match_character_expression". "Both" is implicit when the first argument is not stated and blank is implicit when the second argument is not stated.

Syntax
trim ( [ [ trailing|leading|both ] [ match_character_expression ] , ] string_expression )

Example 1
trim ( trailing 'A' , 'ABCDEFA' )
Result: ABCDEF

Example 2
trim ( both , ' ABCDEF ' )
Result: ABCDEF

upper

Returns "string_expression" with all lowercase characters converted to uppercase.

Syntax
upper ( string_expression )

Example
upper ( 'abcdef' )
Result: ABCDEF

Dimensional Functions

ancestor

Returns the ancestor of "member" at "level" or at "integer" number of levels above "member". Note: The result is not guaranteed to be consistent when there is more than one such ancestor.

Syntax
ancestor ( member, level|integer )

Example 1
ancestor ( [TrailChef Water Bag] , 1 )
Result: Cooking Gear
Example 2

ancestor ( [TrailChef Water Bag] , 2 )

Result: Camping Equipment

Example 3

ancestor ( [TrailChef Water Bag] , [great_outdoors_company].[Products].[Products].[Product type] )

Result: Cooking Gear

ancestors

Returns all the ancestors of "member" at "level" or "index" distance above the member. (Most data sources support only one ancestor at a specified level. If the data source supports more than one ancestor, the result is a member set.)

Syntax

ancestors ( member , level|index )

Example 1

ancestors ( [TrailChef Water Bag] , 1 )

Result: Cooking Gear

Example 2

ancestors ( [TrailChef Water Bag] , 2 )

Result: Camping Equipment

Example 3

ancestors ( [TrailChef Water Bag] , [great_outdoors_company].[Products].[Products].[Product type] )

Result: Cooking Gear

bottomCount

Sorts a set according to the value of "numeric_expression" evaluated at each of the members of "set_expression" and returns the bottom "index_expression" members.

Syntax

countbottom ( set_expression , index_expression , numeric_expression )

Example

bottomCount ( [great_outdoors_company].[Products].[Products].[Product line] , 2 , [Revenue] )

Result: Returns the bottom two members of the set sorted by revenue.
<table>
<thead>
<tr>
<th>Product line</th>
<th>Revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outdoor Protection</td>
<td>$3,171,114.92</td>
</tr>
<tr>
<td>Mountaineering Equipment</td>
<td>$20,891,350.60</td>
</tr>
</tbody>
</table>

**bottomPercent**

Sorts "numeric_expression2", evaluated at the corresponding members of "set_expression", and picks up the bottommost elements whose cumulative total is equal to or less than "numeric_expression1" percent of the total.

**Syntax**

```
bottomPercent ( set_expression , numeric_expression1 , numeric_expression2 )
```

**Example**

```
```

Result: For the set of Camping Equipment, Golf Equipment, and Mountaineering Equipment, returns the members whose percentage total are less than or equal to 40% for 2006.

**bottomSum**

Sorts "numeric_expression2", evaluated at the corresponding member of "set_expression", and picks up the bottommost elements whose cumulative total is equal to or less than "numeric_expression1".

**Syntax**

```
bottomSum ( set_expression , numeric_expression1 , numeric_expression2 )
```

**Example**

```
bottomSum ( members ( [great_outdoors_company].[Products].[Products].[Product line] ) , 6000000 , tuple ( [2006] , [great_outdoors_company].[Measures].[Gross profit] ) )
```

**caption**

Returns the caption values of "level", "member", or "set_expression". The caption is the string display name for an element and does not necessarily match the unique identifier used to generate the business key or member unique name (MUN) for the element. The caption is not necessarily unique; for example, the caption for a month may return the month name without further year details to make the value unique.

**Syntax**

```
caption ( level|member|set_expression )
```

**Example 1**

```
caption ( [TrailChef Water Bag] )
```
Appendix B: Using the Expression Editor

Result: TrailChef Water Bag

**Example 2**

```
caption ( [great_outdoors_company].[Products].[Products].[Product line] )
```

Result: Returns the caption values of the Product line set.

- Camping Equipment
- Mountaineering Equipment
- Personal Accessories
- Outdoor Protection
- Golf Equipment

**children**

Returns the set of children of a specified member.

**Syntax**

```
children ( member )
```

**Example**

```
children ( [Camping Equipment] )
```

Result: Returns the set of children for Camping Equipment.

- Cooking Gear
- Tents
- Sleeping Bags
- Packs
- Lanterns

**closingPeriod**

Returns the last sibling member among the descendants of a member at "level". This function is typically used with a time dimension.

**Syntax**

```
closingPeriod ( level [ , member ] )
```

**Example 1**

```
closingPeriod ( [great_outdoors_company].[Years].[Years].[Month] )
```

Result: 2006/Dec

**Example 2**

```
closingPeriod ( [great_outdoors_company].[Years].[Years].[Year] )
```

Result: 2006

**Example 3**

```
closingPeriod ( [great_outdoors_company].[Years].[Years].[Month] , [2006 Q 4] )
```

Result: 2006/Dec
cousin

Returns the child member of "member2" with the same relative position as "member1" to its parent. This function appears in the Revenue by GO Subsidiary 2005 sample report in the GO Data Warehouse (analysis) package.

Syntax
cousin ( member1 , member2 )

Example 1
cousin ( [Irons] , [Camping Equipment] )
Result: Cooking Gear

Example 2
cousin ( [Putters] , [Camping Equipment] )
Result: Sleeping Bags

completeTuple

Identifies a cell location (intersection) based on the specified members, each of which must be from a different dimension. However, completeTuple () implicitly includes the default member from all dimensions not otherwise specified in the arguments, rather than the current member. CompleteTuple will use the default measure rather than the currentMeasure in the query if the measure is not defined in the completetuple function. This function appears in the Planned Headcount sample report in the GO Data Warehouse (analysis) package.

Syntax
completeTuple ( member { , member } )

Example 1
completeTuple ( [Mountaineering Equipment] , [Fax] )
Result: The completeTuple does not pick up the currentMember by default as the tuple function does. The values in the first column are identical across each year because the default member of the Years dimension, the root member, is used rather than the current member. Likewise, the first column displays Revenue rather than Quantity Sold because the Revenue measure is the default from the Measures dimension. CompleteTuple will use the default measure rather than the currentMeasure in the query if the measure is not defined in the completetuple function.

<table>
<thead>
<tr>
<th>Quantity Sold</th>
<th>Mountaineering Sales by Fax</th>
</tr>
</thead>
<tbody>
<tr>
<td>2004</td>
<td>$1,220,329.38</td>
</tr>
<tr>
<td>2005</td>
<td>$1,220,329.38</td>
</tr>
<tr>
<td>2006</td>
<td>$1,220,329.38</td>
</tr>
</tbody>
</table>
Example 2

completeTuple ( [Mountaineering Equipment], [Fax], [Quantity sold], currentMember ( [great_outdoors_company].[Years].[Years] ) )

Result: The completeTuple function uses the currentMember of the Years dimension and the Quantity sold measure.

<table>
<thead>
<tr>
<th>Year</th>
<th>Quantity Sold</th>
</tr>
</thead>
<tbody>
<tr>
<td>2004</td>
<td>0</td>
</tr>
<tr>
<td>2005</td>
<td>8,746</td>
</tr>
<tr>
<td>2006</td>
<td>7,860</td>
</tr>
</tbody>
</table>

currentMember

Returns the current member of the hierarchy during an iteration. If "hierarchy" is not present in the context in which the expression is being evaluated, its default member is assumed. This function appears in the Rolling and Moving Averages interactive sample report.

Syntax

currentMember ( hierarchy )

defaultMember

Returns the default member of "hierarchy".

Syntax

defaultMember ( hierarchy )

Example 1

defaultMember ( [great_outdoors_company].[Products].[Products] )

Result: Products

Example 2

defaultMember ( [great_outdoors_company].[Years].[Years] )

Result: Year

Example 3

defaultMember ( hierarchy ( [great_outdoors_company].[Measures].[Quantity sold] ) )

Result: Revenue

descendants

Returns the set of descendants of "member" or "set_expression" at "level" (qualified name) or "distance" (integer 0..n) from the root. Multiple options may be specified (separated by a space) to determine which members are returned. self: Only the members at the specified level are included in the final set (this is the default behaviour in the absence of any options). before: If there are any...
intermediate levels between the member's level and the one specified, members from those levels are included. If the level specified is the same as the member upon which the function is applied, the member is included in the final set. beforewithmember: If there are any intermediate levels between the member's level and the one specified, members from those levels are included. The member upon which the function is applied is also included in the final set. after: If other levels exist after the specified level, members from those levels are included in the final set. This function appears in the Sales Commissions for Central Europe sample report in the GO Data Warehouse (analysis) package.

**Syntax**

descendants ( member|set_expression , level|distance [ , { self|before|beforewithmember|after } ] )

**Example 1**

descendants ( [great_outdoors_company].[Products].[Products].[Products] , [great_outdoors_company].[Products].[Products].[Product type] )

Result: Returns the set of descendants of the Products set at the Product type level. Note: [great_outdoors_company].[Products].[Products].[Products] is the root member of the Products hierarchy.

Cooking Gear
Sleeping Bags
Packs
Tents
...
Eyewear
Knives
Watches

**Example 2**

descendants ( [great_outdoors_company].[Products].[Products].[Products] , 1 )

Result: Returns the set of descendants of the Products set at the first level.

Camping Equipment
Golf Equipment
Mountaineering Equipment
Outdoor Protection
Personal Accessories

**Example 3**

descendants ( [great_outdoors_company].[Products].[Products].[Products] , 3 , before )

Result: Returns the descendants of the Products set before the third level.

Camping Equipment
Cooking Gear
Sleeping Bags
Packs
Tents
...
Eyewear
Knives
Watches

**Example 4**

descendants ( [great_outdoors_company].[Products].[Products].[Products] , 2 , self before )
Appendix B: Using the Expression Editor

Result: Returns the set of descendants of the Products set before and including the second level.

Camping Equipment
Cooking Gear
Sleeping Bags
Packs
Tents
...
Eyewear
Knives
Watches

except

Returns the members of "set_expression1" that are not also in "set_expression2". Duplicates are retained only if the optional keyword all is supplied as the third argument.

Syntax

except ( set_expression1 , set_expression2 [ , all ] )

Example


Result: Mountaineering Equipment

filter

Returns the set resulting from filtering a specified set based on the Boolean condition. Each member is included in the result if and only if the corresponding value of "Boolean_expression" is true.

Syntax

filter ( set_expression , Boolean_expression )

Example

filter ( [Product line] , [Gross margin] > .30 )

Result: Mountaineering Equipment

firstChild

Returns the first child of "member".

Syntax

firstChild ( member )

Example 1

firstChild ( [By Product Lines] )

Result: Camping Equipment

Example 2

firstChild ( [Camping Equipment] )

Result: Cooking Gear
firstSibling

Returns the first child of the parent of "member".

Syntax

firstSibling ( member )

Example 1

firstSibling ( [Outdoor Protection] )

Result: Camping Equipment

Example 2

firstSibling ( [Camping Equipment] )

Result: Camping Equipment

_format

Associates a format with the expression. The format_keyword can be PERCENTAGE_0, PERCENTAGE_1, or PERCENTAGE_2. PERCENTAGE_1 returns a percentage with one digit to the right of the decimal point, PERCENTAGE_2 returns a percentage with two digits to the right of the decimal point, and PERCENTAGE_3 returns a percentage value out of one with three digits to the right of the decimal point (for example, 0.965).

Syntax

_format ( expression , format_keyword )

Example

_format ( [Unit Sale Price] / [Unit Price] , PERCENTAGE_2 )

Result: 75.12%

emptySet

Returns an empty member set for "hierarchy". This is most often used as a placeholder during development or with dynamic report design (either with the IBM Cognos Software Development Kit or via report design). By creating a data item that contains the emptyset function, it is possible to build complex expressions that can later be revised by redefining the emptyset data item.

Syntax

emptySet ( hierarchy )

Example

except ( [great_outdoors_company].[Products].[Products].[Product line] , emptyset ( [great_outdoors_company].[Products].[Products] ) )

Result: Returns the Product line set and an empty set for the Products set.

Camping Equipment
Golf Equipment
Mountaineering Equipment
Outdoor Protection
Personal Accessories
generate

Evaluates "set_expression2" for each member of "set_expression1" and joins the resulting sets by union. The result retains duplicates only when the optional keyword "all" is supplied as the third argument.

Syntax

generate ( set_expression1 , set_expression2 [ , all ] )

Example

generate ( [Product line] , topCount ( descendants ( currentMember ( [great_outdoors_company].[Products].[Products] ) , [great_outdoors_company].[Products].[Products].[Product name] ) , 2 , [Revenue] ) )

Result: Returns the top two products by revenue for each product line.

head

Returns the first "index_expression" elements of "set_expression". The default for "index_expression" is 1.

Syntax

head ( set_expression [ , index_expression ] )

Example 1

head ( members ( [great_outdoors_company].[Products].[Products].[Product line] ) )

Result: Camping Equipment

Example 2

head ( members ( [great_outdoors_company].[Products].[Products].[Product line] ) , 2 )

Result: Returns the top two members of the Product line set.

Camping Equipment
Mountaineering Equipment

hierarchize

Orders the members of "set_expression" in a hierarchy. Members in a level are sorted in their natural order. This is the default ordering of the members along a dimension when no other sort conditions are specified.

Syntax

hierarchize ( set_expression )

Example


Result: Returns Camping Equipment, Golf Equipment, Mountaineering Equipment.
**hierarchy**

Returns the hierarchy that contains "level", "member", or "set_expression".

**Syntax**

```
hierarchy ( level|member|set_expression )
```

**Example 1**

```
hierarchy ( [Cooking Gear] )
```

Result: Returns every member in the hierarchy that contains Cooking Gear.

Products
Camping Equipment
Cooking Gear
TrailChef Water Bag
TrailChef Canteen
...
Mountain Man Extreme
Mountain Man Deluxe

**Example 2**

```
hierarchy ( [great_outdoors_company].[Products].[Products].[Product line] )
```

Result: Returns every member in the hierarchy that contains the Product line.

Products
Camping Equipment
Cooking Gear
TrailChef Water Bag
TrailChef Canteen
...
Mountain Man Extreme
Mountain Man Deluxe

**item**

Returns a member from the "index" location within "set_expression". The index into the set is zero based.

**Syntax**

```
item ( set_expression , index )
```

**Example**

```
item ( children ( [Camping Equipment] ) , 2 )
```

Result: Sleeping Bags

**intersect**

Returns the intersection of "set_expression1" and "set_expression2". The result retains duplicates only when the optional keyword "all" is supplied as the third argument.

**Syntax**

```
intersect ( set_expression1 , set_expression2 [ , all ] )
```
Appendix B: Using the Expression Editor

**Example**

`intersect ( set ( [Camping Equipment], [Mountaineering Equipment] ), set ( [Camping Equipment], [Outdoor Protection], ), all )`

Result: Camping Equipment

**lag**

Returns the sibling member that is "index_expression" number of positions prior to "member".

**Syntax**

`lag ( member, index_expression )`

**Example 1**

`lag ( [Tents], 1 )`

Result: Cooking Gear

**Example 2**

`lag ( [Tents], -2 )`

Result: Packs

**lastChild**

Returns the last child of a specified member.

**Syntax**

`lastChild ( member )`

**Example 1**

`lastChild ( Cooking Gear )`

Result: TrailChef Utensils

**Example 2**

`lastChild ( [By Product Line] )`

Result: Golf Equipment

**lastPeriods**

Returns the set of members from the same level that ends with "member". The number of members returned is the absolute value of "integer_expression". If "integer_expression" is negative, members following and including the specified member are returned. Typically used with a time dimension. This function appears in the Rolling and Moving Averages interactive sample report.

**Syntax**

`lastPeriods ( integer_expression, member )`

**Example 1**

`lastPeriods ( 2, [2006 Q 4] )`

Result: Returns the last two members from the level that ends with 2006 Q 4.
Example 2
lastPeriods ( -3 , [2006 Q 4] )

Result: Returns the last three members from the level that starts with 2006 Q 4.

2006 Q 4
2007 Q 1
2007 Q 2

lastSibling

Returns the last child of the parent of a specified member.

Syntax
lastSibling ( member )

Example
lastSibling ( [Camping Equipment] )

Result: Golf Equipment

lead

Returns the sibling member that is "index_expression" number of positions after "member". If "index_expression" is negative, returns the sibling member that is "index_expression" number of positions before "member".

Syntax
lead ( member , index_expression )

Example 1
lead ( [Outdoor Protection] , 1 )

Result: Personal Accessories

Example 2
lead ( [Outdoor Protection] , -2 )

Result: Golf Equipment

level

Returns the level of "member".

Syntax
level ( member )

Example
level ( [Golf Equipment] )

Result: Returns the members on the Golf Equipment level.

Camping Equipment
Mountaineering Equipment
Appendix B: Using the Expression Editor

Personal Accessories
Outdoor Protection
Golf Equipment

**levels**

Returns the level in "hierarchy" whose distance from the root is specified by "index".

**Syntax**

`levels ( hierarchy , index )`

**Example 1**

`levels ( [great_outdoors_company].[Products].[Products] , 2 )`

Result: Returns the members two levels from the root Products hierarchy.

Cooking Gear
Sleeping Bags
Packs
Tents
...  
Irons
Putters
Woods
Golf Accessories

**Example 2**

`levels ( [great_outdoors_company].[Products].[Products] , 1 )`

Result: Returns the members one level from the root Products hierarchy.

Camping Equipment
Mountaineering Equipment
Personal Accessories
Outdoor Protection
Golf Equipment

**linkMember**

Returns the corresponding member in "level" or "hierarchy" (of the same dimension). For level-based hierarchies, a level must be specified as the second argument, and for parent-child hierarchies, a hierarchy must be specified. An exception is thrown when the second parameter does not resolve to a hierarchy of the member's dimension. Note that calculated members are not supported as the first argument.

**Syntax**

`linkMember ( member , level|hierarchy )`

**members**

Returns the set of members in "hierarchy" or "level". In the case of a hierarchy, the order of the members in the result is not guaranteed. If a predictable order is required, an explicit ordering function (such as hierarchize) must be used.

**Syntax**

`members ( hierarchy|level )`
Example 1
members ( [great_outdoors_company].[Years].[Years] )
Result: Returns the members in Years.

Example 2
members ( [great_outdoors_company].[Products].[Products].[Product line] )
Result: Returns the members in Product line.

nextMember
Returns the next member in the "member" level.

Syntax
nextMember ( member )

Example
nextMember ( [Outdoor Protection] )
Result: Golf Equipment

openingPeriod
Returns the first sibling member among the descendants of a member at "level". This function is typically used with a time dimension.

Syntax
openingPeriod ( level [ , member ] )

Example 1
openingPeriod ( [great_outdoors_company].[Years].[Years].[Month] )
Result: 2004/Jan

Example 2
openingPeriod ( [great_outdoors_company].[Years].[Years].[Year] )
Result: 2004

Example 3
openingPeriod ( [great_outdoors_company].[Years].[Years].[Month] , [2006 Q 4] )
Result: 2006/Oct

order
Arranges the members of "set_expression" according to their "value_expression" and the third parameter. ASC and DESC arrange members in ascending or descending order, respectively, according to their position in the set hierarchy. Then the children of each member are arranged according to "value_expression". BASC and BDESC arrange members in the set without regard to the hierarchy. In the absence of an explicit specification, ASC is the default.
Syntax
order ( set_expression , value_expression [ , ASC|DESC|BASC|BDESC ] )

Example 1
order ( members ( [Great Outdoors Company].[Product].[Product].[Product type] ) , [Quantity sold] , BASC )

Result: Returns the quantity sold for each product type in no particular order.

<table>
<thead>
<tr>
<th>Product Line</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Woods</td>
<td>13,924</td>
</tr>
<tr>
<td>Irons</td>
<td>14,244</td>
</tr>
<tr>
<td>Safety</td>
<td>22,332</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Sunscreen</td>
<td>215,432</td>
</tr>
<tr>
<td>Insect Repellents</td>
<td>270,074</td>
</tr>
<tr>
<td>Lanterns</td>
<td>345,096</td>
</tr>
</tbody>
</table>

Example 2
order ( members ( [Great Outdoors Company].[Product].[Product].[Product type] ) , [Quantity sold] , ASC )

Result: Returns the quantity sold for each product type in ascending order.

<table>
<thead>
<tr>
<th>Product Line</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Woods</td>
<td>13,924</td>
</tr>
<tr>
<td>Irons</td>
<td>14,244</td>
</tr>
<tr>
<td>Putters</td>
<td>23,244</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Tents</td>
<td>130,664</td>
</tr>
<tr>
<td>Cooking Gear</td>
<td>198,676</td>
</tr>
<tr>
<td>Lanterns</td>
<td>345,096</td>
</tr>
</tbody>
</table>

ordinal

Returns the zero-based ordinal value (distance from the root level) of "level".

