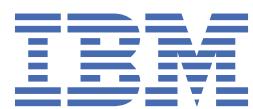


IBM Cognos Analytics
Version 12.0.x

Data Modules User Guide



©

Product Information

This document applies to IBM Cognos Analytics version 12.0.0 and may also apply to subsequent releases.

Copyright

Licensed Materials - Property of IBM

© Copyright IBM Corp. 2015, 2024.

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

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.

© Copyright International Business Machines Corporation 2015, 2020.

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

Contents

Chapter 1. Data modeling in Cognos Analytics.....	1
Modeling user interface	2
Customizing the user interface.....	4
Creating a dashboard within a data module.....	5
Chapter 2. Data modules and their sources.....	7
Data module sources.....	7
Using data servers as data module sources.....	7
Using packages as data module sources.....	8
Using uploaded files as data module sources.....	8
Using data sets as data module sources.....	8
Using other data modules as data module sources.....	9
Creating a data module.....	10
Creating data modules from OLAP cubes.....	12
Discovering related tables.....	13
Adding sources or tables to a data module.....	14
Updating columns in a data module.....	15
Reloading the schema metadata.....	16
Relinking sources.....	16
Defining source overrides.....	18
Enriching packages.....	20
Setting up data caching.....	22
Securing data.....	24
Chapter 3. Modeling metadata	27
Relationships.....	27
Creating a relationship manually.....	28
Generating auto-relationships.....	30
Join operators.....	30
Join optimization.....	31
Custom tables.....	31
Creating custom tables.....	33
Editing the view custom tables.....	35
Editing the union, except, and intersect custom tables.....	36
Creating tables using SQL	38
Converting referenced tables.....	39
Relinking referenced tables.....	40
Column dependencies.....	41
Defining column dependencies.....	44
Configuring column dependencies.....	47
Calculations.....	49
Creating basic calculations.....	49
Creating custom calculations.....	50
Filters.....	51
Creating embedded filters.....	52
Creating selectable filters.....	53
Hiding items	53
Creating data groups.....	54
Cleaning data.....	57
Creating navigation paths.....	58

Formatting data.....	60
SQL in Cognos Analytics.....	61
Supported SQL types	61
Showing the query information.....	62
Generating the query SQL	63
Validating data modules.....	63
Object properties.....	64
Refreshing properties.....	67
Chapter 4. Members in the data tree.....	69
Searching for members.....	71
Setting the members display limits.....	71
Displaying relational members.....	72
Automatic sorting in month, weekday, and season columns.....	73
Defining custom sort order.....	74
Chapter 5. Relative date analysis.....	77
Sample calendars	77
Creating a custom retail calendar	79
Creating a data module for relative date analysis.....	81
Creating relative date filters	83
Creating filter expressions.....	84
Expression variables	86
Example filter: Last 12 months.....	87
Example filter: Next 4 months.....	89
Other examples of relative date filters	90
Customizing the reference date	92
Setting the _as_of_date global parameter.....	93
Setting a dynamic _as_of_date parameter.....	93
Query optimization for relative date measures.....	94
Chapter 6. Multilingual metadata.....	99
Enabling support for multilingual metadata in a data module.....	99
Exporting metadata for translation.....	99
Translation sources.....	101
Creating the translation definition.....	102
Using the sample files to create a multilingual data module.....	104
Appendix A. Supported SQL data types.....	109
Appendix B. Using the expression editor.....	111
Creating expressions.....	112
Expression editor documentation.....	113
Appendix C. Data modules and Framework Manager.....	115
Framework Manager features not supported by data modules.....	115
Governors and data modules.....	117

Chapter 1. Data modeling in Cognos Analytics

IBM® Cognos® Analytics provides web-based, self-service data modeling capabilities.

You can use data modeling in Cognos Analytics to fuse together many sources of data, including relational databases, Hadoop-based technologies, Microsoft Excel spreadsheets, text files, and so on. Using these sources, a data module  is created that can then be used in reports, dashboards, or explorations.

Star schemas are the ideal database structure for data modules, but transactional schemas are equally supported.

You can enhance a data module by creating calculations, defining filters and navigation paths, and more.

After you save a data module, other users can access it. Save the data module in a folder that users, groups, and roles have appropriate permissions to access. This process is the same as saving a report or dashboard.

Tip: Data modeling in Cognos Analytics does not replace IBM Cognos Framework Manager, IBM Cognos Cube Designer, or IBM Cognos Transformer, which remain available for maintaining upgraded projects.

Intent modeling

You can use intent modeling to create a data module. Intent modeling proposes tables to include in the module, based on matches between the terms that you supply and metadata in the underlying sources.

Intent 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 modeling proposal is a star or snowflake of tables. If an appropriate star or snowflake cannot be determined, a single table or a collection of tables is proposed.

For more information, see [“Discovering related tables” on page 13](#).

Automatic joins

Cognos Analytics automatically creates joins between tables in a data module. The autojoin (automatic join) algorithm adopts a diagnostic scoring approach when deciding which columns to use to join two tables. The algorithm uses a set of rules that are applied when choosing the column combinations between the two tables. Each rule produces a score. The score could be negative. The total score of all rules decides if a column combination qualifies to be a join column.

The autojoin algorithm uses the following rules:

- The similarity of two column names must exceed a minimum threshold.

For example, the names SalesCountryCode and CountryCode are highly similar, and can be considered a match.

- Both columns belong to the same semantic category.

For example, the Employee or Product category.

- Both columns have the same semantic attribute.

For example, both are IDs.

- None of the columns is a common row identifier.

The row ID column could be in every table.

- The data in two numeric columns overlaps.

- The relationship between two columns can't be many-to-many.

A join relationship is created if any column combinations between two tables satisfy a minimum qualification score. The collected statistics is used to ensure that cardinality is properly set when building

the relationship. The joins created by the autojoin algorithm are saved as the inferred relationships in the data module.

For more information, see “[Relationships](#)” on page 27.

Modeling user interface

Use the web modeling user interface to view, create, enhance, and edit data modules.

Access to this interface is controlled by the **Web-based modeling** capability that is managed by administrators. For more information, see the *Managing IBM Cognos Analytics* guide.

If some of the user interface elements that are discussed in this topic are not available to you, the user interface could be customized for your role. For more information, see “[Customizing the user interface](#)” on page 4.

You can enter the web modeling user interface from the IBM Cognos Analytics portal in one of the following ways:

- In **Team content**, **My content**, or **Recent**, locate an existing data module, which is an object with this icon , and click it to open.
- Click **New** , and select **Data module**. Then, create a new data module.
- Use the **Quick launch** facility in the Cognos Analytics welcome page to upload a file. Drop the file in the **Data module** box, and start creating your data module.

When working with data modules, 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.

Data module panel

The data module panel contains the data tree that shows the tables, columns, calculations, folders, and other types of items. This is the main 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.

Click the **Add sources and tables** icon  in the panel toolbar to add sources and tables to your data module. Clicking the **Identify navigation path members** icon  underlines columns that are members of navigation paths. If none of the columns are underlined, the data module does not contain navigation paths.

The list of items in the data tree can be long. To find the data faster, use the search box at the top of the data tree. Or sort the items by label or data type, in descending or ascending order. The **Sort** option is available from the context menu.

Grid tab

Use the grid view to examine the actual data in table columns and rows.

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

Review and explore the columns that are available for you. The data fields that are in the columns are not sorted. To sort this data, click a column heading and select the sort type from the menu.

Relationships tab

This tab shows the data module relationships diagram  . The diagram is a graphical representation of table relationships in a data module. You can use the diagram view 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.

Right-click one or more tables in the diagram, and click **Auto-arrange**. The diagram is redrawn around the first selected table allowing you to focus on the selected tables and their relationships.

In the **Diagram settings** box, select the **Cardinality** check box to show 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 degrees of relationships between tables. Select one or more tables in the diagram, and use the **Focus mode** to work with the selected tables.

Custom tables tab

The **Custom tables** tab  is the main space for creating, viewing, and managing custom tables in a data module. This tab is displayed by default, even if the module doesn't contain any custom tables. To start creating a new table, click **Create custom table**. If the data module already contains custom tables, the table names are listed when you click the tab.

For more information, see [“Custom tables” on page 31](#).

