IBM Cognos Analytics
Version 11.0

Data Modeling Guide

IBM

©

Product Information

This document applies to IBM Cognos Analytics version 11.0.0 and may also apply to subsequent releases.

Copyright

Contents

Chapter 1. Data modeling in Cognos Analytics

You can use data modeling in IBM® Cognos® Analytics to access and shape data from data servers or uploaded files. You can create a data module by fusing together many sources of data that includes relational databases, Hadoop-based technologies, Microsoft Excel spreadsheets, and text files. Star schemas are the ideal database structure for data modules but transactional schemas are equally supported.

You can refine a data module in many ways.

- Creating calculations.
- Defining data filters.
- Referencing additional tables.
- Reviewing and updating metadata properties that include aggregation settings and labels.

You can save data modules so that you and other users can access them. When you want other users to access a data module, save the data module in a folder that users, groups, and roles have appropriate permissions to access. This procedure is the same idea as saving a report or dashboard into a folder that controls who can access it.

Data modules can be used in both dashboards and reports. A dashboard can be assembled from multiple data modules.

Data modeling in Cognos Analytics does not replace IBM Cognos Framework Manager, IBM Cognos Cube Designer, or IBM Cognos Transformer, which remain available for more complex modeling.

Intent-driven modeling

You can use intent-driven modeling to add tables into your data module. Intent-driven modeling proposes tables to include in the module, based on matches between the terms you supply and metadata in the underlying sources.

While you are typing in keywords for intent-driven modeling, text from column and table names in the underlying data sources are retrieved by the Cognos Analytics software. The intent field has a type-ahead list that suggests terms that are found in the source metadata.

Intent-driven modeling recognizes the difference between fact tables and dimension tables by the number of rows, data types, and distribution of values within columns. When possible, the intent-driven modeling proposal is a star or snowflake of tables. If an appropriate star or snowflake cannot be determined, then intent-driven modeling proposes a single table or collection of tables.

Chapter 2. Creating a data module

You can create data modules by combining inputs from up to four types of input sources: other data modules, data servers, uploaded files, and data sets.

When you create a new data module from the home screen of IBM Cognos Analytics, you are presented with five possible input sources in **Sources**. These sources are described here.

Data modules
: Data modules are source objects that contain data from data servers, uploaded files, or other data modules, and are saved in **My content** or **Team content**.

Data servers
: Data servers are databases for which connections exist. For more information, see *Managing IBM Cognos Analytics* .

Uploaded files
: Uploaded files are data that are stored with the **Upload files** facility.

Data sets
: Data sets contain extracted data from a package or a data module, and are saved in **My content** or **Team content**.

Packages
: Packages are created in IBM Cognos Framework Manager and contain dimensions, query subjects, and other data contained in Cognos Framework Manager projects. You can use packages as sources for a data module.

You can combine multiple sources into one data module. After you add a source, click **Add sources** () in **Selected sources** to add another source. You can use a combination of data source types in a data module.

Each type of data source is described in the following topics.

Using a data module source

11.0.5 11.0.4

Saved data modules can be used as data sources for other data modules. When a data module is used as a source for another data module, parts of that module are copied into the new data module.

Procedure

1. When you select **Data modules** in the **Sources** slide-out panel, you are presented with a list of data modules to use as input. Check one or more data modules to use as sources.
2. Click **Start** or **Done** in **Selected sources** to expand the data module into its component tables.
3. Drag tables into the new data module.
4. Continue to work with your new data module, and then save it.

5. If the source data module or any of its tables are deleted, then the next time that you open the new data module, tables that are no longer available have a red outline in the diagram and **missing** is listed in the **Source** fields of the **Properties** pane of the table.
6. A table in your new data module that is linked is read-only. You cannot modify it in the new data module in any way. You can break the link to the source data module, and modify the table, by clicking **Break link** in the actions for the table.

Using a data server source

Data servers are databases for which connections exist and can be used as source for data modules.

Before you begin

Data server connections are created in **Manage** > **Data servers**. For more information, see *Managing IBM Cognos Analytics* .

Procedure

1. When you select **Data modules** in the **Sources** slide-out panel, you are presented with a list of data servers to use as input. Check one or more data servers to use as sources.
2. The available schemas in the data server are listed. Choose the schema that you want to use. Only schemas for which metadata is preloaded are displayed. If you want to use other schemas, click **Manage schemas...** to load metadata for other schemas.
3. Click **Start** or **Done** in **Selected sources** to expand the data module into its component tables.
4. To start populating your data module, type some terms into the **Intent** slide-out panel and then click **Go**.
5. You are presented with a proposed model. Click **Add this proposal** to create a data module.
6. You can also drag tables from the data server schema into the data module.

Example

For an example data module created from a data server, see Chapter 3, “Creating a simple data module,” on page 7

What to do next

If the metadata of your data server schemas changes after you create the data module, you can refresh the schema metadata. For more information, see the topic on preloading metadata from a data source connection in the *IBM Cognos Analytics Managing User Guide*.

Using an uploaded file source

Uploaded files are data that is stored with the **Upload files** facility. You can use these files as sources for a data module.

Before you begin

Supported formats for uploaded files are Microsoft Excel (.xlsx and .xls) spreadsheets, and text files that contain either comma-separated, tab-separated, semi colon-separated, or pipe-separated values. Only the first sheet in Microsoft Excel spreadsheets is uploaded. If you want to upload the data from multiple sheets in a spreadsheet, save the sheets as separate spreadsheets. Uploaded files are stored in a columnar format.

To upload a file, click **Upload files** on the navigation bar in the IBM Cognos Analytics home screen.

Procedure

1. When you select **Uploaded files** in the **Sources** slide-out panel, you are presented with a list of uploaded files to use as input. Check one or more uploaded files to use as sources.
2. Click **Start** or **Done** in **Selected sources** to expand the data module into its component tables.
3. Drag the source uploaded file into your data module to start mdeling.

Using a data set source

11.0.4

Data sets contain data that is extracted from a package or a data module, and are saved in **My content** or **Team content**.

About this task

Procedure

1. When you select **Data sets** in the **Sources** slide-out panel, you are presented with a list of data sets to use as input. Check one or more data sets to use as sources.
2. Click **Start** or **Done** in **Selected sources** to expand the data set into its component tables and queries.
3. Drag tables or queries into the new data module.
4. If the data in the data sets changes, this change is reflected in your data module.

Using a package source

11.0.5

Packages are created in IBM Cognos Framework Manager and contain query subject and other data contained in Cognos Framework Manager projects. You can use relational packages as sources for a data module.

Before you begin

The package must be relational and use Dynamic Query Mode.

Procedure

1. When you select **Packages** in the **Sources** slide-out panel, you are presented with a list of packages to use as input. Check one or more packages to use as sources.
2. Click **Start** or **Done** in **Selected sources** to select the packages.
3. Drag the source packages into your data module to start modeling.