Syntax
ordinal ( level )

Example 1
ordinal ( [great_outdoors_company].[Products].[Products].[Product line] )
Result: 1

**Example 2**

`ordinal ( [great_outdoors_company].[Products].[Products].[Product type] )`

Result: 2

**parallelPeriod**

Returns a member from a prior period in the same relative position as "member". This function is similar to the cousin function, but is more closely related to time series. It takes the ancestor of "member" at "level" (called "ancestor") and the sibling of "ancestor" that lags by "integer_expression" positions, and returns the parallel period of "member" among the descendants of that sibling. When unspecified, "integer_expression" defaults to 1 and "member" defaults to the current member.

**Syntax**

`parallelPeriod ( level [ , integer_expression [ , member ] ] )`

**Example 1**

`parallelPeriod ( [great_outdoors_company].[Years].[Years].[Quarter] , -1 , [2006/Aug] )`

Result: 2006/Nov

**Example 2**

`parallelPeriod ( [great_outdoors_company].[Years].[Years].[Quarter] , 1 , [2006/Aug] )`

Result: 2006/May

**Example 3**

`parallelPeriod ( [great_outdoors_company].[Years].[Years].[Year] , 2 , [2006/Aug] )`

Result: 2004/Aug

**parent**

Returns the member that is the parent of "member" or "measure".

**Syntax**

`parent ( member|measure )`

**Example**

`parent ( [Cooking Gear] )`

Result: Camping Equipment

**periodsToDate**

Returns a set of sibling members from the same level as "member", as constrained by "level". It locates the ancestor of "member" at "level" and returns that ancestor’s descendants at the same
level as "member" (up to and including "member"). Typically used with a time dimension. This function appears in the Rolling and Moving Averages interactive sample report.

Syntax

`periodsToDate ( level , member )`

Example

`periodsToDate ( [great_outdoors_company].[Years].[Year] , [2004/Mar] )`

Result: Returns values for [2004/Jan], [2004/Feb], [2004/Mar]

`prevMember`

Returns the member that immediately precedes "member" in the same level. This function appears in the Sales Growth Year Over Year sample report in the GO Data Warehouse (analysis) package.

Syntax

`prevMember ( member )`

Example 1

`prevMember ( [Outdoor Protection] )`

Result: Personal Accessories

Example 2

`prevMember ( [2005] )`

Result: 2004

`member`

Defines a member based on "value_expression" in "hierarchy". "String1" identifies the member created by this function. It must be unique in the query and different from any other member in the same hierarchy. "String2" is the caption of the member; if it is absent, the caption is empty. To ensure predictable results, it is recommended that you supply the "hierarchy". Note: All calculations used as grouping items whose sibling items are other calculations or member sets should be explicitly assigned to a hierarchy using this function. The results are not predictable otherwise. The only exception is where the calculation involves only members of the same hierarchy as the siblings. In this case, the calculation is assumed to belong to that hierarchy.

Syntax

`member ( value_expression [ , string1 [ , string2 [ , hierarchy ] ] ] )`

Example

`member ( total ( currentMeasure within set filter ( [great_outdoors_company].[Products].[Product name] , caption ( [great_outdoors_company].[Products].[Product name] ) starts with 'B' ) ) , 'BProducts', 'B Products' , [great_outdoors_company].[Products] )`

Result: Returns the quantity sold and revenue for all products that start with the letter B.
**nestedSet**

Returns the set of members of "set_expression2" evaluated in the context of the current member of "set_expression1".

**Syntax**

```
nestedSet ( set_expression1 , set_expression2 )
```

**Example**

```
nestedSet ( members ( [Product line] ) , topCount ( descendants ( currentMember ( [great_outdoors_company].[Products].[Products] ) , [great_outdoors_company].[Products].[Product name] ) , 2 , [Revenue] ) )
```

Result: Returns the top two products by revenue for each product line.

**set**

Returns the list of members defined in the expression. The members must belong to the same hierarchy.

**Syntax**

```
set ( member { , member } )
```

**Example**

```
set ( [Golf Equipment] , [Irons] , [TrailChef Cup] )
```

Result: Returns Golf Equipment, Irons, and TrailChef Cup.

**siblings**

Returns the children of the parent of the specified member.

**Syntax**

```
siblings ( member )
```

**Example**

```
siblings ( [Golf Equipment] )
```

Result: Returns the siblings of Golf Equipment.

- Camping Equipment
- Golf Equipment
- Mountaineering Equipment
- Outdoor Protection
- Personal Accessories

**tail**

Returns the last "index_expression" elements of "set_expression". The default for "index_expression" is 1.

**Syntax**

```
tail ( set_expression [ , index_expression ] )
```
Appendix B: Using the Expression Editor

**Example 1**

tail (members { [great_outdoors_company].[Products].[Products].[Product line] })

Result: Returns the last member of the Product line set.

**Example 2**

tail (members { [great_outdoors_company].[Products].[Products].[Product line] }, 2)

Result: Returns the last two members of the Product line set.

**topCount**

Sorts a set according to the values of "numeric_expression" evaluated at each of the members of "set_expression" and returns the top "index_expression" members.

**Syntax**

topCount ( set_expression , index_expression , numeric_expression )

**Example**

topCount ( [great_outdoors_company].[Products].[Products].[Product line] , 2 , [Revenue] )

Result: Returns the top two revenues for the Product line set.

<table>
<thead>
<tr>
<th>Product line</th>
<th>Revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Camping Equipment</td>
<td>$89,713,990.92</td>
</tr>
<tr>
<td>Personal Accessories</td>
<td>$31,894,465.86</td>
</tr>
</tbody>
</table>

**topPercent**

Sorts "numeric_expression2", evaluated at the corresponding members of "set_expression", and picks up the topmost elements whose cumulative total is at least "numeric_expression1" percent of the total.

**Syntax**

topPercent ( set_expression , numeric_expression1 , numeric_expression2 )

**Example**


Result: For the set of Camping Equipment, Golf Equipment, and Mountaineering Equipment, returns the members whose percentage totals are greater than or equal to 40% for 2006.
topSum

Sorts "numeric_expression2", evaluated at the corresponding members of "set_expression", and picks up the topmost elements whose cumulative total is at least "numeric_expression1".

**Syntax**

topSum ( set_expression , numeric_expression1 , numeric_expression2 )

**Example**


tuple

Identifies a cell location (intersection) based on the specified members, each of which must be from a different dimension. This function implicitly includes the current member from all dimensions that are not otherwise specified in the arguments. The current member of any dimension not specified in the evaluating context is assumed to be the default member of that dimension. The value of this cell can be obtained with the "value" function.

**Syntax**

tuple ( member { , member } )

**Example**

tuple ( [Mountaineering Equipment] , [Fax] )

Result: Returns the Mountaineering Equipment sales by fax.

union

Returns data for "set_expression1" and "set_expression2". The result retains duplicates only when the optional keyword "all" is supplied as the third argument.

**Syntax**

union ( set_expression1 , set_expression2 [ , all ] )

**Example 1**


Result: Returns data for both sets as one new set, showing the Golf Equipment column only once.

**Example 2**


Result: Returns data for both sets as one new set, showing the Golf Equipment column twice.

roleValue

Returns the value of the attribute that is associated with the role whose name is specified by "string" within the specified context. "Member" or "set_expression" is optional only in a number of limited circumstances, where it can be derived from another context. Applications can be made portable across different data sources and models by accessing attributes by role rather than by query item.
ID. For dimensionally-modeled relational (DMR) data sources, assignment of roles is the modeler's responsibility. Intrinsic roles that are defined for members of all data source types include: '_businessKey', '_memberCaption', '_memberDescription', '_memberUniqueName'. Additional roles can be defined in Framework Manager for each level in a hierarchy. For example, a Product type level may have an attribute column called "Type Shipping Container", and the Product level may have a "Product Shipping Container" attribute. Each of these could be assigned a custom role in Framework Manager called "Container". The property could then be referenced independently of the actual column name by using the roleValue function.

**Syntax**

```
roleValue ( string [ , member|set_expression ] )
```

**Example 1**

```
roleValue ( '_memberCaption' , [Sales].[Product].[Product].[Product line] -> [all].[1] )
```

Result: Camping Equipment

**Example 2**

```
roleValue ( '_businessKey' , [great_outdoors_company].[Years].[Years].[Year] )
```

Result: Returns the value of the attribute that is associated with the business key role.

("2004-01-01","2004-12-31")
("2005-01-01","2005-12-31")
("2006-01-01","2006-12-31")

**Example 3**

```
roleValue ( '_memberUniqueName' , [great_outdoors_company].[Years].[Years].[Year] )
```

Result: Returns the value of the attribute that is associated with the MUN role.

[great_outdoors_company].[Years].[Years].[Year] ->:[PC].[Years (Root)].[20040101-20041231]
[great_outdoors_company].[Years].[Years].[Year] ->:[PC].[Years (Root)].[20050101-20051231]
[great_outdoors_company].[Years].[Years].[Year] ->:[PC].[Years (Root)].[20060101-20061231]

**rootMember**

Returns the root member of a single-root hierarchy. This function appears in the Promotion Success sample report in the GO Data Warehouse (analysis) package.

**Syntax**

```
rootMember ( hierarchy )
```

**rootMembers**

Returns the root members of a hierarchy.

**Syntax**

```
rootMembers ( hierarchy )
```
Example

```plaintext
rootMembers ( [great_outdoors_company].[Years].[Years] )
```

Result: By Time

**subset**

Returns a subset of members in "set_expression" starting at "index_expression1" from the beginning. If the count "index_expression2" is specified, that many members are returned (if available). Otherwise, all remaining members are returned.

**Syntax**

```plaintext
subset ( set_expression, index_expression1 [ , index_expression2 ] )
```

**Example 1**

```plaintext
subset ( members ( [great_outdoors_company].[Products].[Products].[Product line] ) , 2 )
```

Result: Returns the members of the Product line set starting at the second member.

- Mountaineering Equipment
- Outdoor Protection
- Personal Accessories

**Example 2**

```plaintext
subset ( members ( [great_outdoors_company].[Products].[Products].[Product line] ) , 2 , 2 )
```

Result: Returns two members of the Product line set starting at the second member.

- Mountaineering Equipment
- Outdoor Protection

**unique**

Removes all duplicates from "set_expression". The remaining members retain their original order.

**Syntax**

```plaintext
unique ( set_expression )
```

**value**

Returns the value of the cell identified by "tuple". Note that the default member of the Measures dimension is the Default Measure.

**Syntax**

```plaintext
value ( tuple )
```

**Example 1**

```plaintext
value ( tuple ( [great_outdoors_company].[Years].[Years].[Year] ->:[PC].[Years (Root)].[20040101-20041231] , [great_outdoors_company].[Measures].[Revenue] ) )
```

Result: $34,750,563.50

**Example 2**

```plaintext
value ( tuple ( [2004] , [Camping Equipment] , [Revenue] ) )
```
Result: $20,471,328.88

**DB2**

**ascii**

Returns the ASCII code value of the leftmost character of the argument as an integer.

**Syntax**

```plaintext
ascii ( string_expression )
```

**Example**

```plaintext
ascii ( a )
```

Result: Returns 65, the ASCII code value of "a".

**ceiling**

Returns the smallest integer greater than or equal to "numeric_expression".

**Syntax**

```plaintext
ceiling ( numeric_expression )
```

**Example**

```plaintext
ceiling ( 0.75 )
```

Result: Returns 0.8.

**char**

Returns a string representation of a date/time value or a decimal number.

**Syntax**

```plaintext
char ( expression )
```

**chr**

Returns the character that has the ASCII code value specified by "integer_expression". "Integer_expression" should be between 0 and 255.

**Syntax**

```plaintext
chr ( integer_expression )
```

**Example**

```plaintext
chr ( 65 )
```

Result: Returns a, the character for the ASCII code value of 65.

**concat**

Returns a string that is the result of concatenating "string_expression1" with "string_expression2".

**Syntax**

```plaintext
concat ( string_expression1, string_expression2 )
```
**Example**

```vbnet
concat ( [Sales target (query)].[Sales staff].[First name], [Sales target (query)].[Sales staff].[Last name] )
```

Result: Returns the first name and last name; e.g., Bob Smith.

---

**date**

Returns a date from a single input value. "Expression" can be a string or integer representation of a date.

**Syntax**

date ( expression )

**Example**

date ( '1998-01-08' )


---

**day**

Returns the day of the month (1-31) from "date_expression". "Date_expression" can be a date value or a string representation of a date.

**Syntax**

day ( date_expression )

**Example**

day ( '1998-01-08' )

Result: Returns 8.

---

**dayname**

Returns a character string containing the data source-specific name of the day (for example, Sunday through Saturday or Sun. through Sat. for a data source that uses English, or Sonntag through Samstag for a data source that uses German) for the day portion of "date_expression". "Date_expression" can be a date value or a string representation of a date.

**Syntax**

dayname ( date_expression )

**Example**

dayname ( '1998-01-08' )

Result: Returns Thursday.

---

**dayofweek**

Returns the day of the week in "date_expression" as an integer in the range 1 to 7, where 1 represents Sunday. "date_expression" can be a date value or a string representation of a date.

**Syntax**

dayofweek ( date_expression )
Example
dayofweek ( '1998-01-08' )
Result: Returns 5.

dayofweek_iso
Returns the day of the week in "date_expression" as an integer in the range 1 to 7, where 1 represents Monday. "date_expression" can be a date value or a string representation of a date.

Syntax
dayofweek_iso ( date_expression )

Example
dayofweek_iso ( '1998-01-08' )
Result: Returns 4.

dayofyear
Returns the day of the year in "date_expression" as an integer in the range 1 to 366. "Date_expression" can be a date value or a string representation of a date.

Syntax
dayofyear ( date_expression )

Example
dayofyear ( current_date )
Result: Returns the day of the year for the current date; e.g., if it was January 28, the expression would return 28.

days
Returns an integer representation of a date. "Expression" can be a date value or a string representation of a date.

Syntax
days ( expression )
dec
Returns the decimal representation of "string_expression1" with precision "numeric_expression1", scale "numeric_expression2", and decimal character "string_expression2". "String_expression1" must be formatted as an SQL Integer or Decimal constant.

Syntax

Appendix B: Using the Expression Editor
**decimal**

Returns the decimal representation of "string_expression1" with precision "numeric_expression1", scale "numeric_expression2" and decimal character "string_expression2". "String_expression1" must be formatted as an SQL Integer or Decimal constant.

**Syntax**


difference

Returns an integer value representing the difference between the values returned by the data source-specific soundex function for "string_expression1" and "string_expression2". The value returned ranges from 0 to 4, with 4 indicating the best match. Note that 4 does not mean that the strings are equal.

**Syntax**

difference ( string_expression1 , string_expression2 )

**Example 1**

difference ([Sales target (query)].[Sales staff].[First name], [Sales (query)].[Retailers].[Contact first name])

Result: 0

**Example 2**

difference ([Sales target (query)].[Sales staff].[First name], [Sales target (query)].[Sales staff].[First name])

Result: 4

digits

Returns the character string representation of a non-floating point number.

**Syntax**

digits ( numeric_expression )

double

Returns the floating-point representation of an expression. "Expression" can either be a numeric or string expression.

**Syntax**

double ( expression )

event_mon_state

Returns the operational state of a particular state monitor.

**Syntax**

event_mon_state ( string_expression )
float

Returns the floating-point representation of a number.

Syntax

float ( numeric_expression )

hex

Returns the hexadecimal representation of a value.

Syntax

hex ( expression )

hour

Returns the hour, an integer from 0 (midnight) to 23 (11:00 pm), from "time_expression". "Time_expression" can be a time value or a string representation of a time.

Syntax

hour ( time_expression )

Example

hour ( 01:22:45 )

Result: Returns 1.

insert

Returns a string where "integer_expression2" characters have been deleted from "string_expression1" beginning at "integer_expression1" and where "string_expression2" has been inserted into "string_expression1" at its start. The first character in the string is at position 1.

Syntax

insert ( string_expression1, integer_expression1, integer_expression2, string_expression2 )

integer

Returns the integer representation of an expression. "Expression" can be a numeric value or a string representation of a number.

Syntax

integer ( expression )

Example

integer ( 84.95 )

Result: 85

int

Returns the integer representation of an expression. "Expression" can be a numeric value or a string representation of a number.
Syntax

\texttt{int ( expression )}

Example

\texttt{int ( 84.95 )}

Result: 85

\textbf{julian\_day}

Returns an integer value representing the number of days from January 1, 4712 BC (the start of the Julian date calendar) to the date value specified in "expression". "Expression" can be a date value or a string representation of a date.

Syntax

\texttt{julian\_day ( expression )}

Example

\texttt{julian\_day ( '2009-06-29' )}

Result: 2455012.22130739595741034

\textbf{lcase}

Returns "string\_expression" with all uppercase characters shifted to lowercase.

Syntax

\texttt{lcase ( string\_expression )}

Example

\texttt{lcase ( [Sales (query)].[Sales staff].[Last name] )}

Result: Returns last names with no uppercase letters.

\textbf{left}

Returns the leftmost "integer\_expression" characters of "string\_expression".

Syntax

\texttt{left ( string\_expression, integer\_expression )}

Example

\texttt{left ( [Sales (query)].[Sales staff].[Last name] , 3 )}

Result: Returns the first three characters of each last name.

\textbf{length}

Returns the length of the operand in bytes. Exception: double byte string types return the length in characters.

Syntax

\texttt{length ( expression )}
Example

length ( [Sales (query)].[Sales staff].[Record start date] )

Result: Returns 4; dates always return a value of 4.

locate

Returns the starting position of the first occurrence of "string_expression1" within "string_expression2". The search starts at position start "integer_expression" of "string_expression2". The first character in a string is at position 1. If "string_expression1" is not found, zero is returned.

Syntax

locate ( string_expression1, string_expression2 [ , integer_expression ] )

Example

locate ( A, [Sales (query)].[Sales staff].[Last name] , 2 )

Result: Returns the position of the character A in the last names starting at the second character of the last name.

long_varchar

Returns a long string.

Syntax

long_varchar ( string_expression )

ltrim

Returns "string_expression" with leading spaces removed.

Syntax

ltrim ( string_expression )

Example

ltrim ( [Sales (query)].[Sales staff].[Last name] )

Result: Returns last names with any leading spaces removed.

microsecond

Returns the microsecond (time-unit) part of a value. "Expression" can be a timestamp or a string representation of a timestamp.

Syntax

microsecond ( expression )

Example

microsecond ( 01:45:34.056 )

Result: Returns 056.
**midnight_seconds**

Returns an integer value in the range 0 to 86400 representing the number of seconds between midnight and time value specified in the argument. "Expression" can be a time value, a timestamp or a string representation of a time.

**Syntax**

```plaintext
midnight_seconds ( expression )
```

**Example**

```plaintext
midnight_seconds ( 01:45:34.056 )
```

Result: Returns 6334.

**minute**

Returns the minute (an integer from 0-59) from "time_expression". "Time_expression" can be a time value, a timestamp, or a string representation of a time.

**Syntax**

```plaintext
minute ( time_expression )
```

**Example**

```plaintext
minute ( 01:45:34.056 )
```

Result: Returns 45.

**month**

Returns the month (an integer from 1-12) from "date_expression".

**Syntax**

```plaintext
month ( date_expression )
```

**Example**

```plaintext
month ( 2005-11-01 )
```

Result: Returns 11.

**monthname**

Returns a character string containing the data source-specific name of the month (for example, January through December or Jan. through Dec. for an English data source, or Januar through Dezember for a German data source) for the month portion of "date_expression".

**Syntax**

```plaintext
monthname ( date_expression )
```

**Example**

```plaintext
monthname ( 2005-11-01 )
```

Result: November
Appendix B: Using the Expression Editor

**quarter**

Returns the quarter in "date_expression" as a number in the range 1 to 4, where 1 represents January 1 through March 31.

**Syntax**

\[
\text{quarter ( date_expression )}
\]

**Example**

\[
\text{quarter ( 2005-11-01 )}
\]

Result: Returns 4.

**radians**

Returns the number of radians converted from "numeric_expression" degrees.

**Syntax**

\[
\text{radians ( numeric_expression )}
\]

**repeat**

Returns a string consisting of "string_expression" repeated "integer_expression" times.

**Syntax**

\[
\text{repeat ( string_expression, integer_expression )}
\]

**Example**

\[
\text{repeat ( XYZ, 3 )}
\]

Result: Returns XYZXYZXYZ.

**replace**

Replaces all occurrences of "string_expression2" in "string_expression1" with "string_expression3".

**Syntax**

\[
\text{replace ( string_expression1, string_expression2, string_expression3 )}
\]

**Example**

\[
\text{replace ( [Sales (query)].[Sales staff].[Position code], A, a )}
\]

Result: Returns position codes with all occurrences of "A" replaced by "a".

**right**

Returns the rightmost "integer_expression" characters of "string_expression".

**Syntax**

\[
\text{right ( string_expression, integer_expression )}
\]

**Example**

\[
\text{right ( [Sales (query)].[Sales staff].[Position code], 3 )}
\]
Result: Returns the rightmost 3 characters of each position code.

**round**

Returns "numeric_expression" rounded to "integer_expression" places to the right of the decimal point. If "integer_expression" is negative, "numeric_expression" is rounded to the nearest absolute value "integer_expression" places to the left of the decimal point. Rounding takes place before data formatting is applied.