Sources panel

The **Sources** panel  shows the sources of data that the data module contains. The sources can be data servers, uploaded files, data sets, packages, and other data modules.

Except for packages, you can 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.

From the source context menu , you can initiate actions such as relinking sources or enabling data caching.

Languages panel

The **Languages** panel  is used to view and define metadata translation definitions.

For more information, see [Chapter 6, “Multilingual metadata,” on page 99](#).

Custom sort order panel

The **Custom sort order** panel  is used to view and define custom sorting definitions.

When custom sort order is defined for a column, in the **Data module** panel, the column values are sorted in the custom order. However, in the **Grid** view, the column values are sorted according to the source sort order.

For more information, see [“Defining custom sort order” on page 74](#).

Validation panel

To validate the data module, click the validation icon in the toolbar , or click **Validate** from the data module context menu .

If errors are discovered, the failed validation icon  is displayed for tables, columns, expressions, or joins. Click the error icons to view the validation messages. Click the copy icon  in the error messages to copy the messages to the clipboard for easier analysis or printing.

Expression editor

The expression editor is an SQL editing tool that you can use to create or edit SQL-based tables, calculations, filters, or data groups.

You can create expressions by typing the code or dragging items from the data module tree. The validation and data preview capabilities help to quickly verify and troubleshoot the expressions. The code editing capabilities include: inserting comments, auto-complete function, pretty-print, high-contrast mode, and different font sizes. The information panel shows details and provides examples of supported functions that are used in the expressions.

Customizing the user interface

Users with administrative privileges can customize the modeling user interface by disabling some parts of the interface for some user roles.

Users who are members of those roles can't perform certain tasks when they view or edit data modules. For example, users might not be able to apply data security for data server sources or edit joins when the related user interface features are disabled for them.

About this task

This functionality is available for **Cognos** roles only. For more information, "Customizing roles" in the *IBM Cognos Analytics Managing Guide*.

Procedure

1. In the **Manage** administration interface, click **People > Accounts**.
2. Click the **Cognos** namespace, and locate the role for which you want to customize the user interface.

The role **Modelers** is associated with data modules, but you can also apply the customizations to other roles in the **Cognos** namespace.

3. From the role context menu , click **Properties**.
4. On the **Customization** tab, click the **Features** chevron button .
5. Select the **Data Module** category.

Most of the data module features that you might want to customize are grouped under **Application Bar, Navigation Bar, and Context Menus**.

For example, go to **Data Modules > Context Menus > Shaping options** to customize the following features:

- **Set data security** - ability to apply data security for data servers
- **Create relationship** - ability to create relationships between tables
- **Edit relationship** - ability to edit relationship joins
- **Create basic calculation** and **Edit basic calculation** - ability to create and edit calculations without using the expression editor.

- **Create custom calculation** and **Edit custom calculation** - ability to create and edit calculations by using the expression editor.
 - **Navigation path** - ability to create navigation paths
6. To disable a feature, clear its checkbox.
 7. Click **Apply** to save the changes.

Results

All access points to the disabled feature are not available for the affected roles. For example, when you disable the **Edit relationship** feature, the join editing options are not available in the table and column context menus, the diagram and custom table views, and the properties panel.

Creating a dashboard within a data module

You can start creating a dashboard in the data module user interface.

The data module needs be saved before you start creating the dashboard.

Procedure

In the data module toolbar, click the **Create dashboard** button.

The dashboard user interface is opened with the **Sources** panel showing the data from the data module.

Chapter 2. Data modules and their sources

Data modules are containers that describe data and rules for combining and shaping data to prepare it for analysis and visualization in IBM Cognos Analytics.

Data module sources

Data modules can be based on data servers, packages, uploaded files, data sets, and other data modules. You can combine multiple, different types of sources into one data module.

When you create a new data module in IBM Cognos Analytics, or update an existing module, you choose the input source type from the **Select sources** dialog box.

Using data servers as data module sources

Data servers represent databases for which connections exist in Cognos Analytics.

The data server  connections must already be created in **Manage > Data server connections** or **Manage > Administration console**, and the metadata for one or more schemas in the data server must be loaded. Only schemas where metadata was loaded can be used in data modules.

When a data server schema is updated, you can reload the schema metadata from the data module. On the **Sources** tab, from the schema context menu, select **Reload metadata**.

For the legacy JDBC data source connections, ensure that the **Allow web-based modeling** check box is selected. These connections are created in the **Administration console**. If the **Allow web-based modeling** check box is not selected for this type of connections, the connections are not available in **Manage > Data server connections**, and cannot be used as data module sources. Go to **Manage > Administration console**. On the **Configuration** tab, select **Data source connections**, and find the connection. From the connection properties, click the **Connection** tab where the **Allow web-based modeling** check box is located.

If your data server is Planning Analytics, you create the TM1 cube-based data modules in the administration interface, as soon as the data server connection is created. For more information, see *Creating data modules from Planning Analytics cubes* in the *Managing IBM Cognos Analytics* guide.

For more information, see the *Managing IBM Cognos Analytics*.

Permissions and capabilities that are needed to use data servers

You need specific capabilities and access permissions to create data modules from data servers. The same permissions are needed to edit these data modules.

- Capabilities

You need **Traverse** access for the **Administration** capability, and **Execute** and **Traverse** access for the **Data server connections** capability.

These capabilities give you access to the **Manage > Data server connections** user interface, as shown in the following image:

The screenshot shows the 'Data server connections' page in the IBM Cognos Analytics with Watson interface. At the top, there are navigation links for 'IBM Cognos Analytics with Watson', 'Data server connections', a search bar, and user profile icons. Below the header, the title 'Data server connections' is displayed, followed by a brief description: 'Manage existing connections to data sources or create new connections that can be used across the platform.' A table lists five data server entries:

Name	Modified Time	Tenant
FVT_great_outdoors_DB2	1/20/2023, 5:51 PM	
DB2_Server	1/20/2023, 5:51 PM	
DB2_Server_NFK	1/20/2023, 5:50 PM	
GB18030	1/20/2023, 5:43 PM	LDAP

An 'Add data server' button is located at the top right of the table area.

Tip: The predefined **Modelers** role already has these capabilities assigned.

For more information, see "Setting access to user capabilities" in *Managing IBM Cognos Analytics*.

- Access permissions for data server connections

On the **Data server connections** page, ensure that you have at least **Read** and **Write** access permissions for the connections that your data module is based on.

Use the following steps to view or specify the permissions:

- Select the connection.
- From the connection **Action menu** , click **Properties**.
- Click the **Permissions** tab, and view or specify permissions for the connection.

For more information, see "Access permissions for an entry" in *Managing IBM Cognos Analytics*.

Using packages as data module sources

You can use relational, dynamic query mode packages as sources for data modules.

Packages are created in IBM Cognos Framework Manager and contain dimensions, query subjects, query items, and other data. Packages are located in **Team content** or **My content**.

Tip: Query subjects and query items in packages are equivalents of tables and columns in data modules.

For more information about packages, see the *IBM Cognos Analytics Getting started* guide.

Using uploaded files as data module sources

Uploaded files are supported types of data files, such as Microsoft Excel (.xlsx and .xls) spreadsheets or text (.csv) files, that you can upload into Cognos Analytics by yourself.

Files that are already uploaded to Cognos Analytics are stored in **Team content** or **My content**. You can also upload files after you start creating your data module by using the upload file facility in the **Select sources** dialog box.

For more information about uploaded files, see the *IBM Cognos Analytics Manage* guide.

Using data sets as data module sources

Data sets contain data that is extracted from packages or data modules.

Data sets are stored in **Team content** or **My content**. If the data in the source package or data module changes, the change is reflected in the data set.

For more information about data sets, see the *IBM Cognos Analytics Getting started guide*.

Using other data modules as data module sources

Existing data modules can be used as sources for other data modules.

