Before using this information and the product it supports, read the information in "Notices" on page 265.

Note

Product Information

This document applies to IBM Cognos Business Intelligence Version 10.2.0 and may also apply to subsequent releases. To check for newer versions of this document, visit the IBM Cognos Information Centers [http://publib.boulder.ibm.com/infocenter/cogic/v1r0m0/index.jsp].

Licensed Materials - Property of IBM


US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
## Contents

### Introduction
- vii

### Chapter 1. Metric Designer
- 1
  - Projects
  - The project page
  - Icons

### Chapter 2. Creating extracts
- 7
  - Security
  - Creating a project
    - Starting a project
    - Creating a metric package reference
  - Creating metrics extracts
    - Naming the metrics extract
    - Specifying an import source for the extract
    - Creating a metrics extract for a cube-based framework manager package
    - Creating a metrics extract for a relational framework manager package
    - Creating a metrics extract for an .iqd file
  - Creating a metric type
    - General settings
    - Specifying how metric values are calculated
    - Specifying performance behavior
    - Setting metric type security
  - Filtering categories in a metrics extract
  - Combining metrics from different sources on one scorecard
  - Creating an objects extract
    - Validating referenced objects
    - Checking IBM Cognos Connection references

### Chapter 3. Transferring extracts and data to Metric Studio
- 35
  - Transferring extracts and data to Metric Studio
    - Running an extract
    - Publishing an extract
    - Viewing metrics in Metric Studio

### Chapter 4. Managing projects
- 37
  - Changing a metric package reference
  - Modifying an extract definition
  - Copying a project
  - Moving a project
  - Renaming a project
  - Deleting a project
  - Upgrading a project

### Chapter 5. Tutorial on creating a metrics extract from a PowerCube
- 41
  - Before you begin
  - Creating a project
  - Selecting an import source for your extract
  - Mapping the scorecards
  - Setting time and currency mappings
  - Selecting dimensions to display as reports
  - Selecting a time period
  - Creating a metric type
  - Mapping the metrics
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 (http://publib.boulder.ibm.com/infocenter/cogic/v1r0m0/index.jsp). Release Notes are published directly to Information Centers, and include links to the latest technotes and APARs.

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 used IBM Cognos Data Manager or custom database scripts to load their Metric Studio application. You can use Metric Designer 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 Business Intelligence 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 might 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 by 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 that 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 is displayed. 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 might 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 1. Icons that represent objects and states**

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Project Icon" /></td>
<td></td>
</tr>
<tr>
<td>Represents the project.</td>
<td></td>
</tr>
<tr>
<td><img src="image2" alt="Namespace Icon" /></td>
<td></td>
</tr>
<tr>
<td>Represents the root namespace or any other namespace in the project.</td>
<td></td>
</tr>
<tr>
<td><img src="image3" alt="Folder Icon" /></td>
<td></td>
</tr>
<tr>
<td>Represents a folder that contains one or more data sources.</td>
<td></td>
</tr>
<tr>
<td>Icon</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td><img src="data-source-icon.png" alt="Image" /></td>
<td>Represents a data source.</td>
</tr>
<tr>
<td><img src="package-icon.png" alt="Image" /></td>
<td>Represents a package.</td>
</tr>
<tr>
<td><img src="query-subject-icon.png" alt="Image" /></td>
<td>Represents a query subject within a model.</td>
</tr>
<tr>
<td><img src="query-item-icon.png" alt="Image" /></td>
<td>Represents a query item.</td>
</tr>
<tr>
<td><img src="query-item-under-shortcut-icon.png" alt="Image" /></td>
<td>Represents a query item that is located under a shortcut query subject.</td>
</tr>
<tr>
<td><img src="measure-icon.png" alt="Image" /></td>
<td>Represents a measure.</td>
</tr>
<tr>
<td><img src="calculation-icon.png" alt="Image" /></td>
<td>Represents a calculation.</td>
</tr>
<tr>
<td><img src="filter-icon.png" alt="Image" /></td>
<td>Represents a filter.</td>
</tr>
<tr>
<td><img src="dimension-icon.png" alt="Image" /></td>
<td>Indicates that the object is a dimension (this icon overlays other icons).</td>
</tr>
<tr>
<td><img src="shortcut-icon.png" alt="Image" /></td>
<td>Indicates that the object is a shortcut (this icon overlays other icons).</td>
</tr>
<tr>
<td><img src="metric-studio-connection-folder-icon.png" alt="Image" /></td>
<td>Represents the IBM Cognos Metric Studio connection folder.</td>
</tr>
<tr>
<td><img src="new-connection-icon.png" alt="Image" /></td>
<td>Represents a new connection.</td>
</tr>
<tr>
<td><img src="extracts-folder-icon.png" alt="Image" /></td>
<td>Represents an extracts folder.</td>
</tr>
<tr>
<td><img src="metrics-extract-icon.png" alt="Image" /></td>
<td>Represents a metrics extract.</td>
</tr>
<tr>
<td><img src="objects-or-text-file-extract-icon.png" alt="Image" /></td>
<td>Represents an objects extract or a text file extract.</td>
</tr>
<tr>
<td>Icon</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>![Cube Icon]</td>
<td>Represents a cube data source.</td>
</tr>
<tr>
<td>![Lock Icon]</td>
<td>Represents an .iqd data source.</td>
</tr>
</tbody>
</table>
**Chapter 2. Creating extracts**

You can 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 actions:

- Create a project "Creating a project" on page 8 or open an existing project.
- Create an import source "Naming the metrics extract" on page 10.
- Do one of the following actions:
  - Create a metrics extract for a cube-based IBM Cognos Framework Manager package "Creating a metrics extract for a cube-based framework manager package" on page 11.
  - Create a metrics extract for a relational Framework Manager package "Creating a metrics extract for a relational framework manager package" on page 16.
  - Create a metrics extract for an .iqd file "Creating a metrics extract for an .iqd file" on page 22.
  - Create an objects extract "Creating an objects extract” on page 32.

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 Chapter 3, “Transferring extracts and data to Metric Studio,” on page 35.

**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.</td>
</tr>
<tr>
<td></td>
<td>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 fails.</td>
</tr>
</tbody>
</table>
Table 2. Tasks and the permissions required to perform them (continued)

<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 fails.</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>

Creating 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.

Starting a project

You can use Metric Designer to start a new project.

Procedure

1. If Metric Designer is running, click File, 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.

Creating a metric package reference

You can create a reference for each new metric package.

Procedure

1. On the Create Metric Package Reference page, click a package. If the package that you want to reference is not displayed, 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 Business Intelligence 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 IBM Cognos 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 might want to create them in Metric Studio first, before you create the metrics extracts in Metric Designer. Examples of when you might want to do this include the following situations:

- The scorecard structure, the qualifiers, or both might not be present in the IBM Cognos data source.
- The IBM Cognos data source might provide only part of the scorecard tree.
  For example, the source has data only 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 IBM Cognos 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 are displayed 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 is not displayed. Because you can filter data at various stages while you are creating a metrics extract, the data that is shown might 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.

**Use network paths for file-based data sources.**

If you have a distributed installation with several servers, 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 are 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, store it in a shared directory that can be accessed on your network.

If you installed IBM Cognos Business Intelligence components on UNIX servers, locate the file-based data source on a UNIX server. Then use a UNIX path, such as `/servername/cubes/Great Outdoors Company.mdc` to access the file.

If you installed IBM Cognos Business Intelligence components on Microsoft Windows and UNIX computers, you can use both paths to ensure that the report server that runs 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 that request the data have the appropriate access to the data files on the computer.

For Windows distributed installations, use UNC paths to shared directories for any file-based data sources, such as cubes or XML files.

**Naming the metrics extract**

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

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**.
Procedure
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

In order to specify an import source for the extract, each extract must be associated with an import source.

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) must 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.

Before you begin

You must select one of the following import sources:

- Dimensional IBM Cognos Business Intelligence packages that are based on cubes.
- Relational IBM Cognos Framework Manager packages that were published.
- IBM Cognos Impromptu Query Definition (.iqd) files.

Creating a metrics extract for a cube-based framework manager package

You can use a cube-based Framework Manager package as an import source.

Specifying an import source

You can specify an import source by selecting a specific date source.

Procedure
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, click **Actions**, **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 Business Intelligence 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**.
Mapping the scorecard

You can map a scorecard by selecting the items you want to map.

Procedure

1. In the **Create Extract - Scorecard Mapping** page, if you want to change the parent scorecard, do one of the following actions:
   - 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 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 actions:
   - 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 be displayed.
   - **Combining metrics from different sources on one scorecard** on page 31.
   - 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 results 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 actions:
   - 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**.

**Attention:** 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. The `roleValue` function is in the Functions tab of the expression editor under Common Functions, R-Z, Advanced. For example, if you are creating scorecards for products by using the Great Outdoor Company sample, and you want to filter out the Golf Equipment product category by 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:

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

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

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

8. Click **Preview Level Data** to view the data that is 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.

### Mapping time and currency
You can map time and currency based on the scorecard you want to create.

**Procedure**

1. If the **Time and Currency Mappings** page is displayed, 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 is redisplayed.

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 is displayed.

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 is displayed.

**Selecting a time period**
You can select a time period for the extract you want to create.

**Procedure**
1. If the Create Extract - Time Periods Filtering page is displayed, 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.

**Mapping metrics for several types of columns**
You can map metrics for several types of columns.

**Procedure**
1. If the Create Extract - Metrics Mapping page is displayed, in the Available objects pane do the following actions:
   - 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 "Creating a metric type" on page 27.
   - 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 actions:

   - 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 *IBM Cognos Metric Studio User Guide*.

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

   **Table 3. Rollup Aggregate function rules**

<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>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 in the expression of a data item 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>

You can use the **Rollup Aggregate function** to specify how detail values are summarized for the scorecards.

4. If you want to create a metric, do the following actions:

   - 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 “Creating a metric type” on page 27.
   - Click **OK**.
5. If you want to select a metric to filter on, under the Metric filters pane, do the following actions:
   - 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.

Applying filters
You can create and apply filters based on your design criteria.

Procedure
1. If the Create Extract - Filter Data page is displayed, 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. See "Filtering categories in a metrics extract" on page 30.
5. Click Finish.
6. Save your project.

Results
Your extract is now complete. For information about using the extract in Metric Studio, see Chapter 3, “Transferring extracts and data to Metric Studio,” on page 35.

Creating a metrics extract for a relational framework manager package
You can use a relational Framework Manager package as an import source.

Specifying an import source
You can specify the import source that is needed.

Procedure
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, click Actions, 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 is displayed.

**Mapping the scorecard**

You can map the scorecard based on the required criteria.

**Procedure**

1. In the Create Extract - Scorecard Mapping page, if you want to change the parent scorecard, do one of the following actions:
   - 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, 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 actions:

   - 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 the metric to be displayed. “Combining metrics from different sources on one scorecard” on page 31.
   - 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 results 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 actions:
   - 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. The roleValue function is in the Functions tab of the expression editor under Common Functions, R-Z, Advanced. For example, if you are creating scorecards for products by using the Great Outdoor Company sample, and you want to filter out the Golf Equipment product category by 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:

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

6. Repeat steps 3 to 5 for each level you that 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 is 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 is displayed.
   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.

**Selecting a time period**

You can select the time period that is required for this extract.

**Procedure**

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 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 is displayed 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 actions:
   • 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 is 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.

**Mapping metrics based on required criteria**

You can map metrics as needed based on the required criteria.

**Procedure**

1. If the **Create Extract - Metrics Mapping** page is displayed, in the **Available objects** pane, do the following actions:
   • 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 [“Creating a metric type” on page 27](#).
• 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 actions:
   • 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 **IBM Cognos 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 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>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 in the expression of a data item 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 before the application of any other operations.

For example, a relational fact table might contain transaction records at the granularity of store (geography) and SKU (product), but a metric extract author might be interested in looking at only 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 is associated with a regularAggReate value that is specific in the Framework Manager model. The regularAggReate 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 can 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 metric, do the following actions:
   • 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 Creating a metric type on page 27.
   • Click OK.

5. If you want to select a metric to filter on, under the Metric filters pane, do the following actions:
   • 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 is 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.

Applying filters
You can set up and apply filters based on the required criteria.

Procedure
1. If the Create Extract - Filter Data page is displayed, 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 display 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.
   "Filtering categories in a metrics extract" on page 30.
5. Click Finish.
6. Save your project.

Results

Your extract is now complete. For information about using the extract in Metric Studio, see Chapter 3, “Transferring extracts and data to Metric Studio,” on page 35.

Creating a metrics extract for an .iqd file

You can use an .iqd file as an import source.

To use a relational Framework Manager package as an import source, see
"Creating a metrics extract for a relational framework manager package" on page 16.