**Syntax**

```
round ( numeric_expression, integer_expression )
```

**Example**

```
round ( 3.14159265, 3 )
```

Result: Returns 3.142.

**rtrim**

Returns "string_expression" with trailing spaces removed.

**Syntax**

```
rtrim ( string_expression )
```

**Example**

```
rtrim ( [Sales (query)].[Sales staff].[Last name] )
```

Result: Returns last names with any spaces at the end of the name removed.

**second**

Returns the second (an integer from 0-59) from "time_expression".

**Syntax**

```
second ( time_expression )
```

**Example**

```
second ( 01:45:34.056 )
```

Result: Returns 34.

**sign**

Returns an indicator of the sign of "numeric_expression": +1 if "numeric_expression" is positive, 0 if zero, or -1 if negative.

**Syntax**

```
sign ( numeric_expression )
```

**Example**

```
sign ( [Revenue] )
```

Result: Returns + for positive values and - for negative values.
smallint

Returns the small integer representation of a number.

**Syntax**
smallint ( expression )

soundex

Returns a 4 character string code obtained by systematically abbreviating words and names in "string_expression" according to phonetics. Can be used to determine if two strings sound the same. For example, does sound-of ('SMITH') = sound-of ('SMYTH').

**Syntax**
soundex ( string_expression )

space

Returns a string consisting of "integer_expression" spaces.

**Syntax**
space ( integer_expression )

**Example**
space ( 5 )
Result: Returns 5 spaces.

substr

Returns the substring of "string_expression" that starts at position "integer_expression1" for "integer_expression2" characters. The first character in "string_expression" is at position 1.

**Syntax**
substr ( string_expression , integer_expression1 [ , integer_expression2 ] )

**Example**
substr ( [Sales (query)].[Sales staff].[Position code], 3 , 5 )
Result: Returns characters 3 to 7 of the position codes.

group_expression

Returns an unqualified name of a table or view based on the object name in "string_expression1" and the schema name given in "string_expression2". It is used to resolve aliases.

**Syntax**
table_name ( string_expression1 [ , string_expression2 ] )

table_schema

Returns the schema name portion of the two-part table or view name based on the object name in "string_expression1" and the schema name in "string_expression2". It is used to resolve aliases.
**Syntax**

```
table_schema ( string_expression1 [ , string_expression2 ] )
```

**time**

Returns a time from a value.

**Syntax**

```
time ( expression )
```

**timestamp**

Returns a timestamp from a value or a pair of values. "Expression1" must represent a date value, and "expression2" must represent a time value.

**Syntax**

```
timestamp ( expression1 [ , expression2 ] )
```

**Example**

```
timestamp ( 11 November 2005 , 12:00:00.000000 )
```

Result: Returns 2005-11-11-12:00:00.000000.

**timestamp_iso**

Returns a datetime in the ISO format (yyyy-mm-dd hh:mm:ss.nnnnnn) converted from the IBM format (yyyy-mm-dd-hh.mm.ss.nnnnnn). If "expression" is a time, it inserts the value of the CURRENT DATE for the date elements and zero for the fractional time element.

**Syntax**

```
timestamp_iso ( expression )
```

**Example**

```
timestamp_iso ( 11 November 2005 , 12:00:00.000000 )
```

Result: Returns 2005-11-11 12:00:00.000000.

**timestampdiff**

Returns an estimated number of intervals of type "expression1" based on the difference between two timestamps. "Expression2" is the result of subtracting two timestamp types and converting the result to CHAR. Valid values of "expression1" are: 1 Fractions of a second; 2 Seconds; 4 Minutes; 8 Hours; 16 Days; 32 Weeks; 64 Months; 128 Quarters; 256 Years.

**Syntax**

```
timestampdiff ( expression1, expression2 )
```

**to_char**

Returns the string representation of a timestamp with the format of "string_expression".

**Syntax**

```
to_char ( timestamp_expression , string_expression )
```
translate

Returns "string_expression1" in which characters from "string_expression3" are translated to the equivalent characters in "string_expression2". "String_expression4" is a single character that is used to pad "string_expression2" if it is shorter than "string_expression3". If only "string_expression1" is present, then this function translates it to uppercase characters.

Syntax

translate ( string_expression1 [ , string_expression2, string_expression3 [ , string_expression4 ] ] )

Example 1

translate ( 'abcdefg' )
Result: Returns ABCDEFG.

Example 2

translate ( 'mnlop' , n, m , - )
Result: Returns n-nlop.

trunc

Returns "numeric_expression1" truncated to "numeric_expression2" places to the right of the decimal point. If "numeric_expression2" is negative, "numeric_expression1" is truncated to the absolute value of "numeric_expression2" places to the left of the decimal point.

Syntax

trunc ( numeric_expression1, numeric_expression2 )

Example

trunc ( 3.14159265, 3 )
Result: Returns 3.141.

truncate

Returns "numeric_expression1" truncated to "numeric_expression2" places to the right of the decimal point. If "numeric_expression2" is negative, "numeric_expression1" is truncated to the absolute value of "numeric_expression2" places to the left of the decimal point.

Syntax

truncate ( numeric_expression1, numeric_expression2 )

Example

truncate ( 3141.59265, -3 )
Result: Returns 3.

ucase

Returns "string_expression" with all lowercase characters shifted to uppercase.
**Syntax**

`ucase ( string_expression )`

**Example**

`ucase ( XY896Zbced789 )`

Result: Returns XY896ZBCED789.

**value**

Returns the first non-null argument (or null if all arguments are null). The Value function takes two or more arguments.

**Syntax**

`value ( expression_list )`

**Example**

`value ( [Unit cost], [Unit price], [Unit sale price] )`

Result: Returns the first non-null value.

**varchar**

Returns a VARCHAR representation of expression, with length numeric_expression.

**Syntax**

`varchar ( expression [ , numeric_expression ] )`

**week**

Returns the week of the year in "date_expression" as an integer value in the range 1 to 53.

**Syntax**

`week ( date_expression )`

**Example**

`week ( 11 November 2005 )`

Result: Returns 45.

**year**

Returns the year from "date_expression".

**Syntax**

`year ( date_expression )`

**Example**

`year ( 11 November 2005 )`

Appendix B: Using the Expression Editor

**DB2 Math**

**log**

Returns the natural logarithm of "numeric_expression".

**Syntax**

```sql
log ( numeric_expression )
```

**log10**

Returns the base ten logarithm of "numeric_expression".

**Syntax**

```sql
log10 ( numeric_expression )
```

**rand**

Generates a random number using "integer_expression" as a seed value.

**Syntax**

```sql
rand ( integer_expression )
```

**DB2 Trigonometry**

**acos**

Returns the arccosine of "numeric_expression" in radians. The arccosine is the angle whose cosine is "numeric_expression".

**Syntax**

```sql
acos ( numeric_expression )
```

**asin**

Returns the arcsine of "numeric_expression" in radians. The arcsine is the angle whose sine is "numeric_expression".

**Syntax**

```sql
asin ( numeric_expression )
```

**atan**

Returns the arctangent of "numeric_expression" in radians. The arctangent is the angle whose tangent is "numeric_expression".

**Syntax**

```sql
atan ( numeric_expression )
```

**atanh**

Returns the hyperbolic arctangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.
Syntax
atanh ( numeric_expression )

atan2
Returns the arctangent of the x and y coordinates specified by "numeric_expression1" and "numeric_expression2", respectively, in radians. The arctangent is the angle whose tangent is "numeric_expression2" / "numeric_expression1".

Syntax
atan2 ( numeric_expression1 , numeric_expression2 )

cos
Returns the cosine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax
cos ( numeric_expression )

cosh
Returns the hyperbolic cosine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax
cosh ( numeric_expression )

cot
Returns the cotangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax
cot ( numeric_expression )

degrees
Returns "numeric_expression" radians converted to degrees.

Syntax
degrees ( numeric_expression )

sin
Returns the sine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax
sin ( numeric_expression )
**sinh**
Returs the hyperbolic sine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

**Syntax**
sinh (numeric_expression)

**tan**
Returns the tangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

**Syntax**
tan (numeric_expression)

**tanh**
Returns the hyperbolic tangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

**Syntax**
tanh (numeric_expression)

**Informix**

**cardinality**
Returns the number of elements in a collection column (SET, MULTISET, LIST).

**Syntax**
cardinality (string_expression)

**char_length**
Returns the number of logical characters in "string_expression". The number of logical characters can be distinct from the number of bytes in some East Asian locales.

**Syntax**
char_length (string_expression)

**concat**
Returns a string that is the result of concatenating, or joining, "string_expression1" to "string_expression2".

**Syntax**
concat (string_expression1, string_expression2)

**Example**
concat ([Sales (query)].[Sales staff].[First name], [Sales (query)].[Sales staff].[Last name])
Result: Returns the first name and last name; e.g., Bob Smith.

**date**

Returns the date value of "string_expression", "date_expression", or "integer_expression".

**Syntax**

date ( string_expression|date_expression|integer_expression )

**day**

Returns an integer that represents the day of the month (1-31).

**Syntax**

day ( date_expression )

**extend**

Adjusts the precision of a datetime or date expression. The expression cannot be a quoted string representation of a date value. If you do not specify first and last qualifiers, the default qualifiers are year to fraction (3). If the expression contains fields that are not specified by the qualifiers, the unwanted fields are discarded. If the first qualifier specifies a larger (more significant) field than what exists in the expression, the new fields are filled in with values returned by the current function. If the last qualifier specifies a smaller (less significant) field than what exists in the expression, the new fields are filled in with constant values. A missing month or day field is filled in with 1, and missing hour to fraction fields are filled in with 0.

**Syntax**

extend ( date_expression , ' { year to second } ' )

**Example**

extend ( some_date_column , { year to second } )

**hex**

Returns the hexadecimal encoding of "integer_expression".

**Syntax**

hex ( integer_expression )

**initcap**

Returns "string_expression" with the first letter of each word in uppercase and all other letters in lowercase. A word begins after any character other than a letter. Thus, in addition to a blank space, symbols such as commas, periods, and colons can introduce a new word.

**Syntax**

initcap ( string_expression )
Appendix B: Using the Expression Editor

**length**

Returns the number of bytes in "string_expression", not including any trailing blank spaces. For byte or text "string_expression", length returns the full number of bytes, including any trailing blank spaces.

**Syntax**

```
length ( string_expression )
```

**lpad**

Returns "string_expression1" left-padded by "string_expression2" to the total number of characters specified by "integer_expression". The sequence of "string_expression2" occurs as many times as necessary to make the return string the length specified by "integer_expression".

**Syntax**

```
lpad ( string_expression1 , integer_expression , string_expression2 )
```

**mdy**

Returns a type date value with three expressions that evaluate to integers that represent the month (integer_expression1), day (integer_expression2), and year (integer_expression3).

**Syntax**

```
mdy ( integer_expression1 , integer_expression2 , integer_expression3 )
```

**month**

Returns an integer corresponding to the month portion of "date_expression".

**Syntax**

```
month ( date_expression )
```

**nvl**

Returns the value of "expression1" if "expression1" is not NULL. If "expression1" is NULL, then returns the value of "expression2".

**Syntax**

```
nvl ( expression1 , expression2 )
```

**Example**

```
nvl ( [Unit sale price] , [Unit price] )
```

Result: Returns the unit sale price, or returns the unit price if the unit sale price is NULL.

**octet_length**

Returns the number of bytes in "string_expression", including any trailing spaces.

**Syntax**

```
octet_length ( string_expression )
```
replace

Returns "string_expression1" in which every occurrence of "string_expression2" is replaced by "string_expression3". If you omit the "string_expression3" option, every occurrence of "string_expression2" is omitted from the return string.

Syntax
replace ( string_expression1, string_expression2 [ , string_expression3 ] )

Example
replace ( [Sales (query)].[Products].[Product line code], - )
Result: Returns all product line codes without the character "-"

round

Returns the rounded value of "numeric_expression". If you omit "integer_expression", the value is rounded to zero digits or to the units place. The digit range of 32 (+ and -) refers to the entire decimal value. Rounding takes place before data formatting is applied.

Syntax
round ( numeric_expression [ , integer_expression ] )

Example
round (125, -1)
Result: 130

rpad

Returns "string_expression1" right-padded by "string_expression2" to the total number of characters specified by "integer_expression". The sequence of "string_expression2" occurs as many times as necessary to make the return string the length specified by "integer_expression".

Syntax
rpad ( string_expression1, integer_expression, string_expression2 )

substr

Returns the substring of "string_expression" that starts at position "integer_expression1" for "integer_expression2" characters. The first character in "string_expression" is at position 1. If you omit "integer_expression2", returns the substring of "string_expression" that starts at position "integer_expression1" and ends at the end of "string_expression".

Syntax
substr ( string_expression, integer_expression1 [ , integer_expression2 ] )

Example
substr ( [Sales (query)].[Sales staff].[Position code], 3, 5 )
Result: Returns characters 3 to 7 of the position codes.
to_char

Returns the character string "date_expression" with the specified "string_expression" formatting. You can use this function only with built-in data types.

Syntax

to_char ( date_expression, string_expression )

to_date

Returns "string_expression1" as a date according to the date format you specify in "string_expression2". If "string_expression1" is NULL, then a NULL value is returned.

Syntax

to_date ( string_expression1, string_expression2 )

trunc

Returns the truncated value of "numeric_expression". If you omit "integer_expression", then "numeric_expression" is truncated to zero digits or to the unit’s place. The digit limitation of 32 (+ and -) refers to the entire decimal value.

Syntax

trunc ( numeric_expression [ , integer_expression ] )

weekday

Returns an integer that represents the day of the week of "date_expression". Zero (0) represents Sunday, one (1) represents Monday, and so on.

Syntax

weekday ( date_expression )

year

Returns a four-digit integer that represents the year of "date_expression".

Syntax

year ( date_expression )

Informix Math

log10

Returns the logarithm of "numeric_expression" to base 10.

Syntax

log10 ( numeric_expression )

logn

Returns the natural logarithm of "numeric_expression".
**Syntax**
\[ \logn(\text{numeric}\_\text{expression}) \]

**root**
Returns the root value of "numeric\_expression1". Requires at least one numeric argument (the radians argument). If only "numeric\_expression1" is supplied, 2 is used as a default value for "numeric\_expression2". Zero cannot be used as the value of "numeric\_expression2".

**Syntax**
\[ \text{root}(\text{numeric}\_\text{expression1} [ , \text{numeric}\_\text{expression2} ] ) \]

**Informix Trigonometry**

**acos**
Returns the arccosine of "numeric\_expression" in radians. The arccosine is the angle whose cosine is "numeric\_expression".

**Syntax**
\[ \text{acos}(\text{numeric}\_\text{expression}) \]

**asin**
Returns the arcsine of "numeric\_expression" in radians. The arcsine is the angle whose sine is "numeric\_expression".

**Syntax**
\[ \text{asin}(\text{numeric}\_\text{expression}) \]

**atan**
Returns the arctangent of "numeric\_expression" in radians. The arctangent is the angle whose tangent is "numeric\_expression".

**Syntax**
\[ \text{atan}(\text{numeric}\_\text{expression}) \]

**atan2**
Returns the arctangent of the x and y coordinates specified by "numeric\_expression1" and "numeric\_expression2", respectively, in radians. The arctangent is the angle whose tangent is "numeric\_expression1".

**Syntax**
\[ \text{atan2}(\text{numeric}\_\text{expression1} , \text{numeric}\_\text{expression2} ) \]

**cos**
Returns the cosine of "numeric\_expression" where "numeric\_expression" is an angle expressed in radians.
Appendix B: Using the Expression Editor

**Syntax**

cos ( numeric_expression )

**sin**

Returns the sine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

**Syntax**

sin ( numeric_expression )

tan

Returns the tangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

**Syntax**

tan ( numeric_expression )

**MS Access**

**ascii**

Returns the ascii code value of the leftmost character of "string_expression".

**Syntax**

ascii ( string_expression )

**ceiling**

Returns the smallest integer greater than or equal to "numeric_expression".

**Syntax**

ceiling ( numeric_expression )

**chr**

Returns the character that has the ASCII code value specified by "integer_expression". "Integer_expression" should be between 0 and 255.

**Syntax**

chr ( integer_expression )

**concat**

Returns a string that is the result of concatenating, or joining, "string_expression1" to "string_expression2".

**Syntax**

concat ( string_expression1 , string_expression2 )
Example
concat ([Sales (query)].[Sales staff].[First name], [Sales (query)].[Sales staff].[Last name])

Result: Returns the first name and last name; e.g., Bob Smith.

curdate

Returns a date value representing the current date of the computer that the database software runs on.

Syntax
curdate ()

curtime

Returns a time value representing the current time of the computer that the database software runs on.

Syntax
curtime ()

dayname

Returns a character string containing the data source-specific name of the day (for example, Sunday through Saturday or Sun. through Sat. for an English data source, or Sonntag through Samstag for a German data source) for the day portion of "date_expression".

Syntax
dayname ( date_expression )

dayofmonth

Returns the day of the month (1-31) from "date_expression". Returns the days field (a signed integer) from "interval_expression".

Syntax
dayofmonth ( date_expression|interval_expression )

dayofweek

Returns the day of the week in "date_expression" as an integer (1-7), where 1 represents Monday.

Syntax
dayofweek ( date_expression )

dayofyear

Returns the day of the year in "date_expression" as an integer (1-366).

Syntax
dayofyear ( date_expression )
hour

Returns the hour from "time_expression" as an integer from 0 (midnight) to 23 (11:00 pm).

Syntax
hour ( time_expression )

instr

Searches "string_expression1" for the first occurrence of "string_expression2" and returns an integer specifying the position of "string_expression2". "Integer_expression1" sets the starting position for the search. If "integer_expression1" is omitted, the search begins at the first character position of "string_expression1". "Integer_expression2" specifies the type of string comparison. "Integer_expression1" is required if "integer_expression2" is specified.

Syntax
instr ( [ integer_expression1 , ] string_expression1 , string_expression2 [ , integer_expression2 ] )

lcase

Returns "string_expression" with all uppercase characters converted to lowercase.

Syntax
lcase ( string_expression )

left

Returns the leftmost "integer_expression" characters of "string_expression".

Syntax
left ( string_expression , integer_expression )

Example
left ( [Sales (query)].[Sales staff].[Last name] , 3 )

Result: Returns the first three characters of each last name.

length

Returns the number of characters in "string_expression", excluding trailing blanks and the string termination character.

Syntax
length ( string_expression )

locate

Returns the starting position of the first occurrence of "string_expression1" within "string_expression2". The search starts at position "integer_expression" of "string_expression2". The first character in a string is at position 1. If "string_expression1" is not found, then zero is returned.
Syntax
locate ( string_expression1 , string_expression2 [ , integer_expression ] )

ltrim

Returns "string_expression" with leading spaces removed.

Syntax
ltrim ( string_expression )

minute

Returns the minute (an integer from 0-59) from "time_expression".

Syntax
minute ( time_expression )

month

Returns the month (an integer from 1-12) from "date_expression".

Syntax
month ( date_expression )

monthname

Returns a character string containing the data source-specific name of the month (for example, January through December or Jan. through Dec. for an English data source, or Januar through Dezember for a German data source) for the month portion of "date_expression".

Syntax
monthname ( date_expression )

Example

monthname ( 2005-11-01 )
Result: November

now

Returns a datetime value representing the current date and time of the computer that the database software runs on.

Syntax
now ()

position

Returns the starting position of "string_expression1" in "string_expression2". The first character in a string is at position 1.

Syntax
position ( string_expression1 , string_expression2 )
quarter

Returns the quarter in "date_expression" as a number (1-4), where 1 represents January 1 through March 31.

Syntax
quarter ( date_expression )

right

Returns the rightmost "integer_expression" characters of "string_expression".

Syntax
right ( string_expression , integer_expression )

round

Returns "numeric_expression" rounded to the nearest value "integer_expression" places right of the decimal point. If "integer_expression" is negative, "numeric_expression" is rounded to the nearest absolute value "integer_expression" places to the left of the decimal point. Rounding takes place before data formatting is applied.

Syntax
round ( numeric_expression , integer_expression )

Example
round (125, -1)
Result: 130

rtrim

Returns "string_expression" with trailing spaces removed.

Syntax
rtrim ( string_expression )

Example
rtrim ( [Sales (query)].[Sales staff].[Last name] )
Result: Returns last names with any spaces at the end of the name removed.

sign

Returns an indicator of the sign of "numeric_expression", +1 if positive, 0 if zero, or -1 if negative.

Syntax
sign ( numeric_expression )

space

Returns a string consisting of "integer_expression" spaces.
Syntax
space ( integer_expression )

substr

Returns the substring of "string_expression" that starts at position "integer_expression1" for "integer_expression2" characters. The first character in "string_expression" is at position 1.

Syntax
substr ( string_expression , integer_expression1 , integer_expression2 )

Example
substr ( [Sales (query)].[Sales staff].[Position code], 3 , 5 )
Result: Returns characters 3 to 7 of the position codes.

substring

Returns the substring of "string_expression" that starts at position "integer_expression1" for "integer_expression2" characters. The first character in "string_expression" is at position 1.