What to do next

When you use a package as your data source, you cannot select individual tables. You must drag the entire package into your data module. The only actions you can take are to create relationships between query subjects in the package and query subjects in the data module.

Chapter 3. Creating a simple data module

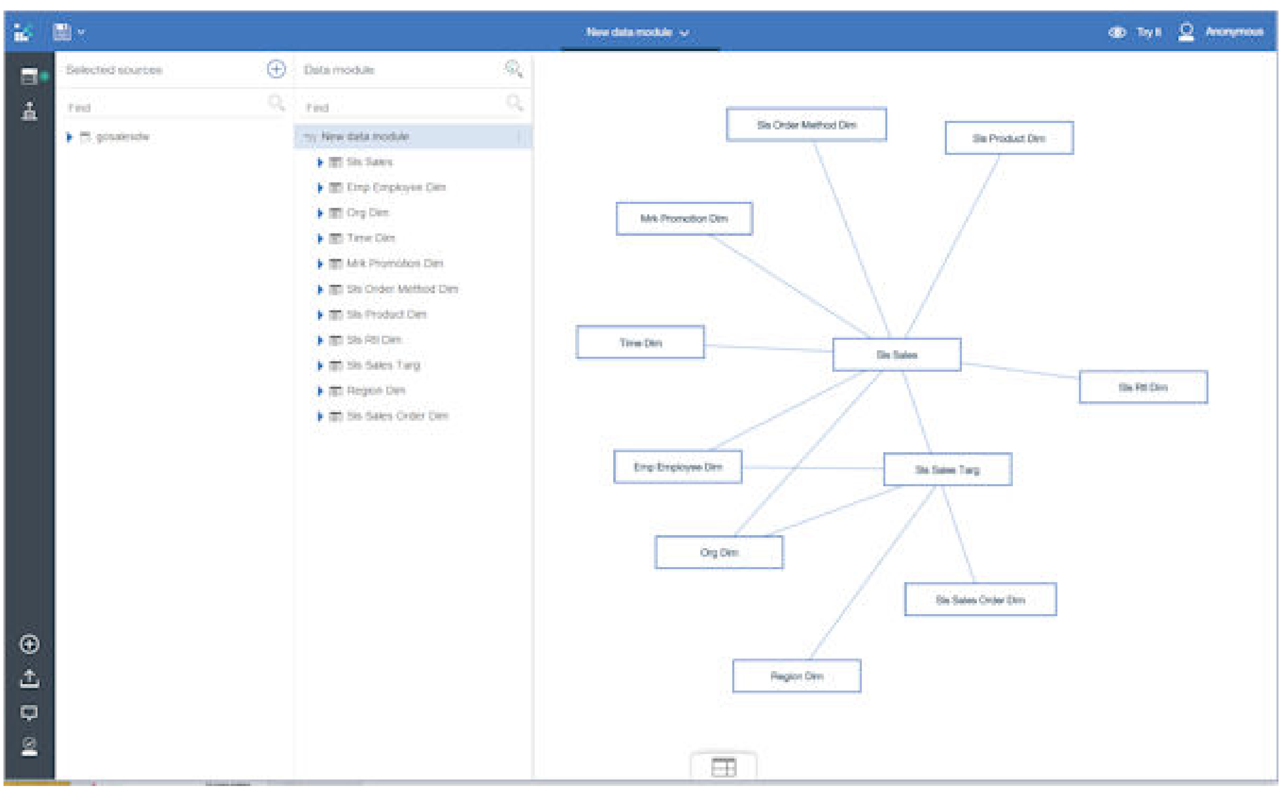
You can create a simple data module based on the Great Outdoors Warehouse sales data database that is included in IBM Cognos Analytics samples.

Before you begin

Install the Great Outdoors sales data warehouse database and create a connection to the database. For more information, see *Samples for IBM Cognos Analytics*.

Procedure

1. In the IBM Cognos Analytics welcome screen, click **New** → **Data module**.
2. In **Sources**, select **Data servers**.
3. In **Data servers**, select **great_outdoors_warehouse**.
4. In **great_outdoors_warehouse**, check **GOSALESDW/gosalesdw**.
5. In **Selected sources**, click **Start**.
6. In **Intent**, type `sales revenue` and then click **Go**. A proposed model is displayed in **Intent**
7. In **Intent**, click **Add this proposal** A base module is created and displayed.



8. You can now explore the model. For example, click an item in **Data module**, and then click to view and modify the properties of the item.
9. To save the model, click **Save** ( ).

10. To create a report from your data module, click **Try it**. A new tab opens in your browser with IBM Cognos Analytics - Reporting open within it. Your data module is shown in **Source Data items**.
11. Drag **Product Line Code** from **Sls Product Dim** and **Quantity** from **Sls Sales Fact** into the report.
12. Click **Run options** () to select an output format, and then click **Run HTML** to run the report and view the output as a web page.

Chapter 4. Refining a data module

The initial data module that you create manually or using intent modeling might contain data that is not required for your reporting purposes. Your goal is to create a data module that contains only the data that meets your reporting requirements and that is properly formatted and presented.

For example, you can delete some tables from your initial data module, or add different tables. You can also apply different data formatting, filter and group the data, and change the metadata properties.

You can refine your data module by applying the following modifications:
- Add or delete tables.
- Edit or create new relationships between tables.
- Change column properties.
- Create basic and custom calculations.
- Define filters.
- Group data.
- Clean the text data.
- Hide tables and columns.

You can initiate these actions from the **Data module** panel or from the diagram.

When working in a data module, you can use the undo and redo actions in the application bar to revert or restore changes to the data module in the current editing session. You can undo or redo up to 20 times.

The following graphic shows the user interface for viewing and editing data modules.