To use a cube-based package as an import source, see “Creating a metrics extract for a cube-based framework manager package” on page 11.

To use a relational Framework Manager package as an import source, see
"Creating a metrics extract for a relational framework manager package" on page 16.

Specifying an import source

You can specify an import source needed for your extract.

Procedure

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, click Actions, 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 actions:
   • 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 is re-displayed.
   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 Business Intelligence Administration and Security Guide*.

**Mapping the scorecard**

You can map the scorecard based on the required criteria.

**Procedure**

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 action:
   - 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 be displayed.
   - 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 results 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 actions:
   - 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. The roleValue function is in the Functions tab of the expression editor under Common Functions, R-Z, Advanced. For example, if you are creating scorecards for products by using the Great Outdoor Company sample, and you want to filter out the Golf Equipment product category by 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] )
```

6. Click Next or Finish.
   - If you click Next, the Create Extract - Time Hierarchy Attributes page is displayed.
   - 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.

### Selecting a time period
You can select a time period for the extract you want to create.

**Procedure**

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 is displayed 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 actions:
• 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 is displayed, where you map measures to metric types.

Mapping metrics
You can map metrics based on a number of required criteria.

Procedure
1. If the Create Extract - Metrics Mapping page is displayed, in the Available Objects pane, do the following actions:
   • 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 "Creating a metric type" on page 27.
   • 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 actions:
   • 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 IBM Cognos 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 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>Minimum</td>
<td>The minimum metric value.</td>
</tr>
<tr>
<td>Maximum</td>
<td>The maximum metric value.</td>
</tr>
</tbody>
</table>
**Rule** | **Description**
---|---
Calculated | This can also be thought of as aggregate-then-calculate. That is, all the terms in the expression of a data item are aggregated according to their own rollup rules and the results of those aggregations are then computed within the overall expression.
Summarize | Metric values are aggregated in the most obvious, or simplistic, manner, based on either model or data type information.
Count Distinct | A distinct count of metric values.

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

For example, a relational fact table might contain transaction records at the granularity of store (geography) and SKU (product), but a metric extract author might be interested in looking at only metric values for city, and product type scorecards. You can use the aggregate attribute 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 is associated with 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 can 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 actions:
   - Under **Metric**, 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 “Creating a metric type” on page 27.
   - Click **OK**.

5. If you want to select a metric to filter on, under the **Metric filters** pane, do the following actions:
   - 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**.

6. Click **Preview metric data** to view the data that is 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.

**Applying filters**

You can apply the filters needed based on the required criteria.

**Procedure**

1. If the **Create Extract - Filter Data** page is displayed, 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 ("Filtering categories in a metrics extract" on page 30).

5. Click **Finish**.

6. Save your project.

**Results**

Your extract is now complete. For information about using the extract in Metric Studio, see Chapter 3, “Transferring extracts and data to Metric Studio,” on page 35.

---

**Creating a metric type**

You can 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.

**General settings**

You can complete the general settings based on the required criteria.

**Procedure**

1. Do one of the following actions:
Complete the steps in the Create Extract wizard until you reach the Create Extract - Metrics Mapping page. 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 actions:
   - 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 IBM Cognos Metric Studio User Guide.
10. Under Calendar Details, do the following actions:
    - 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 actions:
    - 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 actions:
    - Type the value for Minimum value.
    - Type the value for Maximum value.
    - If you want, select the Show zero value check box.
Specifying how metric values are calculated

You can specify how metric values are calculated.

**Procedure**

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 *IBM Cognos Metric Studio User Guide*.
3. If you select **Define calculation**, do the following actions:
   - Click **Edit**.
   - Define the expression for the calculation.
   - Click **OK**.
4. Under **Actual** and **Target**, do the following actions:
   - 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 actions:
   - 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 actions:
   - 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.

Specifying performance behavior

You can specify performance behavior based on a defined criteria.

**Procedure**

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 *IBM Cognos Metric Studio User Guide*.

3. Under **Score Settings**, do the following actions:
   - 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.

**Setting metric type security**

You can set your own metric type security.

**Procedure**

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*.

**Filtering 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, 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 can 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.

**Procedure**

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 level to display in the scorecard hierarchy. If the filter happens at an intermediate level then any of the items excluded at the level and their descendants are 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 is not 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 is excluded, all of its descendant members from the Product Type and Product levels are also excluded. In the expression editor, this would look as follows:

```
[great outdoors].[Products].[Products].[Product Line] not in ([great outdoors].[Products].[Products].[Product Line]->:[PC].[@MEMBER].[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 is that no child members of Golf Accessories at the Product level are included in the scorecard tree and the aggregated metric value calculated for the Golf Equipment scorecard at the Product Line level does not include the value for Golf Accessories. In the expression editor, this would look as follows:

```
[great outdoors].[Products].[Products].[Product Type] not in ([great outdoors].[Products].[Products].[Product Type]->:PC.[@MEMBER].[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 example, you could also decide that this particular extract have only product data related to Canada. In the expression editor, this would look as follows:

```
[great outdoors].[Locations].[Locations].[Country] in ([great outdoors].[Locations].[Locations].[Country]->:PC.[@MEMBER].[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 are created for Golf Accessories, however no metrics are 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 might 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 are displayed on the same scorecard.

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

Creating 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 you are advised to use the metrics extract wizard [Creating metrics extracts" on page 9].

Before you begin

You must already have created a project.

Procedure

1. In the main page, in the Project Explorer pane, click the extract folder that you want to create an objects extract under.
2. Click Actions, 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 is displayed 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 actions:
   • 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 is redisplayed.
• Under **Specify connection**, click the database that you want and click the ellipse (...) 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 Business Intelligence Administration and Security Guide*.

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

7. If you want to import a new package import source, do the following actions:
   • 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 Business Intelligence Administration and Security Guide*.

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

8. When the **Create Objects Extract - Select Object Type** page is displayed, 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 actions:
    • Click the ellipsis (...) next to a value.

      The **Expression Editor** page is displayed, 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 actions:
    • 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.

Results

You can now publish or run your extract. For more information, see Chapter 3, “Transferring extracts and data to Metric Studio,” on page 35.

Validating 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 might want to validate the referenced objects after you create the extract.

Procedure

1. In the main page, in the Projects Explorer pane, click the package or metrics extract that you want.
2. Click Actions, Show all referenced objects.
   The Metrics object page is displayed 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 actions:
   • Click the metrics extract that contains the object.
   • Click Actions, Edit definition.
   • Click the Metrics mapping tab and correct the problem.

Checking 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 can 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.

Procedure

1. In the main page, click Project, 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 Business Intelligence 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 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.

**Transferring 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. 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 Business Intelligence Administration and Security Guide* and the *IBM Cognos Metric Studio User Guide*.

**Running an extract**

You can run an extract that specifies the media to write to.

**Procedure**

1. In the main page, in the Project Explorer pane, click the metrics extract that you want to run.
2. Click Actions, 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.

**Publishing an extract**

You can publish an extract to the content store.

**Procedure**

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. Click Actions, 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 Business Intelligence Administration and Security Guide.

**Viewing 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 IBM Cognos Metric Studio User Guide.
Chapter 4. Managing projects

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

You can also change a metric package reference and modify an extract definition.

Projects can also be upgraded from an older version of Metric Designer to a newer version.

Changing a metric package reference

You can change the metric package reference through the Project Explorer pane.

Procedure
1. In the main page, in the Project explorer pane, click the metric package reference that you want to change.
2. Click Actions, 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 Business Intelligence Administration and Security Guide.

Modifying 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.

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

Copying a project

You might 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 change 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: 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 overwrite the project in the target location.

You can also move a project "Moving a project."

**Before you begin**

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

**Procedure**

1. Click File, 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 is displayed 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.

**Moving a project**

You might 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 which can have the same effect as moving a project "Copying a project" on page 37.

**Before you begin**

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

**Procedure**

1. Click File, 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 is displayed 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.

**Renaming 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 is displayed 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 by using the Open command from the File menu.

**Before you begin**

Before you can rename a project, the project must be closed in Metric Designer.
Procedure
1. Click File, 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 is displayed in the text box.
3. In the To text box, type the new name for the project.

Results
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.

Deleting 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 begin
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.

Procedure
1. Click File, 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 is displayed in the text box.

Results
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.

Upgrading 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 is displayed indicating that the project was created in an earlier version of Metric Designer and asking if you want to specify a location for the backup file.

Before you begin
Before upgrading a project with a model that is 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.

Procedure
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 does 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 generate scorecards by creating a metrics extract from an IBM Cognos PowerCube and transferring data into a metric store. You then view the scorecards in IBM Cognos Metric Studio.

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

Before you begin

Before you begin the tutorial, you must ensure that settings and access to servers are in place.
- 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.

Creating a project

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

Procedure
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.

Selecting an import source for your extract

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

Procedure
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 file name into either the **Windows location** box or **UNIX or Linux location** box. This file is in the installation folder for IBM Cognos Business Intelligence: Cognos\c10\webcontent\samples\datasources\cubes\PowerCubes\En\.

10. At the bottom of the page, click **Test the Connection**. After you confirm 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**.

---

**Mapping the scorecards**

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

**Procedure**

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.

![](Create%20Extract%20-%20Scorecard%20Mapping.png)

**Figure 1. Mapping the scorecards**

2. Click **Preview Scorecard Tree** to generate the following preview.
3. Close the preview page and click Next.

Setting 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 be displayed. You can access the page by clicking Edit Metric Studio Time Mappings at the bottom of the Time Periods Filtering page.

Procedure

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.


**Selecting dimensions to display as reports**

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

**Procedure**

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.

**Selecting a time period**

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

**Procedure**

Select Use All Time Periods and click Next.

**Creating a metric type**

Before you can map the metrics, you must create at least one metric type.
Procedure
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.

Mapping the metrics
You 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.

Procedure
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.

Figure 4. A dimension dragged from the Available Objects pane to the target attribute

3. Click Next.
Selecting the metric values to include on the scorecards

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

Procedure

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.
Executing the extract

You create a metrics extract, and now you 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 use the Execute command to write to the metric staging area.

Procedure

1. In the Project Explorer pane, right-click Tutorial_Extract, and then click Execute.
2. Select Write to metric staging area and click OK.

Transferring data from the staging area to the metric store

You 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.
Procedure
1. Open Metric Studio and click Tutorial_Package.
2. Click Tools, Metric maintenance.
3. Click Transfer data from staging area into metric store.
The transfer might take several minutes.

Viewing your scorecards

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

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

![Figure 7. A list of the various order methods](image)

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, you generate scorecards that use 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 begin the tutorial, you must ensure that settings and access to servers are in place.

- 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 Sample Outdoors Company that references an empty metric store called Sample Outdoors Company.

Creating 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.

Procedure

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 ppds.cfg.xml file in the configuration folder.

12. Locate the Sales and Marketing.mdc file and paste or type its path and file name into either the Windows location box or UNIX or Linux location box. This file is in the installation folder for IBM Cognos Business Intelligence: Cognos\c10\webcontent\samples\datasources\cubes\PowerCubes\En\ 

13. At the bottom of the page, click Test the Connection. After you confirm 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 is displayed.

Mapping the scorecards

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

Procedure

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 else you 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 **Preview Scorecard Tree**, the duplicate IDs are resolved.
6. Click Close.

7. To set time and currency mappings, click Next.

**Setting time and currency mappings**

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

**Procedure**

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 that follow 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**.

Selecting dimensions to display as reports

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

Procedure
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.

Selecting a time period

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

Procedure
Select Use All Time Periods and click Next.

Creating a metric type

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

Procedure
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.

Mapping the revenue metric

You can 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.

Procedure
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.

### Executing the extract

You create a metrics extract, and now you 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 use the **Execute** command to write to the metric staging area.

**Procedure**

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**.

### Transferring data from the staging area to the metric store

You can 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.

**Procedure**

1. Open Metric Studio and click **Great Outdoors Company**.
2. Click **Tools, Metric maintenance**.
3. Click **Transfer data from staging area into metric store**.
   - The transfer might take several minutes.

### Viewing your scorecards

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

**Procedure**

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

In IBM Cognos Metric Studio, you can now view the metrics you created under the Revenue metric type.

Procedure

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

Even though you created unique scorecards IDs, there seem 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.
Customizing metric names

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

Procedure
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. Click Available Components, 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.

**Figure 15.** List of metrics with the region appended to create unique names
Appendix A. Troubleshooting

This chapter describes some common problems you might 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 by 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

IBM Cognos Metric Designer might not generate any rollups for an extract that references a calculated member of an SAP Info Query data source. The affected measures 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, IBM Cognos Metric Designer creates an outer join between the first pair of nonnumeric, 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 by using IBM Cognos Impromptu.