Syntax
substring ( string_expression , integer_expression1 , integer_expression2 )

Example
substring ( [Sales (query)].[Sales staff].[Position code], 3 , 5 )
Result: Returns characters 3 to 7 of the position codes.

truncate

Returns "string_expression" with trailing spaces removed.

Syntax
truncate ( string_expression )

ucase

Returns "string_expression" with all lowercase characters converted to uppercase.

Syntax
ucase ( string_expression )

week

Returns the week of the year in "date_expression" as an integer value (1-53), where 1 represents the first week of the year.

Syntax
week ( date_expression )

year

Returns the year from "date_expression".
Appendix B: Using the Expression Editor

Syntax
year ( date_expression )

MS Access Cast

cast_decimal
Returns the value of "expression" cast as a decimal.

Syntax
cast_decimal ( expression )

cast_float
Returns the value of "expression" cast as a float.

Syntax
cast_float ( expression )

cast_integer
Returns the value of "expression" cast as an integer.

Syntax
cast_integer ( expression )

Example
cast_integer ( 84.95 )
Result: 84

cast_numeric
Returns "string_expression" cast as a numeric value.

Syntax
cast_numeric ( string_expression )

cast_real
Returns the value of "expression" cast as a real value.

Syntax
cast_real ( expression )

cast_smallint
Returns "expression" cast as a small integer.

Syntax
cast_smallint ( expression )
**cast_varchar**
Returns the value of "expression" cast as a variable character field.

**Syntax**
cast_varchar ( expression )

**MS Access Math**

**log**
Returns the natural logarithm of "numeric_expression".

**Syntax**
log ( numeric_expression )

**rand**
Generates a random number using "integer_expression" as a seed value.

**Syntax**
rnd ( integer_expression )

**MS Access Trigonometry**

**atan**
Returns the arctangent of "numeric_expression" in radians. The arctangent is the angle whose tangent is "numeric_expression".

**Syntax**
atan ( numeric_expression )

**cos**
Returns the cosine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

**Syntax**
cos ( numeric_expression )

**sin**
Returns the sine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

**Syntax**
sin ( numeric_expression )

**tan**
Returns the tangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.


**Syntax**

\[\text{tan} \left( \text{numeric-expression} \right)\]

**Netezza**

**ascii**

Returns a number representing the ASCII code value of the leftmost character of "string_expression"; for example, ascii ( 'A' ) is 65.

**Syntax**

\[\text{ascii} \left( \text{string-expression} \right)\]

**chr**

Returns the character that has the ASCII code value specified by "integer_expression". "Integer_expression" should be between 0 and 255.

**Syntax**

\[\text{chr} \left( \text{integer-expression} \right)\]

**decode**

Compares "expr" to each search value one by one. If "expr" is equal to a search, then it returns the corresponding result. If no match is found, it returns "default". If "default" is omitted, it returns null.

**Syntax**

\[\text{decode} \left( \text{expr}, \text{search}, \text{result} \ldots \right) \left[ , \text{default} \right]\]

**initcap**

Returns "string_expression", with the first letter of each word in uppercase, all other letters in lowercase. Words are delimited by white space or characters that are not alphanumerical.

**Syntax**

\[\text{initcap} \left( \text{string-expression} \right)\]

**instr**

Searches "string_expression1" starting at position "integer_expression1" for the "integer_expression2" occurrence of "string_expression2". If "integer_expression1" is negative then the search is backwards from the end of "string_expression1". Returns an integer indicating the position of "string_expression2".

**Syntax**

\[\text{instr} \left( \text{string_expression1}, \text{string_expression2} \left[ , \text{integer_expression1} \left[ , \text{integer_expression2} \right] \right] \right)\]
lpad
Returns "string_expression1" padded to length "integer_expression" with occurrences of "string_expression2". If "string_expression1" is longer than "integer_expression", the appropriate portion of "string_expression1" is returned.

Syntax
lpad ( string_expression1 , integer_expression [ , string_expression2 ] )

ltrim
Returns "string_expression1", with leading characters removed up to the first character not in "string_expression2"; for example, ltrim ( 'xyxXxyAB' , 'xy' ) returns XxyAB.

Syntax
ltrim ( string_expression1 [ , string_expression2 ] )

months_between
Returns the number of months from "date_expression1" to "date_expression2". If "date_expression1" is later than "date_expression2" then the result will be a positive number. The days and time portions of the difference are ignored, i.e., the months are not rounded, except if "date_expression1" and "date_expression2" are the last days of a month.

Syntax
months_between ( date_expression1 , date_expression2 )

next_day
Returns the datetime of the first weekday named by "string_expression" that is later than "datetime_expression". The return value has the same hours, minutes, and seconds as "datetime_expression".

Syntax
next_day ( datetime_expression , string_expression )

nvl
Returns "expression" if not null, otherwise returns "constant". Valid for "numeric_expression", "string_expression", "date_expression", and "time_expression".

Syntax
nvl ( expression , constant )

round
Returns "numeric_expression" rounded to the nearest value "integer_expression" places right of the decimal point. If "integer_expression" is negative, "numeric_expression" is rounded to the nearest absolute value "integer_expression" places to the left of the decimal point; for example, round (125, -1) rounds to 130.

Syntax
round ( numeric_expression [ , integer_expression ] )
Appendix B: Using the Expression Editor

**rpad**

Returns "string_expression1" right-padded to length "integer_expression" with occurrences of "string_expression2". If "string_expression1" is longer than "integer_expression", the appropriate portion of "string_expression1" is returned. If "string_expression2" is not specified, then spaces are used.

**Syntax**

```
rpad ( string_expression1 , integer_expression [ , string_expression2 ] )
```
### to_date
Converts "string_expression1" to a datetime value as specified by the format "string_expression2".

**Syntax**
to_date ( string_expression1 , string_expression2 )

### to_number
Converts "string_expression1" to a numeric value as specified by the format "string_expression2".

**Syntax**
to_number ( string_expression1 , string_expression2 )

### translate
Returns "string_expression1", with all occurrences of each character in "string_expression2" replaced by its corresponding character in "string_expression3".

**Syntax**
translate ( string_expression1 , string_expression2 , string_expression3 )

### date_trunc
Truncates "date_expression1" to a value as specified by the format "string_expression1".

**Syntax**
date_trunc ( string_expression1 , date_expression1 )

### trunc
Truncates digits from "numeric_expression1" using "numeric_expression2" as the precision.

**Syntax**
trunc ( numeric_expression1 [ , numeric_expression2 ] )

### version
Returns the "string_expression1" value of the database version.

**Syntax**
version ()

### Netezza Math

#### log
Returns the logarithm of "numeric_expression2" to the base "numeric_expression1".

**Syntax**
log ( numeric_expression1 , numeric_expression2 )
Netezza Trigonometry

acos
Returns the arccosine of "numeric_expression" in radians. The arccosine is the angle whose cosine is "numeric_expression".

Syntax
acos ( numeric_expression )

asin
Returns the arcsine of "numeric_expression" in radians. The arcsine is the angle whose sine is "numeric_expression".

Syntax
asin ( numeric_expression )

atan
Returns the arctangent of "numeric_expression" in radians. The arctangent is the angle whose tangent is "numeric_expression".

Syntax
atan ( numeric_expression )

atan2
Returns the arctangent of the x and y coordinates specified by "numeric_expression1" and "numeric_expression2", respectively, in radians. The arctangent is the angle whose tangent is "numeric_expression2" / "numeric_expression1".

Syntax
atan2 ( numeric_expression1 , numeric_expression2 )

cos
Returns the cosine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax
cos ( numeric_expression )

degrees
Returns the degrees where "numeric_expression" is an angle expressed in radians.

Syntax
degrees ( numeric_expression )

radians
Returns the radians where "numeric_expression" is an angle expressed in degrees.
Syntax
radians ( numeric_expression )

sin
Returns the sine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax
sin ( numeric_expression )

tan
Returns the tangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax
tan ( numeric_expression )

Netezza Fuzzy

le_dst
Returns a value indicating how different the two input strings are, calculated according to the Levenshtein edit distance algorithm.

Syntax
le_dst ( string_expression1 , string_expression2 )

dle_dst
Returns a value indicating how different the two input strings are, calculated according to the Damerau-Levenshtein distance algorithm.

Syntax
dle_dst ( string_expression1 , string_expression2 )

Netezza Phonetic

nysiis
Returns a Soundex representation of "string_expression" using the New York State Identification and Intelligence System (NYSIIS) variation of Soundex.

Syntax
nysiis ( string_expression )

dbl_mp
Returns a composite 32-bit value of "string_expression".

Syntax
dbl_mp ( string_expression )
Appendix B: Using the Expression Editor

pri_mp
Returns the 4 character primary metaphone string from "numeric_expression" returned by dbl_mp.

Syntax
pri_mp ( numeric_expression )

sec_mp
Returns the 4 character secondary metaphone string from "numeric_expression" returned by dbl_mp.

Syntax
sec_mp ( numeric_expression )

score_mp
Returns a score for how closely "numeric_expression" and "numeric_expression2" match.

Syntax
score_mp ( numeric_expression , numeric_expression2 , numeric_expression3 , numeric_expression4 , numeric_expression5 , numeric_expression6 )

Oracle

add_months
Returns the datetime resulting from adding "integer_expression" months to "date_expression".

Syntax
add_months ( date_expression , integer_expression )

ascii
Returns a number representing the ASCII code value of the leftmost character of "string_expression".

Syntax
ascii ( string_expression )

Example
ascii ( 'A' )
Result: Returns '65'

ceil
Returns the smallest integer greater than or equal to "numeric_expression".

Syntax
ceil ( numeric_expression )
char_length

Returns the number of logical characters in "string_expression". The number of logical characters can be distinct from the number of bytes in some East Asian locales.

**Syntax**

\`char_length ( string_expression )\`

chr

Returns the character that has the ASCII code value specified by "integer_expression". "Integer_expression" should be between 0 and 255.

**Syntax**

\`chr ( integer_expression )\`

concat

Returns a string that is the result of concatenating, or joining, "string_expression1" to "string_expression2".

**Syntax**

\`concat ( string_expression1 , string_expression2 )\`

**Example**

\`concat ( [Sales (query)].[Sales staff].[First name], [Sales (query)].[Sales staff].[Last name] )\`

Result: Returns the first name and last name; e.g., Bob Smith.

decode

Compares "expression" to each search value one by one. If "expression" is equal to a search, then it returns the corresponding result. If no match is found, it returns "default", or if "default" is omitted, it returns null.

**Syntax**

\`decode ( expression , search , result [ , search , result ]... [ , default ] )\`

dump

Returns internal representation of "expression" with the format of "numeric_expression1" starting from position "numeric_expression2" for "numeric_expression3" characters.

**Syntax**

\`dump ( expression [ , numeric_expression1 [ , numeric_expression2 [ , numeric_expression3 ] ] ] )\`

greatest

Returns the greatest value in "expression_list".

**Syntax**

\`greatest ( expression_list )\`
initcap

Returns "string_expression" with the first letter of each word in uppercase and all other letters in lowercase. Words are delimited by white space or characters that are not alphanumeric.

**Syntax**

initcap ( string_expression )

instr

Searches "string_expression1" starting at position "integer_expression1" for the "integer_expression2" occurrence of "string_expression2". If "integer_expression1" is negative, then the search occurs backwards from the end of "string_expression1". Returns an integer indicating the position of "string_expression2".

**Syntax**

instr ( string_expression1, string_expression2 [ , integer_expression1 [ , integer_expression2 ] ] )

instrb

Searches "string_expression1" starting at position "integer_expression1" for the "integer_expression2" occurrence of "string_expression2". If "integer_expression1" is negative, then the search occurs backwards from the end of "string_expression1". Returns the position (byte number) where "string_expression2" was found.

**Syntax**

instrb ( string_expression1, string_expression2 [ , integer_expression1 [ , integer_expression2 ] ] )

least

Returns the least value in "expression_list".

**Syntax**

least ( expression_list )

length

Returns the number of characters in "string_expression".

**Syntax**

length ( string_expression )

lengthb

Returns the number of bytes in "string_expression".

**Syntax**

lengthb ( string_expression )
**lpad**

Returns "string_expression1" left-padded to the length defined by "integer_expression" with occurrences of "string_expression2". If "string_expression1" is longer than "integer_expression", the appropriate portion of "string_expression1" is returned.

**Syntax**

\[ lpad \ ( \ string\_expression1, \ integer\_expression \ [ , \ string\_expression2 \ ] ) \]

**ltrim**

Returns "string_expression1" with leading characters removed up to the first character not in "string_expression2".

**Syntax**

\[ ltrim \ ( \ string\_expression1 \ [ , \ string\_expression2 \ ] ) \]

**Example**

\[ ltrim \ ( \ 'xyxXxyAB' , \ 'xy' ) \]

Result: XxyAB

**months_between**

Returns the number of months from "date_expression1" to "date_expression2". If "date_expression1" is later than "date_expression2" then the result will be a positive number. The days and time portion of the difference are ignored, so the months are not rounded unless "date_expression1" and "date_expression2" are the last days of a month.

**Syntax**

\[ months\_between \ ( \ date\_expression1 , \ date\_expression2 ) \]

**new_time**

Returns the datetime in "new_timezone" for "datetime_expression" in "old_timezone". "Old_timezone" and "new_timezone" can be one of 'AST', 'ADT', 'BST', 'BDT', 'CST', 'CDT', 'EST', 'EDT', 'HST', 'HDT', 'MST', 'MDT', 'NST', 'PST', 'PDT', 'YST', or 'YDT'.

**Syntax**

\[ new\_time \ ( \ datetime\_expression , \ old\_timezone , \ new\_timezone ) \]

**next_day**

Returns the datetime of the first weekday named by "string_expression" that is later than "datetime_expression". The return value has the same format as "datetime_expression".

**Syntax**

\[ next\_day \ ( \ datetime\_expression , \ string\_expression ) \]

**nls_initcap**

Returns "string_expression1" with the first letter of each word in uppercase and all other letters in lowercase. A word begins after any character other than a letter. Thus, in addition to a blank...
space, symbols such as commas, periods, and colons can introduce a new word. "String_expression2" specifies the sorting sequence.

**Syntax**

```
nls_initcap ( string_expression1 [ , string_expression2 ] )
```

**nls_lower**

Returns "string_expression1" with all letters in lowercase. "String_expression2" specifies the sorting sequence.

**Syntax**

```
nls_lower ( string_expression1 [ , string_expression2 ] )
```

**nls_upper**

Returns "string_expression1" with all letters in uppercase. "String_expression2" specifies the sorting sequence.

**Syntax**

```
nls_upper ( string_expression1 [ , string_expression2 ] )
```

**nvl**

Returns "expression" unless it is null. If "expression" is null, returns "constant". Valid for "numeric_expression", "string_expression", "date_expression", and "time_expression".

**Syntax**

```
nvl ( expression , constant )
```

**Example**

```
nvl ( [Unit sale price] , [Unit price] )
```

Result: Returns the unit sale price, or returns the unit price if the unit sale price is NULL.

**replace**

Replaces all occurrences of "string_expression2" in "string_expression1" with "string_expression3". If "string_expression3" is not specified, then it removes all occurrences of "string_expression2".

**Syntax**

```
replace ( string_expression1 , string_expression2 [ , string_expression3 ] )
```

**round**

Returns "numeric_expression" rounded to the nearest value "integer_expression" places right of the decimal point. If "integer_expression" is negative, "numeric_expression" is rounded to the nearest absolute value "integer_expression" places to the left of the decimal point. Rounding takes place before data formatting is applied.

**Syntax**

```
round ( numeric_expression [ , integer_expression ] )
```
Example
round ( 125 , -1 )
Result: Returns 130

rpad

Returns "string_expression1" right-padded to length "integer_expression" with occurrences of "string_expression2". If "string_expression1" is longer than "integer_expression", the appropriate portion of "string_expression1" is returned. If "string_expression2" is not specified, then occurrences of "string_expression2" are replaced with spaces.

Syntax
rpad ( string_expression1 , integer_expression [ , string_expression2 ] )

rtrim

Returns "string_expression1" with the final characters removed after the last character not in "string_expression2". If "string_expression2" is not specified, the final space characters are removed.

Syntax
rtrim ( string_expression1 [ , string_expression2 ] )

Example
rtrim ( 'ABxXyx' , 'xy' )
Result: Returns 'ABxX'

sign

Returns an indicator of the sign of "numeric_expression", +1 if positive, 0 if zero, or -1 if negative.

Syntax
sign ( numeric_expression )

soundex

Returns a character string containing the phonetic representation of "string_expression".

Syntax
soundex ( string_expression )

substr

Returns the substring of "string_expression" that starts at position "integer_expression1" for "integer_expression2" characters or to the end of "string_expression" if "integer_expression2" is omitted. The first character in "string_expression" is at position 1.

Syntax
substr ( string_expression , integer_expression1 [ , integer_expression2 ] )

Example
substr ( [Sales (query)].[Sales staff].[Position code], 3 , 5 )
Appendix B: Using the Expression Editor

Result: Returns characters 3 to 7 of the position codes.

**subr**

Returns the substring of "string_expression" that starts at position "numeric_expression1" and ends after "numeric_expression2" bytes. The first byte in "string_expression" is at position 1. If you omit "numeric_expression2", returns the substring of "string_expression" that starts at position "numeric_expression1" and ends at the end of "string_expression".

**Syntax**

```
substrb ( string_expression , numeric_expression1 [ , numeric_expression2 ] )
```

**Example**

```
substrb ( [Sales (query)].[Sales staff].[Position code], 3 , 5 )
```

Result: Returns characters 3 to 7 of the position codes.

**{sysdate}**

Returns a datetime value representing the current date and time of the computer that the database software runs on.

**Syntax**

```
{ sysdate }
```

**to_char**

Returns the string representation of "expression" with the format of "string_expression". "Expression" can be either a date value or a numeric value.

**Syntax**

```
to_char ( expression [ , string_expression ] )
```

**to_date**

Converts "string_expression1" to a datetime value as specified by the format "string_expression2". "String_expression3" specifies the format elements, such as language.

**Syntax**

```
to_date ( string_expression1 [ , string_expression2 [ , string_expression3 ] ] )
```

**to_number**

Converts "string_expression1" to a numeric value as specified by the format "string_expression2". "String_expression3" specifies the format elements, such as currency information.

**Syntax**

```
to_number ( string_expression1 , string_expression2 , string_expression3 )
```

**translate**

Returns "string_expression1" with all occurrences of each character in "string_expression2" replaced by the corresponding character in "string_expression3".
**Syntax**
`translate ( string_expression1 , string_expression2 , string_expression3 )`

**trunc**
Truncates "date_expression" using the format specified by "string_expression". For example, if "string_expression" is 'year', then "date_expression" is truncated to the first day of the year.

**Syntax**
`trunc ( date_expression , string_expression )`

**Example**
`trunc ( 2003-08-22 , 'year' )`
**Result:** Returns 2003-01-01.

**trunc**
Truncates digits from "numeric_expression1" using "numeric_expression2" as the precision.

**Syntax**
`trunc ( numeric_expression1 , numeric_expression2 )`

**{user}**
Returns the username of the current Oracle user.

**Syntax**
`{ user }`

**vsize**
Returns the number of bytes in the internal representation of "expression". "Expression" must be a string expression.

**Syntax**
`vsize ( expression )`

**Oracle Math**

**log**
Returns the logarithm of "numeric_expression2" to the base "numeric_expression1".

**Syntax**
`log ( numeric_expression1 , numeric_expression2 )`

**Oracle Trigonometry**

**acos**
Returns the arccosine of "numeric_expression" in radians. The arccosine is the angle whose cosine is "numeric_expression".
Appendix B: Using the Expression Editor

**Syntax**

`acos ( numeric_expression )`

**asin**

Returns the arcsine of "numeric_expression" in radians. The arcsine is the angle whose sine is "numeric_expression".

**Syntax**

`asin ( numeric_expression )`

**atan**

Returns the arctangent of "numeric_expression" in radians. The arctangent is the angle whose tangent is "numeric_expression".

**Syntax**

`atan ( numeric_expression )`

**atan2**

Returns the arctangent of the x and y coordinates specified by "numeric_expression1" and "numeric_expression2", respectively, in radians. The arctangent is the angle whose tangent is "numeric_expression2" / "numeric_expression1".

**Syntax**

`atan2 ( numeric_expression1 ,numeric_expression2 )`

**cos**

Returns the cosine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

**Syntax**

`cos ( numeric_expression )`

**cosh**

Returns the hyperbolic cosine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

**Syntax**

`cosh ( numeric_expression )`

**sin**

Returns the sine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

**Syntax**

`sin ( numeric_expression )`
**sinh**  
Returns the hyperbolic sine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

**Syntax**  
sinh ( numeric_expression )

**tan**  
Returns the tangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

**Syntax**  
tan ( numeric_expression )

**tanh**  
Returns the hyperbolic tangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

**Syntax**  
tanh ( numeric_expression )

---

**Red Brick**

**ceil**  
Returns the smallest integer greater than or equal to "numeric_expression" or "string_expression". Note that "string_expression" must represent a valid numeric value.