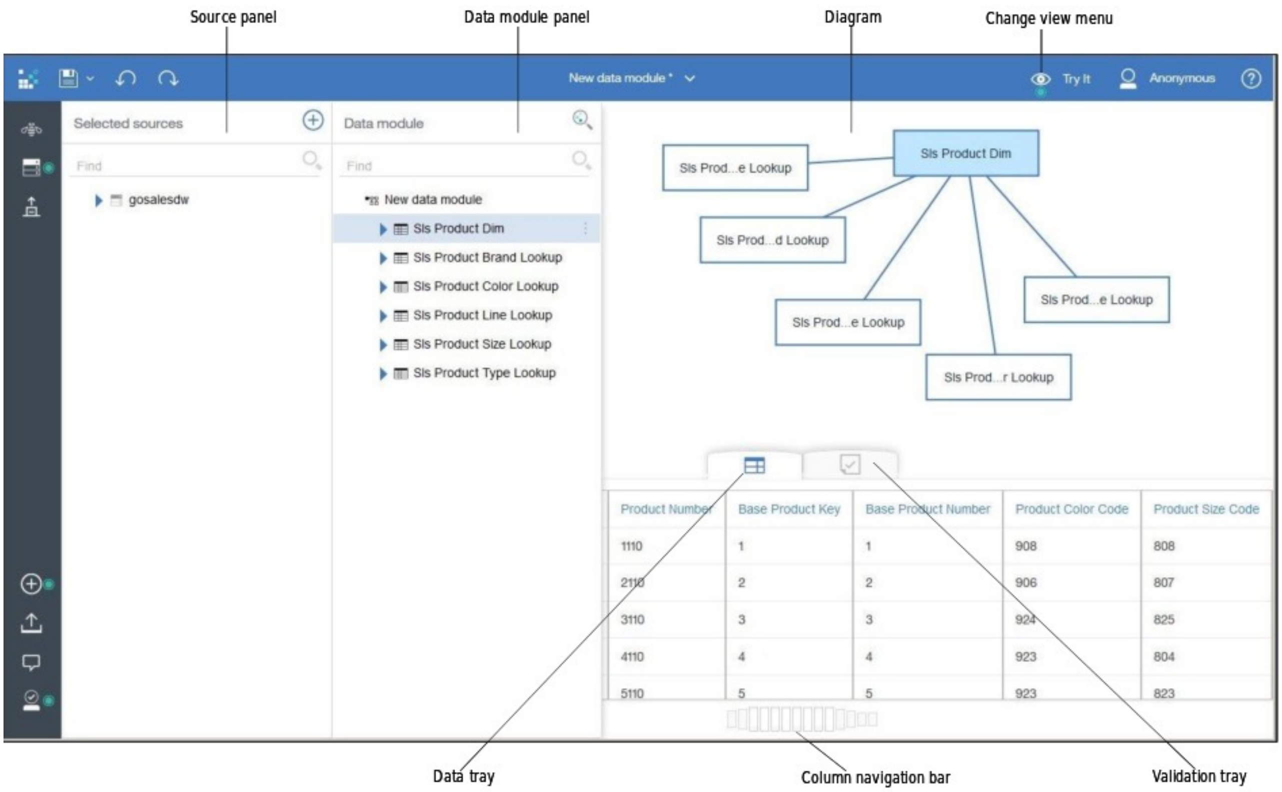


Figure 1. User interface for viewing and editing data modules

Source panel

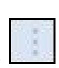
The source panel shows the sources of data that were selected when the data module was created. The types of sources include data servers, uploaded files, and other data modules.

Expand the specific source to view its tables and columns.

Drag tables onto the data module panel or onto the diagram to add them to the data module.

Data module panel

The data module tree shows the tables and columns of data that is included in the data module. This is the primary space for editing the data module.

Click the context menu icon  for the module, table, or column to view its modeling and editing context menu options. Here you can start joining tables, creating filters and calculations, or renaming and deleting items.

Diagram

The diagram is a graphical representation of table relationships in a data module. You can use the diagram to examine the relationships, edit the data module, and view the cardinality information for the relationships.

Right-click a table in the diagram to view the table context menu that can be your starting point for creating joins or filters, renaming the table, viewing the table properties, or removing it from the module.

Click any table join to see the join summary information that includes the matching keys. When you right-click the join line, the context menu appears with options for editing or deleting the join.

Click the change view icon in the application toolbar. In the box that appears, select the **Cardinality** check box. The diagram now shows the cardinality of relationships between different tables in your data module. Move the **Degrees of separation** slider. Depending on the slider position, the diagram shows different distances of relationships between tables.

Data tray

You can use the data tray to view the actual data in table columns and rows.

Select a table in the data module tree or in the diagram, and click the data tray icon to open the data tray.

To navigate between rows of data, drag the data tray icon up and down. To navigate between the columns of data, use the column navigation bar at the bottom of the data tray. To close the data tray, click its icon again.

Validation tray

11.0.4

You can use the validation tray to view errors that are identified by the validation process.

The messages are displayed after you start the **Validate** operation anywhere in the modeling user interface, and the failed validation icon is displayed for tables, columns, expressions, or joins where errors are discovered.

To open or close the validation tray, click its icon . To expand the message viewing area, drag the tray icon up.

Relationships

A relationship joins logically related objects that the users want to combine in a single query. Relationships exist between two tables.

You can modify or delete relationships, or create new ones so that the data module properly represents the logical structure of your business. Verify that the relationships that you require exist in the data module, the cardinality is set correctly, and referential integrity is enforced.

The diagram provides a graphical view of table relationships in a data module. You can use the diagram to examine the relationships, create and edit the joins, or view the join cardinality.

Creating a relationship from scratch

You need to create relationships whenever the required relationships are not detected by the IBM Cognos software.

About this task

Relationships can be created between tables from the same source and from different sources.

The diagram is the most convenient place to view all data module relationships, and quickly discover the disconnected tables.

Tip: The list of possible keys in the relationship editor excludes measures. This means that if a row in a column was misidentified as a measure, but you want to use it as an identifier, you will not see the row in the key drop-down list. You need to examine the data module to confirm that the usage property is correct on each column in the table.

Procedure

1. In the data module tree or in the diagram, click the table for which you want to create a relationship, and from the context menu, click **Join**.

   Tip: You can also start creating a relationship using the following methods:
   - In the data module tree or in the diagram, control-click the two tables that you want to join in a relationship, and click **Join**.
   - On the **Relationships** tab in the table properties, click **Create new**.

   If the data module does not include the table that you need, you can drag this table from **Selected sources** directly onto the diagram.
2. In the relationship editor, specify the second table to include in the relationship, and then select the matching columns in both tables.

   Depending on the method that you used to start the relationship, the second table might already be added, and you only need to match the columns. You can include more than one set of matching rows in both tables.
3. Specify the **Join type** for the relationship by selecting one of the available options.
4. Specify the **Cardinality** of the relationship by selecting one of the available options.
5. Specify the **Filter join**. If you apply this filter, the performance of a join can be improved by filtering one side of the join with the values that are retrieved by the other side.
6. Click **OK**.

Results

The new relationship appears on the **Relationships** tab in the properties page of the tables that you joined, and in the diagram view.

To view or edit all relationships defined for a table, go to the **Relationships** tab in the table properties. To edit a relationship, click its expression, and make the modifications. To view a relationship from the diagram, click the join line to open a small graphical view of the relationship. To edit a relationship from the diagram, right-click the join line, and click **Edit**.

To delete a relationship for a table, go to the **Relationships** tab in the table properties, and click **Edit**. Then, click the remove icon for the required relationship, and click **Done**. To delete the relationship from the diagram, right-click the line joining the two tables, and click **Remove**.

Calculations

Calculations allow you to answer questions that cannot be answered by the source columns.