**Previewed scorecard hierarchy shows blanks**

If you use if () then () else () statements in an expression for level attributes, you 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:

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

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

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

```plaintext
[great_outdoors].[Locations].[Locations].[CountryRegion].[CountryRegion] = 'Canada' or [great_outdoors].[Locations].[Locations].[CountryRegion].[CountryRegion] = '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 might return unexpected results

In the expression editor, when searching for values for a data item, the results you obtain might contain unexpected results if the data item is not a string data type. Because you can edit the expression for a data item, IBM Cognos Business Intelligence cannot determine with certainty what the data type is.

Therefore, IBM Cognos Business Intelligence guesses the data type of the data item by looking at its aggregate and rollup aggregate set.

Calculation components

You can build calculations, or expressions, in the expression editor using the components that are defined in this section.

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**
value1 / value2

||
Concatenates, or joins, strings.

**Syntax**
string1 || string2

+
Adds two numeric values.

**Syntax**
value1 + value2

-
Subtracts two numeric values or negates a numeric value.

**Syntax**
value1 - value2
or
- value

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

**Syntax**
value1 < value2

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

**Syntax**
value1 <= value2

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

**Syntax**
value1 <> 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 value1 and value2

Example  
[Revenue] between 200 and 300

Result  
Returns the number of results with revenues between 200 and 300.

Result data

<table>
<thead>
<tr>
<th>Revenue</th>
<th>Between</th>
</tr>
</thead>
<tbody>
<tr>
<td>$332.06</td>
<td>false</td>
</tr>
<tr>
<td>$230.55</td>
<td>true</td>
</tr>
<tr>
<td>$107.94</td>
<td>false</td>
</tr>
</tbody>
</table>
**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**
if ( condition ) then .... else ( expression ) , or case .... else ( expression ) end

**end**
Indicates the end of a case or when construct.

**Syntax**
ends with
Determines if "string1" ends with "string2".

Syntax
string1 ends with string2

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

Syntax
aggregate_function ( expression for expression { , 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
aggregate_function ( expression for ALL expression { , 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
aggregate_function ( expression for ANY expression { , 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 ).

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

**Syntax**
value is missing

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

**Syntax**
value is null

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

**Syntax**
value is not missing
**is not null**
Determines 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_function ([expression] prefilter)

**Example**
total ( [Quantity] for report prefilter )
summaryFilter: total([Quantity] for [ProductNo]) > 50000

**Result**
Sums the quantities in a report before the summary filter is applied.

**Result data**

<table>
<thead>
<tr>
<th>ProductNo</th>
<th>Total_forProductNo</th>
<th>Total_forReport</th>
<th>Total_forReport_Prefilter</th>
</tr>
</thead>
<tbody>
<tr>
<td>88</td>
<td>54928</td>
<td>298140</td>
<td>2215354</td>
</tr>
<tr>
<td>89</td>
<td>51126</td>
<td>298140</td>
<td>2215354</td>
</tr>
<tr>
<td>ProductNo</td>
<td>Total_forProductNo</td>
<td>Total_forReport</td>
<td>Total_forReport_Prefilter</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------------</td>
<td>-----------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>90</td>
<td>69996</td>
<td>298140</td>
<td>2215354</td>
</tr>
<tr>
<td>94</td>
<td>69004</td>
<td>298140</td>
<td>2215354</td>
</tr>
<tr>
<td>95</td>
<td>53086</td>
<td>298140</td>
<td>2215354</td>
</tr>
</tbody>
</table>

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

**Syntax**
```sql
count ( ROWS )
```

**starts with**
Determines if "string1" starts with "string2".

**Syntax**
```sql
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**
```sql
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**
```sql
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**
```sql
aggregate ( expression [ auto ] )
aggregate ( expression for [ all|any ] expression { , expression } )
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**
average ( [ distinct ] expression [ auto ] )
average ( [ distinct ] expression for [ all|any ] expression ( , expression ) )
average ( [ distinct ] expression for report )

Example
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
count ( [ distinct ] expression [ auto ] )
count ( [ distinct ] expression for [ all|any ] expression ( , expression ) )
count ( [ distinct ] expression for report )

Example
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
maximum ( [ distinct ] expression [ auto ] )
maximum ( [ distinct ] expression for [ all|any ] expression ( , expression ) )
maximum ( [ distinct ] expression for report )

Example
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.

**Result data**

<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.

Result data

<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 ] ] )

Example

percentage ( Sales 98 )

Result

Returns the percentage of the total sales for 1998 that is attributed to each sales representative.

Result data

<table>
<thead>
<tr>
<th>Sales Rep</th>
<th>Sales 98</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gibbons</td>
<td>60646</td>
<td>7.11%</td>
</tr>
<tr>
<td>Flertjan</td>
<td>62523</td>
<td>7.35%</td>
</tr>
<tr>
<td>Cornel</td>
<td>22396</td>
<td>2.63%</td>
</tr>
</tbody>
</table>

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

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

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).

Result data
<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>
</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 ] )
<for-option> ::= for expression { , expression }|for report|auto

Example
quartile ( Qty )

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

Result data
<table>
<thead>
<tr>
<th>Qty</th>
<th>Quartile (Qty)</th>
</tr>
</thead>
<tbody>
<tr>
<td>450</td>
<td>1</td>
</tr>
<tr>
<td>400</td>
<td>1</td>
</tr>
<tr>
<td>350</td>
<td>2</td>
</tr>
<tr>
<td>300</td>
<td>2</td>
</tr>
<tr>
<td>250</td>
<td>3</td>
</tr>
<tr>
<td>200</td>
<td>3</td>
</tr>
<tr>
<td>150</td>
<td>4</td>
</tr>
<tr>
<td>100</td>
<td>4</td>
</tr>
</tbody>
</table>

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

Example
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.

Result data

<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

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

Example

```plaintext
running-average ( Qty )
```

Result

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

Result data

<table>
<thead>
<tr>
<th>Name</th>
<th>Qty</th>
<th>Avg</th>
<th>Running-Average for name</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

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

Example

```plaintext
running-count ( Qty )
```
running-count ( Qty )

Result

For each row, returns the quantity and a running count of the position of the current row.

Result data

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

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 ] )
running-difference ( numeric_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.

Result data

<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.

Result data

<table>
<thead>
<tr>
<th>Name</th>
<th>Qty</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>Smith</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>Smith</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Smith</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Wong</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Wong</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.

Result data

<table>
<thead>
<tr>
<th>Name</th>
<th>Qty</th>
<th>Min</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>Smith</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Smith</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>Smith</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Wong</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Wong</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>
**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**

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

**Example**

running-total ( Qty )

**Result**

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

**Result data**

<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**

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

**Example**

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**

standard-deviation-pop ( expression [ auto ] )  
standard-deviation-pop ( expression for [ all|any ] expression { , expression } )  
standard-deviation-pop ( expression for report )
Example
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
total ( [ distinct ] expression [ auto ] )
total ( [ distinct ] expression for [ all|any ] expression { , expression } )
total ( [ distinct ] expression for report )

Example
total ( Sales )

Result

Returns the total value of all Sales values.

variance
Returns the variance of selected data items.

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

Example
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
variance-pop ( expression [ auto ] )
variance-pop ( expression for [ all|any ] expression { , expression } )
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.

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

Example
maximum ( Sales )

Result
Returns the maximum value out of all Sales values.

median
Returns the median value of selected data items.

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

minimum
Returns the minimum value of selected data items.

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

Example
minimum ( 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
percentage ( numeric_expression [ tuple member_expression { , member_expression } ] within set set_expression )

Example
percentage ( [gosales].[sales measures].[quantity] tuple [gosales].[Staff].[].[department] -> [West] within set children ( [gosales].[Staff].[].[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
percentile ( numeric_expression [ 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**

\[
\text{variance (} \text{< currentMeasure|numeric_expression>} \text{ within set set_expression)}
\]

\[
\text{variance (} \text{< currentMeasure|numeric_expression>} \text{ within } \text{< detail|aggregate > expression)}
\]

**variance-pop**
Returns the variance population of the selected data items.

**Syntax**

\[
\text{variance-pop (} \text{< currentMeasure|numeric_expression>} \text{ within set set_expression)}
\]

\[
\text{variance-pop (} \text{< currentMeasure|numeric_expression>} \text{ within } \text{< detail|aggregate > expression)}
\]

**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**

```sql
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**

```sql
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

_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
_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**

```plaintext
_firstFromSet ( set_expression , numeric_expression_maximum ,
   numeric_expression_overflow )
```

**Example 1**

```plaintext
_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.

**Result data**

- 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
**Aster Data**

**Aster Data String**

**overlay:**

Returns the "string_exp1" replacing "string_exp2" from character position numeric_exp.

**Syntax**

overlay ( string_exp1, string_exp2, numeric_exp1 [, numeric_exp2] )

**btrim:**

Returns string_exp1 after removing the longest string of characters in string_exp2.

**Syntax**

btrim ( string_exp1 [, string_exp2] )

**initcap:**

Returns "string_exp", 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_exp )

**lpad:**

Returns "string_exp1" padded to length "integer_exp" with occurrences of "string_exp2". If "string_exp1" is longer than "integer_exp", the appropriate portion of "string_exp1" is returned.

**Syntax**

lpad ( string_exp1, integer_exp [, string_exp2 ] )

**ltrim:**

Returns "string_exp1", with leading characters removed up to the first character not in 'string_exp2', e.g., ltrim('xyxXxyAB', 'xy') returns 'XxyAB'.

**Syntax**

ltrim ( string_exp1 [, string_exp2 ] )

**md5:**

Returns the MD5 hash of "string_exp1".

**Syntax**

md5 ( string_exp1 )

**to_hex:**

Returns the hexadecimal string representation of "numeric_exp1".
**Syntax**

to_hex ( numeric_exp1 )

**repeat:**

Returns the "string_exp" repeated "numeric_exp1" times.

**Syntax**

repeat ( string_exp, numeric_exp1 )

**replace:**

Returns "string_exp" having replaced "string_exp2" with "string_exp3".

**Syntax**

replace( string_exp, string_exp2, string_exp3)

**rpad:**

Returns "string_exp1" right-padded to length "integer_exp" with occurrences of "string_exp2". If "string_exp1" is longer than "integer_exp", the appropriate portion of "string_exp1" is returned. If "string_exp2" is not specified, then spaces are used.

**Syntax**

rpad ( string_exp1, integer_exp [ , string_exp2 ] )

**rtrim:**

Returns "string_exp1", with final characters removed after the last character not in "string_exp2", e.g., rtrim('ABxXyx', 'xy') returns 'ABxX'. If "string_exp2" is not specified, the final space characters are removed.

**Syntax**

rtrim ( string_exp1 [ , string_exp2 ] )

**split_part:**

Returns "numeric_exp" field having split "string_exp1" on "string_exp2".

**Syntax**

split_part ( string_exp1 , string_exp2 , numeric_exp )

### Aster Data Data Type Formatting

**to_char:**

Returns the string representation of "exp" with the format of "string_exp". "Exp" can be either a date value or a numeric value.

**Syntax**

to_char ( exp , string_exp )

**to_date:**

Converts "string_exp1" to a date value as specified by the format "string_exp2".
Syntax
to_date ( string_exp1, string_exp2 )

to_number:
Converts "string_exp1" to a numeric value as specified by the format "string_exp2".

Syntax
to_number ( string_exp1, string_exp2 )

to_timestamp:
Converts "string_exp1" to a timestamp value as specified by the format "string_exp2".

Syntax
to_timestamp ( string_exp1, string_exp2 )

**Aster Data Math**

log:
Returns the base 10 logarithm of "numeric_exp1" or logarithm to the base "numeric_exp2".

Syntax
log ( numeric_exp1[, numeric_exp2] )

ln:
Returns the natural logarithm of "numeric_exp1".

Syntax
ln ( numeric_exp )

cbrt:
Returns the cube root of "numeric_exp1".

Syntax
cbrt ( numeric_exp )

pi:
Returns the constant of pi.

Syntax
pi( )

**Aster Data Trigonometry**

acos:
Returns the arccosine of "numeric_exp" in radians. The arccosine is the angle whose cosine is "numeric_exp".
Syntax
acos ( numeric_exp )

asinh:

Returns the arcsine of "numeric_exp" in radians. The arcsine is the angle whose sine is "numeric_exp".

Syntax
asin ( numeric_exp )

atan:

Returns the arctangent of "numeric_exp" in radians. The arctangent is the angle whose tangent is "numeric_exp".

Syntax
atan ( numeric_exp )

atan2:

Returns the arctangent of the x and y coordinates specified by "numeric_exp1" and "numeric_exp2", respectively, in radians. The arctangent is the angle whose tangent is "numeric_exp2" / "numeric_exp1".