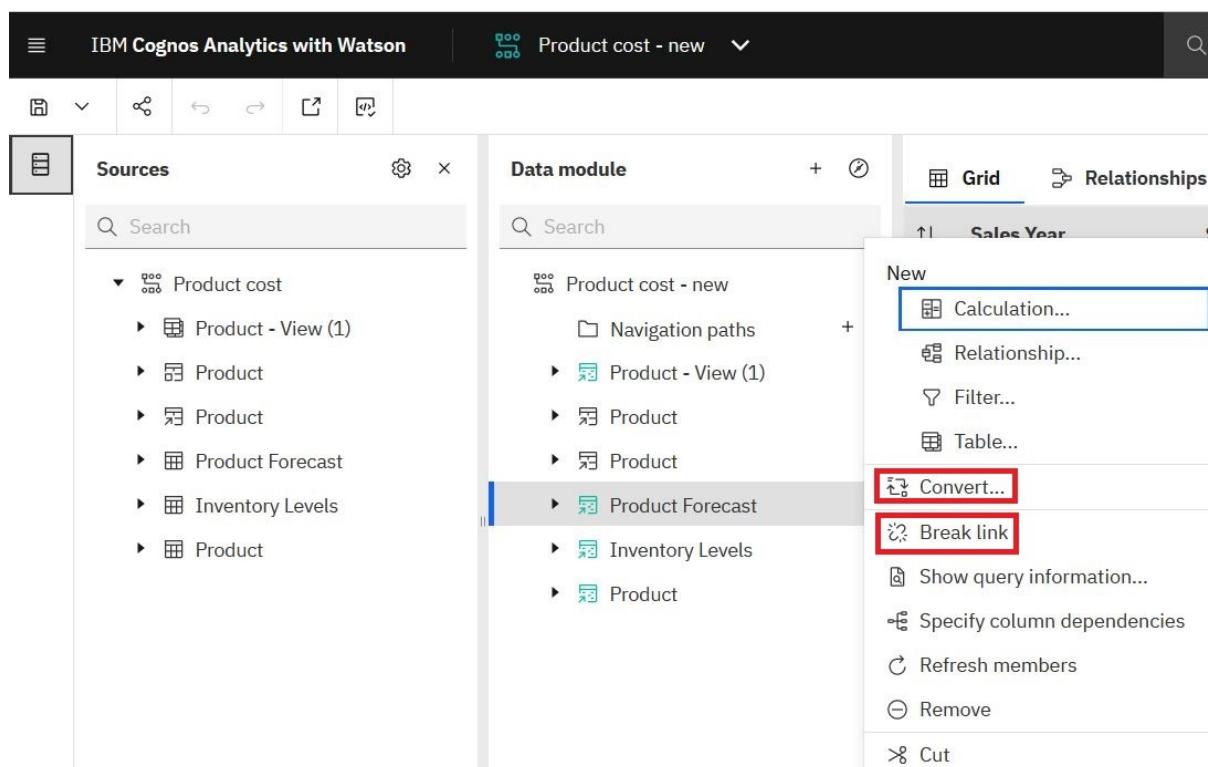
Data modules  are saved in **Team content** or **My content**.

The tables in the new data module remain linked to the source data module, which is indicated by the linked table icon .

Linked tables cannot be modified and are read-only. Relationships that are defined in the source module can be included in the data module as referenced relationships. If these relationships don't exist, they need to be recreated in the new module.

As long as a table remains linked to the source data module, any changes to the table in the source module are reflected in the new data module. If you break the link by using the **Break link** or **Convert** context menu, you can edit the table. However, changes to the table in the source module are no longer reflected in the new module.

In the following image from Cognos Analytics 11.2.3, the **Product cost - new** (**Data module** panel) data module is created from the **Product cost** (**Source** panel) data module. The tables in the **Product cost - new** data module are linked to the tables in the **Product cost** data module, which is indicated by the teal-colored icons.



The screenshot shows the IBM Cognos Analytics interface with the title bar "IBM Cognos Analytics with Watson". The main area is divided into two panels: "Sources" on the left and "Data module" on the right. In the "Sources" panel, there is a tree view under "Product cost" with nodes for "Product - View (1)", "Product", "Product Forecast", "Inventory Levels", and "Product". In the "Data module" panel, there is a tree view under "Product cost - new" with nodes for "Navigation paths", "Product - View (1)", "Product", "Product", "Product Forecast", "Inventory Levels", and "Product". A context menu is open over the "Product Forecast" node in the "Data module" panel, listing options: "Calculation...", "Relationship...", "Filter...", "Table...", "Convert..." (highlighted with a red box), "Break link" (highlighted with a red box), "Show query information...", "Specify column dependencies", "Refresh members", "Remove", and "Cut".

Important: Starting with Cognos Analytics 11.2.4, the **Break link** option is completely replaced by the **Convert** option and removed from the product.

When you use source data modules that contain either shortcuts  or aliases , shortcuts are created in the new data module for both the shortcut and alias tables. These shortcuts continue referencing the shortcut and alias tables in the source data module. Changes to the shortcuts and aliases in the source module are reflected in the new data module. One of the benefits of using shortcuts is their ability to inherit the relationships that are defined in the source data module.

Tip: When you drag and drop a table from a source data module to the new data module, a linked table is created in the data tree of the new module, which is indicated by the teal color icon . If the table that you drag and drop is a shortcut or alias, a shortcut  (black color icon) is created in the new data module. This shortcut references the shortcut or alias in the source module. This action is an equivalent of creating a shortcut in the new module and relinking the shortcut to a shortcut or alias in the source module.

For more information, “[Converting referenced tables](#)” on page 39 and “[Relinking referenced tables](#)” on page 40.

Creating a data module

A user can quickly create a data module that includes data from one or more sources of different types.

The data module can be shared with other users, and used as a source to create reports, dashboards, stories, and explorations.

Before you begin

Prepare the sources that you plan to use to create the data module.

- Save the sources to **Team content** or **My content**.
The only exception are your data files that can be uploaded while the data module is created.
- For data server sources, create connections in **Manage > Data server connections**.

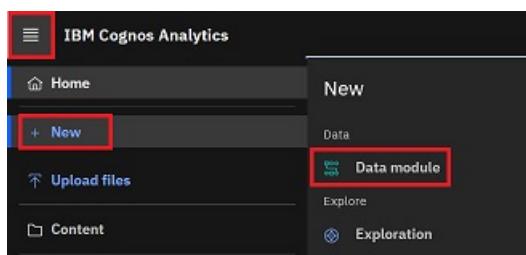
For more information, see “[Using data servers as data module sources](#)” on page 7.

About this task

To access the data modeling user interface, users need execute and traverse permissions for the **Web-based modeling** capability. For more information about capabilities, see the *Managing IBM Cognos Analytics* guide.

Procedure

- In the Cognos Analytics welcome page, from the **Open menu** icon  in the application bar, click **New > Data module**.



Tip: An alternative way to start creating a data module is to upload data files first by using the **Quick launch** facility. When you drop the files onto the Cognos Analytics welcome page, in the **Data module** box, you can immediately start creating your data module. Other sources can be added to the data module later.

- In the **Select sources** dialog box, select one or more sources of any type.

- To select a saved data module, data set, uploaded file, or package, click the **Team content** , **My content** , or **Recently viewed content**  folder, and locate the source that you want to add. If needed, use search and filtering options to find the sources.

- To select a data server, click the **Data servers and schemas**  folder. Select the data server connection that you want. 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.
 - To upload a data file from your hard drive or LAN location, click the **Upload** icon , and browse for the file. By default, the file is saved to **My content**.
3. If all selected sources contain one table each, the basic data module is created, and you can proceed to step 5.
4. If any of the selected sources, such as a multi-tab spreadsheet or a data server, contain multiple tables, you have two options to add the tables to your data module:
- Select tables**
You select the tables manually, and click **OK** to create the data module.
 - Discover related tables**
A word cloud visualization is displayed that contains keywords from sources that the data module is based on. Select one or more keywords, and click **Next**. A data module proposal is generated for you. You can accept the proposal, or click **Previous** to try different keywords. To accept the suggested proposal, click **OK**. The data module is created for you.
For more information, see [“Discovering related tables” on page 13](#).
- The data module is created based on the chosen tables.
5. Examine the data module.
- In the **Data module** panel, view the sources that are included in your data module.
You can expand the sources to view their tables, columns, and members.
The link icon  on tables indicates that the tables are linked to the source data module. For more information, see [“Relinking sources” on page 16](#).
For data server schemas and uploaded files, the table and column labels are cleaned up in English and some other languages in the following way:
 - For uploaded files, the file extension, such as .xls or .csv, is removed from the table label. For example, Customer.csv is changed to Customer.
 - If all label characters in the source are in uppercase, and the label contains special characters, such as underscore (_), dash (-), or slash (\), in the data module, each word in the label is capitalized, and the special characters are replaced with the space characters. For example, VEHICLE_CLASS in the source is changed to Vehicle Class in the data module.
If this cleanup produces unintended results, you can disable it at the service level. For more information, see "Disabling the default cleanup of labels in data modules" in *Managing IBM Cognos Analytics*.
- To view data, select a table or a column in a table, and click the data grid view .
 - To view relationships between tables, click the **Relationships** tab . Typically, the relationships are detected by the system and joins between tables are created automatically. If the tables are not joined, you need to join them manually. For more information, see [“Creating a relationship manually” on page 28](#).
 - The data module is validated automatically. If there are any broken references, the failed validation icon  is displayed in the data module tree and in the diagram. For more information, see [“Validating data modules” on page 63](#).
6. To create a test report from your data module, click the **Try this data module in Reporting** icon  in the data module toolbar.