The following product functionality that you can use to refine your data modules is based on underlying calculations:

- Basic arithmetic calculations and field concatenations.
- Custom groups.
- Cleaning text data.
- Custom calculations.

Creating basic calculations

You can create basic arithmetic calculations for columns with numeric data types, and concatenate text values for columns with the text data type.

About this task

The expression for these calculations is predefined and you only need to select it. For example, you can create a column `Revenue` by multiplying values for `Quantity` and `Unit price`. You can create a column `Name` by combining two columns: `First name` and `Last name`.

Procedure

1. To create a simple arithmetic calculation for columns with numeric data types, use the following steps:
   a. In the data module tree, right-click the column for which you want to create a calculation. For calculations that are based on two columns, use control-click to select the columns.
   b. In the **Create a calculation** box, type a name for the calculation.
   c. If the calculation is based on one column, type the number to use in the calculation.
   d. Click **Create**.
2. To create a calculation that concatenates values for columns with the text data type, use the following steps:
   a. In the data module tree, control-click the two columns that you want to combine into a single column. Depending on which column you select first, its value appears at the beginning of the combined string.
   b. Click **Create a calculation**, and select the suggested option.
   c. Type a name for the calculation.
   d. Click **Create**.

Results

In the table that you added the calculation to, you can now see a new column with the new calculation at the end of the list of columns. An expression for the calculation is automatically created in the expression editor. To view the expression, go to the column properties page and click on the expression that is shown for the **Expression** property.

Grouping data

You can organize the column data into custom groups so that the data is easier to read and analyze.

About this task

You can create two types of custom groups depending on the data type of the column: one group type for columns with numeric data and the second group type for columns with text data. For example, in the `Employee code` column you can group employees into ranges, such as 0-100, 101-200, 200+. In the `Manager` column, you can group managers according to their rank, such as `First line manager`, `Senior manager`, and so on.

Procedure

1. In the data module tree, right-click the column that you want to group on, and click **Custom groups**.
2. If you selected a numeric column, specify the grouping in the following way:
   a. Specify how many groups you want to create.
   b. Specify the distribution of the values to be either **Equal distribution** or **Custom**.
   c. If you chose **Equal distribution**, specify the values to be contained in each group by typing the numbers or clicking the scroll bars.
   d. If you chose **Custom**, you can enter your own range values for the group.
   e. Optional: Change the group name.
   f. Click **Create**.
3. If you selected a text column, specify the grouping in the following way:
   a. Control-select the values to include in the first group.
   b. In the **Groups** column, click the plus sign.
   c. Specify the name for the group, and click **OK**. The values are added in the **Group members** column, and the name of the group appears in the **Groups** column. You can add additional values to a group after it is created, and you can remove values from a group. You can also remove a group.
   d. Optional: To add another group, repeat the steps for the first group.
   e. Optional: To create a group that contains all of the values that aren't already included in a group, select the **Group remaining and future values in** check box, and specify a name for the group.
   f. Click **Create**.

Results

The custom group column that is based on your selections appears at the end of the list of columns in the table. A group expression is automatically created in the expression editor. To view the expression, go to the column properties page and click on the expression that is shown for the **Expression** property.

Tip: To complete the action of creating the custom group, you can click **Replace** instead of **Create**. This option will replace the column name in the table with the group name.

Cleaning data

Data is often messy and inconsistent. You might want to impose some formatting order on your data so that it's clearer and easier to read.

About this task

The **Clean** options that are available for a column depend on the column data type. Some options can be specified for multiple columns with the same data type, and some for singular columns only.

The following options are available to clean your data:

Whitespace

Trim leading and trailing whitespace

Select this check box to remove leading and trailing whitespace from strings.

Convert case to

UPPERCASE, **lowercase**, **Do not change**

Use this option to change the case of all characters in the string to either uppercase or lowercase, or to ensure that the case of each individual character is unchanged.

Return a substring of characters

Return a string that includes only part of the original string in each value. For example, an employee code can be stored as CA096670, but you need only the number 096670 so you use this option to remove the CA part. You can specify this option for singular columns only.

For the **Start** value, type a number that represents the position of a character in the string that will start the substring. Number 1 represents the first character in the string. For the **Length** value, specify the number of characters that will be included in the substring.

NULL values

11.0.4

Specify NULL-handling options for columns with text, numeric, date, and time data types that allow NULL values. When Cognos Analytics detects that a column does not allow NULL values, these options are not available for that column.
The default value for each option depends on the column data type. For text data, the default is an empty string. For numbers, the default is 0. For dates, the default is 2000-01-01. For time, the default is 12:00:00. For date and time (timestamp), the default is 2000-01-01T12:00:00.
The entry field for each option also depends on the column data type. For text, the entry field accepts alphanumeric characters, for numbers, the entry field accepts only numeric input. For dates, a date picker is provided to select the date, and for time, a time picker is provided to select the time.
The following NULL-handling options are available:

Replace this value with NULL
Replaces the text, numbers, date, and time values, as you specify in the entry field, with NULL.
For example, if you want to use an empty string instead of NULL in a given column, but your uploaded file sometimes uses the string `n/a` to indicate that the value is unknown, you can replace `n/a` with NULL, and then choose to replace NULL with the empty string.

Replace NULL values with
Replaces NULL values with text, numbers, date, and time values, as you specify in the entry field.
For example, for the Middle Name column, you can specify the following values to be used for cells where middle name does not exist: `n/a`, `none`, or the default empty string. For the Discount Amount column, you can specify `0.00` for cells where the amount is unknown.

Procedure

1. In the data module tree, click the context menu icon  for a column, and click **Clean**.

   Tip: To clean data in multiple columns at once, control-select the columns that you want to clean. The **Clean** option is available only if the data type of each selected column is the same.
2. Specify the options that are applicable for the selected column or columns.
3. Click **Clean**.

Results

After you complete the **Clean** operation, the expression editor automatically creates an expression for the modified column or columns. To view the expression, open the column properties panel, and click the expression that is shown for the **Expression** property.

Creating custom calculations

To create a custom calculation, you must define your own expression using the expression editor.

About this task

Custom calculations can be created at the data module level or at the table level. The module-level calculations can reference columns from multiple tables.

For information about the functions that you can use to define your expressions, see Appendix A, “Using the expression editor,” on page 23.

Procedure

1. In the data module tree, right-click the data module name or a specific table name, and click **Create custom calculation**.
2. In the **Expression editor** panel, define the expression for your calculation, and specify a name for it.
   - To enter a function for your expression, type the first character of the function name, and select the function from the drop-down list of suggested functions.
   - To add table columns to your expression, drag-and-drop one or more columns from the data module tree to the expression editor panel. The column name is added where you place the cursor in the expression editor.

     Tip: You can also double-click the column in the data module tree, and the column name appears in the expression editor.
3. Click **Validate** to check if the expression is valid.