Syntax
atan2 ( numeric_exp1, numeric_exp2 )

cos:

Returns the cosine of "numeric_exp", where "numeric_exp" is an angle expressed in radians.

Syntax
cos ( numeric_exp )

cot:

Returns the cotangent of "numeric_exp", where "numeric_exp" is an angle expressed in radians.

Syntax
cot ( numeric_exp )

degrees:

Returns the degrees where "numeric_exp" is an angle expressed in radians.

Syntax
degrees ( numeric_exp )

radians:

Returns the radians where "numeric_exp" is an angle expressed in degrees.
Syntax
radians ( numeric_exp )

sin:

Returns the sine of "numeric_exp", where "numeric_exp" is an angle expressed in radians.

Syntax
sin ( numeric_exp )

tan:

Returns the tangent of "numeric_exp", where "numeric_exp" is an angle expressed in radians.

Syntax
tan ( numeric_exp )

ascii

Returns a number representing the ASCII code value of the leftmost character of "string_exp", e.g., ascii('A') is 65.

Syntax
ascii ( string_exp )

chr

Returns the character that has the ASCII code value specified by "integer_exp". "Integer_exp" should be between 0 and 255.

Syntax
chr ( integer_exp )

current_schema

Returns the name of the current schema

Syntax
current_schema ()

translate

Returns "string_exp1", with each occurrence of each character in "string_exp2" replaced by its corresponding character in "string_exp3".

Syntax
translate ( string_exp1, string_exp2, string_exp3 )

date_trunc

Returns the timestamp to the specified precision.

Syntax
date_trunc ( string_exp, timestamp_exp)
**version**

Returns the string value of the database version.

**Syntax**

`version()`

---

**DB2 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 )`

**rand:**

Generates a random number using "integer_expression" as a seed value.

**Syntax**

`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**

`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 )

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:
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 )

ascii
Returns the ASCII code value of the leftmost character of the argument as an integer.

Syntax
ascii ( string_expression )

Example
ascii ( a )

Result
Returns 65, the ASCII code value of "a".

ceiling
Returns the smallest integer greater than or equal to "numeric_expression".

Syntax
ceiling ( numeric_expression )

Example
ceiling ( 0.75 )

Result
Returns 0.8.
**char**
Returns a string representation of a date/time value or a decimal number.

**Syntax**
`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**
`chr ( integer_expression )`

**Example**
`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**
`concat ( string_expression1, string_expression2 )`

**Example**
`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' )`

**Result**
Returns 8 January 1998.

**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
Returs 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
dec ( string_expression1 [, numeric_expression1 [, numeric_expression2 [, string_expression2 ] ] ] )

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
decimal ( string_expression1 [, numeric_expression1 [, numeric_expression2 [, string_expression2 ] ] ] )

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])
Example 2

\[
\text{difference ([Sales target (query)].[Sales staff].[First name], [Sales target (query)].[Sales staff].[First name])}
\]

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

104
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
```plaintext
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
```plaintext
integer ( expression )
```

Example
```plaintext
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
```plaintext
int ( expression )
```

Example
```plaintext
int ( 84.95 )
```

Result
85

**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
```plaintext
julian_day ( expression )
```

Example
```plaintext
julian_day ( '2009-06-29' )
```

Result
2455012.22130739595741034
**lcase**
Returns "string_expression" with all uppercase characters shifted to lowercase.

**Syntax**
lcase ( string_expression )

**Example**
lcase ( [Sales (query)].[Sales staff].[Last name] )

**Result**
Returns last names with no uppercase letters.

**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 length of the operand in bytes. Exception: double byte string types return the length in characters.

**Syntax**
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**

```
midnight_seconds ( expression )
```

**Example**

```
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
minute ( time_expression )

Example
minute ( 01:45:34.056 )

Result
Returns 45.

month
Returns the month (an integer from 1-12) from "date_expression".

Syntax
month ( date_expression )

Example
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
monthname ( date_expression )

Example
monthname ( 2005-11-01 )

Result
November

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
quarter ( date_expression )

Example
quarter ( 2005-11-01 )

Result
Returns 4.
radians
Returns the number of radians converted from "numeric_expression" degrees.

Syntax
radians ( numeric_expression )

repeat
Returns a string consisting of "string_expression" repeated "integer_expression" times.

Syntax
repeat ( string_expression, integer_expression )

Example
repeat ( XYZ, 3 )

Result
Returns XYZXYZXYZ.

replace
Replaces all occurrences of "string_expression2" in "string_expression1" with "string_expression3".

Syntax
replace ( string_expression1, string_expression2, string_expression3 )

Example
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
right ( string_expression, integer_expression )

Example
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.

**table_name**
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.nnnnn). 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**

\[
\text{translate} \left( \text{string_expression1} \ [, \text{string_expression2}, \text{string_expression3} \ [, \text{string_expression4} \ ] \right)
\]

**Example 1**

\[
\text{translate} \left( \text{'abcdefg'} \right)
\]

**Result**

Returns ABCDEFG.

**Example 2**

\[
\text{translate} \left( \text{'mnlop'}, \text{n}, \text{m}, \text{-} \right)
\]

**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**

\[
\text{trunc} \left( \text{numeric_expression1}, \text{numeric_expression2} \right)
\]

**Example**

\[
\text{trunc} \left( \text{3.14159265}, \text{3} \right)
\]

**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**

\[
\text{truncate} \left( \text{numeric_expression1}, \text{numeric_expression2} \right)
\]

**Example**

\[
\text{truncate} \left( \text{3141.59265}, \text{-3} \right)
\]

**Result**
Returns 3.

**ucase**

Returns "string_expression" with all lowercase characters shifted to uppercase.

**Syntax**

```
ucase ( string_expression )
```

**Example**

```
ucase ( XY896Zbcd789 )
```

**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 )

Result

Returns 2005.

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 ( ‘xxyXxyAB’ , ‘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 )

**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**

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 )

**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 )

**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
```sql
current_database()
```

**current_schema**
Returns the name of the current schema.

Syntax
```sql
current_schema()
```

**{current_user}**

Syntax
```sql
{current_user}
```

**{session_user}**

Syntax
```sql
{session_user}
```

**translate**
Returns "string_expression1" with each occurrence of each character in "string_expression2" replaced by its corresponding character in "string_expression3".

Syntax
```sql
translate (string_expression1, string_expression2, string_expression3)
```

**date_trunc**
Returns the timestamp to the specified precision.

Syntax
```sql
date_trunc (string_expression, timestamp_expression)
```

**version**
Returns the string value of the database version.

Syntax
```sql
version()
```

**Informix**

**Informix Math**

**log10:**
Returns the logarithm of "numeric_expression" to base 10.

Syntax
```sql
log10 ( numeric_expression )
```
Returns the natural logarithm of "numeric_expression".

Syntax

\[ \text{logn} \ ( \text{numeric\_expression} ) \]

\text{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

\[ \text{root} \ ( \text{numeric\_expression1} \ [ \ , \text{numeric\_expression2} \ ] ) \]

\textbf{Informix Trigonometry}

\text{acos:}

Returns the arccosine of "numeric\_expression" in radians. The arccosine is the angle whose cosine is "numeric\_expression".

Syntax

\[ \text{acos} \ ( \text{numeric\_expression} ) \]

\text{asin:}

Returns the arcsine of "numeric\_expression" in radians. The arcsine is the angle whose sine is "numeric\_expression".

Syntax

\[ \text{asin} \ ( \text{numeric\_expression} ) \]

\text{atan:}

Returns the arctangent of "numeric\_expression" in radians. The arctangent is the angle whose tangent is "numeric\_expression".

Syntax

\[ \text{atan} \ ( \text{numeric\_expression} ) \]

\text{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

\[ \text{atan2} \ ( \text{numeric\_expression1} \ , \text{numeric\_expression2} \ ) \]

\text{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 )

cardinality
Returns the number of elements in a collection column (SET, MULTISER, 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 )

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**

\texttt{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**

\texttt{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**

\texttt{weekday ( date\_expression )}

**year**

Returns a four-digit integer that represents the year of "date\_expression".

**Syntax**

\texttt{year ( date\_expression )}

**MS Access**

**MS Access Cast**

**cast\_decimal:**

Returns the value of "expression" cast as a decimal.

**Syntax**

\texttt{cast\_decimal ( expression )}

**cast\_float:**

Returns the value of "expression" cast as a float.

**Syntax**

\texttt{cast\_float ( expression )}

**cast\_integer:**

Returns the value of "expression" cast as an integer.

**Syntax**

\texttt{cast\_integer ( expression )}

**Example**

\texttt{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**

\[ \text{cos ( numeric_expression )} \]

\[ \text{sin ( numeric_expression )} \]

\[ \text{tan ( numeric_expression )} \]

**ascii**

Returns the ascii code value of the leftmost character of "string_expression".

**Syntax**

\[ \text{ascii ( string_expression )} \]

**ceiling**

Returns the smallest integer greater than or equal to "numeric_expression".

**Syntax**

\[ \text{ceiling ( numeric_expression )} \]

**chr**

Returns the character that has the ASCII code value specified by "integer_expression". "Integer_expression" should be between 0 and 255.

**Syntax**

\[ \text{chr ( integer_expression )} \]

**concat**

Returns a string that is the result of concatenating, or joining, "string_expression1" to "string_expression2".

**Syntax**

\[ \text{concat ( string_expression1 , string_expression2 )} \]

**Example**

\[ \text{concat ( [Sales (query)].[Sales staff].[First name], [Sales (query)].[Sales staff].[Last name] )} \]

**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".

**Syntax**
```
year ( date_expression )
```

**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**
```
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 ] )

**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 )

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.

Syntax

tan ( 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 )

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 ()

Netezza

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 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 )
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 )

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 )

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
decode ( expr , search , result [ , search , result]... [ , default] )

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 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 is 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 ] ] )
```

**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, \texttt{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 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] , 0 )
```
**Result**

Returns the unit sale price, or returns 0 if the unit sale price is NULL.

**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 ] )
```

**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 ] )
```

**substr**

Returns the substring of "string_expression" that starts at position "integer_expression1". The first character in "string_expression" is at position 1. "Integer_expression2" can be used to select fewer characters; by default it selects characters to the end of the string.

**Syntax**

```
substr ( string_expression , integer_expression1 [ , integer_expression2 ] )
```
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".

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 ()
Oracle

**Oracle Math**

**log:**

Returns the logarithm of "numeric_expression2" to the base "numeric_expression1".

**Syntax**

\[ \text{log} \left( \text{numeric_expression1} , \text{numeric_expression2} \right) \]

**Oracle Trigonometry**

**acos:**

Returns the arccosine of "numeric_expression" in radians. The arccosine is the angle whose cosine is "numeric_expression".

**Syntax**

\[ \text{acos} \left( \text{numeric_expression} \right) \]

**asin:**

Returns the arcsine of "numeric_expression" in radians. The arcsine is the angle whose sine is "numeric_expression".

**Syntax**

\[ \text{asin} \left( \text{numeric_expression} \right) \]

**atan:**

Returns the arctangent of "numeric_expression" in radians. The arctangent is the angle whose tangent is "numeric_expression".

**Syntax**

\[ \text{atan} \left( \text{numeric_expression} \right) \]

**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**

\[ \text{atan2} \left( \text{numeric_expression1} , \text{numeric_expression2} \right) \]

**cos:**

Returns the cosine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

**Syntax**

\[ \text{cos} \left( \text{numeric_expression} \right) \]
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 )

**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**

**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 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] , 0 )

Result
Returns the unit sale price, or returns 0 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 ( 'ABxXxyx' , '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 )
```
Result

Returns characters 3 to 7 of the position codes.

**substrb**

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**

```sql
substrb ( string_expression , numeric_expression1 [ ,
          numeric_expression2 ] )
```

**Example**

```sql
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**

```sql
{ 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**

```sql
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**

```sql
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**

```sql
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**

```sql
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**

```sql
trunc ( date_expression, string_expression )
```

**Example**

```sql
trunc ( '2003-08-22', 'year' )
```

**Result**

Returns 2003-01-01.

**trunc**

Truncates digits from "numeric_expression1" using "numeric_expression2" as the precision.

**Syntax**

```sql
trunc ( numeric_expression1, numeric_expression2 )
```

**{user}**

Returns the username of the current Oracle user.

**Syntax**

```sql
{ user }
```

**vsize**

Returns the number of bytes in the internal representation of "expression". "Expression" must be a string expression.

**Syntax**

```sql
vsize ( 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 ABxX. If "string_expression2" is not specified, the final space characters are removed.

Syntax
rtrim (string_expression1 [, string_expression2])

Paracel 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
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)
Paraccel Math

cbrt:
Returns the cube root of "numeric_expression1".

Syntax
cbrt ( numeric_expression )

pi:
Returns the constant of pi.