A new tab opens in your browser with IBM Cognos Analytics - Reporting open within it. Your data module is shown in the **Insertable objects** pane.

Note: The data module object capabilities are not applied with the option **Try this data module in Reporting** when editing the data module.

7. To save the data module, click **Save** or **Save as**.

Results

The data module  is created in the location that you saved it to, in **Team content**  or **My content** .

What to do next

You can enhance the data module by adding calculations, filters, groups, custom tables, and more. For more information, see [Chapter 3, “Modeling metadata ,” on page 27](#).

Creating data modules from OLAP cubes

Starting in Cognos Analytics 12.0.3, after successfully establishing a connection to an OLAP data server, you can browse its cubes and use them to create data modules.

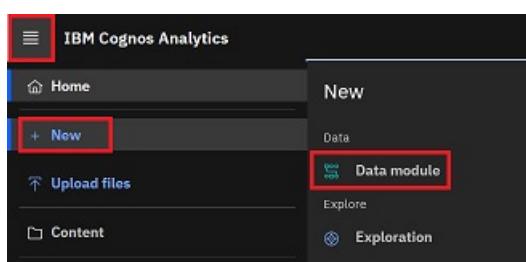
Data modules that are based on OLAP cubes can be used to create reports, dashboards, stories, and other Cognos Analytics content in the same way as packages that contain OLAP cubes are used.

Before you begin

A successful connection of type **IBM Planning Analytics**, **IBM Planning Analytics as a Service**, **Microsoft Analysis Services**, **Microsoft Azure Analysis Services**, or other supported OLAP data server connection must already be created. For more information, see the "Creating a data server connection" topic in the *IBM Cognos Analytics Manage Guide*.

Procedure

1. In the Cognos Analytics welcome page, from the **Open menu** icon  in the application bar, click **New > Data module**.



2. In the **Select sources** dialog box, click the **Data servers and schemas**  folder.
3. Click an OLAP data server to view its cubes.

Note: Only cubes for which metadata was loaded are available to create data modules. For more information, see the "Loading OLAP metadata" topic in the *IBM Cognos Analytics Manage Guide*.

4. Select one or multiple cubes, and click **OK**.

In the data module user interface, the cube data is shown in the **Data module** panel.

5. Save the data module in **Team content** or **My content**.

Results

The data module is created in the location that you specified.

What to do next

Use cube-based data modules to create dashboards, reports, explorations, and other Cognos Analytics content.

Discovering related tables

You can engage the system to suggest the most appropriate tables for your data module. Using natural language processing and AI-based functionality, the system generates a data module that best represents your use case.

This functionality is used when creating a data module or adding new sources or tables to a data module.

The choice of tables for the data module is based on keywords that you select. An interactive word cloud visualization shows the keywords that exist in the available sources.

The following example shows a word cloud for a data module proposal that is based on four sources:

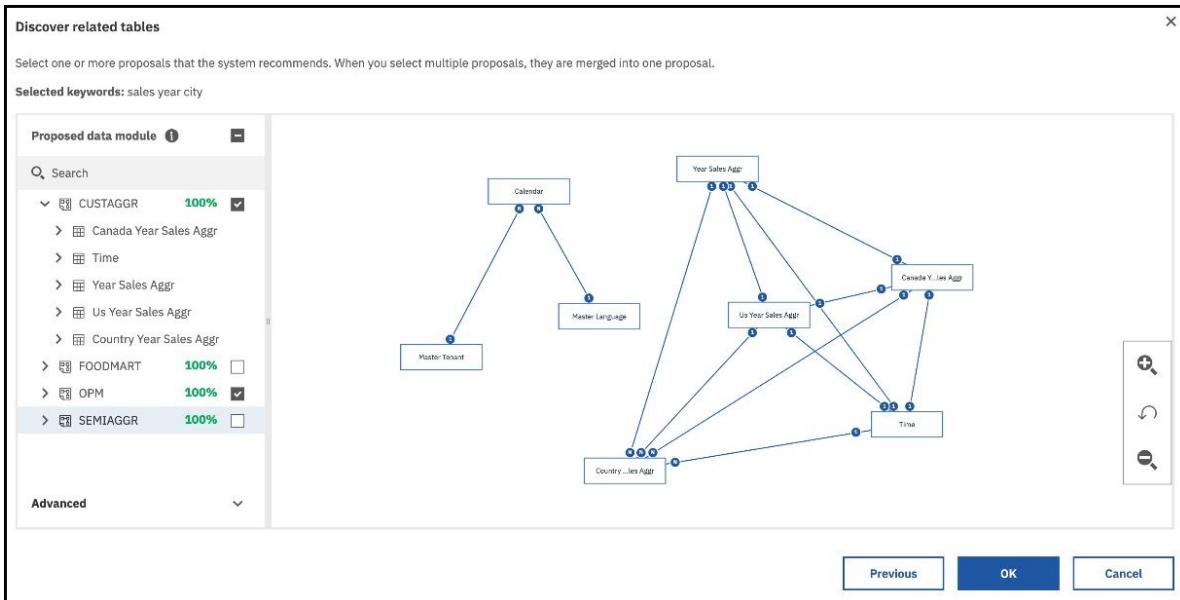


The font colors represent the different sources. The font size indicates the keyword weight, which is the measure of the keyword importance in the source. Selecting keywords with higher weight increases the probability of creating the most relevant data module for your use case.

To increase or decrease the number of keywords in the word cloud, expand the **Keywords** section in the right pane, and enter a number for the **Keywords limit** option.

You can select the keywords from the word cloud, or type them in the search bar. The selected keywords are automatically added to the search bar. To deselect the keywords, delete them from the search bar.

After you click **Next**, a data module proposal is generated, as shown in the following example:



The **Proposed data module** pane shows the tables that the system suggests to use for the data module. By default, one proposal is generated for each source. A percentage confidence score is assigned to each proposal. The confidence score reflects the predicted ability of the proposal to fulfill your modeling objective.

Select one or more of the suggested proposals. The selected proposals are merged into one proposal, and table relationships are generated.

Note: To increase the number of proposals per source, expand the **Advanced** section in the **Proposed data module** pane, and increase the number.

Click **OK** to accept the proposal, or click **Previous** and try to generate different proposals.

Adding sources or tables to a data module

After a data module is created, you can add new sources or different tables from the sources that are already in the data module.

About this task

You can add a combination of source types in a data module.

Procedure

1. Open an existing data module.
2. In the **Data module** panel, click the **Add sources and tables** icon
3. Select one of the following options:

- **Add new sources**

Select and add new sources to the data module.

- **Add more tables**

Add tables from sources that are already in the data module. Only tables that aren't already included in the data module can be selected.

- **Discover related tables**

Add tables from sources that are already in the data module. This option is available only for sources that contain multiple tables, such as data servers and multi-sheet uploaded files. Based on

keywords that you select, related tables are suggested to be added to the data module. For more information, see “[Discovering related tables](#)” on page 13.

4. Save the data module.

Updating columns in a data module

After a table in a data module source is updated, you can add or remove individual columns in the data module without updating the whole table.