**Syntax**  
ceil ( numeric_expression|string_expression )

**concat**  
Returns a string that is the result of concatenating, or joining, "string_expression1" to "string_expression2".

**Syntax**  
concat ( string_expression1 , string_expression2 )

**Example**  
concat ( [Sales (query)].[Sales staff].[First name], [Sales (query)].[Sales staff].[Last name] )

Result: Returns the first name and last name; e.g., Bob Smith.

**{current_user}**  
Returns the database username (authorization ID) of the current user.
Syntax
{ current_user }

date

Returns a date value. "Expression" can be either characters or a timestamp.

Syntax
date ( expression )

dateadd

Adds "interval" to "datetime_expression" and returns a result that is the same datetime data type as "datetime_expression". "Datepart" refers to the year, month, day, hour, minute, second. "Interval" must be an integer and "datetime_expression" can be a date, time, or timestamp.

Syntax
dateadd ( { datepart }, interval, datetime_expression )

datediff

Determines the difference between two datetime expressions and returns an integer result in "datepart" units. "Datepart" refers to a year, month, day, hour, minute, or second. "DateTime_expression1" and "datetime_expression2" can be dates, times, or timestamps.

Syntax
datediff ( { datepart }, datetime_expression1, datetime_expression2 )

datename

Extracts "datepart" of "datetime_expression" and returns its value as a character string. "Datepart" refers to a year, month, day, hour, minute, or second. "DateTime_expression" can be a date, a time, or a timestamp.

Syntax
datename ( { datepart }, datetime_expression )

dec

Converts "expression" to a decimal value with the data type decimal (precision, scale). The default value of precision is 9. The default value of scale is 0.

Syntax
dec ( expression, [ precision, scale ] )

decimal

Converts "expression" to a decimal value with the data type decimal (precision, scale). The default value of precision is 9. The default value of scale is 0.

Syntax
decimal ( expression, [ precision, scale ] )
decode

Compares and converts "expression" to another value. If "expression" matches "target", it is replaced, otherwise it is replaced by "default" or null if no default is specified. The expressions can be any data type as long as they are all the same data type.

**Syntax**

```plaintext
decode ( expression , target , replacement [ ,default ] )
```  

float

Converts "numeric_expression" into a double-precision floating-point value.

**Syntax**

```plaintext
float ( numeric_expression )
```  

ifnull

Tests "expression" for missing values and replaces each one with "substitute". If "expression" is null, "substitute" is returned, otherwise it returns the value of "expression". The expressions can be any data type as long as they are all the same data type.

**Syntax**

```plaintext
ifnull ( expression, substitute )
```  

int

Converts "numeric_expression" into an integer value and returns an integer value. If "numeric_expression" is null, it returns null.

**Syntax**

```plaintext
int ( numeric_expression )
```  

integer

Converts "numeric_expression" into an integer value and returns an integer value. If "numeric_expression" is null, it returns null.

**Syntax**

```plaintext
integer ( numeric_expression )
```  

Example

```plaintext
integer ( 84.95 )
```

Result: 85

length

Returns an integer result specifying the number of characters in "string_expression". If "string_expression" is null, it returns null.

**Syntax**

```plaintext
length ( string_expression )
```
lengthb

Returns an integer result specifying the number of bytes in "string_expression". If "string_expression" is null, it returns null.

**Syntax**

```sql
lengthb ( string_expression )
```

ltrim

Removes leading blanks from "string_expression". If "string_expression" is null, it returns null.

**Syntax**

```sql
ltrim ( string_expression )
```

nullif

Returns null if both "expression1" and "expression2" have the same value. If they have different values, the value of "expression1" is returned. "Expression1" and "expression2" can be any data type as long as they are the same data type.

**Syntax**

```sql
nullif ( expression1 , expression2 )
```

positionb

Returns an integer that is relative to the beginning byte position of "string_expression1" in "string_expression2". If "string_expression1" is not located, the result is 0. If "string_expression1" is of zero length, the result is 1. If "string_expression1" is null, an error message is returned. If "string_expression2" is null, the result is 0.

**Syntax**

```sql
positionb ( string_expression1, string_expression2 )
```

real

Returns a real value. If "numeric_expression" is null, it returns null.

**Syntax**

```sql
real ( numeric_expression )
```

round

Returns "numeric_expression" rounded to the nearest value "integer_expression" places to the right of the decimal point. If "integer_expression" is negative, "numeric_expression" is rounded to the nearest absolute value "integer_expression" places to the left of the decimal point. Rounding takes place before data formatting is applied.

**Syntax**

```sql
round ( numeric_expression , integer_expression )
```
Example
round (125, -1)
Result: 130

rtrim
Removes trailing blanks from "string_expression". If "string_expression" is null, it returns null.

Syntax
rtrim ( string_expression )

Example
rtrim ( [Sales (query)].[Sales staff].[Last name] )
Result: Returns last names with any spaces at the end of the name removed.

sign
Determines the sign of "numeric_expression", and returns 1 for a positive value, –1 for a negative value, and 0 for zero.

Syntax
sign ( numeric_expression )

string
Converts "expression" to a character string. "Expression" can be either numeric or datetime.

Syntax
string ( expression [ , length [ , scale ] ] )

substr
Returns a substring of "string_expression" that begins at position "start_integer" and continues for "length_integer" characters. If "length_integer" is not specified, a substring from "start_integer" to the end of "string_expression" is returned.

Syntax
substr ( string_expression, start_integer, length_integer )

Example
substr ( [Sales (query)].[Sales staff].[Position code], 3, 5 )
Result: Returns characters 3 to 7 of the position codes.

substrb
Returns a substring of "string_expression" that begins at position "start_integer" and continues for "length_integer" bytes. If "length_integer" is not specified, a substring from "start_integer" to the end of "string_expression" is returned.

Syntax
substrb ( string_expression, start_integer, length_integer )
time

Creates a time value from "expression", which can be a character string or a time-stamp data type expression.

**Syntax**

```plaintext
time ( expression )
```

timestamp

Creates a time-stamp value from "timestamp_expression", which is a character string.

**Syntax**

```plaintext
timestamp ( timestamp_expression )
```

timestamp

Creates a time-stamp value from "time_expression" and "date_expression". If either "time_expression" or "date_expression" is null, the resulting time-stamp expression is also null.

**Syntax**

```plaintext
timestamp ( date_expression , time_expression )
```

to_char

Converts "source_date" to the character string specified by "format_string". "Source_date" can be a date, time, or timestamp data type.

**Syntax**

```plaintext
to_char ( source_date, format_string )
```

**SQL Server**

ascii

Returns a number representing the ascii code value of the leftmost character of "string_expression".

**Syntax**

```plaintext
ascii ( string_expression )
```

**Example**

```plaintext
ascii ( 'A' )
```

Result: 65

char

Returns the character that has the ASCII code value specified by "integer_expression". "Integer_expression" should be between 0 and 255.

**Syntax**

```plaintext
char ( integer_expression )
```
**Example**

char ( 65 )

Result: A

**charindex**

Searches "string_expression2" for the first occurrence of "string_expression1" and returns an integer. "Start_location" is the character position to start searching for "string_expression1" in "string_expression2". If "start_location" is not specified, is a negative number, or is zero, the search starts at the beginning of "string_expression2".

**Syntax**

`charindex ( string_expression1 , string_expression2 [ , start_location ] )`

**{current_user}**

Returns the name of the current user.

**Syntax**

`{ current_user }`

**datalength**

Returns the length in bytes of "string_expression".

**Syntax**

`datalength ( string_expression )`

**dateadd**

Returns the date resulting from the addition of "integer_expression" units (indicated by "datepart" (day, month, year)) to "date_expression".

**Syntax**

`dateadd ( { datepart } , integer_expression , date_expression )`

**datediff**

Returns the number of "datepart" (day, month, year) units between "date_expression1" and "date_expression2".

**Syntax**

`datediff ( {datepart} , date_expression1 , date_expression2 )`

**Example**

datediff ( {yy} , 1984-01-01 , 1997-01-01 )

Result: 13
datename

Returns "datepart" from "date_expression", which can be a datetime, smalldatetime, date, or time value as an ASCII string. Note that "datepart" must be a keyword representing a datepart or its abbreviation recognized by Microsoft® SQL Server and must be enclosed in curly brackets.

Syntax

\[
\text{datename} \left( \{' \ ' \text{datepart} \ ' \}' , \text{date_expression} \right)
\]

Example

datename \( \{\text{mm}\} , 2000-01-01 \)

Result: January

datepart

Returns part of "date_expression" (for example, the month) as an integer. "date_expression" can be a datetime, smalldatetime, date, or time value. Note that "datepart" must be a keyword representing a datepart or its abbreviation recognized by Microsoft® SQL Server and must be enclosed in curly brackets.

Syntax

\[
\text{datepart} \left( \{' \ ' \text{datepart} \ ' \}' , \text{date_expression} \right)
\]

Example

datepart \( \{\text{wk}\} , 2000-01-01 \)

Result: 1 (first week of the year)

day

Returns the day portion of "date_expression". Same as extract (day from date_expression).

Syntax

\[
\text{day} \ ( \text{date_expression} )
\]

difference

Returns an integer value representing the difference between the values returned by the data source-specific soundex function for "string_expression1" and "string_expression2". The value returned ranges from 0 to 4, with 4 indicating the best match. Note that 4 does not mean that the strings are equal.

Syntax

\[
\text{difference} \ ( \text{string_expression1} , \text{string_expression2} )
\]

Example 1

difference ([Sales target (query)].[Sales Staff].[First name],[Sales (query)].[Retailers].[Contact first name])

Result: 0
Example 2

difference ([Sales target (query)].[Sales Staff].[First name],[Sales target (query)].[Sales Staff].[First name])

Result: 4

getdate

Returns a datetime value representing the current date and time of the computer that the database software runs on.

Syntax

getdate ()

left

Returns the leftmost "integer_expression" characters of "string_expression".

Syntax

left ( string_expression , integer_expression )

Example

left ( [Sales (query)].[Sales staff].[Last name] , 3 )

Result: Returns the first three characters of each last name.

ltrim

Returns "string_expression" with leading spaces removed.

Syntax

ltrim ( string_expression )

month

Returns the month portion of "date_expression". Same as extract (month from date_expression).

Syntax

month ( date_expression )

patindex

Returns an integer that represents the starting position of the first occurrence of "string_expression1" in the "string_expression2". Returns 0 if "string_expression1" is not found. The % wildcard character must precede and follow "string_expression1", except when searching for first or last characters.

Syntax

patindex ( string_expression1 , string_expression2 )

Example

patindex ( '%po%', 'Report' )

Result: 3
replace

Replaces all occurrences of "string_expression2" in "string_expression1" with "string_expression3".

Syntax

replace ( string_expression1 , string_expression2 , string_expression3 )

replicate

Returns a string consisting of "string_expression" repeated "integer_expression" times.

Syntax

replicate ( string_expression , integer_expression )

reverse

Returns "string_expression" in reverse order.

Syntax

reverse ( string_expression )

right

Returns the rightmost "integer_expression" characters of "string_expression".

Syntax

right ( string_expression , integer_expression )

round

Returns "numeric_expression" rounded to the nearest value "integer_expression" places to the right of the decimal point. Rounding takes place before data formatting is applied.

Syntax

round ( numeric_expression , integer_expression )

Example

round (125, -1)
Result: 130

rtrim

Returns "string_expression" with trailing spaces removed.

Syntax

rtrim ( string_expression )

Example

rtrim ( [Sales (query)].[Sales staff].[Last name] )
Result: Returns last names with any spaces at the end of the name removed.
sign

Returns an indicator of the sign "numeric_expression": +1 if "numeric_expression" is positive, 0 if zero or -1 if negative.

Syntax

```
sign ( numeric_expression )
```

soundex

Returns a four character string representing the sound of the words in "string_expression".

Syntax

```
soundex ( string_expression )
```

space

Returns a string consisting of "integer_expression" spaces.

Syntax

```
space ( integer_expression )
```

str

Returns a string representation of "numeric_expression" where "integer_expression1" is the length of the string returned and "integer_expression2" is the number of decimal digits.

Syntax

```
str ( numeric_expression [ , integer_expression1 [ , integer_expression2 ] ] )
```

stuff

Returns a string where "integer_expression2" characters have been deleted from "string_expression1" beginning at "integer_expression1", and where "string_expression2" has been inserted into "string_expression1" at its start. The first character in a string is at position 1.

Syntax

```
stuff ( string_expression1 , integer_expression1 , integer_expression2 , string_expression2 )
```

year

Returns the year portion of "date_expression". Same as extract (year from date_expression).

Syntax

```
year ( date_expression )
```

**SQL Server Math**

log

Returns the natural logarithm of "numeric_expression".
Appendix B: Using the Expression Editor

**Syntax**

\[
\text{log} \ ( \text{numeric-expression} )
\]

**log10**

Returns the base ten logarithm of "numeric-expression".

**Syntax**

\[
\text{log10} \ ( \text{numeric-expression} )
\]

**pi**

Returns the constant value of pi as a floating point value.

**Syntax**

\[
\text{pi} ()
\]

**rand**

Generates a random number using "integer-expression" as the seed value.

**Syntax**

\[
\text{rand} \ ( \text{integer-expression} )
\]

**SQL Server Trigonometry**

**acos**

Returns the arccosine of "numeric-expression" in radians. The arccosine is the angle whose cosine is "numeric-expression".

**Syntax**

\[
\text{acos} \ ( \text{numeric-expression} )
\]

**asin**

Returns the arcsine of "numeric-expression" in radians. The arcsine is the angle whose sine is "numeric-expression".

**Syntax**

\[
\text{asin} \ ( \text{numeric-expression} )
\]

**atan**

Returns the arctangent of "numeric-expression" in radians. The arctangent is the angle whose tangent is "numeric-expression".

**Syntax**

\[
\text{atan} \ ( \text{numeric-expression} )
\]
atn2
Returns the arctangent of the x and y coordinates specified by "numeric_expression1" and "numeric_expression2", respectively, in radians. The arctangent is the angle whose tangent is "numeric_expression1".

Syntax
atn2 ( numeric_expression1, numeric_expression2 )

cos
Returns the cosine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax
cos ( numeric_expression )

cot
Returns the cotangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax
cot ( numeric_expression )

degrees
Returns "numeric_expression" radians converted to degrees.

Syntax
degrees ( numeric_expression )

radians
Returns the number of radians converted from "numeric_expression" degrees.

Syntax
radians ( numeric_expression )

sin
Returns the sine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax
sin ( numeric_expression )

tan
Returns the tangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.
Appendix B: Using the Expression Editor

**Syntax**
\[
\tan ( \text{numeric_expression} )
\]

**Teradata**

**account**

Returns the account string for the current user.

**Syntax**
\[
(\text{account})
\]

**add_months**

Returns the date or the datetime resulting from adding "integer_expression" months to "date_expression" or "datetime_expression".

**Syntax**
\[
\text{add_months} ( \text{date_expression|datetime_expression} , \text{integer_expression} )
\]

**bytes**

Returns the number of bytes contained in "byte_expression". "Byte_expression" is restricted to BYTE or VARBYTE.

**Syntax**
\[
\text{bytes} ( \text{byte_expression} )
\]

**case_n**

Evaluates "condition_expression_list" and returns the position of the first true condition, provided that no prior condition in the list evaluates to unknown. The keywords must be enclosed in curly brackets. No case is an optional condition that evaluates to true if every expression in the list evaluates to false. No case or unknown is an optional condition that evaluates to true if every expression in the list evaluates to false, or if an expression evaluates to unknown and all prior conditions in the list evaluate to false. Unknown is an optional condition that evaluates to true if an expression evaluates to unknown and all prior conditions in the list evaluate to false.

**Syntax**
\[
\text{case}_n ( \text{condition_expression_list} [ , \text{NO CASE|UNKNOWN|NO CASE OR UNKNOWN} [ , \text{UNKNOWN} ] ] )
\]

**char2hexint**

Returns the hexadecimal representation for "string_expression".

**Syntax**
\[
\text{char2hexint} ( \text{string_expression} )
\]
characters

Returns an integer value representing the number of logical characters or bytes contained in "string_expression".

Syntax
characters ( string_expression )

database

Returns the name of the default database for the current user.

Syntax
database

date

Returns the current date.

Syntax
date

format

Returns the declared format for "expression" as a variable character string of up to 30 characters.

Syntax
format ( expression )

index

Returns the starting position of "string_expression2" in "string_expression1".

Syntax
index ( string_expression1 , string_expression2 )

log

Computes the base 10 logarithm of "numeric_expression". "Numeric_expression" must be a non-zero, positive, numeric expression.

Syntax
log ( numeric_expression )

nullif

Returns null if "scalar_expression1" and "scalar_expression2" are equal. Otherwise, it returns "scalar_expression1". "Scalar_expression1" and "scalar_expression2" can be any data type.

Syntax
nullif ( scalar_expression1 , scalar_expression2 )
nullifzero

If "numeric_expression" is zero, converts it to null to avoid division by zero.

**Syntax**

nullifzero ( numeric_expression )

profile

Returns the current profile for the session or null if none.

**Syntax**

{profile}

random

Returns a random integer number for each row of the results table. "Lower_bound" and "upper_bound" are integer constants. The limits for "lower_bound" and "upper_bound" range from -2147483648 to 2147483647 inclusive. "Upper_bound" must be greater than or equal to "lower_bound".

**Syntax**

random ( lower_bound , upper_bound )

role

Returns the current role for the session or null if none.

**Syntax**

{role}

session

Returns the number of the session for the current user.

**Syntax**

{session}

soundex

Returns a character string that represents the Soundex code for "string_expression".

**Syntax**

soundex ( string_expression )

substr

Returns the substring of "string_expression" that starts at position "integer_expression1" for "integer_expression2" characters. The first character in "string_expression" is at position 1. If "integer_expression2" is omitted, returns the substring of "string_expression" that starts at position "integer_expression1" and ends at the end of "string_expression".
Syntax
substr ( string_expression , integer_expression1 [ , integer_expression2 ] )

Example
substr ( [Sales (query)].[Sales staff].[Position code], 3 , 5 )
Result: Returns characters 3 to 7 of the position codes.

time
Returns the current time based on a 24-hour day.

Syntax
{}time{}

type
Returns the data type defined for "expression".

Syntax
type ( expression )

user
Returns the user name of the current user.

Syntax
{}user{}

vargraphic
Returns a character string that represents the vargraphic code for "string_expression".

Syntax
vargraphic ( string_expression )

zeroifnull
Converts data from null to 0 to avoid errors created by a null value. If "numeric_expression" is not null, returns the value of "numeric_expression". If "numeric_expression" is a character string, it is converted to a numeric value of float data type. If "numeric_expression" is null or zero, it returns zero.

Syntax
zeroifnull ( numeric_expression )

Teradata Trigonometry

acos
Returns the arccosine of "numeric_expression" in radians. The arccosine is the angle whose cosine is "numeric_expression". "Numeric_expression" must be between -1 and 1, inclusive.
**Syntax**

acos ( numeric_expression )

**acosh**
Returns the inverse hyperbolic cosine of "numeric_expression" where "numeric_expression" can be any real number equal to or greater than 1.

**Syntax**

acosh ( numeric_expression )

**asin**
Returns the arcsine of "numeric_expression" in radians. The arcsine is the angle whose sine is "numeric_expression". "Numeric_expression" must be between -1 and 1, inclusive.

**Syntax**

asin ( numeric_expression )

**asinh**
Returns the inverse hyperbolic sine of "numeric_expression" where "numeric_expression" can be any real number.

**Syntax**

asinh ( numeric_expression )

**atan**
Returns the arctangent of "numeric_expression" in radians where the arctangent is the angle whose tangent is "numeric_expression".

**Syntax**

atan ( numeric_expression )

**atan2**
Returns the arctangent of the x and y coordinates specified by "numeric_expression1" and "numeric_expression2", respectively, in radians. The returned angle will be between - and π radians, excluding π.

**Syntax**

atan2 ( numeric_expression1, numeric_expression2 )

**atanh**
Returns the inverse hyperbolic tangent of "numeric_expression" where "numeric_expression" can be any real number between 1 and -1, excluding 1 and -1.

**Syntax**

atanh ( numeric_expression )
\textbf{cos}  
Returns the cosine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

\textbf{Syntax}
\begin{equation*}
cos \ ( \text{numeric\_expression} )
\end{equation*}

\textbf{cosh}  
Returns the hyperbolic cosine of "numeric_expression" where "numeric_expression" can be any real number.

\textbf{Syntax}
\begin{equation*}
cosh \ ( \text{numeric\_expression} )
\end{equation*}

\textbf{sin}  
Returns the sine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

\textbf{Syntax}
\begin{equation*}
sin \ ( \text{numeric\_expression} )
\end{equation*}

\textbf{sinh}  
Returns the hyperbolic sine of "numeric_expression" where "numeric_expression" can be any real number.