Syntax
pi ()

**current_database**
Returns the name of the current database.

Syntax
current_database ()

**current_schema**
Returns the name of the current schema.

Syntax
current_schema ()

**{current_user}**

Syntax
{current_user}

**{session_user}**

Syntax
{session_user}

**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 )

**version**
Returns the string value of the database version.

Syntax
version ()
Postgres

Postgres 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 ] )

md5:

Returns the MD5 hash of "string_expression1".

Syntax
md5 ( string_expression1 )
to_hex:

Returns the hexadecimal string representation of "numeric_expression1".

Syntax

\texttt{to\_hex\ (\ numeric\_expression1\ )}

repeat:

Returns the "string_expression" repeated "numeric_expression1" times.

Syntax

\texttt{repeat\ (\ string\_expression\ ,\ numeric\_expression1\ )}

replace:

Returns "string_expression" with "string_expression2" replaced with "string_expression3".

Syntax

\texttt{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

\texttt{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

\texttt{rtrim\ (\ string\_expression1\ [\ ,\ string\_expression2\ ]\ )}

split_part:

Returns "numeric_expression" field having split "string_expression1" on "string_expression2".

Syntax

\texttt{split\_part\ (\ string\_expression1\ ,\ string\_expression2\ ,\ numeric\_expression\ )}

\textbf{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**

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 )

**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".

**Syntax**

div ( numeric_expression1, numeric_expression2 )

**pi:**

Returns the constant of pi.

**Syntax**

pi()

**Postgres 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 )

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 )
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 ()

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**
```
decode ( expression , target , replacement [ ,default ])
```

**float**
Converts "numeric_expression" into a double-precision floating-point value.

**Syntax**
```
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**
```
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**
```
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**
```
icenter ( numeric_expression )
```

**Example**
```
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**

```
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**

```
lengthb ( string_expression )
```

**ltrim**

Removes leading blanks from "string_expression". If "string_expression" is null, it returns null.

**Syntax**

```
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**

```
nul1f ( 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**

```
positionb ( string_expression1, string_expression2 )
```

**real**

Returns a real value. If "numeric_expression" is null, it returns null.

**Syntax**

```
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
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**
time ( expression )

**timestamp**
Creates a time-stamp value from "timestamp_expression", which is a character string.

**Syntax**
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**
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**
to_char ( source_date , format_string )

---

**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 )

**Salesforce.com**

**Date Functions**

**CALENDAR_MONTH:**

Returns a number representing the calendar month of "date_expression".

**Syntax**

CALENDAR_MONTH ( date_expression )

**Example**

CALENDAR_MONTH ( '2012-02-29' )

**Result**

2

**CALENDAR_QUARTER:**

Returns a number representing the calendar quarter of "date_expression".

**Syntax**

CALENDAR_QUARTER ( date_expression )

**Example**

CALENDAR_QUARTER ( '2012-02-29' )

**Result**

1

**CALENDAR_YEAR:**

Returns a number representing the calendar year of "date_expression".

**Syntax**

CALENDAR_YEAR ( date_expression )

**Example**

CALENDAR_YEAR ( '2012-02-29' )

**Result**
2012

**DAY_IN_MONTH:**

Returns a number representing the day in the month of "date_expression".

**Syntax**

```
DAY_IN_MONTH ( date_expression )
```

**Example**

```
DAY_IN_MONTH ( '2012-02-29' )
```

**Result**

29

**DAY_IN_WEEK:**

Returns a number representing the day of the week for "date_expression" 1 for Sunday, 7 for Saturday.

**Syntax**

```
DAY_IN_WEEK ( date_expression )
```

**Example**

```
DAY_IN_WEEK ( '2012-02-29' )
```

**Result**

4 (Wednesday)

**DAY_IN_YEAR:**

Returns a number representing the day in the year for "date_expression".

**Syntax**

```
DAY_IN_YEAR ( date_expression )
```

**Example**

```
DAY_IN_YEAR ( '2012-02-29' )
```

**Result**

60

**DAY_ONLY:**

Returns a date representing the day portion of 'dateTime_expression'.

**Syntax**

```
DAY_ONLY ( dateTime_expression )
```

**Example**

```
DAY_ONLY ( '2012-02-29T23:00:01Z' )
```
FISCAL_MONTH:

Returns a number representing the fiscal month of "date_expression". This differs from CALENDAR_MONTH() if your organization uses a fiscal year that does not match the Gregorian calendar. If your fiscal year starts in March, 1 for March 12 for February.

Syntax
FISCAL_MONTH( date_expression )

Example
FISCAL_MONTH( '2012-02-29' )

Result
12

FISCAL_QUARTER:

Returns a number representing the fiscal quarter of "date_expression". This differs from CALENDAR_QUARTER() if your organization uses a fiscal year that does not match the Gregorian calendar. If your fiscal year starts in July, 1 for July 15 and 4 for June 6

Syntax
FISCAL_QUARTER( date_expression )

Example
FISCAL_QUARTER( '2012-02-29' )

Result
3

FISCAL_YEAR:

Returns a number representing the fiscal year of "date_expression". This differs from CALENDAR_YEAR() if your organization uses a fiscal year that does not match the Gregorian calendar.

Syntax
FISCAL_YEAR( date_expression )

Example
FISCAL_YEAR( '2012-02-29' )

Result
2012

HOUR_IN_DAY:
Returns a number representing the hour in the day for "dateTime_expression".

**Syntax**

```
HOUR_IN_DAY ( dateTime_expression )
```

**Example**

```
HOUR_IN_DAY ( '2012-02-29T23:00:01Z' )
```

**Result**

23

**WEEK_IN_MONTH**:

Returns a number representing the week in the month for "date_expression". The first week is from the first through the seventh day of the month.

**Syntax**

```
WEEK_IN_MONTH ( date_expression )
```

**Example**

```
WEEK_IN_MONTH ( '2012-02-29' )
```

**Result**

5

**WEEK_IN_YEAR**:

Returns a number representing the calendar week of a "date_expression". The first week is from January 1 through January 7.

**Syntax**

```
WEEK_IN_YEAR ( date_expression )
```

**Example**

```
WEEK_IN_YEAR ( '2012-02-29' )
```

**Result**

9

**convertCurrency**

Converts "numeric_expression" to the user's currency when multicurrency is enabled.

**Syntax**

```
convertCurrency ( numeric_expression )
```

**convertTimezone**

Converts "dateTime_expression" to the user's time zone.

**Syntax**

```
convertTimezone ( dateTime_expression )
```
SQL Server

SQL Server 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 ()

rand:

Generates a random number using "integer_expression" as the seed value.

Syntax
rand ( 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
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_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 "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.

Syntax
\[ \tan(\text{numeric_expression}) \]

**ascii**

Returns a number representing the ASCII code value of the leftmost character of "string_expression".

**Syntax**

\[ \text{ascii}(\text{string_expression}) \]

**Example**

\[ \text{ascii}(\text{'A'}) \]

**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**

\[ \text{char}(\text{integer_expression}) \]

**Example**

\[ \text{char}(65) \]

**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**

\[ \text{charindex}(\text{string_expression1}, \text{string_expression2} [, , \text{start_location}]) \]

**{current_user}**

Returns the name of the current user.

**Syntax**

\[ \{\text{current_user}\} \]

**datalength**

Returns the length in bytes of "string_expression".

**Syntax**

\[ \text{datalength}(\text{string_expression}) \]

**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

datename ( ' { datepart } ' , date_expression )

Example

datename ( {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

datepart ( ' { datepart } ' , date_expression )

Example

datepart ( {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

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 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

**getdate**

Returns a datetime value representing the current date and time of the computer that the database software runs on.

**Syntax**

getdate ()

**isnull**

Returns the first non-null argument (or null if both arguments are null). It is recommended to use the sql standard COALESCE function instead.

**Syntax**

isnull ( expression , expression )

**Example**

isnull ( [Sales (query)].[Sales Fact].[Sales quantity] , 0 )

Result

Returns the sales quantity if it's not null, otherwise returns 0.

**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 )

**Sybase**

**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**

```
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 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**
 radians ( numeric_expression )

sin:

Returns the sine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

**Syntax**
sin ( numeric_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
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.

**Syntax**
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**

```
charindex ( string_expression1 , string_expression2 )
```

**datalength**

Returns the length in bytes of "string_expression".

**Syntax**

```
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**

```
dateadd ( ' { datepart } ' , integer_expression ,
           date_expression )
```

**Example**

```
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**

```
datediff ( ' { datepart } ' , date_expression1 ,
           date_expression2 )
```

**Example**

```
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**

```
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

getdate
Returns current system date and time.

Syntax
**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.

**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 )

Teradata

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 -π 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 )

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" can be any real number.

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" can be any real number.

**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" can be any real number.

**Syntax**
```
tanh(numeric_expression)
```

**account**

Returns the account string for the current user.

**Syntax**
```
{account}
```

**add_months**

Returns the date or the datetime resulting from adding "integer_expression" months to "date_expression" or "datetime_expression".

**Syntax**
```
add_months(date_expression|datetime_expression, integer_expression)
```

**bytes**

Returns the number of bytes contained in "byte_expression". "Byte_expression" is restricted to BYTE or VARBYTE.

**Syntax**
```
bytes(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**

```c
case_n ( condition_expression_list [, NO CASE|UNKNOWN|NO CASE OR UNKNOWN [, UNKNOWN ] ])
```

**char2hexint**

Returns the hexadecimal representation for "string_expression".

**Syntax**

```c
char2hexint ( string_expression )
```

**characters**

Returns an integer value representing the number of logical characters or bytes
contained in "string_expression".

**Syntax**

```c
characters ( string_expression )
```

**database**

Returns the name of the default database for the current user.

**Syntax**

```c
{database}
```

**date**

Returns the current date.

**Syntax**

```c
{date}
```

**format**

Returns the declared format for "expression" as a variable character string of up to
30 characters.

**Syntax**

```c
format ( expression )
```

**index**

Returns the starting position of "string_expression2" in "string_expression1".

**Syntax**

```c
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**

```c
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 )

Vectorwise

Vectorwise String

lpad:

Returns "string_exp1" padded to length "integer_exp" with occurrences of "string_exp2". If "string_exp1" is longer than "integer_exp", the appropriate portion of "string_exp1" is returned.

Syntax
lpad ( string_exp1 , integer_exp [ , string_exp2 ] )
ltrim:
Returns "string_exp1", with leading blank characters removed.

Syntax
ltrim ( string_exp1 )

rtrim:
Returns "string_exp1", with trailing blank characters removed.

Syntax
ltrim ( string_exp1 )

shift:
Returns "string_exp1" shifted by numeric_exp character. If "numeric_exp" is greater
than zero shift to the right else shift to the left.

Syntax
shift ( string_exp1, numeric_exp )

soundex:
Returns a four character code for the "string_exp1".

Syntax
soundex ( string_exp1 )

squeeze:
Returns "string_exp1" with multiple whitespace characters collapsed into a single
whitespace character.

Syntax
squeeze ( string_exp1 )

Vectorwise Math
log:
Returns the base 10 logarithm of "numeric_exp1".

Syntax
log ( numeric_exp1 )

ln:
Returns the natural logarithm of "numeric_exp1".

Syntax
ln ( numeric_exp )

pi:
Returns the constant of pi.
Vectorwise Trigonometry

acos:
Returns the arccosine of "numeric_exp" in radians. The arccosine is the angle whose cosine is "numeric_exp".

Syntax
acos ( numeric_exp )

asin:
Returns the arcsine of "numeric_exp" in radians. The arcsine is the angle whose sine is "numeric_exp".

Syntax
asin ( numeric_exp )

atan:
Returns the arctangent of "numeric_exp" in radians. The arctangent is the angle whose tangent is "numeric_exp".

Syntax
atan ( numeric_exp )

atan2:
Returns the arctangent of the x and y coordinates specified by "numeric_exp1" and "numeric_exp2", respectively, in radians. The arctangent is the angle whose tangent is "numeric_exp2" / "numeric_exp1".

Syntax
atan2 ( numeric_exp1 ,numeric_exp2 )

cos:
Returns the cosine of "numeric_exp" where "numeric_exp" is an angle expressed in radians.

Syntax
cos ( numeric_exp )

sin:
Returns the sine of "numeric_exp" where "numeric_exp" is an angle expressed in radians.

Syntax
sin ( numeric_exp )

tan:
Returns the tangent of "numeric_exp" where "numeric_exp" is an angle expressed in radians.

Syntax
tan ( numeric_exp )

{initial_user}
Returns the initial username.

Syntax
{ initial_user }

{session_user}
Returns the session username.

Syntax
{ session_user }

numeric_trunc
Returns the numeric_exp1 truncated at the decimal place or numeric_exp2 digits to the left or right of the decimal place.