This functionality can be used for the following scenarios:

- A source column was deleted from a data module table, the data module was modified and saved, and the modeler wants to re-add the deleted column to the data module.
- A new column was added to an existing table in a database, and the modeler wants to use this column in the data module.
- A column was removed or renamed in a database, and the modeler must update the data module to avoid validation errors.

About this task

For data modules that are based on data servers, use the **Reload metadata** function to reload the latest schema metadata from the database. For more information, see “[Reloading the schema metadata](#)” on page 16.

Procedure

1. Open the data module that you want to update.
 2. Click the **Source view** icon  to open the **Sources** panel, and expand the source tree.
 3. If the source is a database server schema, from the schema context menu , click **Reload metadata**.
- Tip:** You don't need to perform this step if the database server schema was already reloaded in the **Data server connections** administration user interface after a database change that needs to be reflected in Cognos Analytics. For more information, see “[Loading metadata](#)” in the *IBM Cognos Analytics Managing* guide.
- The tables and columns are reloaded based on the latest state of the data server.
4. In the **Sources** panel toolbar, click the **Source tree settings** icon  , and select the **Show unused items** check box.
 - The columns (and their tables) that are not in the data module are highlighted in the **Sources** panel.
 - If the source contains renamed columns, or columns were removed from the source, validation error icons  appear next to the affected columns in the **Data module** panel.
 5. In the **Sources** panel, identify the column that you need to add to the data module, and drag the column to the related table in the **Data module** panel. Expand the table if you want to drop the column into a specific place in the table.
- You can drag multiple columns from the same table at once.
6. If the data module contains validation errors, the columns in the module might be missing or renamed in the source. Use the following steps to resolve the errors:
 - a) Drag the renamed columns to the **Data module** panel.
 - b) Remove the columns that no longer exist in the source from the data module.
 - c) Validate the data module.
- Potential validation errors might be related to the broken references in expressions, such as filters or calculations, that might still refer to the removed columns. Using information in the error messages, manually update these expressions.

7. Save the data module.

Reloading the schema metadata

When a database schema is updated, the schema must be reloaded for the data module to remain synchronized with the database.

To create a data module that is based on a data server source, the database schema metadata must be loaded and saved to the content store. This task is performed by administrators in the administration component after the data server connection is created. For more information, see "Loading metadata" in the *IBM Cognos Analytics Managing* guide.

For existing data modules, the modeler can reload the schema metadata after the schema is updated. This step might be needed when you encounter validation errors in the data module.

Before you begin

To access the **Reload metadata** action in a data module, you need the following permissions:

- Data server - Write permission
- **Administration** capability - Traverse permission
- **Data Source Connections** capability (child of the **Administration** capability) - Access permission

Procedure

1. Open the data module that is based on a relational data server.
2. Click the **Source view** icon  to open the **Sources** panel.
3. From the schema context menu, click **Reload metadata**.

If this action is not available for a data server schema, ensure that you have the required permissions for the data server and the **Administration** and **Data Source Connections** capabilities, as documented earlier in this topic.

Results

The tables and columns are reloaded based on the latest state of the data server. If the source contains renamed items, or items were removed from the source, validation error icons  appear next to the affected columns in the **Data module** panel.

What to do next

To compare the data in the schema and the data module, in the **Sources** panel toolbar, click the **Source tree settings** icon, and select the **Show unused items** check box. The tables and columns that are not in the data module are highlighted in the **Sources** panel. You can proceed to update the columns in the data module.

Relinking sources

You can relink a data module source to a different source. After a successful relink, global calculations and relationships in the data module remain valid.

Here are some scenarios in which relinking a source can be useful:

- You build and test a data module against a test source. When the data module is ready, you relink the source to the intended production source.
- The current source in your data module is invalid, and you must use a new, valid source.
- You want to relink your data module from one data server to another data server, or from one schema to another schema.

Relink between different types of data servers is supported, as well as between schemas and catalogs within data servers.

Tip: Data server sources can be organized into schemas, catalogs, both, or none.

About this task

The relinked (target) source must be of the same type as the original source. A data server can be relinked only to a data server, an uploaded file to an uploaded file, and so on.

In addition to the matching source types, the following conditions must be met:

- All columns from the original source must exist in the target source, and the columns **Identifier** properties (case-sensitive) and data types must match.

For example, file A with columns ColA and ColB can be relinked to file B with columns ColA and ColB. Relinking to file B with columns colA and colB would not work.

The data types of the matching columns must be compatible for the data module calculations and relationships to remain valid. For example, if the column data type in the original source is date, the column data type in the target source must also be date, and not string or timestamp.

- For data servers, packages, and data modules, all tables from the original source must exist in the target source, and the tables **Identifier** properties (case-insensitive) must match. If a matching table can't be found based on these criteria, the system also considers the table labels and matching columns identifiers (case-sensitive) when trying to find the right match.

If a duplicate match is found in the target source, the last table in the list is used for the match.

- Extra columns and tables can exist in the target source.

When relinking to a source that contains a table with extra columns, you can add the extra columns to the table in the data module by dragging the table from the **Sources** pane to the **Data module** pane.

- The source names, such as file and package names or data server connection names, do not need to match.

Tip: Columns and tables matching is done by comparing their **Identifier** property. The column or table **Identifier** value can be, but not always is, the same as the column or table name (**Label**). You can view the **Identifier** value in the column or table **Properties** pane, **Advanced** section.

Procedure

1. From **Team content** or **My content**, open your data module.
2. In the **Sources** pane, find the source that you want to relink.
 - 3. From the source context menu , select **Relink**.
 - 4. Select the source type that matches the original source type. If the original source is a data server, select a data server. If it's an uploaded file, select a file, and so on.
 - 5. Click **Done**.

If the relink was successful, a confirmation message is displayed.

If the relink finished with errors, a message is displayed that suggests opening the [validation view](#) where the relink issues are listed. Resolve the issues, and save the data module. You can also save the data module with unresolved issues.

Important: The validation process does not detect incompatible data types on columns. If there are columns with incompatible data types in your sources, and all other relink conditions are met, the successful relink message is displayed. This type of data issues must be reconciled in the sources.

Results

After you successfully relink a source in a data module, reports and dashboards that are based on this data module can start using the new source without any involvement from report authors.

Defining source overrides

You can override the default data server connections, catalogs, or schemas that the data module is based on. As a result, the module can dynamically switch to a different source (data server connection, schema, or catalog) at run time, without any intervention from a user.

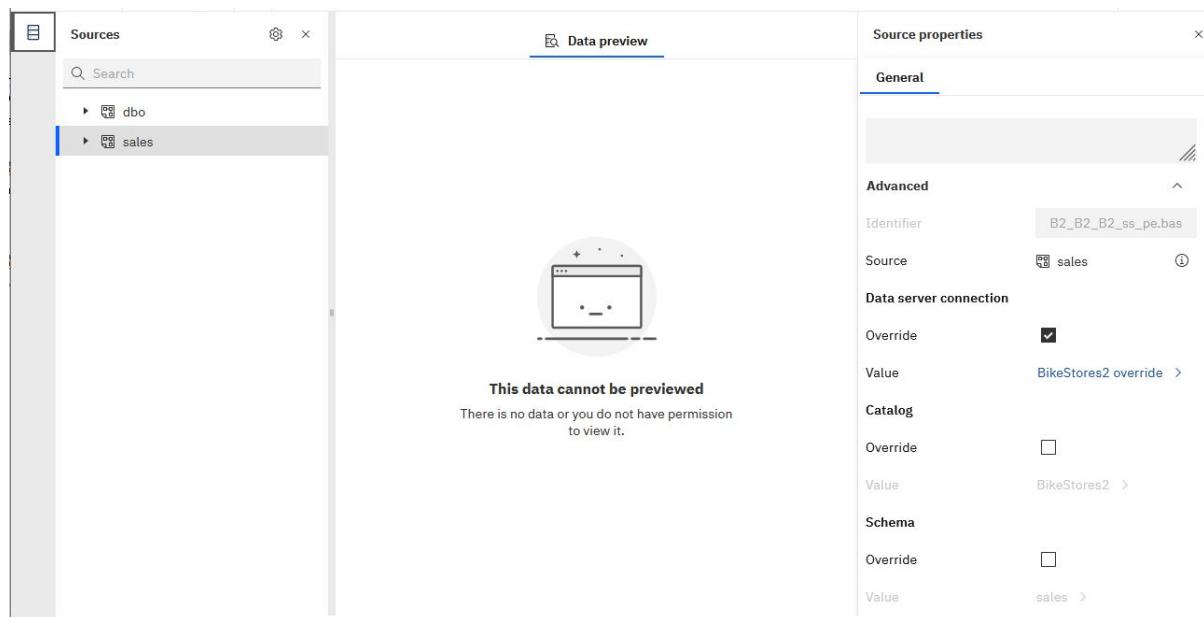
Source overrides are supported only for data modules that are based on data servers. If you reference a data module in which override values are specified in another data module, the override values are maintained.