\textbf{Syntax}
\begin{equation*}
sinh \ ( \text{numeric\_expression} )
\end{equation*}

\textbf{tan}  
Returns the tangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

\textbf{Syntax}
\begin{equation*}
tan \ ( \text{numeric\_expression} )
\end{equation*}

\textbf{tanh}  
Returns the hyperbolic tangent of "numeric_expression" where "numeric_expression" can be any real number.

\textbf{Syntax}
\begin{equation*}
tanh \ ( \text{numeric\_expression} )
\end{equation*}
SAP BW

SAP BW Trigonometry

**arccos**
Returns the arccosine of "numeric_expression" in radians. The arccosine is the angle whose cosine is "numeric_expression".

**Syntax**
arccos ( numeric_expression )

**arcsin**
Returns the arcsine of "numeric_expression" in radians. The arcsine is the angle whose sine is "numeric_expression".

**Syntax**
arcsin ( numeric_expression )

**arctan**
Returns the arctangent of "numeric_expression" in radians. The arctangent is the angle whose tangent is "numeric_expression".

**Syntax**
arctan ( numeric_expression )

**cos**
Returns the cosine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

**Syntax**
cos ( numeric_expression )

**sin**
Returns the sine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

**Syntax**
sin ( numeric_expression )

**tan**
Returns the tangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

**Syntax**
tan ( numeric_expression )
coshyp
Returns the hyperbolic cosine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

coshyp ( numeric_expression )

sinhyp
Returns the hyperbolic sine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

sinhyp ( numeric_expression )

tanhyp
Returns the hyperbolic tangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

tanhyp ( numeric_expression )

SAP BW Math

log10
Returns the base ten logarithm of "numeric_expression".

Syntax

log10 ( numeric_expression )

Sybase

ascii
Returns a number representing the ascii code value of the leftmost character of "string_expression".

Syntax

ascii ( string_expression )

Example

ascii ( 'A' )

Result: 65

char
Converts "integer_expression" to a character value. Char is usually used as the inverse of ascii where "integer_expression" must be between 0 and 255. If the resulting value is the first byte of a multibyte character, the character may be undefined.
Appendix B: Using the Expression Editor

**Syntax**
\[
\text{char ( integer_expression )}
\]

**charindex**

Returns an integer that represents the starting position of "string_expression1" within "string_expression2". If "string_expression1" is not found, zero is returned. If "string_expression1" contains wildcard characters, charindex treats them as literals.

**Syntax**
\[
\text{charindex ( string_expression1 , string_expression2 )}
\]

**datalength**

Returns the length in bytes of "string_expression".

**Syntax**
\[
\text{datalength ( string_expression )}
\]

**dateadd**

Returns the date resulting from adding "integer_expression" units indicated by datepart (day, month, year) to "date_expression". Note that "datepart" must be enclosed in curly brackets.

**Syntax**
\[
\text{dateadd ( ' { ' datepart ' } ' , integer_expression , date_expression )}
\]

**Example**
\[
\text{dateadd ( {dd} , 16 , 1997-06-16 )}
\]

Result: Jul 2, 1997

**datediff**

Returns the number of units indicated by "datepart" (day, month, year) between "date_expression1" and "date_expression2". Note that "datepart" must be enclosed in curly brackets.

**Syntax**
\[
\text{datediff ( ' { ' datepart ' } ' , date_expression1 , date_expression2 )}
\]

**Example**
\[
\text{datediff ( {yy} , 1984-01-01 , 1997-01-01 )}
\]

Result: 13

**datename**

Returns "datepart" of "date_expression" as an ASCII string. "Date_expression" can be a datetime, smalldatetime, date, or time value. Note that "datepart" must be enclosed in curly brackets.

**Syntax**
\[
\text{datename ( ' { ' datepart ' } ' , date_expression )}
\]
Example
datename ( {mm} , 1999-05-01 )

Result: May

datepart

Returns "datepart" of "date_expression" as an integer. "Date_expression" can be a datetime, smalldatetime, date, or time value. Note that the datepart argument must be enclosed in curly brackets.

Syntax
datepart ( ' { ' datepart ' } ' , date_expression )

Example
datepart ( {mm} , 1999-05-01 )

Result: 5

day

Returns the day of the month (1-31) from "date_expression".

Syntax
day ( date_expression )

difference

Returns an integer value representing the difference between the values returned by the data source-specific soundex function for "string_expression1" and "string_expression2". The value that is returned ranges from 0 to 4, with 4 indicating the best match. Note that 4 does not mean that the strings are equal.

Syntax
difference ( string_expression1 , string_expression2 )

Example 1
difference ([Sales target (query)].[Sales staff].[First name],[Sales (query)].[Retailers].[Contact first name])

Result: 0

Example 2
difference ([Sales target (query)].[Sales staff].[First name],[Sales target (query)].[Sales staff].[First name])

Result: 4

gdate

Returns current system date and time.

Syntax
gdate ()
left

Returns the leftmost "integer_expression" characters of "string_expression".

**Syntax**

```
left ( string_expression , integer_expression )
```

**Example**

```
left ( [Sales (query)].[Sales staff].[Last name] , 3 )
```

Result: Returns the first three characters of each last name.

ltrim

Returns "string_expression" with any leading spaces removed.

**Syntax**

```
ltrim ( string_expression )
```

month

Returns the month number (1-12) from "date_expression".

**Syntax**

```
month ( date_expression )
```

patindex

Returns an integer representing the starting position of the first occurrence of "string_expression1" in "string_expression2" or returns 0 if "string_expression1" is not found. By default, patindex returns the offset in characters. The offset can be returned in bytes by setting the return type to bytes. The % wildcard character must precede and follow the pattern in "string_expression1", except when searching for first or last characters.

**Syntax**

```
patindex ( string_expression1 , string_expression2 [ using {bytes|chars|characters} ] )
```

rand

Returns a random float value between 0 and 1, using the optional "integer_expression" as a seed value.

**Syntax**

```
rand ( integer_expression )
```

replicate

Returns a string with the same datatype as "string_expression", containing the same expression repeated "integer_expression" times or as many times as will fit into a 225-byte space, whichever is less.
Syntax
replicate ( string_expression , integer_expression )

reverse

Returns the reverse of "string_expression".

Syntax
reverse ( string_expression )

right

Returns the rightmost "integer_expression" characters of "string_expression".

Syntax
right ( string_expression , integer_expression )

round

Returns "numeric_expression" rounded to the nearest value "integer_expression" places to the right of the decimal point. Rounding takes place before data formatting is applied.

Syntax
round ( numeric_expression, integer_expression )

Example
round (125, -1)
Result: 130

rtrim

Returns "string_expression" with trailing spaces removed.

Syntax
rtrim ( string_expression )

Example
rtrim ( [Sales (query)].[Sales staff].[Last name] )
Result: Returns last names with any spaces at the end of the name removed.

soundex

Returns a four-character soundex code for character strings that are composed of a contiguous sequence of valid single- or double-byte Roman letter.

Syntax
soundex ( string_expression )

space

Returns a string with "integer_expression" single-byte spacing.
Appendix B: Using the Expression Editor

Syntax
space ( integer_expression )

str

Returns a string representation of "numeric_expression". "Integer_expression1" is the length of the returned string and has a default setting of 10. "Integer_expression2" is the number of decimal digits and has a default setting of 0. Both are optional values.

Syntax
str ( numeric_expression [ , integer_expression1 [ , integer_expression2 ] ] )

stuff

Deletes "integer_expression2" characters from "string_expression1" starting at "integer_expression1", and inserts "string_expression2" into "string_expression1" at that position. To delete characters without inserting other characters, "string_expression2" should be null and not " ", which indicates a single space.

Syntax
stuff ( string_expression1 , integer_expression1 , integer_expression2 , string_expression2 )

substring

Returns the substring of "string_expression" that starts at position "integer_expression1". "Integer_expression2" specifies the number of characters in the substring.

Syntax
substring ( string_expression , integer_expression1 , integer_expression2 )

Example
substring ( [Sales (query)].[Sales staff].[Position code] , 3 , 5 )

Result: Returns characters 3 to 7 of the position codes.

to_unichar

Returns a unichar expression with the value "integer_expression". If "integer_expression" is in the range 0xD800..0xDFFF, the operation is aborted. If the "integer_expression" is in the range 0..0xFFFF, a single Unicode value is returned. If "integer_expression" is in the range 0x10000..0x10FFFF, a surrogate pair is returned.

Syntax
to_unichar ( integer_expression )

uhighsurr

Returns 1 if the Unicode value at "integer_expression" is the high half of a surrogate pair (which should appear first in the pair). Otherwise, it returns 0. This function allows you to write explicit code for surrogate handling. Particularly, if a substring starts on a Unicode character where
uhighsurr () is true, extract a substring of at least 2 Unicode values, as substr() does not extract just 1. Substr () does not extract half of a surrogate pair.

**Syntax**

```
uhighsurr ( string_expression , integer_expression )
```

**ulowsurr**

Returns 1 if the Unicode value at "integer_expression" is the low half of a surrogate pair (which should appear second in the pair). Otherwise, it returns 0. This function allows you to explicitly code around the adjustments performed by substr(), stuff(), and right(). Particularly, if a substring ends on a Unicode value where ulowsurr () is true, extract a substring of 1 less characters (or 1 more), since substr () does not extract a string that contains an unmatched surrogate pair.

**Syntax**

```
ulowsurr ( string_expression , integer_expression )
```

**uscalar**

Returns the Unicode scalar value for the first Unicode character in "string_expression". If the first character is not the high-order half of a surrogate pair, then the value is in the range 0..0xFFFF. If the first character is the high-order half of a surrogate pair, a second value must be a low-order half, and the return value is in the range 0x10000..0x10FFFF. If this function is called on a Unicode character expression containing an unmatched surrogate half, the operation is aborted.

**Syntax**

```
uscalar ( string_expression )
```

**year**

Returns the year from date_expression.

**Syntax**

```
year ( date_expression )
```

**Sybase Math**

**log**

Returns the natural logarithm of "numeric_expression".

**Syntax**

```
log ( numeric_expression )
```

**log10**

Returns the base ten logarithm of "numeric_expression".

**Syntax**

```
log10 ( numeric_expression )
```
**pi**
Returns the constant value of pi as a floating point value.

**Syntax**
pi ()

**sign**
Returns an indicator denoting the sign of "numeric_expression": +1 if "numeric_expression" is positive, 0 if "numeric_expression" is zero, or -1 if "numeric_expression" is negative.

**Syntax**
sign ( numeric_expression )

**Sybase Trigonometry**

**acos**
Returns the arccosine of "numeric_expression" in radians. The arccosine is the angle whose cosine is "numeric_expression".

**Syntax**
acos ( numeric_expression )

**asin**
Returns the arcsine of "numeric_expression" in radians. The arcsine is the angle whose sine is "numeric_expression".

**Syntax**
asin ( numeric_expression )

**atan**
Returns the arctangent of "numeric_expression" in radians. The arctangent is the angle whose tangent is "numeric_expression".

**Syntax**
atan ( numeric_expression )

**tan**
Returns the tangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

**Syntax**
tan ( numeric_expression )

**atn2**
Returns the angle, in radians, whose tangent is "numeric_expression1" / "numeric_expression2".
Syntax

\texttt{atn2 ( numeric_expression1, numeric_expression2 )}

cos

Returns the cosine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

\texttt{cos ( numeric_expression )}

cot

Returns the cotangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

\texttt{cot ( numeric_expression )}

degrees

Returns "numeric_expression" radians converted to degrees.

Syntax

\texttt{degrees ( numeric_expression )}

radians

Returns the degree equivalent of "numeric_expression". Results are of the same type as "numeric_expression". For numeric or decimal expressions, the results have an internal precision of 77 and a scale equal to that of "numeric_expression". When the money datatype is used, an internal conversion to float may cause some loss of precision.

Syntax

\texttt{radians ( numeric_expression )}

sin

Returns the sine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax

\texttt{sin ( numeric_expression )}

**Postgres**

**Postgres String**

\texttt{overlay}

Returns the "string_expression1" replacing "string_expression2" from character position numeric_expression.
Appendix B: Using the Expression Editor

**Syntax**

_overlay ( string_expression1 , string_expression2 , numeric_expression1 [ , numeric_expression2 ] )_

**btrim**

Returns string_expression1 after removing the longest string of characters in "string_expression2".

_Syntax_

btrim ( string_expression1 [ , string_expression2 ] )

**initcap**

Returns "string_expression", with the first letter of each word in uppercase and all other letters in lowercase. Words are delimited by white space or characters that are not alphanumeric.

_Syntax_

initcap ( string_expression )

**lpad**

Returns "string_expression1" padded to length "integer_expression" with occurrences of "string_expression2". If "string_expression1" is longer than "integer_expression", the appropriate portion of "string_expression1" is returned.

_Syntax_

lpad ( string_expression1 , integer_expression [ , string_expression2 ] )

**ltrim**

Returns "string_expression1", with leading characters removed up to the first character not in "string_expression2"; for example, ltrim ( 'xyxXxyAB' , 'xy' ) returns XxyAB.

_Syntax_

ltrim ( string_expression1 [ , string_expression2 ] )

**md5**

Returns the MD5 hash of "string_expression1".

_Syntax_

md5 ( string_expression1 )

**to_hex**

Returns the hexadecimal string representation of "numeric_expression1".

_Syntax_

to_hex ( numeric_expression1 )

**repeat**

Returns the "string_expression" repeated "numeric_expression1" times.
Syntax
repeat ( string_expression , numeric_expression1 )

replace
Returns "string_expression" with "string_expression2" replaced with "string_expression3".

Syntax
replace ( string_expression , string_expression2 , string_expression3 )

rpad
Returns "string_expression1" right-padded to length "integer_expression" with occurrences of"string_expression2". If "string_expression1" is longer than "integer_expression", the appropriateportion of "string_expression1" is returned. If "string_expression2" is not specified, then spacesare used.

Syntax
rpad ( string_expression1 , integer_expression [ , string_expression2 ] )

rtrim
Returns "string_expression1", with final characters removed after the last character not in "string_ expression2"; for example, rtrim ( 'ABxXyx' , 'xy' ) returns ABxX. If "string_expression2" is notspecified, the final space characters are removed.

Syntax
rtrim ( string_expression1 [ , string_expression2 ] )

split_part
Returns "numeric_expression" field having split "string_expression1" on "string_expression2".

Syntax
split_part ( string_expression1 , string_expression2 , numeric_expression )

ascii
Returns a number representing the ASCII code value of the leftmost character of "string_expression";for example, ascii ( 'A' ) is 65.

Syntax
ascii ( string_expression )

chr
Returns the character that has the ASCII code value specified by "integer_expression". "Integer_expression" should be between 0 and 255.

Syntax
chr ( integer_expression )
Appendix B: Using the Expression Editor

### {current_catalog}

**Syntax**

```sql
{current_catalog}
```

### {current_db}

**Syntax**

```sql
{current_db}
```

### {current_schema}

**Syntax**

```sql
{current_schema}
```

### {current_user}

**Syntax**

```sql
{current_user}
```

### {session_user}

**Syntax**

```sql
{session_user}
```

## Postgres Data type formatting

### `to_char`

Returns the string representation of "expression" with the format of "string_expression". "Expression" can be either a date value or a numeric value.

**Syntax**

```sql
to_char ( expression , string_expression )
```

### `to_date`

Converts "string_expression1" to a date value as specified by the format "string_expression2".

**Syntax**

```sql
to_date ( string_expression1 , string_expression2 )
```

### `to_number`

Converts "string_expression1" to a numeric value as specified by the format "string_expression2".

**Syntax**

```sql
to_number ( string_expression1 , string_expression2 )
```

### `to_timestamp`

Converts "string_expression1" to a timestamp value as specified by the format "string_expression2".

218 IBM Cognos Metric Designer
Syntax
to_timestamp ( string_expression1 , string_expression2 )

translate
Returns "string_expression1", with each occurrence of each character in "string_expression2"
replaced by its corresponding character in "string_expression3".

Syntax
translate ( string_expression1 , string_expression2 , string_expression3 )

date_trunc
Returns the timestamp to the specified precision.

Syntax
date_trunc ( string_expression , timestamp_expression )

version
Returns the string value of the database version.

Syntax
version ()

Postgres Math
log
Returns the base 10 logarithm of "numeric_expression1" or logarithm to the base "numeric_expression2".

Syntax
log ( numeric_expression1 [ , numeric_expression2 ] )

ln
Returns the natural logarithm of "numeric_expression1".

Syntax
ln ( numeric_expression )

cbrt
Returns the cube root of "numeric_expression1".

Syntax
cbrt ( numeric_expression )

div
Returns the integer quotient of "numeric_expression1" divided by "numeric_expression2".
Appendix B: Using the Expression Editor

**Syntax**

```plaintext
div ( numeric_expression1 , numeric_expression2 )
```

**pi**

Returns the constant of pi.

**Syntax**

```plaintext
pi ()
```

---

**Postgres Trigonometry**

**acos**

Returns the arccosine of "numeric_expression" in radians. The arccosine is the angle whose cosine is "numeric_expression".

**Syntax**

```plaintext
acos ( numeric_expression )
```

**asin**

Returns the arcsine of "numeric_expression" in radians. The arcsine is the angle whose sine is "numeric_expression".

**Syntax**

```plaintext
asin ( numeric_expression )
```

**atan**

Returns the arctangent of "numeric_expression" in radians. The arctangent is the angle whose tangent is "numeric_expression".

**Syntax**

```plaintext
atan ( numeric_expression )
```

**atan2**

Returns the arctangent of the x and y coordinates specified by "numeric_expression1" and "numeric_expression2", respectively, in radians. The arctangent is the angle whose tangent is "numeric_expression2"/"numeric_expression1".

**Syntax**

```plaintext
atan2 ( numeric_expression1 , numeric_expression2 )
```

**cos**

Returns the cosine of "numeric_expression", where "numeric_expression" is an angle expressed in radians.

**Syntax**

```plaintext
cos ( numeric_expression )
```
cot
Returns the cotangent of "numeric_expression", where "numeric_expression" is an angle expressed in radians.

**Syntax**
cot ( numeric_expression )

degrees
Returns the degrees where "numeric_expression" is an angle expressed in radians.

**Syntax**
degrees ( numeric_expression )

radians
Returns the radians where "numeric_expression" is an angle expressed in degrees.

**Syntax**
radians ( numeric_expression )

sin
Returns the sine of "numeric_expression", where "numeric_expression" is an angle expressed in radians.

**Syntax**
sin ( numeric_expression )

tan
Returns the tangent of "numeric_expression", where "numeric_expression" is an angle expressed in radians.

**Syntax**
tan ( numeric_expression )

Vertica

Vertica String

overlay
Returns the "string_expression1", replacing "string_expression2" from character position numeric_expression.

**Syntax**
overlay ( string_expression1 , string_expression2 , numeric_expression1 [ , numeric_expression2 ] )
btrim
Returns string_expression1 after removing the longest string of characters in string_expression2.

**Syntax**
```
btrim ( string_expression1 [ , string_expression2 ] )
```

initcap
Returns "string_expression", with the first letter of each word in uppercase and all other letters in lowercase. Words are delimited by white space or characters that are not alphanumeric.

**Syntax**
```
initcap ( string_expression )
```

lpad
Returns "string_expression1" padded to length "integer_expression" with occurrences of "string_expression2". If "string_expression1" is longer than "integer_expression", the appropriate portion of "string_expression1" is returned.

**Syntax**
```
lpad ( string_expression1 , integer_expression [ , string_expression2 ] )
```

ltrim
Returns "string_expression1", with leading characters removed up to the first character not in "string_expression2"; for example, ltrim ( 'xyxXxyAB' , 'xy' ) returns XxyAB.

**Syntax**
```
ltrim ( string_expression1 [ , string_expression2 ] )
```

to_hex
Returns the hexadecimal string representation of "numeric_exp1".

**Syntax**
```
to_hex ( numeric_expression1 )
```

repeat
Returns the "string_expression" repeated "numeric_expression1" times.

**Syntax**
```
repeat ( string_expression , numeric_expression1 )
```

replace
Returns "string_expression" having replaced "string_expression2" with "string_expression3".

**Syntax**
```
replace ( string_expression , string_expression2 , string_expression3 )
```
**rpad**

Returns "string_expression1" right-padded to length "integer_expression" with occurrences of "string_expression2". If "string_expression1" is longer than "integer_expression", the appropriate portion of "string_expression1" is returned. If "string_expression2" is not specified, then spaces are used.

**Syntax**

rpad ( string_expression1 , integer_expression [ , string_expression2 ] )

**rtrim**

Returns "string_expression1", with final characters removed after the last character not in "string_expression2"; for example, rtrim ( 'ABxXxyx' , 'xy' ) returns ABxX. If "string_expression2" is not specified, the final space characters are removed.

**Syntax**

rtrim ( string_expression1 [ , string_expression2 ] )

**ascii**

Returns a number representing the ASCII code value of the leftmost character of "string_expression"; for example, ascii ( 'A' ) is 65.

**Syntax**

ascii ( string_expression )

**chr**

Returns the character that has the ASCII code value specified by "integer_expression". "Integer_expression" should be between 0 and 255.

**Syntax**

chr ( integer_expression )

**current_database**

Returns the name of the current database.