Syntax
trunc ( numeric_exp1 , numeric_exp2 )

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 ( 'xyXxyAB' , '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 ('ABxXyx', 'xy') returns ABxX. If "string_expression2" is not specified, the final space characters are removed.

Syntax
rtrim ( string_expression1 [, string_expression2 ] )

**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
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 )

**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 ( \text{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.
cos (numeric_expression)

cot:

Returns the cotangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

cot (numeric_expression)

degrees:

Returns the degrees where "numeric_expression" is an angle expressed in radians.

degrees (numeric_expression)

radians:

Returns the radians where "numeric_expression" is an angle expressed in degrees.

radians (numeric_expression)

sin:

Returns the sine of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

sin (numeric_expression)

tan:

Returns the tangent of "numeric_expression" where "numeric_expression" is an angle expressed in radians.

tan (numeric_expression)

ascii

Returns a number representing the ASCII code value of the leftmost character of "string_expression"; for example, ascii ("A") is 65.

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.

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 ()

{current_user}

Syntax
{current_user}

{session_user}

Syntax
{session_user}

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 ()

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  
value1 + value2  

Example  
# '{ ' + $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_

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.

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**
getConfigurationEntry ( entry_string , force_decode_flag )

Example
# getConfigurationEntry ( 'serverLocale' ) #

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

Syntax
grep ( pattern_string , array_expression )

Example
# 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
_first_of_month ( string_expression )

Example 1
# _first_of_month ( '2005-11-11 12:00:00.000-05:00' )#

Result
2005-11-01 12:00:00.000000000-05:00

Example 2
# timestampMask ( _first_of_month ( '2005-11-11 12:00:00.000-05:00' ), 'yyyy-mm-dd' )#

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
_last_of_month ( string_expression )

Example 1
# _last_of_month ( '2005-11-11 12:00:00.000-05:00' )#

Result
2005-11-30 12:00:00.000000000-05:00

Example 2
# timestampMask ( _last_of_month ( '2005-11-11 12:00:00.000-05:00' ), 'yyyy-mm-dd' )#
join
Joins the elements of an array using "separator_string".

Syntax
join ( separator_string , array_expression )

Example
# sq ( join ( ' | ' , array ( 'as' , 'an' , 'arts' ) ) ) #

Result
'as | an | arts'

lstrip
Strips the leading characters from the first argument. The optional second argument defines the set of characters to strip. By default, this function strips white space (ie. space, tab, carriage return and line feed).

Syntax
lstrip ( string_expression [ , set_of_characters ] )

Example 1
# sq ( lstrip ( ' abc ' ) ) #

Result
'abc '

Example 2
' ) #

Result
53.2100

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
select . . . where COUNTRY_MULTILINGUAL.COUNTRY_CODE > #prompt ( 'Starting CountryCode' , 'integer' , '10' ) #
Result

select ... where COUNTRY_MULTILINGUAL.COUNTRY_CODE > 10

Example 2
[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.COUNTRY XX where
COUNTRY_MULTILINGUAL.COUNTRY_CODE = XX.COUNTRY_CODE # promptmany ( 'Selected CountryCodes' , 'integer' , '' , '' and
COUNTRY_MULTILINGUAL.COUNTRY_CODE in ( '' , '' , '') )#

Result
select ... from gosales.gosalesdbo.COUNTRY_MULTILINGUAL
COUNTRY_MULTILINGUAL, gosales.gosalesdbo.COUNTRY XX
where COUNTRY_MULTILINGUAL.COUNTRY_CODE = XX.COUNTRY_CODE and
COUNTRY_MULTILINGUAL.COUNTRY_CODE in ('Canada', 'The
Netherlands', 'Russia')

rstrip
Strips the trailing characters from the first argument. The optional second
argument defines the set of characters to strip. By default, this function strips
white space (ie. space, tab, carriage return and line feed).

Syntax
rstrip ( string_expression [ , set_of_characters ] )

Example 1
# sq(rstrip (' abc'))#

Result
' abc'

Example 2
' ' ) #

Result
0053.21

sb
Surrounds "string_expression" with square brackets.

Syntax
sb ( string_expression )

Example
# sb( 'abc' ) #

Result
[ abc]

sq
Surrounds "string_expression" with single quotes.

Syntax
sq ( string_expression )

Example
# sq ( 'zero' ) #

Result
'zero'
**sort**
Sorts the elements of the array in alphabetical order. Duplicates are retained.

**Syntax**
```sql
sort ( array_expression )
```

**Example**
```sql
# csv ( sort ( array ( 's3', 'a', 'x' )))#
```

**Result**
'\'a', 's3', 'x'\'

**split**
Splits a string or string elements of the array into separate elements.

**Syntax**
```sql
split ( pattern_string, string_expression|array_expression )
```

**Example 1**
```sql
# csv ( split ( '::', 'ab=c::de=f::gh=i'))#
```

**Result**
'ab=c', 'de=f', 'gh=i'

**Example 2**
```sql
# csv ( split ( '=' , split ( '::', 'ab=c::de=f::gh=i')))#
```

**Result**
'ab', 'c', 'de', 'f', 'gh', 'i'

**strip**
Strips the leading and trailing characters from the first argument. The optional second argument defines the set of characters to strip. By default, this function strips white space (ie. space, tab, carriage return and line feed).

**Syntax**
```sql
strip ( string_expression [ , set_of_characters ] )
```

**Example 1**
```sql
# sq( strip ( ' abc '))#
```

**Result**
'abc'

**Example 2**
```sql
' ' )#
```

**Result**
53.21
**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', 'yyyymm', 'yyyy-mm-dd', 'yyyyyymmd', 'yyyy-mm-dd hh:mm:ss', 'yyyy-mm-dd hh:mm:ss+hh:mm', 'yyyy-mm-dd hh:mm:ss.ff3', 'yyyy-mm-dd hh:mm:ss+hh:mm', 'yyyy-mm-ddThh:mm:ss', 'yyyy-mm-ddThh:mm:ss+hh:mm', 'yyyy-mm-ddThh:mm:ss.ff3', '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+00:00', 'yyyy-mm-dd hh:mm:ss' )#
```
Example 3

```c
# 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**

```c
# 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

**Example 2**

```c
# 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**

```c
# 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

**tolower**

Returns the string "string_expression" with all the characters converted to lower case using the rules of the locale "locale_string". If no locale is specified, the locale 'en' is used.

**Syntax**

tolower ( string_expression [, locale_string ] )

**Example 1**

```c
# tolower ( 'ABC' ) #
```

Result
Example 2
# tolower ( 'ABC' , 'fr' ) #

Result

abc

toupper
Returns the string "string_expression" with all the characters converted to upper case using the rules of the locale defined in "locale_string". If "locale_string" is not specified, the locale 'en' is used.

Syntax
toupper ( string_expression [ , locale_string ] )

Example 1
# toupper ( 'abc' ) #

Result

ABC

Example 2
# toupper ( 'abc' , 'fr' ) #

Result

ABC

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
# toUTC ( '2005-11-01 12:00:00.000-05:00' ) #

Result

2005-11-01 17:00:00.000000000-00:00

Example 2
# timestampMask( toUTC ( '2005-11-01 12:00:00.000-05:00' ), 'yyyy-mm-dd hh:mm:ss.fffffffff+hh:mm' ) #

Result

2005-11-01 17:00:00.000000-00:00
Example 3
# 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
# 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
eurlencode ( 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**

```python
CSVIdentityNameList ( [ separator_string ] )
```

**Example**

```
# CSVIdentityNameList ( ) #
```

**Result**

'Eeveryone', 'Report Administrators', 'Query User'

**CAMPassport**

Returns the Cognos® Access Manager passport.

**Syntax**

```python
CAMPassport ( )
```

**Example**

```
# CAMPassport ( ) #
```

**Result**

111:98812d62-4fd4-037b-4354-26414cf7ebe6:3677162321

**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.

**Syntax**

```python
CAMIDList ( [ separator_string ] )
```

**Example**

```
# 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**

```python
CAMIDListForType ( identity type )
```

**Example**

```
[qs].[userRole] in ( # csv ( CAMIDListForType ( 'role' ) ) # )
```
simple case
This macro construct is the template for a simple case, including the case, when, else, and end functions.

Syntax
CASE <expression> WHEN <literal> THEN <expression> [ELSE <expression>] END

Example
#CASE prompt('pDateRange','token') WHEN 'Current Week' THEN '[PR Current Week]' ELSE '[PR Prior Week]' END#

Result
[PR Current Week]

Common Functions

abs
Returns the absolute value of "numeric_expression". Negative values are returned as positive values.

Syntax
abs ( numeric_expression )

Example 1
abs ( 15 )

Result
15

Example 2
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 "datatypeSpecification": 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

```sql
cast ( expression , datatype_specification )
```

**Example 1**
```sql
cast ( '123' , integer )
```

**Result**

123

**Example 2**
```sql
cast ( 12345 , varchar ( 10 ) )
```

**Result**

a string containing 12345

**ceil**

Returns the smallest integer that is greater than or equal to "numeric_expression".

Syntax

```sql
ceil ( numeric_expression )
```

```sql
ceiling ( numeric_expression )
```

**Example 1**
```sql
ceiling ( 4.22 )
```

**Result**

5

**Example 2**
```sql
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 )
```

**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**
exttract ( 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".

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
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**

\[ \text{ancestors}( \text{member}, \text{level}|\text{index} ) \]

**Example 1**

\[ \text{ancestors}( \text{[TrailChef Water Bag]}, 1 ) \]

**Result**

Cooking Gear

**Example 2**

\[ \text{ancestors}( \text{[TrailChef Water Bag]}, 2 ) \]

**Result**

Camping Equipment

**Example 3**

\[ \text{ancestors}( \text{[TrailChef Water Bag]}, \text{[great_outdoors_company].[Products].[Products].[Product type]} ) \]

**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**

\[ \text{bottomCount}( \text{set_expression}, \text{index_expression}, \text{numeric_expression} ) \]

**Example**

\[ \text{bottomCount}( \text{[great_outdoors_company].[Products].[Products].[Product line]}, 2, [\text{Revenue}] ) \]

**Result**

Returns the bottom two members of the set sorted by revenue.

**Result data**

<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 the set specified in "set_expression" in ascending order and returns the bottommost elements from the sorted set whose cumulative percentage of the total is greater than or equal to "percentage".
Syntax
bottomPercent ( set_expression, percentage, numeric_expression )

Example
bottomPercent ( set ( [Camping Equipment], [Golf Equipment],
[Mountaineering Equipment]), 40, [2006] )

Result
For the set of Camping Equipment, Golf Equipment, and Mountaineering
Equipment, returns the members with the smallest Gross profit whose total for the
year 2006 is at least 40% of the overall total.

bottomSum
Sorts the set specified in "set_expression" in ascending order and returns the
bottommost elements from the sorted set whose cumulative total is greater than or
equal to "value".

Syntax
bottomSum ( set_expression, value, numeric_expression )

Example
bottomSum ( members ( [great_outdoors_company].[Products].[Products].
[Product line]), 6000000, tuple ( [2006],
[great_outdoors_company].[Measures].[Gross profit] ) )

Result
For the Product line members, returns the members with the smallest Gross profit
whose total for the year 2006 is at least $6,000,000.

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] )

Result
TrailChef Water Bag

Example 2
caption ( [great_outdoors_company].[Products].[Products].[Product
line] )

Result
Returns the caption values of the Product line set.
Result data
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.

Result data
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.

Result data
<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.

Result data
<table>
<thead>
<tr>
<th>Quantity Sold</th>
<th>Mountaineering Sales by Fax</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.

Result data
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.

Result data
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.

Result data
Camping Equipment
Cooking Gear
Sleeping Bags
Packs
Tents
...
Eyewear
Knives
Watches

Example 4
descendants ( [great_outdoors_company].[Products].[Products]. [Products], 2, self before )

Result

Returns the set of descendants of the Products set before and including the second level.

Result data
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
except ( set ( [Camping Equipment] , [Mountaineering Equipment] ) ,
set ( [Camping Equipment] , [Golf Equipment] ) )

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.

**Result data**

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.

**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**
hierarchize ( set ( [Golf Equipment], [Mountaineering Equipment], [Camping Equipment] ) )

**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.
Result data
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.

Result data
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 ])

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.

**Result data**

- 2006 Q 3
- 2006 Q 4

**Example 2**

\[ \text{lastPeriods}(-3, [2006 Q 4]) \]

**Result**

Returns the last three members from the level that starts with 2006 Q 4.

**Result data**

- 2006 Q 4
- 2007 Q 1
- 2007 Q 2

**lastSibling**

Returns the last child of the parent of a specified member.