Note: Source overrides are not supported for data modules that are based on packages, uploaded files, and data sets.

Dynamic source overrides are useful when the development, test, and production environments use different data server connections, schemas, or catalogs, or in a multi-tenant application where each tenant uses a different instance of the same database schema or catalog.

About this task

Use the **Override** property on a data server connection, schema, or catalog to specify the override values. The following screen capture shows the override properties for the sales schema:



The **Override** checkbox determines if the default value or the override value of the property is used. If the checkbox is selected, the specified override value is used. If the checkbox is clear, the default value is used.

Note: The default values are derived from the data server schema or catalog that was used during metadata import, or from the schema or catalog that was used for relinking.

The override value can be a string, which is a name of a data server connection, schema, or catalog that exists in your data server, or a macro. In a macro, you can use session parameters, such as `tenantID`, `machine`, or `account.personalInfo.userName`.

Procedure

1. From **Team content**, open your data module.
2. Expand the **Sources**  panel.
3. From the source (schema or catalog) context menu , select **Properties**.
4. On the **General** tab, scroll down to the **Advanced** section, and expand this section.

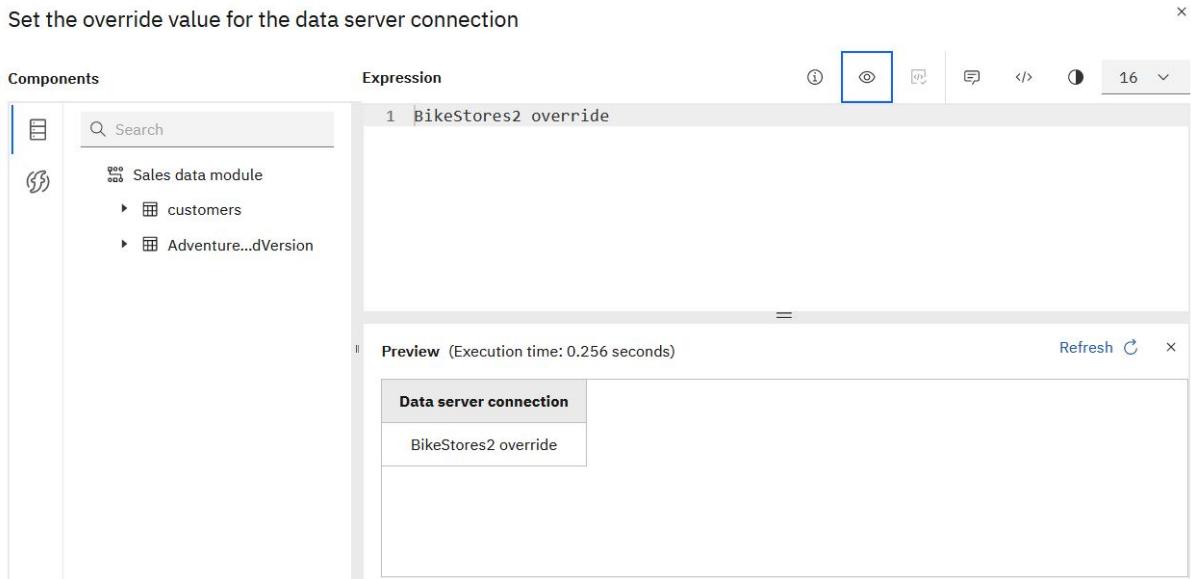
The **Override** property is available for **Data server connection**, **Schema**, and **Catalog**.

5. Select the **Override** checkbox, and click the **Value** link.

The override expression editor is displayed.

6. Add the override value in the **Expression** area.

If the value is a simple string, type the string.



If the value is a macro, type the macro text within the macro tags `##`. Comments that are associated with a macro must be included within the macro itself.

For example, the following macro is used to dynamically switch to another source when the data module is used in the development environment, testing environment, or production environment.

```
#let datasource =
case $machine
when 'Holst' then 'GOSLDW_DNB_DB2'
/* this is the development machine */
when 'Scarlatti' then 'GOSLDW_DNB_MSSQL_TEST'
/* this is the testing machine */
when 'Buxtehude' then 'GOSLDW_DNB_DB2_SECURED'
/* this is the production machine */
else 'nothing'
end;
datasource #
```

In this example, the preview shows the value of GOSLDW_DNB_DB2 because the machine name is Holst. This is the override value for the data server connection.

Set the override value for the data server connection

The screenshot shows the 'Components' and 'Expression' tabs. The 'Components' tab lists various items like 'Dynamic sc...t and prod', 'Star sch...roupings', etc. The 'Expression' tab contains the following code:

```
1 #let datasource=
2 case $machine
3 when 'Holst' then 'GOSLDW_DNB_DB2'
4 /* this is the development machine */
5 when 'Scarlatti' then 'GOSLDW_DNB_MSSQL_TEST'
6 /* this is the testing machine */
7 when 'Buxtehude' then 'GOSLDW_DNB_DB2_SECURED'
8 /* this is the production machine */
9 else 'nothing'
10 end;
11 datasource#
```

Below the code, there's a 'Preview' section showing a table with one row:

Data server connection
GOSLDW_DNB_DB2

Note: If the database doesn't use catalogs, the default override value for **Catalog** is null. And if the database doesn't use schemas, the default override value for **Schema** is null.

Results

Both the default values and the override values are stored in the data module, and persist from session to session. If you clear any of the **Override** checkboxes, the default value is restored and displayed. If you select the checkbox again, the override value is restored and displayed.

If you relink a schema or catalog, the override property values are removed from the data module. To restore these properties, click the undo button . However, consider whether you need to relink manually since the override properties allow you to relink dynamically.

Enriching packages

To optimize the user experience in IBM Cognos Analytics components, such as dashboards and explorations, Framework Manager packages must be enriched.

The enrichment process associates the Cognos Analytics data characteristics, such as **Time** and **Geographic location**, to query items in the packages. The information from the enrichment process complements the information, such as the data type, column name, or **Usage** property value, that is derived from the package metadata.

An enriched package includes the data characteristics that are required for the artificial intelligence (AI) based functionality in the product, such as visualization recommendations or intelligently set default values on column properties. For example, to display the relationships diagram in **Explore**, an enriched package must be used. Otherwise, the relationships diagram isn't displayed.

The enrichment process can be time and memory-intensive so it should be performed only when the original package has changed. Consider reenriching the package after the following changes to the package:

- Names of query subjects, query items, and namespaces are changed.
- Data types on query items are changed. For example, number changed to string.
- New query items are added.

- Filters or expressions are changed that significantly alter the values that the query subject would return.
- A deployment archive is imported into a new environment that uses different data from the source used for a previous enrichment.

When a package is republished, existing enriched metadata isn't removed or refreshed.

Before you begin

To minimize the impact of the enrichment process on the system, consider creating smaller packages that include only a subset of purpose-specific query subjects, and enriching only the smaller packages. For example, a package used by advanced report authors might expose many query subjects where many of the query subjects aren't relevant when creating dashboards or explorations. You can create a smaller package off the original package, and include only those query subjects that you need in your dashboards and explorations. Enriching this smaller package requires less time and memory.

About this task

You can enrich a package metadata by using the automatic or manual process. The automatic process evaluates all query items of all selected query subjects in the package, and automatically applies the data characteristics to them. To minimize the impact on the system, you can deselect namespaces or individual query subjects to exclude them from the enrichment process.

In the manual process, you explicitly apply the data characteristics to individual query items. The manual process is not applicable for dimensional data.