**Syntax**

current_database ()

**current_schema**

Returns the name of the current schema

**Syntax**

current_schema ()
Vertica Data type formatting

**to_char**
Returns the string representation of "expression" with the format of "string_expression". "Expression" can be either a date value or a numeric value.

**Syntax**
tag textile ( expression , string_expression )

**to_date**
Converts "string_expression1" to a date value as specified by the format "string_expression2".

**Syntax**
tag textile ( string_expression1 , string_expression2 )

**to_number**
Converts "string_expression1" to a numeric value as specified by the format "string_expression2".

**Syntax**
tag textile ( string_expression1 , string_expression2 )

**to_timestamp**
Converts "string_expression1" to a timestamp value as specified by the format "string_expression2".

**Syntax**
tag textile ( string_expression1 , string_expression2 )

**translate**
Returns "string_expression1", with each occurrence of each character in "string_expression2" replaced by its corresponding character in "string_expression3".

**Syntax**
tag textile ( string_expression1 , string_expression2 , string_expression3 )

**date_trunc**
Returns the timestamp to the specified precision.
Syntax

date_trunc ( string_expression , timestamp_expression)

version

Returns the string value of the database version.

Syntax

version ()

Vertica Math

log

Returns the base 10 logarithm of "numeric_expression1" or logarithm to the base "numeric_expression2".

Syntax

log ( numeric_expression1 [ , numeric_expression2 ] )

ln

Returns the natural logarithm of "numeric_expression1".

Syntax

ln ( numeric_expression )

cbrt

Returns the cube root of "numeric_expression1".

Syntax

cbrt ( numeric_expression )

pi

Returns the constant of pi.

Syntax

pi ()

Vertica Trigonometry

acos

Returns the arccosine of "numeric_expression" in radians. The arccosine is the angle whose cosine is "numeric_expression".

Syntax

acos ( numeric_expression )
asin
Returns the arcsine of "numeric_expression" in radians. The arcsine is the angle whose sine is "numeric_expression".

**Syntax**
```
asin ( numeric_expression )
```

atan
Returns the arctangent of "numeric_expression" in radians. The arctangent is the angle whose tangent is "numeric_expression".

**Syntax**
```
atan ( numeric_expression )
```

atan2
Returns the arctangent of the x and y coordinates specified by "numeric_expression1" and "numeric_expression2", respectively, in radians. The arctangent is the angle whose tangent is "numeric_expression2" / "numeric_expression1".

**Syntax**
```
atan2 ( numeric_expression1 , numeric_expression2 )
```

cos
Returns the cosine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

**Syntax**
```
cos ( numeric_expression )
```

cot
Returns the cotangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

**Syntax**
```
cot ( numeric_expression )
```

degrees
Returns the degrees where "numeric_expression" is an angle expressed in radians.

**Syntax**
```
degrees ( numeric_expression )
```

radians
Returns the radians where "numeric_expression" is an angle expressed in degrees.
Syntax
radians ( numeric_expression )

sin
Returns the sine of "numeric_exp" where "numeric_expression" is an angle expressed in radians.

Syntax
sin ( numeric_expression )

tan
Returns the tangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax
tan ( numeric_expression )

Paraccel
Paraccel String

overlay
Returns the "string_expression1", replacing "string_expression2" from character position numeric_expression.

Syntax
overlay ( string_expression1 , string_expression2 , numeric_expression1 [ ,
numeric_expression2 ] )

ltrim
Returns "string_expression1", with leading characters removed up to the first character not in "string_expression2"; for example, ltrim ( 'xyxXxyAB' , 'xy' ) returns XxyAB.

Syntax
ltrim ( string_expression1 [ , string_expression2 ] )

replace
Returns "string_expression", having replaced "string_expression2" with "string_expression3".

Syntax
replace ( string_expression , string_expression2 , string_expression3 )

rtrim
Returns "string_expression1", with final characters removed after the last character not in "string_expression2"; for example, rtrim ( 'ABxXxyx' , 'xy' ) returns ABx. If "string_expression2" is not specified, the final space characters are removed.
Appendix B: Using the Expression Editor

**Syntax**

```
rtrim ( string_expression1 [ , string_expression2 ] )
```

**current_database**

Returns the name of the current database.

```
current_database ()
```

**current_schema**

Returns the name of the current schema

```
current_schema ()
```

**{current_user}**

```
{current_user}
```

**{session_user}**

```
{session_user}
```

**Paraccel Data type formatting**

**to_char**

Returns the string representation of "expression" with the format of "string_expression". "Expression" can be either a date value or a numeric value.

```
to_char ( expression , string_expression )
```

**to_date**

Converts "string_expression1" to a date value as specified by the format "string_expression2".

```
to_date ( string_expression1 , string_expression2 )
```

**to_number**

Converts "string_expression1" to a numeric value as specified by the format "string_expression2".

```
to_number ( string_expression1 , string_expression2 )
```
**translate**

Returns "string_expression1", with each occurrence of each character in "string_expression2" replaced by its corresponding character in "string_expression3".

**Syntax**

\[
\text{translate ( string_expression1 , string_expression2 , string_expression3 )}
\]

**version**

Returns the string value of the database version.

**Syntax**

\[
\text{version ()}
\]

**Paracel Math**

**cbrt**

Returns the cube root of "numeric_expression1".

**Syntax**

\[
\text{cbrt ( numeric_expression )}
\]

**pi**

Returns the constant of pi.

**Syntax**

\[
\text{pi ()}
\]

**MySQL**

**MySQL String**

**lpad**

Returns "string_expression1" padded to length "integer_expression" with occurrences of "string_expression2". If "string_expression1" is longer than "integer_expression", the appropriate portion of "string_expression1" is returned.

**Syntax**

\[
\text{lpad ( string_expression1 , integer_expression [ , string_expression2 ] )}
\]

**ltrim**

Returns "string_expression1", with leading characters removed up to the first character not in "string_expression2"; for example, ltrim ( 'xyxXxyAB' , 'xy' ) returns XxyAB.

**Syntax**

\[
\text{ltrim ( string_expression1 [ , string_expression2 ] )}
\]
Appendix B: Using the Expression Editor

hex
Returns the hexadecimal string representation of "numeric_expression1".

Syntax
hex ( numeric_expression1 )

repeat
Returns the "string_expression" repeated "numeric_expression1" times.

Syntax
repeat ( string_expression , numeric_expression1 )

replace
Returns "string_expression" having replaced "string_expression2" with "string_expression3".

Syntax
replace ( string_expression , string_expression2 , string_expression3 )

reverse
Returns "string_expression" reversed.

Syntax
reverse ( string_expression )

right
Returns the rightmost "numeric_expression" characters from "string_expression1".

Syntax
right ( string_expression1 , numeric_expression )

rpad
Returns "string_expression1" right-padded to length "integer_expression" with occurrences of "string_expression2". If "string_expression1" is longer than "integer_expression", the appropriate portion of "string_expression1" is returned. If "string_expression2" is not specified, then spaces are used.

Syntax
rpad ( string_expression1 , integer_expression [ , string_expression2 ] )

rtrim
Returns "string_expression1", with final characters removed after the last character not in "string_expression2"; for example, rtrim ( 'ABxXyx' , 'xy' ) returns ABxX. If "string_expression2" is not specified, the final space characters are removed.

Syntax
rtrim ( string_expression1 [ , string_expression2 ] )
soundex
Returns a soundex string of "string_expression1".

Syntax
soundex ( string_expression1 )

ascii
Returns a number representing the ASCII code value of the leftmost character of "string_expression"; for example, ascii ( 'A' ) is 65.

Syntax
ascii ( string_expression )

database
Returns the current database name

Syntax
database ()

schema
Returns the current schema name

Syntax
schema ()

session_user
Return the user name returned by the client

Syntax
session_user ()

system_user
Return the user name returned by the client

Syntax
system_user ()

version
Returns the string value of the database version.

Syntax
version ()
Appendix B: Using the Expression Editor

**MySQL Math**

**log**
Returns the base 10 logarithm of "numeric_expression1" or logarithm to the base "numeric_expression2".

**Syntax**

```
log ( numeric_expression )
```

**ln**
Returns the natural logarithm of "numeric_expression1".

**Syntax**

```
ln ( numeric_expression )
```

**pi**
Returns the constant of pi.

**Syntax**

```
pi()
```

**MySQL Trigonometry**

**acos**
Returns the arccosine of "numeric_expression" in radians. The arccosine is the angle whose cosine is "numeric_expression".

**Syntax**

```
acos ( numeric_expression )
```

**asin**
Returns the arcsine of "numeric_expression" in radians. The arcsine is the angle whose sine is "numeric_expression".

**Syntax**

```
asin ( numeric_expression )
```

**atan**
Returns the arctangent of "numeric_expression" in radians. The arctangent is the angle whose tangent is "numeric_expression".

**Syntax**

```
atan ( numeric_expression )
```
atan2
Returns the arctangent of the x and y coordinates specified by "numeric_expression1" and "numeric_expression2", respectively, in radians. The arctangent is the angle whose tangent is "numeric_expression2" / "numeric_expression1".

Syntax
atan2 ( numeric_expression1 , numeric_expression2 )

cos
Returns the cosine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax
cos ( numeric_expression )

cot
Returns the cotangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax
cot ( numeric_expression )

degrees
Returns the degrees where "numeric_expression" is an angle expressed in radians.

Syntax
degrees ( numeric_expression )

radians
Returns the radians where "numeric_expression" is an angle expressed in degrees.

Syntax
radians ( numeric_expression )

sin
Returns the sine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax
sin ( numeric_expression )

tan
Returns the tangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.
Greenplum

Greenplum String

**overlay**
Returns the "string_expression1" replacing "string_expression2" from character position "numeric_expression".

**Syntax**
overlay ( string_expression1 , string_expression2 , numeric_expression1 [ , numeric_expression2 ] )

**btrim**
Returns "string_expression1" after removing the longest string of characters in "string_expression2".

**Syntax**
btrim ( string_expression1 [ , string_expression2 ] )

**initcap**
Returns "string_expression" with the first letter of each word in uppercase and all other letters in lowercase. Words are delimited by white space or characters that are not alphanumeric.

**Syntax**
initcap ( string_expression )

**lpad**
Returns "string_expression1" padded to length "integer_expression" with occurrences of "string_expression2". If "string_expression1" is longer than "integer_expression", the appropriate portion of "string_expression1" is returned.

**Syntax**
lpad ( string_expression1 , integer_expression [ , string_expression2 ] )

**ltrim**
Returns "string_expression1", with leading characters removed up to the first character not in "string_expression2"; for example, ltrim ( 'yxXxyAB' , 'xy' ) returns XxyAB.

**Syntax**
ltrim ( string_expression1 [ , string_expression2 ] )

**md5**
Returns the MD5 hash of "string_expression1".
Syntax
md5 ( string_expression1 )

to_hex
Returns the hexadecimal string representation of "numeric_expression1".

Syntax
to_hex ( numeric_expression1 )

repeat
Returns the "string_expression" repeated "numeric_expression1" times.

Syntax
repeat ( string_expression , numeric_expression1 )

replace
Returns "string_expression" having replaced "string_expression2" with "string_expression3".

Syntax
replace ( string_expression , string_expression2 , string_expression3 )

rpad
Returns "string_expression1" right-padded to length "integer_expression" with occurrences of "string_expression2". If "string_expression1" is longer than "integer_expression", the appropriate portion of "string_expression1" is returned. If "string_expression2" is not specified, then spaces are used.

Syntax
rpad ( string_expression1 , integer_expression [ , string_expression2 ] )

rtrim
Returns "string_expression1", with final characters removed after the last character not in "string_expression2"; for example, rtrim ( 'ABxXxyx' , 'xy' ) returns ABxX. If "string_expression2" is not specified, the final space characters are removed.

Syntax
rtrim ( string_expression1 [ , string_expression2 ] )

split_part
Returns "numeric_expression" field having split "string_expression1" on "string_expression2".

Syntax
split_part ( string_expression1 , string_expression2 , numeric_expression )
Appendix B: Using the Expression Editor

**ascii**

Returns a number representing the ascii code value of the leftmost character of "string_expression"; for example, ascii ( 'A' ) is 65.

**Syntax**

```plaintext
ascii ( string_expression )
```

**chr**

Returns the character that has the ASCII code value specified by "integer_expression". "Integer_expression" should be between 0 and 255.

**Syntax**

```plaintext
chr ( integer_expression )
```

**current_database**

Returns the name of the current database.

**Syntax**

```plaintext
current_database ()
```

**current_schema**

Returns the name of the current schema.

**Syntax**

```plaintext
current_schema ()
```

**{current_user}**

**Syntax**

```plaintext
{current_user}
```

**{session_user}**

**Syntax**

```plaintext
{session_user}
```

**Greenplum Data type formatting**

**to_char**

Returns the string representation of "expression" with the format of "string_expression". "Expression" can either be a date value or a numeric value.

**Syntax**

```plaintext
to_char ( expression , string_expression )
```

**to_date**

Converts "string_expression1" to a date value as specified by the format "string_expression2".
**Syntax**

```

to_date ( string_expression1 , string_expression2 )
```

**to_number**

Converts "string_expression1" to a numeric value as specified by the format "string_expression2".

**Syntax**

```

to_number ( string_expression1 , string_expression2 )
```

**to_timestamp**

Converts "string_expression1" to a timestamp value as specified by the format "string_expression2".

**Syntax**

```

to_timestamp ( string_expression1 , string_expression2 )
```

**translate**

Returns "string_expression1" with each occurrence of each character in "string_expression2" replaced by its corresponding character in "string_expression3".

**Syntax**

```
translate ( string_expression1 , string_expression2 , string_expression3 )
```

**date_trunc**

Returns the timestamp to the specified precision.

**Syntax**

```
date_trunc ( string_expression , timestamp_expression)
```

**version**

Returns the string value of the database version.

**Syntax**

```
version ()
```

**Greenplum Math**

**log**

Returns the base 10 logarithm of "numeric_expression1" or logarithm to the base "numeric_expression2".

**Syntax**

```
log ( numeric_expression1 [ , numeric_expression2 ] )
```

**ln**

Returns the natural logarithm of "numeric_expression1".
Syntax

ln ( numeric_expression )

cbrt
Returns the cube root of "numeric_expression1".

Syntax
cbrt ( numeric_expression )

pi
Returns the constant of pi.

Syntax
pi ()

Greenplum Trigonometry

acos
Returns the arccosine of "numeric_expression" in radians. The arccosine is the angle whose cosine is "numeric_expression".

Syntax
acos ( numeric_expression )

asin
Returns the arcsine of "numeric_expression" in radians. The arcsine is the angle whose sine is "numeric_expression".

Syntax
asin ( numeric_expression )

atan
Returns the arctangent of "numeric_expression" in radians. The arctangent is the angle whose tangent is "numeric_expression".

Syntax
atan ( numeric_expression )

atan2
Returns the arctangent of the x and y coordinates specified by "numeric_expression1" and "numeric_expression2", respectively, in radians. The arctangent is the angle whose tangent is "numeric_expression2" / "numeric_expression1".

Syntax
atan2 ( numeric_expression1 ,numeric_expression2 )
cos
Returns the cosine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax
```
cos( numeric_expression )
```

cot
Returns the cotangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax
```
cot( numeric_expression )
```

degrees
Returns the degrees where "numeric_expression" is an angle expressed in radians.

Syntax
```
degrees( numeric_expression )
```

radians
Returns the radians where "numeric_expression" is an angle expressed in degrees.

Syntax
```
radians( numeric_expression )
```

sin
Returns the sine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax
```
sin( numeric_expression )
```

tan
Returns the tangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

Syntax
```
tan( numeric_expression )
```

Report Functions

_add_days
Returns the datetime resulting from adding "integer_expression" days to "timestamp_expression".
Syntax
_add_days ( timestamp_expression , integer_expression )

Example
_add_days ( 2007-01-14 00:00:00.000 , 3 )
Result: 2007-01-17 00:00:00.000

_add_months
Returns the datetime resulting from adding "integer_expression" months to "timestamp_expression".

Syntax
_add_months ( timestamp_expression , integer_expression )

_add_years
Returns the datetime resulting from adding "integer_expression" years to "timestamp_expression".

Syntax
_add_years ( timestamp_expression , integer_expression )

_age
Returns a number by subtracting "timestamp_expression" from today's date.

Syntax
_age ( timestamp_expression )

Example
_age ([Query1].[Date]), where [Query1].[Date] is March 2, 2004, and today is July 8, 2009
Result: 50,406, where 5 is the number of years, 04 is the number of months, and 06 is the number of days.

day_of_week
Returns the day of the week (between 1 and 7) for "timestamp_expression" where "integer_expression" indicates which day of that week is day 1. To determine "integer_expression", choose the day of the week and count from Monday; for example, if you choose Wednesday, "integer_expression" would be 3 because Wednesday is the third day from Monday.

Syntax
_day_of_week ( timestamp_expression , integer_expression )

Example
_day_of_week ( 2009-01-01 , 7 ), where 7 means that Sunday is the first day of the week.
Result: 5
__day_of_year__

Returns the ordinal for the day of the year in "timestamp_expression" (1 to 366). Also known as Julian day.

**Syntax**

```
_day_of_year ( timestamp_expression )
```

__days_between__

Returns a positive or negative number representing the number of days between "timestamp_expression1" and "timestamp_expression2". If "timestamp_expression1" < "timestamp_expression2", the result will be a negative number.

**Syntax**

```
_days_between ( timestamp_expression1 , timestamp_expression2 )
```

__days_to_end_of_month__

Returns a number representing the number of days remaining in the month represented by "timestamp_expression".

**Syntax**

```
_days_to_end_of_month ( timestamp_expression )
```

__first_of_month__

Returns a datetime that is the first day of the month represented by "timestamp_expression".

**Syntax**

```
_first_of_month ( timestamp_expression )
```

**Example 1**

```
_first_of_month ( 2009-05-04 00:00:00.000 )
```

Result: Returns 2009-05-01 00:00:00.000

**Example 2**

```
_first_of_month (current_date)
```

Result: Returns Jul 1, 2009 if the current date is July 30, 2009.

__last_of_month__

Returns a datetime that is the last day of the month represented by "timestamp_expression".

**Syntax**

```
_last_of_month ( timestamp_expression )
```

__make_timestamp__

Returns a timestamp constructed from "integer_expression1" (the year), "integer_expression2" (the month), and "integer_expression3" (the day). The time portion defaults to 00:00:00.000.
Appendix B: Using the Expression Editor

Syntax
_make_timestamp ( integer_expression1 , integer_expression2 , integer_expression3 )

_months_between

Returns a positive or negative number representing the number of months between "timestamp_expression1" and "timestamp_expression2". If "timestamp_expression1" < "timestamp_expression2", the result will be a negative number.

Syntax
_months_between ( timestamp_expression1 , timestamp_expression2 )

_week_of_year

Returns the week number (1-53) of the year for "timestamp_expression". According to the ISO 8601, week 1 of the year is the first week to contain a Thursday, which is equivalent to the first week containing January 4th. A week starts on a Monday (day 1) and ends on a Sunday (day 7).

Syntax
_week_of_year ( timestamp_expression )

_years_between

Returns a positive or negative integer representing the number of years between "timestamp_expression1" and "timestamp_expression2". If "timestamp_expression1" < "timestamp_expression2", a negative value is returned.

Syntax
_years_between ( timestamp_expression1 , timestamp_expression2 )

_ymdint_between

Returns a number representing the difference between "timestamp_expression1" and "timestamp_expression2". This value has the form YYMMDD, where YY represents the number of years, MM represents the number of months, and DD represents the number of days.

Syntax
_ymdint_between ( timestamp_expression1 , timestamp_expression2 )

Example
_ymdint_between ( [Query1].[Date (close date)] , [Query1].[Date (ship date)] ),
where [Query1].[Date (close date)] is February 20, 2004, and [Query1].[Date (ship date)] is January 19, 2004.

Result: 101, where 1 is the number of months and 01 is the number of days.

abs

Returns the absolute value of "numeric_expression". If "numeric_expression" is negative, a positive value is returned.
Syntax
abs ( numeric_expression )

AsOfDate
Returns the date value of the AsOfDate expression, if it is defined. Otherwise, AsOfDate returns the report execution date.

Syntax
AsOfDate ()

AsOfTime
Returns the time value of the AsOfTime expression, if it is defined. Otherwise, AsOfTime returns the report execution time.

Syntax
AsOfTime ()

BurstKey
Returns the burst key.

Syntax
BurstKey ()

BurstRecipients
Returns the distribution list of burst recipients.

Syntax
BurstRecipients ()

ceiling
Returns the smallest integer that is greater than or equal to "numeric_expression".

Syntax
ceiling ( numeric_expression )

CellValue
Returns the value of the current crosstab cell.

Syntax
CellValue ()

character_length
Returns the number of characters in "string_expression".

Syntax
character_length ( string_expression )
Appendix B: Using the Expression Editor

**ColumnNumber**

Returns the current column number.

**Syntax**

ColumnNumber()

**CubeCreatedOn**

Returns the date and time when the cube was created. "Dimension" specifies from which cube to retrieve the metadata. If the dimension source is an IBM Cognos PowerCube (.mdc), the function returns a blank string because the initial creation date of a PowerCube is not maintained.