**Syntax**

\[ \text{lastSibling} \ ( \text{member} ) \]

**Example**

\[ \text{lastSibling} \ ( [\text{Camping Equipment}] ) \]

**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**

\[ \text{lead} \ ( \text{member}, \text{index_expression} ) \]

**Example 1**

\[ \text{lead} \ ( [\text{Outdoor Protection}], 1 ) \]

**Result**

Personal Accessories

**Example 2**

\[ \text{lead} \ ( [\text{Outdoor Protection}], -2 ) \]

**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.

**Result data**
Camping Equipment
Mountaineering Equipment
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.

**Result data**
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.

**Result data**
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.

Result data

<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>345,696</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.

Result data

<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].[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].[Products].[Product name] , caption ( [great_outdoors_company].[Products].[Products].[Product name] ) starts with 'B' ) ) , 'BProducts' , 'B Products' , [great_outdoors_company].[Products].[Products] )

**Result**
Returns the quantity sold and revenue for all products that start with the letter B.

**nestedSet**
This function is intended for use only by Analysis Studio

**Syntax**
nestedSet ( set_expression1 , set_expression2 )
**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.

**Result data**
- 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 ] )
```

**Example 1**
```
tail (members ( [great_outdoors_company].[Products].[Products]. [Product line] ) )
```

**Result**
Returns the last member of the Product line set.

**Result data**
- Personal Accessories

**Example 2**
```
tail ( members ( [great_outdoors_company].[Products].[Products]. [Product line] ), 2 )
```
Result

Returns the last two members of the Product line set.

Result data
Outdoor Protection
Personal Accessories

**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.

Result data

<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 the set specified in "set_expression" in descending order and returns the topmost elements from the sorted set whose cumulative percentage of the total is greater than or equal to "percentage".

**Syntax**
topPercent ( set_expression , percentage , numeric_expression2 )

**Example**

Result

For the set of Camping Equipment, Golf Equipment, and Mountaineering Equipment, returns the members with the largest Gross profit whose total for the year 2006 is at least 40% of the overall total.

**topSum**
Sorts the set specified in "set_expression" in descending order and returns the topmost elements from the sorted set whose cumulative total is greater than or equal to "value".

**Syntax**
topSum ( set_expression , value , numeric_expression2 )
Example

topSum ( children ( [Products] ) , 16000000 , tuple ( [2006] ,
[great_outdoors_company].[Measures].[Gross profit] ) )

Result

For the Product line members, returns the members with the largest Gross profit
whose total for the year 2006 is at least $6,000,000.

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

union ( set ( [Camping Equipment] , [Golf Equipment] ) , set ( [Golf
Equipment] , [Mountaineering Equipment] ) )

Result

Returns data for both sets as one new set, showing the Golf Equipment column
only once.

Example 2

union ( set ( [Camping Equipment] , [Golf Equipment] ) , set ( [Golf
Equipment] , [Mountaineering Equipment] ) , all )

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.

Result data
("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.

Result data
[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**
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**
subset ( set_expression, index_expression1 [ , index_expression2 ] )

**Example 1**
subset ( members ( [great_outdoors_company].[Products].[Products].[Product line] ), 2 )

**Result**

Returns the members of the Product line set starting at the second member.

**Result data**
Mountaineering Equipment
Outdoor Protection
Personal Accessories

**Example 2**
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.

**Result data**
Mountaineering Equipment
Outdoor Protection

**unique**
Removes all duplicates from "set_expression". The remaining members retain their original order.

**Syntax**
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
value ( tuple )

Example 1
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
value ( tuple ( [2004] , [Camping Equipment] , [Revenue] ) )

Result
$20,471,328.88

Report Functions

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 ()

CellValue
Returns the value of the current crosstab cell.

Syntax
CellValue ()

**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

```plaintext
CubeSchemaUpdatedOn ( dimension )
```

**GetLocale**
Returns the run locale (deprecated).

Syntax

```plaintext
GetLocale ()
```

**HorizontalPageCount**
Returns the current horizontal page count.

Syntax

```plaintext
HorizontalPageCount ()
```

**HorizontalPageNumber**
Returns the current horizontal page number.

Syntax

```plaintext
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

```plaintext
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

```plaintext
IsAccessible()
```

**IsBursting**
Returns Boolean 1 (true) when the report will be distributed to the recipient; otherwise, returns Boolean 0 (false).

Syntax

```plaintext
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.

Syntax
IsOuterMostCrosstabColumnNodeMember ()

Appendix B. Using the expression editor  249
**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 ()

---

**Locale**
Returns the run locale.

**Syntax**

Locale ()

---

**ModelPath**
Returns the model path.

**Syntax**

ModelPath ()

---

**Now**
Returns the current system time.

**Syntax**

Now ()