When enriching a package, you typically start with the automatic process. Use the manual process to enrich only a small subset of query items or to override values that were set incorrectly by the automatic option.

The automatic enrichment includes the option to retrieve sample data. When this option is selected, the Cognos Analytics query engine connects to the data source and reads a sample of its data. The enrich dialog box allows the sample size to be changed. Setting the sample size to a low value, or not sampling at all, reduces the amount of information that the enrichment can gather. The amount of sampled data also depends on the signons that are used to access the package underlying data sources. An ideal signon can access the tables, views, and columns that the query subjects are based on, and a representative number of rows and values in the queried tables and views.

To access the **Enrich package** functionality, you need write permissions for the package.

Procedure

1. Locate the package or its shortcut in **Team content** or **My content**.
2. From the package or shortcut context menu , select **Enrich package**.

Tip: If a package was used as a data module source, you can enrich the package in the modeling user interface, from the **Sources** pane.

3. Select one of the following options.

- **Enrich automatically**

Most of the time, start with this option. The status information shows you the dates when the package was last published and enriched (if it was enriched before).

- In the **Select tables** panel, you can deselect the query subjects that you don't want to be evaluated by the enrichment process. By default, all visible query subjects in the package are evaluated.

This option gives you the opportunity to exclude the query subjects that aren't used in your dashboards or explorations, and therefore reduce the time and memory usage by the system during the enrichment process.

- To enable data sampling, select the **Retrieve sample data** checkbox, and specify the number of rows of data to be retrieved.

The data sample includes some deeper data characteristics that support the product functions that are behind the optimized user experience in dashboards, explorations, and other components. Extracting too many rows might impact the system performance. No data sampling, or too few rows might not provide enough information. Clearing this checkbox reduces the time and memory usage during the enrichment process, but the expected information might not be gathered.

For more information, see *Data sampling* in the *Managing IBM Cognos Analytics* guide.

- Click **Run**.

Depending on the number of query subjects involved, the enrichment process can take some time, potentially even hours. After the process is finished, an information message shows you the results of the process. Even if only a certain percentage of the query subjects were enriched, you might have enough data to support the AI-functions in your dashboards and explorations.

- Click **Close**.

- **Enrich manually**

Use this option to enrich individual query items.

- Expand the package.

Then, expand a query subject, and select one or more query items.

- From the **Define data representation** drop-down menu, select the option that you want the data in the query to represent.

Select either **Time** or **Geographic Location**, and their specific values. The **Default** value allows to propagate settings from the source.

- Click **OK**.

Setting up data caching

You can enable data caching in a data module, and specify the cache expiry options.

The cache is populated by the result sets from queries to data servers. These queries originate from widgets (visualizations in dashboards, reports, stories, and explorations).

The cached result sets are reused when a subsequent request that generates exactly the same or compatible statement is made. An example of a compatible statement is one that's identical to another statement, except that it has one more filter.

To see how caching is used, open a dashboard, report, story, or exploration, and then open another dashboard, report, and so on, where at least one of the widgets has exactly the same or compatible statement.

Results that are cached from one user's request can be used for a different user's request only if both users have the same data security profiles, which means that the following information is the same for both users:

- Sign-on information (user name and password) to the data server.
- Expanded values of data source connection command blocks.
- Data security defined in the data module.
- Expanded values of macros within the queries used to populate the cache.

About this task

Data caching can be enabled at a source level, or at a table level. Tables don't automatically inherit the cache options from their sources.

The following data cache options can be specified for sources and tables:

- **No cache**

Data caching is disabled.

- **Automatic**

Reflects the data cache option that was specified for a source. This option is available for tables only.

- **Custom**

Enables data caching, and allows to specify the length of time to keep the cached data.

- **Macro**

Enables data caching that is based on a macro.

Data caching is not applicable to OLAP cube data, data sets, and uploaded files even if the **Data cache** setting is available in the user interface.

Procedure

1. From **Team content** or **My content**, open a data module.
2. To specify data caching for a source, do the following steps:
 - a) Click the **Source view** pane ▾ to expand it, and locate the source.
 - b) From the source context menu ⚙, select **Data cache**.
 - c) Specify one of the cache options, and click **OK**.
3. To specify data caching for a table, do the following steps:
 - a) In the **Data module** pane, select one or more tables, and from the context menu, click **Properties**.
 - b) Under **Advanced** properties, locate the **Data cache** property.
 - c) Specify one of the cache options, and click **OK**.
4. Save the data module.

Results

Cached results are retained for the length of time that is specified in the **Data cache** setting. The timestamp of the cache entry is the time at the beginning of the request, the moment before the query execution starts.

When fields from different tables that have different cache settings are used together, the cache is retained as long as the cache for the table with the smallest setting. For example, if the data cache for one table is set to 5 min, and for another table to **No cache**, there's no caching for a visualization that uses fields from both tables.

What to do next

The **Data cache** setting in data modules can be overwritten in dashboards and stories. In these components, you can also enable local caching that allows the client applications to store the query results in the browser. Local caching is not available in Cognos Analytics reports.

Securing data

You can secure data at the value level by creating security filters.

A security filter defines which users, groups, or roles have access to specific data values in a table. When the users work with dashboards, reports, or explorations that use the table, only the data that is included in the security filter is visible to them.

There are business reasons for restricting access to data at such low level of granularity. For example, you have confidential data that only specific users are allowed to see. Or, a table contains many records, and your users need only a subset of those records.

Note: When you define data security filters, the filters are applied to the data server schema and are reflected in new and existing data modules that use the schema as a source.

Before you begin

The schema metadata for the associated data server connections must be loaded, and you must have write permissions for the connections and their signons.

Tables that are based on typed-in SQL bypass security filters. To avoid potential security risks, specify the **ibmcognos.typeinsqldisabled** property on the data server connection that your data module is based on. If an attempt is made to create an SQL-based table after this property is specified, the table is not created. If this property is specified after an SQL-based table was created, the query execution is stopped. For more information about Cognos-specific connection parameters, see the *IBM Cognos Analytics Managing Guide*.

About this task

This type of data security can be implemented only for data server sources.

Users who perform this task must belong to a role that has the **Set data security** feature that is enabled in the administration interface. Otherwise, the table context menus in the data module won't show the **Set data security** option that is needed to perform this task. For more information, see [“Customizing the user interface” on page 4](#).

Procedure

1. From **Team content** or **My content**, open a data module.

The data module source must be a data server, or another source that includes data server tables.

2. Click the **Sources** pane  to expand it.

3. Expand the data server schema to view its tables.

4. From a table context menu , select **Set data security**, and click **Add security definition**.

5. In the **Set data security** dialog box, create the filters by associating specific users, groups, or roles with columns in the table. Your security definition can include one or multiple filters.

- a) In the **Users, groups and roles** pane, click the add icon . In your authentication namespace, locate the users, groups, or roles for which you want to define access to the table data, and select their associated check boxes. The selected names appear in the **Selected users, groups and roles** pane.

- b) In the **Filters** pane, from the **Select a column** drop-down list, select one column, and click **Create a filter**. Specify the required filter conditions, and click **OK**. You can add filters for other columns in the same way.

- c) Use the **via expression editor** option from the **Select a column** drop-down menu to add filters for multiple columns at once.

When this option is used, the filter can reference columns from other tables in the data module, not only the table that was selected in step 4. The syntax to reference another table uses an IN clause

with an expression that contains the required filter condition. A CASE statement is the easiest one to use as a condition, but you can also use an IF THEN ELSE statement in this role. When the CASE statement is used in the IN predicate of the filter condition, a subquery is generated that does not require to be joined to the originally selected table (step 4).

For example, if the 'employee' table needs to be secured by using the 'department' table, the following filter expression can be used to set the column 'department_id' to '2':

```
department_id IN (CASE WHEN department.department_id=2 THEN department.department_id END)
```

- d) Specify a name for the security definition, and click **OK**.

Tip: Your security definition can include one or multiple filters.

Results

The security definition is added to the **Security filters** tab in the table properties. In the **Sources** pane, the padlock icon  appears next the table name.

Chapter 3. Modeling metadata

The initial data module that you create manually or using intent modeling can be modified, edited, and enhanced.