4. After successful validation, click **OK**.

Results

If you created your calculation at the data module level, the calculation is added after the last table in the data module tree. If you created your calculation at the table level, the calculation is added at the end of the list of columns in the table. To view the expression for the calculation, open the calculation properties panel, and click on the expression that is shown for the **Expression** property.

Creating navigation groups

11.0.5

A navigation group is a collection of non-measure columns that business users might associate for data exploration.

When a data module contains navigation groups, the dashboard users can drill down and back to change the focus of their analysis by moving between levels of information. The users can drill down from column to column in the navigation group by either following the order of columns in the navigation group, or by choosing the column to which they want to proceed.

About this task

You can create a navigation group with columns that are logically related, such as year, month, quarter, week. You can also create a navigation group with columns that are not logically related, such as product, customer, state, city.

Columns from different tables can be added to a navigation group. The same column can be added to multiple navigation groups.

Procedure

1. In the data module panel, start creating a navigation group by using one of the following methods:
   - From the data module context menu, click **Properties**, and then click the **Navigation groups** tab. Click **Add a navigation group**, and drag columns from the data module panel to the **Create navigation group** panel.
   - In the data module tree, select one or more columns, and from the context menu of any of the selected columns, click **Create navigation group**.

   The selected columns are listed in the **Create navigation group** panel. If there are multiple columns, the name of the navigation group includes names of the first and last column in the group.
2. If needed, modify the navigation group in the following way:
   - To add different columns, drag the columns from the data module panel to the navigation group panel. You can multi-select columns and drag them all at once.
   - To remove columns, click the remove icon for the column.
   - To change the order of columns, drag them up or down.
   - To change the navigation group name, overwrite the default name.

The default name reacts to the changed order of columns. If you override the default name, it does no longer change when you modify the group definition. The name cannot be blank.

3. Click **Apply**, and save the data module.

   If you select the option **Identify navigation group members** in the data module toolbar, the columns that are members of navigation groups are underlined.

Results

The navigation group is added to the data module and will be available to users in dashboards and stories.

What to do next

The modeler can modify navigation groups at any time, and re-save the data module.

To view the navigation group or groups that a column belongs to, from the column context menu, click **Properties** > **Navigation groups**. Click the navigation group name to view or modify its definition.

To view all navigation groups in a data module, from the data module context menu, click **Properties** > **Navigation groups**. Click the navigation group names to view or modify their definitions. To delete a navigation group, click **Remove**, click the remove icon for the group, and click **Done**.

Filtering data

A filter specifies the conditions that rows must meet to be retrieved from a table.

About this task

The filter is based on a specific column in a table, but it affects the whole table. Also, only rows that meet the filter criteria are retrieved from other tables.

You can create filters at the table level, which allows you to add multiple filters at once, or at the column level.

Procedure

1. In the data module tree or in the diagram, locate the table for which you want to create filters.

   - To create a filter at the table level, click the table context menu icon, and then click **Filter**.

     Tip: You can also right-click the table in the diagram, and click **Filter** from there.

     On the **Filters** tab in the table properties pane, click **Add a filter**, select a column, and click **Enter a value**.
   - To create a filter at the column level, expand the table in the data module panel, and from the column context menu, click **Filter**.

2. Select the filter values in the following way:
   a. If the column data type is integer, you have two options to specify the values: **Range** and **Individual values**. When you choose **Range**, use the slider to specify the value ranges. When you choose **Individual values**, select the check boxes associated with the values.
   b. For columns with numeric data types other than integer, use the slider to specify the range values.
   c. For columns with date and time (timestamp) data types, specify a range of values before, after, or between the selected date and time, or select individual values.
   d. For columns with text data types, select the check boxes associated with the values.
3. Optional: To select values that are outside the range that you specified, click **Invert**.
4. Specify a meaningful label for the filter.
5. Click **Apply**.

Results

After you create a filter, the filter icon is added for the table and column in the data module panel and in the diagram.

What to do next

To view, edit, or remove the filters defined for a table, go to the **Filters** tab in the table properties. To edit the filter, click its expression, make the modifications, and click **Done**.

Tip: To edit a filter on a single column, from the column context menu in the data module panel, click **Filter** to open the filter definition.

To remove any filter from a table, go to the **Filters** tab in the table properties, and click **Edit**. Then, click the remove icon that appears before the filter name, and click **Done**.

Hiding tables and columns

11.0.4

You can hide a table or column in a data module. The hidden tables or columns remain visible in the modeling interface, but they are not visible in the reporting and dashboarding interfaces. The hidden items are fully functional in the product.

About this task

Use this feature to provide an uncluttered view of metadata for the report and dashboard users. For example, when you hide columns that are referenced in a calculation, the metadata tree in the reporting and dashboarding interfaces shows only the calculation column, but not the referenced columns. When you hide the identifier columns used as keys for joins, the keys are not exposed in the dashboarding and reporting interfaces, but the joins are functional in all interfaces.

Procedure

1. In the data module tree, click the context menu icon for a table or column, and click **Hide**.

   You can also select multiple tables or columns to hide them at once.

   Tip: To un-hide the items, click the context menu icon for the hidden table or column, and click **Show**.
2. Save the data module.

Results

The labels on the hidden tables and columns are grayed out in the data module tree and in the diagram. Also, on the **General** tab of the table or column properties, the check box **This item is hidden from users** is selected.

The hidden tables and columns are not visible in the reporting and dashboarding interfaces.

Validating data modules

11.0.4

Use the validation feature to check for invalid object references within a data module.

About this task

Validation identifies the following errors:

- A table or column that a data module is based on no longer exists in the source.
- A calculation expression is invalid.
- A filter references a column that no longer exists in the data module.
- A table or column that is referenced in a join no longer exists in the data module.

Errors in the data module are identified by the failed validation icon . The descriptions of errors are shown in the validation tray when it is open .

Procedure

1. In the data module tree, click the data module context menu icon , and click **Validate**

   If errors are identified, the failed validation icon is displayed in the data module tree, in the diagram, and in the properties panel, next to the column or expression where the error exists. The descriptions of errors are displayed in the validation tray.

   Tip: To open or close the validation tray, click its icon .

2. Click the failed validation icon for a module, column, expression, or join to view a pop-up box that informs you of the number of errors for the selected item. Double-click the failed validation icon in the pop-up box to view the error details.

Results

Using the validation messages, try to resolve the errors. You can save a data module with validation errors in it.

Appendix A. 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 that are contained within a data item. A filter is an expression that you use to retrieve a specific subset of records.

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
```

/

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
```

>=

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
```

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

Revenue	Between
$332.06	false
$230.55	true
$107.94	false

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
```

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.

Syntax

```
if ( condition ) then .... else ( expression ) , or case .... else (
 expression ) end
```

end

Indicates the end of a case or when construct.

Syntax

```
case .... end
```

ends with

Determines if "string1" ends with "string2".