---

**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 ()

---

**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')
```

**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 ()

**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, 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.
**ReportSaveDate**
Returns the date when the report was last saved.

**Syntax**
ReportSaveDate ()

**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 web server where the run request originated from. The value may be empty if the request is executed from the scheduler.

**Syntax**
ServerName ()

**TOCHeadingCount**
Returns the table of contents heading count for a specified heading level.

**Syntax**
TOCHeadingCount ( headingLevel )

**Today**
Returns the current system date.

**Syntax**
Today ()

**URLEncode**
Returns the URL encoded value of the input text.

**Syntax**
URLEncode ('text')

### Data Type Casting 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.

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 )

ceiling
Returns the smallest integer that is greater than or equal to "numeric_expression".

Syntax
ceiling ( numeric_expression )
**character_length**
Returns the number of characters in "string_expression".

**Syntax**
character_length ( string_expression )

**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 )

**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**
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".

**Syntax**
floor ( numeric_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**
in2DTinterval ( integer_expression , string_expression )

**Example 1**
in2DTinterval (1020,"h")

**Result**
42 days 12 hours

**Example 2**
in2DTinterval (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 )

**ln**
Returns the natural logarithm of "numeric_expression".

**Syntax**

\[
\text{ln} ( \text{numeric\_expression} )
\]

**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**

\[
\text{lower} ( \text{string\_expression} )
\]

**mapNumberToLetter**
Adds "integer\_expression" to "string\_expression".

**Syntax**

\[
\text{mapNumberToLetter} ( \text{string\_expression} , \text{integer\_expression} )
\]

**Example**

mapNumberToLetter ( 'a', 1 )

**Result**

b

**mod**
Returns an integer value representing the remainder (modulo) of "integer\_expression1" / "integer\_expression2".

**Syntax**

\[
\text{mod} ( \text{integer\_expression1} , \text{integer\_expression2} )
\]

**nullif**
Returns null if "string\_expression1" equals "string\_expression2" (case-insensitive), otherwise returns "string\_expression1".

**Syntax**

\[
\text{nullif} ( \text{string\_expression1} , \text{string\_expression2} )
\]

**number2string**
Converts "numeric\_expression" to a string, using the %g format specifier (C/C++ syntax).

**Syntax**

\[
\text{number2string} ( \text{numeric\_expression} )
\]

**octet\_length**
Returns the number of bytes in "string\_expression".

**Syntax**

\[
\text{octet\_length} ( \text{string\_expression} )
\]
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

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

sqrt
Returns the square root of "numeric_expression". "Numeric_expression" must not be a negative value.

Syntax
sqrt ( 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]"
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[.FFFFFF] format.

**Syntax**
string2time ( string_expression )

**string2timestamp**
Returns "string_expression" as a timestamp in YYYY-MM-DD [T|t][white space]+ HH:MM:SS[.FFFFFF] 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 )

**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 )

Result
Returns characters 3 to 7 of the position codes.

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.FFFFFFF 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.

Syntax
timestampTZ2date ( timestamp_time_zone_expression )

timestampTZ2string
Returns a timestamp with the time zone as a string in YYYY-MM-DD
HH:MM:SS.FFFFFFF +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 )

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 )

YMinterval2string
Returns "year_month_interval_expression" as a string in (YY MM) or -(YY MM) format.

Syntax
YMinterval2string ( year_month_interval_expression )
Notices

This information was developed for products and services offered worldwide.

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

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

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

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

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

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

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.
IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Software Group
Attention: Licensing
3755 Riverside Dr
Ottawa, ON K1V 1B7
Canada

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM’s future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.
Trademarks

IBM, the IBM logo, ibm.com, Impromptu, and Cognos are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "[Copyright and trademark information]" at www.ibm.com/legal/copytrade.shtml.

The following terms are trademarks or registered trademarks of other companies:

- Netezza is a registered trademark or trademark of Netezza Corporation, an IBM Company.
- Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
- Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
- UNIX is a registered trademark of The Open Group in the United States and other countries.
Glossary

This glossary includes terms and definitions for IBM Cognos Business Intelligence.

The following cross-references are used in this glossary:

- See refers you from a term to a preferred synonym, or from an acronym or abbreviation to the defined full form.
- See also refers you to a related or contrasting term.

To view glossaries for other IBM products, go to www.ibm.com/software/globalization/terminology (opens in new window).

A

access permission
A privilege that permits the access or use of an object.

accountability scorecard
A scorecard that Metric Studio automatically builds for each user which contains the metrics and projects they own.

agent
A process that performs an action on behalf of a user or other program without user intervention or on a regular schedule, and reports the results back to the user or program.

alias
An alternative name used instead of a primary name.

anonymous access
A type of access that allows users and servers to access a server without first authenticating with it.

application tier component
For installation, the set of processors that access the query databases to gather information and then render the results as PDF and HTML reports and metrics. Application tier components also pass requests to Content Manager and render the results that Content Manager retrieves from the content store.

attribute
In BI Modeling, a characteristic of an entity which is descriptive rather than a unique identifier or an aggregative measure.

authentication
The process of validating the identity of a user or server.

authentication provider
The communication mechanism to an external authentication source. Functionalities, such as user authentication, group membership, and namespace searches, are made available through authentication providers.

B

burst
To create several report results by running a single report once. For example, the user can create a report that shows sales for each employee, and run it once, sending different results to regional managers by bursting on region.

burst key
The dimension or level of a query in the report specification that is used to create, or burst, a set of report results.

C

CA
See certificate authority.

calculated member
A member of a dimension whose measure values are not stored but are calculated at run time using an expression.

canvas
An area within a dashboard or workspace that users interact with to create, view, and manipulate content and data.

capability
A group of functions and features that...
can be hidden or revealed to simplify the user interface. Capabilities can be enabled or disabled by changing preference settings, or they can be controlled through an administration interface.

cardinality
1. For relational data sources, a numerical indication of the relationship between two query subjects, query items, or other model objects.
2. For OLAP data sources, the number of members in a hierarchy. The cardinality property for a hierarchy is used to assign solve orders to expressions.

cascading prompt
A prompt that uses values from a previous prompt to filter the values in the current prompt or pick list.

certificate
In computer security, a digital document that binds a public key to the identity of the certificate owner, thereby enabling the certificate owner to be authenticated. A certificate is issued by a certificate authority and is digitally signed by that authority. See also certificate authority.

certificate authority (CA)
A component that issues certificates to each computer on which components are installed.

CGI  See Common Gateway Interface

cipher suite
The combination of authentication, key exchange algorithm, and the Secure Sockets Layer (SSL) cipher specification used for the secure exchange of data.

class style
A combination of formatting characteristics, such as font, font size, and border, that the user names and stores as a set.

CM  See Content Manager

Common Gateway Interface (CGI)
An Internet standard for defining scripts that pass information from a web server to an application program, through an HTTP request, and vice versa.

condition
An expression that can be evaluated as true, false, or unknown. It can be expressed in natural language text, in mathematically formal notation, or in a machine-readable language.

constraint
1. A security specification that denies one or more users the ability to access a model component or to perform a modeling or authoring task.
2. A restriction on the possible values that users can enter in a field.

contact
A named email address to which reports and agent e-mails can be sent. Contacts are never authenticated.

content locale
A code that is used to set the language or dialect used for browsers and report text, and the regional preferences, such as formats for time, date, money, money expressions, and time of day.

Content Manager (CM)
The service that retrieves information from the content store, and saves information to the content store.

content store
The database that contains the data needed to operate, such as report specifications, published models, and security rights.

credential
A set of information that grants a user or process certain access rights.

cube
A multidimensional representation of data needed for online analytical processing, multidimensional reporting, or multidimensional planning applications.

custom set
In Analysis Studio, a named object which can include filter rules, calculations, and sort rules. Custom sets can define a set of members that is different from any set originally defined in the cube model. See also predefined set.
dashboard
A web page that can contain one or more widgets that graphically represent business data.

data source
The source of data itself, such as a database or XML file, and the connection information necessary for accessing the data.

data source connection
The named information that defines the type of data source, its physical location, and any sign-on requirements. A data source can have more than one connection.

data tree
Within a studio, a structure that contains objects such as query subjects, query items, dimensions, levels, and members. A data tree is used as a palette of the available data that can be inserted into calculations, filters, display areas, and other authoring gestures.

deployment
The process of moving an application (such as a report or model) to a different instance. For example, reports are often created in a test environment and then deployed to production. When an application is deployed, it is exported, transferred, and imported.

deployment archive
A file used for deployment. A deployment archive contains the data from the content store that is being moved.

deployment specification
A definition of what packages to move (deploy) between source and target environments, the deployment preferences, and the archive name. Deployment specifications are used for import and export.

derived index
A calculated metric that provides a status and a score based on other metrics.

details-based set
A set based on an item and its immediate details. See also set.

dimension
A broad grouping of descriptive data about a major aspect of a business, such as products, dates, or locations. Each dimension includes different levels of members in one or more hierarchies and an optional set of calculated members or special categories.

dimensional data source
A data source containing data modeled using OLAP concepts, including dimensions, hierarchies, and measures.

drill down
In a multidimensional representation of data, to access information by starting with a general category and moving downwards through the hierarchy of information, for example from Years to Quarters to Months.

event
A change to a state, such as the completion or failure of an operation, business process, or human task, that can trigger a subsequent action, such as persisting the event data to a data repository or invoking another business process.

event key
A combination of data items that uniquely defines an event instance. Identifying an event instance enables the agent to determine if it is new, ongoing or stopped.

event list
The set of detected event instances evaluated by the task execution rules to determine which agent tasks should be performed.

F

fact
See measure.

G

gateway
An extension of a web server program that transfers information from the web server to another server. Gateways are
often CGI programs, but may follow other standards such as ISAPI and Apache modules.

glyph  The actual shape (bit pattern, outline) of a character image. For example, italic A and roman A are two different glyphs representing the same underlying character. Strictly speaking, any two images which differ in shape constitute different glyphs. In this usage, glyph is a synonym for character image, or simply image (The Unicode Standard – Version 1.0).

group  A collection of users who can share access authorities for protected resources.

**grouping**  In reporting, the process of organizing common values of query items together and only displaying the value once.

H

**hierarchy**  The organization of a set of entities into a tree structure, with each entity (except the root) having one or more parent entities and an arbitrary number of child entities.

I

**information card**  A display of high-level information about dashboard, workspace, or report content, such as owner, contact information, date modified, and an optional thumbnail view of the dashboard, workspace, or report.

**information pane**  In Analysis Studio, a pane that helps the user to confirm their selection in the data tree by displaying related information, such as the level and attributes.

**initiative**  A task developed to achieve objectives or close the gap between performance and targets. Initiatives are associated with individual objectives and often known as projects, actions, or activities.

item  See **member**

J

**job**  A group of runnable objects, such as reports, agents, and other jobs that the user runs and schedules as a batch.

**job step**  The smallest part of a job that can be run separately. A job step can be a report or it can be another job.

L

**layout**  The arrangement of printed matter on a screen or page, including margins, line spacing, type specification, header and footer information, indents, and more.

**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 region, state, and city.

**locale**  A setting that identifies language or geography and determines formatting conventions such as collation, case conversion, character classification, the language of messages, date and time representation, and numeric representation.

M

**MDX**  See Multidimensional Expression Language.

**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.

**member**  A unique item within a hierarchy. For example, Camping Equipment and 4 Man tent are members of the Products hierarchy.

**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. See also **package**.

**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.

**model**
A physical or business representation of the structure of the data from one or more data sources. A model describes data objects, structure, and grouping, as well as relationships and security. In Cognos BI, a model is created and maintained in Framework Manager. The model or a subset of the model must be published to the Cognos server as a package for users to create and run reports.

**multidimensional data source**
See **dimensional data source**.

**Multidimensional Expression Language (MDX)**
The multidimensional equivalent of Structured Query Language (SQL).

**N**

**named set**
See **predefined set**.

**namespace**
A part of the model in which the names may be defined and used. Within a namespace, each name has a unique meaning.

**news item**
A single entry in a Really Simple Syndication (RSS) compatible format. It can include a headline, text, and a link to more information. A news item task in an agent can be used to create news items for display in a Cognos Connection portlet.

**O**

**object**
In Report Studio, an empty information container that can be dragged to a report from the Toolbox tab and then filled with data. Reports are made up of objects, which include crosstabs, text items, calculations, graphics, and tables.

**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.

**P**

**package**
A subset of a model, which can be the whole model, to be made available to the Cognos server. See also **metric package**.

**page set**
In Report Studio, a set of one or more designed pages which repeat in the report output for each instance of a chosen query item. See also **set**.

**passport**
Session-based information, stored and encrypted in Content Manager memory, regarding authenticated users. A passport is created the first time a user accesses Cognos 8, and it is retained until a session ends, either when the user logs off or after a specified period of inactivity.

**portlet**
A reusable component that is part of a web application that provides specific information or services to be presented in the context of a portal.

**predefined set**
A set of members defined inside an OLAP data source as a list or by an expression. Predefined sets can be used in analysis and report authoring. See also **custom set**.

**product locale**
The code or setting that specifies which language, regional settings, or both to use for parts of the product interface, such as menu commands.
project

1. In Metric Studio, a task or set of tasks undertaken by a team and monitored on a scorecard. A project tracks dates, resources, and status.
2. 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.

prompt

A report element that asks for parameter values before the report is run.

properties pane

Within a studio, a pane that provides an overview of the properties for selected data. The properties pane can also be used to make several changes and apply them at the same time, instead of repeating several different commands.

publish

In Cognos BI, to expose all or part of a Framework Manager model or Transformer PowerCube, through a package, to the Cognos server, so that the data can be used to create reports and other content.

Q

query

The simple report specifications created and edited by Query Studio.

query item

A representation of a column of data in a data source. Query items may appear in a model or in a report and contain a reference to a database column, a reference to another query item, or a calculation.

query subject

A named collection of query items that are closely functionally related. Query subjects are defined using Framework Manager to represent relational data and form the set of available data for authoring reports in Query Studio and Report Studio. A query subject is similar to a relational view in that it can be treated as a table but does not necessarily reflect the data storage.

R

Really Simple Syndication (RSS)

An XML file format for syndicated web content that is based on the Really Simple Syndication specification (RSS 2.0). The RSS XML file formats are used by Internet users to subscribe to websites that have provided RSS feeds. See also Rich Site Summary.

repeater

In Report Studio, a cell container that repeats values within itself with no predefined internal structure.

repeater table

In Report Studio, a table-like container that repeats cells across and down the page or row in the associated query.

report

A set of data deliberately laid out to communicate business information.

report output

The output produced as a result of executing a report specification against a data set.

report specification

An executable definition of a report, including query and layout rules, which can be combined with data to produce a report output.

report view

A reference to another report that has its own properties, such as prompt values, schedules, and results. Report views can be used to share a report specification instead of making copies of it.

response file

An ASCII file that can be customized with the setup and configuration data that automates an installation. During an interactive installation, the setup and configuration data must be entered, but with a response file, the installation can proceed without any intervention.

Rich Site Summary (RSS)

An XML-based format for syndicated web content that is based on the RSS 0.91 specification. The RSS XML file formats are used by Internet users to subscribe to websites that have provided RSS feeds. See also Really Simple Syndication.

RSS
1. See [Really Simple Syndication](#).
2. See [Rich Site Summary](#).

**S**

**score** A number or ranking that expresses applicability in relation to a standard.

**scorecard** A collection of metrics representing the performance of one unit or aspect of an organization.

**scorecard structure** The hierarchy of scorecards that reflects how an enterprise organizes its metrics.

**security provider** See [authentication provider](#).

**selection-based set** A collection of individual items that the user has explicitly selected. The items or members may be selected from one or more levels of the same hierarchy. See also [set](#).

**session** The time during which an authenticated user is logged on.

**set** A collection of related items or members. Members in a set may be specifically chosen, or selected by one or more filter rules. See also [custom set](#), [details-based set](#), [page set](#), [predefined set](#), [selection-based set](#), [stacked set](#).

**stacked set** Two or more sets arranged one above another in rows or side-by-side in columns. See also [set](#).

**strategy** The overall plan of action (such as for a brand unit, business unit, channel, or company) to achieve a stated goal. Strategies normally cover a period of more than one year.

**strategy map** In Metric Studio, a visual representation of the strategy and the objectives of that strategy for an organization. For example, a strategy map may show employees how their jobs are aligned to the overall objectives of the organization.

**summary** In reporting and analysis, an aggregate value that is calculated for all the values of a particular level or dimension. Examples of summaries include total, minimum, maximum, average, and count.

**T**

**task** An action performed by an agent if the event status meets the task execution rules. For example, an agent can send an email, publish a news item, or run a report.

**task execution rule** A user-specified option within an agent that determines which statuses and values cause a task to be run. It determines which tasks to execute for each event instance.

**template** In report authoring, a reusable report layout or style that can be used to set the presentation of a query or report.

**thumbnail** An icon-sized rendering of a larger graphic image that permits a user to preview the image without opening a view or graphical editor.

**tuple** An ordered collection of two or more members from different dimensions. For example, the tuple (2007, Camping Equipment, Japan) returns the value for the intersection of the three members: 2007, Camping Equipment, and Japan. Tuples can be used to filter and sort data, and to create calculations.

**U**

**union set** See [stacked set](#).

**user** Any individual, organization, process, device, program, protocol, or system that uses the services of a computing system.

**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.
watch list
A list of metrics that each user has chosen to monitor closely. If notification is enabled in Metric Studio, the user will receive email notification of changes to these metrics. Users can also choose to display their watch list as a portlet within Cognos Connection.

watch rule
A user-defined condition that determines whether a report is delivered to the user. When the rule is run, the output is evaluated and, if it satisfies the condition or rule, the report is delivered by email or news item. Watch rules limit report delivery to those reports containing data of significance to the user.

Web Services for Remote Portlets
A standard for creating presentation-oriented web services so that they can be easily integrated within other applications, such as web portals.

widget
A portable, reusable application or piece of dynamic content that can be placed into a web page, receive input, and communicate with an application or with another widget.

work area
The area within a studio that contains the report, analysis, query, or agent currently being used.

workspace
See dashboard
Index

Special characters
  .cpf 1
  .iqd 22
  .xml 1
  .xsd 1

C
  categories
    filtering 30
  CCLAssert error message 61
  cpf files 1

D
  data loading 35
  double-byte characters 35

E
  errors
    troubleshooting 61
  executing extracts 35
  expression editor
    Aster Data 93
    Aster Data Data Type Formatting 94
    Aster Data Math 95
    Aster Data String 93
    Aster Data Trigonometry 95
    Block Functions 91
    Business Date/Time Functions 86
    Common Functions 213
    Constants 84
    Constructs 85
    Date Functions 167
    DB2 98
    DB2 Math 98
    DB2 Trigonometry 98
    Dimensional Functions 221
    Greenplum 115
    Greenplum Data type formatting 117
    Greenplum Math 117
    Greenplum String 115
    Greenplum Trigonometry 118
    Informix 120
    Informix Math 120
    Informix Trigonometry 121
    Macro Functions 200
    Member Summaries 81
    MS Access 126
    MS Access Cast 126
    MS Access Math 127
    MS Access Trigonometry 127
    MySQL 133
    MySQL Math 135
    MySQL String 133
    MySQL Trigonometry 135
    Netezza 137
    Netezza Fuzzy 139

expression editor (continued)
  Netezza Math 137
  Netezza Phonetic 139
  Netezza Trigonometry 138
  Operators 63
  Oracle 144
  Oracle Math 144
  Oracle Trigonometry 144
  Paracel 152
  Paracel Data type formatting 153
  Paracel Math 154
  Paracel String 152
  Postgres 155
  Postgres Data type formatting 156
  Postgres Math 157
  Postgres String 155
  Postgres Trigonometry 158
  Red Brick 160
  Report Functions 246
  Salesforce.com 167
  SAP BW 165
  SAP BW Math 167
  SAP BW Trigonometry 165
  searching for values 63
  SQL Server 171
  SQL Server Math 171
  SQL Server Trigonometry 171
  Summaries 70
  Sybase 178
  Sybase Math 178
  Sybase Trigonometry 179
  Teradata 186
  Teradata Trigonometry 186
  Vectorwise 191
  Vectorwise Math 192
  Vectorwise String 191
  Vectorwise Trigonometry 193
  Vertica 194
  Vertica Data type formatting 196
  Vertica Math 196
  Vertica String 194
  Vertica Trigonometry 197

extracts
  changing 37
  creating 7
  executing 35
  metrics from multiple sources 31
  objects 32
  publishing 35
  running 35
  validating 7

F
  files
    .cpf 1
    .iqd 22
    .xml 1
    .xsd 1
  filtering categories for metrics 30