**Syntax**

CubeCreatedOn ( dimension )

**CubeCurrentPeriod**

Returns the current period for the cube. "Dimension" specifies from which cube to retrieve the metadata.

**Syntax**

CubeCurrentPeriod ( dimension )

**CubeDataUpdatedOn**

Returns the date time that data in the cube was last updated. "Dimension" specifies from which cube to retrieve the metadata.

**Syntax**

CubeDataUpdatedOn ( dimension )

**CubeDefaultMeasure**

Returns the name of the default measure for the cube. "Dimension" specifies from which cube to retrieve the metadata.

**Syntax**

CubeDefaultMeasure ( dimension )

**CubeDescription**

Returns the description of the cube. "Dimension" specifies from which cube to retrieve the metadata.

**Syntax**

CubeDescription ( dimension )

**CubeIsOptimized**

Returns "true" if the cube is optimized. "Dimension" specifies from which cube to retrieve the metadata.
Syntax

CubeIsOptimized ( dimension )

CubeName

Returns the name of the cube. "Dimension" specifies from which cube to retrieve the metadata.

Syntax

CubeName ( dimension )

CubeSchemaUpdatedOn

Returns the date time that the cube schema was last updated. "Dimension" specifies from which cube to retrieve the metadata.

Syntax

CubeSchemaUpdatedOn ( dimension )

exp

Returns the constant 'e' raised to the power of "numeric_expression". The constant 'e' is the base of the natural logarithm.

Syntax

exp ( numeric_expression )

Example

exp ( 2 )
Result: 7.389056

extract

Returns an integer representing the value of "date_part_expression" in "datetime_expression". "Date_part_expression" could be the year, month, day, hour, minute, or second.

Syntax

extract ( date_part_expression , datetime_expression )

Example 1

eextract ( year , 2003-03-03 16:40:15.535 )
Result: 2003

Example 2

eextract ( hour , 2003-03-03 16:40:15.535 )
Result: 16

floor

Returns the largest integer that is less than or equal to "numeric_expression".

User Guide 245
Syntax
floor ( numeric_expression )

GetLocale
Returns the run locale (deprecated).

Syntax
GetLocale ()

HorizontalPageCount
Returns the current horizontal page count.

Syntax
HorizontalPageCount ()

HorizontalPageNumber
Returns the current horizontal page number.

Syntax
HorizontalPageNumber ()

InScope
Returns Boolean 1 (true) when the cell is in the scope of the data items and MUNs; otherwise, returns Boolean 0 (false).

Syntax
InScope ( dataItem , MUN, ... )

IsAccessible
Returns Boolean 1 (true) if the report is run with the accessibility features enabled. Use this function as a variable expression with a conditional block to make your reports accessible. For example, you can add a list or crosstab equivalent to a chart in reports that are run with accessibility features enabled.

Syntax
IsAccessible()

IsBursting
Returns Boolean 1 (true) when the report will be distributed to the recipient; otherwise, returns Boolean 0 (false).

Syntax
IsBursting ('recipientName')

IsCrosstabColumnNodeMember
Returns Boolean 1 (true) if the current node is a crosstab column node member.
Syntax
IsCrosstabColumnNodeMember ()

**IsCrosstabRowNodeMember**

Returns Boolean 1 (true) if the current node is a crosstab row node member.

Syntax
IsCrosstabRowNodeMember ()

**IsFirstColumn**

Returns Boolean 1 (true) if the current column is the first column.

Syntax
IsFirstColumn ()

**IsInnerMostCrosstabColumnNodeMember**

Returns Boolean 1 (true) if the current node is an innermost crosstab column node member.

Syntax
IsInnerMostCrosstabColumnNodeMember ()

**IsInnerMostCrosstabRowNodeMember**

Returns Boolean 1 (true) if the current node is an innermost crosstab row node member.

Syntax
IsInnerMostCrosstabRowNodeMember ()

**IsLastColumn**

Returns Boolean 1 (true) if the current column is the last column.

Syntax
IsLastColumn ()

**IsLastInnerMostCrosstabColumnNodeMember**

Returns Boolean 1 (true) if the current node is the last innermost crosstab column node member.

Syntax
IsLastInnerMostCrosstabColumnNodeMember ()

**IsLastInnerMostCrosstabRowNodeMember**

Returns Boolean 1 (true) if the current node is the last innermost crosstab row node member.

Syntax
IsLastInnerMostCrosstabRowNodeMember ()

**IsOuterMostCrosstabColumnNodeMember**

Returns Boolean 1 (true) if the current node is an outermost crosstab column node member.
Appendix B: Using the Expression Editor

Syntax
IsOuterMostCrosstabColumnNodeMember ()

IsOuterMostCrosstabRowNodeMember

Returns Boolean 1 (true) if the current node is an outermost crosstab row node member.

Syntax
IsOuterMostCrosstabRowNodeMember ()

IsPageCountAvailable

Returns Boolean 1 (true) if the page count is available for the current execution of the report; otherwise, returns Boolean 0 (false).

Syntax
IsPageCountAvailable ()

ln

Returns the natural logarithm of "numeric_expression".

Syntax
ln ( numeric_expression )

Locale

Returns the run locale.

Syntax
Locale ()

lower

Returns "string_expression" with all uppercase characters converted to lowercase. This function appears in the Bursted Sales Performance Report sample report in the GO Data Warehouse (query) package.

Syntax
lower ( string_expression )

mapNumberToLetter

Adds "integer_expression" to "string_expression".

Syntax
mapNumberToLetter ( string_expression , integer_expression )

Example
mapNumberToLetter ( 'a' , 1 )

Result: b
mod

Returns an integer value representing the remainder (modulo) of "integer_expression1" / "integer_expression2".

Syntax
mod ( integer_expression1 , integer_expression2 )

ModelPath

Returns the model path.

Syntax
ModelPath ()

Now

Returns the current system time.

Syntax
Now ()

nullif

Returns null if "string_expression1" equals "string_expression2" (case-insensitive), otherwise returns "string_expression1".

Syntax
nullif ( string_expression1 , string_expression2 )

octet_length

Returns the number of bytes in "string_expression".

Syntax
octet_length ( string_expression )

PageCount

Returns the current page count. This function works only when the report output is Adobe® PDF or Microsoft® Excel. If you save the report output, this function works for all formats.

Syntax
PageCount ()

PageName

Returns the current page name.

Syntax
PageName ()
Appendix B: Using the Expression Editor

**PageNumber**

Returns the current page number.

**Syntax**

PageNumber ()

**ParamCount**

Returns the parameter count of "parameterName".

**Syntax**

ParamCount ('parameterName')

**ParamDisplayValue**

Returns a string that is the parameter display value of "parameterName". This function appears in the Recruitment Report sample report in the GO Data Warehouse (analysis) package.

**Syntax**

ParamDisplayValue ('parameterName')

**ParamName**

Returns the parameter name of "parameterName".

**Syntax**

ParamName ('parameterName')

**ParamNames**

Returns all parameter names.

**Syntax**

ParamNames ()

**ParamValue**

Returns the parameter value of "parameterName".

**Syntax**

ParamValue ('parameterName')

**position**

Returns the integer value representing the starting position of "string_expression1" in "string_expression2". Returns 0 if "string_expression1" is not found.

**Syntax**

position ( string_expression1 , string_expression2 )

**power**

Returns "numeric_expression1" raised to the power of "numeric_expression2".
**Syntax**

power ( numeric_expression1 , numeric_expression2 )

**Example**

```
power ( 3 , 2 )
```

Result: 9

**ReportAuthorLocale**

Returns the author locale.

**Syntax**

```
ReportAuthorLocale ()
```

**ReportCreateDate**

Returns the date when the report was created.

**Syntax**

```
ReportCreateDate ()
```

**ReportDate**

Returns the report execution date and time.

**Syntax**

```
ReportDate ()
```

**ReportDescription**

Returns the report description. This function works only when the report is run from IBM Cognos Connection.

**Syntax**

```
ReportDescription ()
```

**ReportID**

Returns the report ID.

**Syntax**

```
ReportID ()
```

**ReportLocale**

Returns the run locale.

**Syntax**

```
ReportLocale ()
```
Appendix B: Using the Expression Editor

**ReportName**

Returns the report name. This function works only when the report is run from IBM Cognos Connection.

**Syntax**

ReportName ()

**ReportOption**

Returns the value of the run option variable identified by "optionName", such as attachmentEncoding, burst, cssURL, email, emailAsAttachment, emailAsURL, emailBody, emailSubject, emailTo, emailToAddress, history, metadataModel, outputEncapsulation, outputFormat, outputLocale, outputPageDefinition, outputPageOrientation, primaryWaitThreshold, print, printer, printerAddress, prompt, promptFormat, saveAs, saveOutput, secondaryWaitThreshold, verticalElements, or xslURL.

**Syntax**

ReportOption ('optionName')

**ReportOutput**

Returns the name of the output format, such as CSV, HTML, layoutDataXML, MHT, PDF, rawXML, singleXLS, spreadsheetML, XLS, XML, or XLWA.

**Syntax**

ReportOutput ()

**ReportPath**

Returns the report path. This function works only when the report is run from IBM Cognos Connection.

**Syntax**

ReportPath ()

**ReportProductLocale**

Returns the product locale.

**Syntax**

ReportProductLocale ()

**ReportSaveDate**

Returns the date when the report was last saved.

**Syntax**

ReportSaveDate ()

**round**

Returns "numeric_expression" rounded to the nearest value with "integer_expression" significant digits to the right of the decimal point. If "integer_expression" is negative, "numeric_expression"
is rounded to the nearest absolute value with "integer_expression" significant digits to the left of the decimal point. Rounding takes place before data formatting is applied.

**Syntax**

round ( numeric_expression , integer_expression )

**Example**

round (125, -1)

Result: 130

**RowNumber**

Returns the current row.

**Syntax**

RowNumber ()

**ServerLocale**

Returns the locale of the server that runs the report.

**Syntax**

ServerLocale ()

**ServerName**

Returns the name of the server that runs the report.

**Syntax**

ServerName ()

**sqrt**

Returns the square root of "numeric_expression". "Numeric_expression" must not be a negative value.

**Syntax**

sqrt ( numeric_expression )

**substring**

Returns the substring of "string_expression" that starts at position "integer_expression1" for "integer_expression2" characters or to the end of "string_expression" if "integer_expression2" is -1. The first character in "string_expression" is at position 1.

**Syntax**

substring ( string_expression , integer_expression1 , integer_expression2 )

**Example**

substring ( [Sales (analysis)].[Sales staff].[Sales staff].[Sales staff].[Position code], 3 , 5 )
Appendix B: Using the Expression Editor

Result: Returns characters 3 to 7 of the position codes.

TOCHeadingCount

Returns the table of contents heading count for a specified heading level.

Syntax
TOCHeadingCount ( headingLevel )

Today

Returns the current system date.

Syntax
Today ()

trim

Returns "string_expression" trimmed of any leading and trailing blanks or trimmed of the character specified by "match_character_expression". "Trim_what_expression" may be "leading", "trailing", or "both" (default). "Match_character_expression" can be an empty string to trim blanks or can specify a character to be trimmed.

Syntax
trim ( trim_what_expression , match_character_expression , string_expression )

upper

Returns "string_expression" with all lowercase characters converted to uppercase.

Syntax
upper ( string_expression )

URLEncode

Returns the URL encoded value of the input text.

Syntax
URLEncode ('text')

Data Type Casting Functions

date2string

Returns a date as a string in YYYY-MM-DD format.

Syntax
date2string ( date_expression )

date2timestamp

Converts "date_expression" to a timestamp. The time part of the timestamp will equal zero.
Syntax

date2timestamp ( date_expression )

date2timestampTZ
Converts "date_expression" to a timestamp with a time zone. The time and time zone parts of the
timestamp will equal zero.

Syntax

date2timestampTZ ( date_expression )

DTinterval2string
Returns a date time interval as a string in DDDD HH:MM:SS.FFFFFFF or -DDDD HH:MM:SS.FFF
format.

Syntax

DTinterval2string ( date_time_interval_expression )

DTinterval2stringAsTime
Returns a date time interval as a string in HHHH:MM:SS.FFFFFFF or HH:MM:SS.FFF format.
Days are converted to hours.

Syntax

DTinterval2stringAsTime ( date_time_interval_expression )

int2DTinterval
Converts an integer to a date time interval. "String_expression" specifies what "integer_expression"
represents: "ns" = nanoseconds, "s" = seconds (default), "m" = minutes, "h" = hours, "d" = days.

Syntax

int2DTinterval ( integer_expression , string_expression )

Example 1
int2DTinterval (1020,"h")
Result: 42 days 12 hours

Example 2
int2DTinterval (1020,"s")
Result: 17 minutes

int2YMinterval
Converts "integer_expression" to a year month interval. "String_expression" specifies what "integer_
expression" represents: "y" = years, "m" = months (default).

Syntax

int2YMinterval ( integer_expression , string_expression )
number2string
Converts "numeric_expression" to a string, using the %g format specifier (C/C++ syntax).

Syntax
number2string ( numeric_expression )

string2date
Returns "string_expression" as a date in YYYY-MM-DD format.

Syntax
string2date ( string_expression )

string2double
Returns a floating point number. "String_expression" has the following form: "[whitespace] [sign] [digits] [digits] [ {d|D|e|E }[sign]digits]"

Syntax
string2double ( string_expression )

string2DTinterval
Returns "string_expression" as a date time interval in [-]DD HH:MM[:SS[.FFF]] format.

Syntax
string2DTinterval ( string_expression )

string2int32
Returns an integer. "String_expression" has the following form: "[whitespace] [+|-] [digits]"

Syntax
string2int32 ( string_expression )

string2int64
Returns a long integer. "String_expression" has the following form: "[whitespace] [+|-] [digits]"

Syntax
string2int64 ( string_expression )

string2time
Returns "string_expression" as a time in HH:MM:SS.FFFFFFF format.

Syntax
string2time ( string_expression )

string2timestamp
Returns "string_expression" as a timestamp in YYYY-MM-DD [Tt][white space]+ HH:MM:SS.FFFFFFF format.
**Syntax**

`string2timestamp ( string_expression )`

**string2timestampTZ**


**Syntax**

`string2timestampTZ ( string_expression )`

**string2YMinterval**

Returns "string_expression" as a Year Month Interval in [-]YY MM format.

**Syntax**

`string2YMinterval ( string_expression )`

**time2string**

Returns a time as a string in HH:MM:SS.FFF format.

**Syntax**

`time2string ( time_expression )`

**timestamp2date**

Converts "timestamp_expression" to a date. The time part of the timestamp will be ignored.

**Syntax**

`timestamp2date ( timestamp_expression )`

**timestamp2string**

Returns a timestamp as a string in YYYY-MM-DD HH:MM:SS.FFFFFFFF format.

**Syntax**

`timestamp2string ( timestamp_expression )`

**timestamp2timestampTZ**

Converts "timestamp_expression" to a timestamp with a time zone. The displacement part of the timestamp with the time zone will be zero.

**Syntax**

`timestamp2timestampTZ ( timestamp_expression )`

**timestampTZ2date**

Converts "timestamp_time_zone_expression" to a date. The time and time zone parts of the timestamp will be ignored.
Appendix B: Using the Expression Editor

Syntax

timestampTZ2date ( timestamp_time_zone_expression )

timestampTZ2string
Returns a timestamp with the time zone as a string in YYYY-MM-DD HH:MM:SS.FFFFFF +HHMM or YYYY-MM-DD HH:MM:SS.FFF -HHMM format.

Syntax

timestampTZ2string ( timestamp_time_zone_expression )

timestampTZ2timestamp
Converts "timestamp_time_zone_expression" to a timestamp. The displacement part of the timestamp with the time zone will be ignored.

Syntax

timestampTZ2timestamp ( timestamp_time_zone_expression )

timeTZ2string
Returns a time with the time zone as a string in HH:MM:SS.FFF +HHMM or HH:MM:SS.FFFFFFFF -HHMM format. For example, -05:30 means a timezone of GMT minus 5 hours and 30 minutes

Syntax

timeTZ2string ( timeTZ_expression )

YMinterval2string
Returns "year_month_interval_expression" as a string in (YY MM) or -(YY MM) format.

Syntax

YMinterval2string ( year_month_interval_expression )
content store
The database that contains the data needed to operate, such as report specifications, published models, and security rights.

data source
The source of data itself, such as a database or XML file, and the connection information necessary for accessing the data.

In TM1®, the file or data used as the source for the TurboIntegrator import process.

level
A set of entities or members that form one section of a hierarchy in a dimension and represent the same type of object. For example, a geographical dimension might contain levels for country, region, and city.

measure
A performance indicator that is quantifiable and used to determine how well a business is operating. For example, measures can be Revenue, Revenue/Employee, and Profit Margin percent.

metric
A measure to assess performance in a key area of a business.

metric extract
A set of mappings between an existing Cognos data source and a Metric Studio object or value. For example, a cube measure named Revenue is mapped to a Metric Studio metric named Revenue Actual Value.

metric package
In Cognos Connection, a representation of a Metric Studio application. A metric package contains connection information, reports, and metric management tasks for that application.

metric store
A database that contains content for metric packages. A metric store also contains Metric Studio settings, such as user preferences.

metric type
A category of metrics that defines the business rules such as performance pattern, units, and meaning of a group of metrics. For example, Revenue can be a metric type, and European Revenue and North American Revenue would be metrics of this type.
**object extract**
An extract that defines the metadata for a Metric Studio object, such as a user defined column, a scorecard, or a data source.

**package**
A subset of a model, which can be the whole model, to be made available to the Cognos server.

**project**
In Framework Manager, a set of models, packages, and related information for administration, and for sharing model information.

In Metric Studio, a task or set of tasks undertaken by a team and monitored on a scorecard. A project tracks the dates, resources, and status of the project.

In Metric Designer, a group of extracts. Each extract contains the metadata that is used to populate the Metric Studio data store or to create applications.

**scorecard**
A collection of metrics representing the performance of one unit or aspect of an organization.

**user-defined column**
In metric management, a column used to represent a value other than the actual or target. It may be an industry benchmark or any other useful additional numerical information for a period, including a calculation based on the other values of the metric. User-defined columns may be different for each metric type.
Symbols
   .cpf, 9
   .iqd, 29
   .xml, 9
   .xsd, 9

C
   categories
      filtering, 38
   CCLAssert error message, 69
   content stores
      definition, 259
   cpf files, 9

D
   data loading, 45
   data sources
      definition, 259
   double-byte characters, 45

E
   errors
      troubleshooting, 69
   executing extracts, 45
   expression editor
      searching for values, 71
   extracts
      changing, 47
      creating, 13
      executing, 45
   metrics from multiple sources, 39
   objects, 40
   publishing, 45
   running, 45
   validating, 13

F
   files
      .cpf, 9
      .iqd, 29
      .xml, 9
      .xsd, 9

filtering categories for metrics, 38

I
   IBM Cognos Connection references, 43
   import sources, 17
   iqd files, 29

L
   language settings for PowerCubes, 45
   levels
      definition, 259
   loading data, 45

M
   measures
      definition, 259
   Metric Designer
      adding IQD files to an import source, 69
      blanks in scorecard hierarchy preview, 70
      CCLAssert error message, 69
      error processing template, 69
      no rollups generated, 69
      report not displayed, 69
   metric extracts
      definition, 259
   metric package references, 47
   metric packages
      definition, 259
   metrics
      creating extracts for .iqd files, 29
      creating extracts for cube-based Framework Manager packages, 17
      creating extracts for relational Framework Manager Packages, 23
      creating metric a package reference, 14
      definition, 259
      from different sources in one scorecard, 39
      viewing in Metric Studio, 46
metrics extract
definition, 9
metrics extracts
creating, 15
filtering categories, 38
naming, 16
metric stores
definition, 259
Metric Studio
viewing metrics, 46
metric types, 35
definition, 259
N
network paths, 15
O
object diagram, 10
object explorer, 10
object extracts
definition, 259
objects extracts, 40
P
packages
creating a metric package reference, 14
definition, 260
paths, 15
project explorer, 10
project page, 10
projects, 9
copying, 47
creating, 14
definition, 260
deleting, 49
managing, 47
moving, 48
object diagram, 10
object explorer views, 10
project explorer, 10
properties pane, 10
renaming, 48
upgrading, 49
properties pane, 10
publishing extracts, 45
R
referenced objects
validating, 42
S
scorecards
definition, 260
including metrics from different sources, 39
T
troubleshooting, 69
tutorial
create an import source, 52
create a project, 51
creating a metrics extract from a cube, 51
creating a scorecard hierarchy from multiple
dimensions, 59
execute extract, 56, 64
map metrics, 55, 64
map scorecards, 52, 60
metric type, 54, 63
select metric values, 55
select reports, 54, 63
select time period, 54, 63
time and currency mappings, 53, 63
transfer data from staging area to metrics store, 57,
65
view scorecards, 57, 65
U
user-defined columns
definition, 260
V
validating
extracts, 13
referenced objects, 42
X
xml files, 9
xsd files, 9