Syntax

```
string1 ends with string2
```

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.

Syntax

```
if ( condition ) then ( expression ) else ( expression )
```

in

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

Syntax

```
expression1 in ( expression_list )
```

is missing

Determines if "value" is undefined in the data.

Syntax

```
value is missing
```

like

Determines if "string1" matches the pattern of "string2", with the character "char" optionally used to escape characters in the pattern string.

Syntax

```
string1 LIKE string2 [ ESCAPE char ]
```

Example 1

```
[PRODUCT_LINE] like 'G%'
```

Result

All product lines that start with 'G'.

Example 2

```
[PRODUCT_LINE] like '%Ga%' escape 'a'
```

Result

All the product lines that end with 'G%'.

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
```

starts with

Determines if "string1" starts with "string2".

Syntax

```
string1 starts with string2
```

then

Works with the if or case constructs. When the if condition or the when expression are true, the then expression is used.

Syntax

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

when

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

Syntax

```
case [expression] when ... end
```

Summaries

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

Statistical functions

This list contains predefined summary functions of statistical nature.

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.

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.

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. All is supported in DQM mode only and it avoids the presumption of double counting a data item of a dimension table.

Syntax

```
count ( [ all | distinct ] expression [ auto ] )
count ( [ all | distinct ] expression for [ all|any ] expression { ,
 expression } )
count ( [ all | 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.

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.

Syntax

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

Example

```
percentage ( Sales 98 )
```

Result

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

Result data

Employee	Sales 98	Percentage
Gibbons	60646	7.11%
Flertjan	62523	7.35%
Cornel	22396	2.63%

percentile

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

Syntax

```
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

Qty	Percentile (Qty)
800	1
700	0.875
600	0.75
500	0.625
400	0.5
400	0.5
200	0.25
200	0.25

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

Qty	Rank	Quantile (Qty, 4)
800	1	1
700	2	1
600	3	2
500	4	2
400	5	3

Qty	Rank	Quantile (Qty, 4)
400	5	3
200	7	4
200	7	4

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

Qty	Quartile (Qty)
450	1
400	1
350	2
300	2
250	3
200	3
150	4
100	4

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.

Syntax

```
rank ( expression [ ASC|DESC ] { , expression [ ASC|DESC ] } [ at
expression { , expression } ] [ <for-option> ] [ prefilter ] )
rank ( [ 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

Employee	Sales 98	Rank
Gibbons	60000	1
Flertjan	50000	2
Cornel	50000	2
Smith	48000	4

tertile

Returns the rank of a value as High, Middle, or Low relative to a group of values.

Syntax

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

Example

```
tertile ( Qty )
```

Result

Returns the quantity, the quantile rank value of the quantity as broken down into tertiles, and the quantile rank label as broken down into tertiles.

Result data

Qty	Quantile (Qty, 3)	Tertile (Qty)
800	1	H
700	1	H
500	2	M
400	2	M
200	3	L
200	3	L

total

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

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.

Business Date/Time Functions

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

_add_seconds

Returns the time or datetime, depending on the format of "time_expression", that results from adding "integer_expression" seconds to "time_expression".

Syntax

```
_add_seconds ( time_expression, integer_expression )
```

Example 1

```
_add_seconds ( 13:04:59 , 1 )
```

Result

13:05:00

Example 2

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

Result

2002-04-30 12:10:11.000

Example 3

```
_add_seconds ( 2002-04-30 00:00:00.000, 1/100 )
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 00:00:00.010

_add_minutes

Returns the time or datetime, depending on the format of "time_expression", that results from adding "integer_expression" minutes to "time_expression".

Syntax

```
_add_minutes ( time_expression, integer_expression )
```

Example 1

```
_add_minutes ( 13:59:00 , 1 )
```

Result

14:00:00

Example 2

```
_add_minutes ( 2002-04-30 12:59:10.000, 1 )
```

Result

2002-04-30 13:00:10.000

Example 3

```
_add_minutes ( 2002-04-30 00:00:00.000, 1/60 )
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 00:00:01.000

_add_hours

Returns the time or datetime, depending on the format of "time_expression", that results from adding "integer_expression" hours to "time_expression".

Syntax

```
_add_hours ( time_expression, integer_expression )
```

Example 1

```
_add_hours ( 13:59:00 , 1 )
```

Result

14:59:00

Example 2

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

Result

2002-04-30 13:10:10.000,

Example 3

```
_add_hours ( 2002-04-30 00:00:00.000, 1/60 )
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 00:01:00.000

_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

Adds "integer_expression" months to "date_expression". If the resulting month has fewer days than the day of month component, then the last day of the resulting month is returned. In all other cases the returned value has the same day of month component as "date_expression".

Syntax

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

Example 1

```
_add_months ( 2012-04-15 , 3 )
```

Result

2012-07-15

Example 2

```
_add_months ( 2012-02-29 , 1 )
```

Result

2012-03-29

Example 3

```
_last_of_month ( _add_months ( 2012-02-29 , 1 ) )
```

Result

2012-03-31

Example 4

```
_add_months ( 2012-01-31 , 1 )
```

Result

2012-02-29

Example 5

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

Result

2002-05-30 12:10:10.000

_add_years

Adds "integer_expression" years to "date_expression". If the "date_expression" is February 29 and resulting year is non leap year, then the resulting day is set to February 28. In all other cases the returned value has the same day and month as "date_expression".

Syntax

```
_add_years ( date_expression, integer_expression )
```

Example 1

```
_add_years ( 2012-04-15 , 1 )
```

Result

2013-04-15

Example 2

```
_add_years ( 2012-02-29 , 1 )
```

Result

2013-02-28

Example 3

```
_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.

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 IBM Cognos Analytics server.

Syntax

```
current_time
```

Example

```
current_time
```

Result

16:33:11.354+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 server.

Syntax

```
current_timestamp
```

Example

```
current_timestamp
```

Result

2003-03-03 16:40:15.535+05:00

_day_of_week

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

Syntax

```
_day_of_week ( date_expression, integer )
```

Example

```
_day_of_week ( 2003-01-01 , 1 )
```

Result

3

_day_of_year

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

Syntax

```
_day_of_year ( date_expression )
```

Example

```
_day_of_year ( 2003-03-01 )
```

Result

61

_days_between

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

Syntax

```
_days_between ( date_expression1 , date_expression2 )
```

Example

```
_days_between ( 2002-04-30 , 2002-06-21 )
```

Result

-52

_days_to_end_of_month

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

Syntax

```
_days_to_end_of_month ( date_expression )
```

Example

```
_days_to_end_of_month ( 2002-04-20 14:30:22.123 )
```

Result

10

_end_of_day

Returns the end of today as a timestamp.

Syntax

```
_end_of_day
```

Example

```
_end_of_day
```

Result

```
2014-11-23 23:59:59
```

_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

_from_unixtime

Returns the unix time specified by an integer expression as a timestamp with time zone.

Syntax

```
_from_unixtime ( integer_expression )
```

Example

```
_from_unixtime ( 1417807335 )
```

Result

```
2014-12-05 19:22:15+00:00
```

_hour

Returns the value of the hour field in a date expression.

Syntax

```
_hour( date_expression )
```

Example

```
_hour (  2002-01-31 12:10:10.254 )
```

Result

```
12
```

_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

_minute

Returns the value of the minute field in a date expression.

Syntax

```
_minute( date_expression )
```

Example

```
_minute (  2002-01-31 12:10:10.254 )
```

Result

```
10
```

_month

Returns the value of the month field in a date expression.

Syntax

```
_month( date_expression )
```

Example

```
_month ( 2003-03-01 )
```

Result

```
3
```

_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

_second

Returns the value of the second field in a date expression.

Syntax

```
_second( date_expression )
```

Example

```
_second (  2002-01-31 12:10:10.254 )
```

Result

```
10.254
```

_shift_timezone

Shifts a timestamp value from one time zone to another time zone. This function honors the Daylight Savings Time when applicable. If the first argument is of type "timestamp", then the second and third arguments represent the "from" and "target" time zones, respectively. If the first argument is of type "timestamp with time zone", then the "from" time zone is already implied in the first argument therefore the second argument represents the "target" time zone. The data type of the first argument will also determine the data type of the return value. The second and third arguments are of type "string" and represent time zone identifiers. A list of these identifiers can be found below. Note: using this function will cause local processing.

Syntax

```
_shift_timezone ( timestamp_value , from_time_zone ,
 target_time_zone )
_shift_timezone ( timestamp_with_time_zone_value , target_time_zone )
```

Example 1

```
_shift_timezone( 2013-06-30 12:00:00 , 'EST' , 'GMT' )
```

Result

2013-06-30 16:00:00

Example 2

```
_shift_timezone( 2013-11-30 12:00:00-05:00 , 'PST' )
```

Result

2013-11-30 09:00:00-08:00

Example 3

```
Time zone abbreviations:
```

Result data

```
GMT (GMT+00:00) Greenwich Mean Time
UTC (GMT+00:00) Coordinated Universal Time
WET (GMT+00:00) Western Europe Time: Lisbon, Faeroe Islands, Canary
Islands
ECT (GMT+01:00) European Central Time: Amsterdam, Brussels, Paris,
Rome, Vienna
MET (GMT+01:00) Middle European Time
ART (GMT+02:00) Egypt Time: Cairo, Damascus, Beirut, Amman, Nicosia
CAT (GMT+02:00) Central African Time: Johannesburg, Blantyre, Harare,
Tripoli
EET (GMT+02:00) Eastern Europe Time: Athens, Kiev, Sofia, Minsk,
Bucharest, Vilnius, Tallinn
EAT (GMT+03:00) East Africa Time: Addis Ababa, Asmera, Kampala,
Nairobi, Mogadishu, Khartoum
NET (GMT+04:00) Near East Time
PLT (GMT+05:00) Pakistan Lahore Time
IST (GMT+05:30) Indian Time
BST (GMT+06:00) Bangladesh Time
VST (GMT+07:00) Vietnam Time
CTT (GMT+08:00) Asia, Hong Kong S.A.R. of China
JST (GMT+09:00) Japan Time: Tokyo
ACT (GMT+09:30) Australian Central Time: Darwin
AET (GMT+10:00) Australian Eastern Time: Sydney, Melbourne, Canberra
SST (GMT+11:00) Solomon Time
AGT (GMT-03:00) Argentina Time
BET (GMT-03:00) Brazil Eastern Time: Sao Paulo, Buenos Aires
CNT (GMT-03:30) Newfoundland Time: St. Johns
PRT (GMT-04:00) Puerto Rico and U.S. Virgin Islands Time
EST (GMT-05:00) Eastern Time: Ottawa, New York, Toronto, Montreal,
```

```
Jamaica, Porto Acre
CST (GMT-06:00) Central Time: Chicago, Cambridge Bay, Mexico City
MST (GMT-07:00) Mountain Time: Edmonton, Yellowknife, Chihuahua
PST (GMT-08:00) Pacific Time: Los Angeles, Tijuana, Vancouver
AST (GMT-09:00) Alaska Time: Anchorage, Juneau, Nome, Yakutat
HST (GMT-10:00) Hawaii Time: Honolulu, Tahiti
MIT (GMT-11:00) Midway Islands Time: Midway, Apia, Niue, Pago Pago
```

Example 4

```
A customized time zone identifier may also be used, using the format
GMT(+|-)HH:MM. For example, GMT-06:30 or GMT+02:00.
A more complete
list of time zone idenfitiers (including longer form identifiers such
as "Europe/Amsterdam") may be found in the "i18n_res.xml" file from
the product's configuration folder.
```

_start_of_day

Returns the start of today as a timestamp.

Syntax

```
 _start_of_day
```

Example

```
 _start_of_day
```

Result

```
2014-11-23 00:00:00
```

_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

_timezone_hour

Returns the value of the timezone hour field in a date expression.

Syntax

```
_timezone_hour( date_expression )
```

Example

```
_timezone_hour (  2002-01-31 12:10:10.254-05:30 )
```

Result

```
5
```

_timezone_minute

Returns the value of the timezone minute field in a date expression.

Syntax

```
_timezone_minute( date_expression )
```

Example

```
_timezone_minute (  2002-01-31 12:10:10.254-05:30 )
```

Result

```
30
```

_unix_timestamp

Returns the unix time specified by an integer expression as a timestamp with time zone.

Syntax

```
_unix_timestamp
```

Example

```
_unix_timestamp
```

Result

```
1416718800
```

_year

Returns the value of the year field in a date expression.

Syntax

```
_year( date_expression )
```

Example

```
_year ( 2003-03-01 )
```

Result

```
2003
```

_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.

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 "datatype_specification": character, varchar, char, numeric, decimal, integer, bigint, smallint, real, float, date, time, timestamp, time with time zone, timestamp with time zone, and interval. When type casting to an interval type, one of the following interval qualifiers must be specified: year, month, or year to month for the year-to-month interval datatype; day, hour, minute, second, day to hour, day to minute, day to second, hour to minute, hour to second, or minute to second for the day-to-second interval datatype. Notes: When you convert a value of type timestamp to type date, the time portion of the timestamp value is ignored. When you convert a value of type timestamp to type time, the date portion of the timestamp is ignored. When you convert a value of type date to type timestamp, the time components of the timestamp are set to zero. When you convert a value of type time to type timestamp, the date component is set to the current system date. It is invalid to convert one interval datatype to the other (for instance because the number of days in a month is variable). Note that you can specify the number of digits for the leading qualifier only, i.e. YEAR(4) TO MONTH, DAY(5). Errors will be reported if the target type and size are not compatible with the source type and size.

Syntax

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

Example 1

```
cast ( '123' , integer )
```

Result

123

Example 2

```
cast ( 12345 , varchar ( 10 ) )
```

Result

a string containing 12345

ceiling

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

Syntax

```
ceiling ( numeric_expression )
```

Example 1

```
ceiling ( 4.22 )
```

Result

5

Example 2

```
ceiling ( -1.23 )
```

Result

-1

char_length

Returns the number of logical characters in "string_expression". The number of logical characters can be distinct from the number of bytes in some East Asian locales.

Syntax

```
char_length ( string_expression )
```

Example

```
char_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.

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

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

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 )
```

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

position_regex

Returns the integer value representing the beginning or ending position of the substring in "string_expression" that matches the regular expression "regex_expression". The search starts at position "integer_expression1", which has a default value of 1. The occurrence of the pattern to search for is specified by "integer_expression2", which has a default value of 1. The return option, specified by the first argument, specifies what is returned in relation to the occurrence. If you specify "start", the position of the first character of the occurrence is returned. If you specify "after", the position of the character following the occurrence is returned. If you don't specify a return option, "start" is implicit. Flags to set options for the interpretation of the regular expression are specified by "flags_expression". Individual letters are used to define the flags, with valid values being 's', 'm', 'i', and 'x'.

Syntax

```
position_regex ([ start|after ]  regex_expression , string_expression
[ , integer_expression1 [ , integer_expression2 [ , flags_expression ]]] )
```

Example 1

```
position_regex ( '.er' , 'Flicker Lantern' )
```

Result

5

Example 2

```
position_regex ( after '.er' , 'Flicker Lantern' )
```

Result

8

Example 3

```
position_regex ( '.er' , 'Flicker Lantern' , 1 , 2 )
```

Result

12

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" decimal places. 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

substring_regex

Returns a substring of "string_expression" that matches the regular expression "regex_expression". The search starts at position "integer_expression1", which has a default value of 1. The occurrence of the pattern to search for is specified by "integer_expression2", which has a default value of 1. Flags to set options for the interpretation of the regular expression are specified by "flags_expression". Individual letters are used to define the flags, with valid values being 's', 'm', 'i', and 'x'.

Syntax

```
substring_regex ( regex_expression , string_expression [ , integer_expression1
[ , integer_expression [ , flags_expression ]]] )
```

Example 1

```
substring_regex ( '.er' , 'Flicker Lantern')
```

Result

ker

Example 2

```
substring_regex ( '.er' , 'Flicker Lantern' , 1 , 2 )
```

Result

ter

trim

Returns "string_expression" trimmed of leading and trailing blanks or trimmed of a certain character specified in "match_character_expression". "Both" is implicit when the first argument is not stated and blank is implicit when the second argument is not stated.

Syntax

```
trim ( [ [ trailing|leading|both ] [ match_character_expression ] , ]
string_expression )
```

Example 1

```
trim ( trailing 'A' , 'ABCDEFA' )
```

Result

ABCDEF

Example 2

```
trim ( both , ' ABCDEF ' )
```

Result

ABCDEF

upper

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

Syntax

```
upper ( string_expression )
```

Example

```
upper ( 'abcdef' )
```

Result

ABCDEF

Trigonometric functions

arccos

Returns the arc cosine of the argument, where the argument is in the range of -1 to 1 and the result is a value expressed in radians.

Syntax

```
arccos ( numeric_expression )
```

Example

```
arccos ( -1 )
```

Result

3.1415

arcsin

Returns the arc sine of the argument, where the argument is in the range of -1 to 1 and the result is a value expressed in radians.

Syntax

```
arcsin ( numeric_expression )
```

Example

```
arcsin ( 0 )
```

Result

3.1415

arctan

Returns the arc tangent of the argument, where the argument is in the range of -1 to 1 and the result is a value expressed in radians.

Syntax

```
arctan ( numeric_expression )
```

Example

```
arctan ( 0 )
```

Result

3.1415

cos

Returns the cosine of the argument, where the argument is expressed in radians.

Syntax

```
cos ( numeric_expression )
```

Example

```
cos ( 0.3333 * 3.1415 )
```

Result

0.5

coshyp

Returns the hyperbolic cosine of the argument, where the argument is expressed in radians.

Syntax

```
coshyp ( numeric_expression )
```

Example

```
coshyp ( 0 )
```

Result

1

sin

Returns the sine of the argument, where the argument is expressed in radians.

Syntax

```
sin ( numeric_expression )
```

Example

```
sin  ( 0.1667 * 3.1415 )
```

Result

0.5

sinhyp

Returns the hyperbolic sine of the argument, where the argument is expressed in radians.

Syntax

```
sinhyp ( numeric_expression )
```

Example

```
sinhyp ( 0 )
```

Result

0

tan

Returns the tangent of the argument, where the argument is expressed in radians.

Syntax

```
tan ( numeric_expression )
```

Example

```
tan ( 0.25 * 3.1415 )
```

Result

1

tanhyp

Returns the hyperbolic tangent of the argument, where the argument is expressed in radians.

Syntax

```
tanhyp ( numeric_expression )
```

Example

```
tanhyp ( 0 )
```

Result

0

Appendix B. About this guide

This document is intended for use with IBM Cognos Analytics. Cognos Analytics integrates reporting, modeling, analysis, dashboards, metrics, and event management so you can understand your organization's data, and make effective business decisions.

To find product documentation on the web, including all translated documentation, access IBM Knowledge Center (http://www.ibm.com/support/knowledgecenter).

Accessibility features

Accessibility features help users who have a physical disability, such as restricted mobility or limited vision, to use information technology products successfully. For information on accessibility features in Cognos Analytics, see the *Cognos Analytics Accessibility Guide*.

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.

Samples disclaimer

The Sample Outdoors Company, Great Outdoors Company, GO Sales, any variation of the Sample Outdoors or Great Outdoors names, 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.

Notices

This information was developed for products and services offered worldwide.

This material may be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

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.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, 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 is for planning purposes only. The information herein is subject to change before the products described become available.

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.

Depending upon the configurations deployed, this Software Offering may use session and persistent cookies that collect each user's

- name
- user name
- password

for purposes of

- session management
- authentication
- enhanced user usability
- single sign-on configuration
- usage tracking or functional purposes other than session management, authentication, enhanced user usability and single sign-on configuration

These cookies cannot be disabled.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at http://www.ibm.com/privacy and IBM's Online Privacy Statement at http://www.ibm.com/privacy/details in the section entitled "Cookies, Web Beacons and Other Technologies" and the "IBM Software Products and Software-as-a-Service Privacy Statement" at http://www.ibm.com/software/info/product-privacy.

Trademarks

IBM, the IBM logo and ibm.com 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.

Index