You can enhance your data module by adding new tables or sources, applying filters, creating calculations and navigation paths, changing column formatting, and more.

Relationships

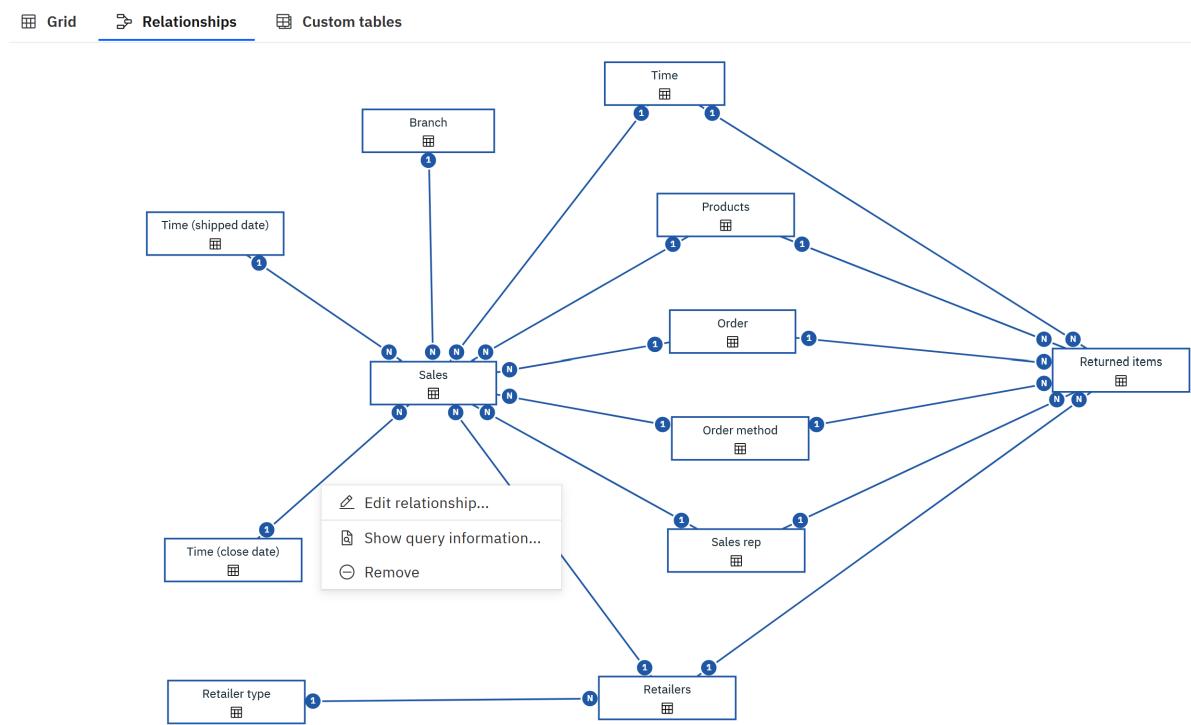
A relationship joins logically related tables that the users want to combine in a single query.

Cognos Analytics automatically detects relationships between tables in a data module by using the autojoin algorithm. For more information, see “[Automatic joins](#)” on page 1.

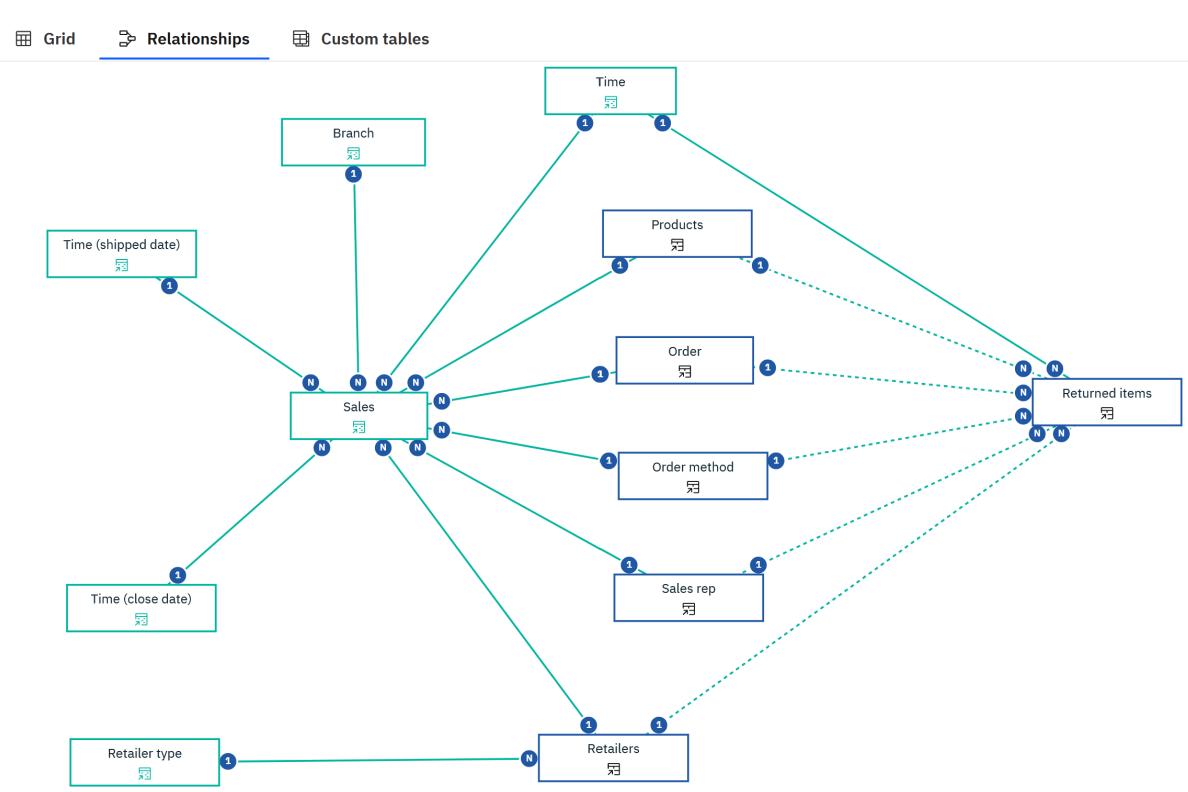
You can modify or delete relationships, or create new ones so that the data module properly represents the logical structure of your business. Verify that the relationships that you require exist in the data module, the cardinality is set correctly, and referential integrity is enforced.

The Relationships diagram

The **Relationships** diagram provides a graphical view of table relationships in a data module. The relationships are represented by solid lines between tables. You can use the diagram to examine, create, and edit the relationships.



The following image shows the relationships diagram for a data module that is sourced from another data module.



The teal table icons indicate that the tables are linked to tables in the source data module. The black icons indicate that the linked tables were converted to shortcuts, or that the shortcuts represent custom tables from the source data module.

Tip: All types of custom tables (views, shortcuts, aliases, joined views, unions, intersects, and excepts) from the source data module appear as shortcuts in the new data module. For more information, see “Custom tables” on page 31.

The dotted line appears between the shortcut tables whose target tables in the source data module have a relationship. This way, different users of the data module know that the shortcut tables are related through their targets. Both ends of the relationship must be shortcuts for the dotted line to appear. This feature is available as of Cognos Analytics 12.0.2.

For more information, see [“Using other data modules as data module sources” on page 9](#).

Creating a relationship manually

You need to create relationships when they are not detected by IBM Cognos Analytics.

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.

Important: 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 table context menu , click **New > Relationship**.

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 **Relationship**.
- On the **Relationships** tab in the table properties, click **Add a relationship**.

If the data module does not include the table that you need, you can drag the table from **Selected sources** directly onto the diagram.

2. In the **Create relationship** dialog box, select the second table to include in the relationship.

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.

3. Find the matching columns in both tables, and select **Match selected columns**. For example, you can match on `Product_id` columns.

The matching columns are highlighted in the data grid. You might need to click **Refresh** to retrieve the data.

4. Click **Matched columns** to specify the join operator for the match.

In the **Defined matches** dialog box, select a join operator. By default, columns are matched based on similar values, by using the equal (=) operator. However, you can also match columns based on a range of values by choosing a different join operator. For more information, see [“Join operators” on page 30](#).

5. Click the relationship settings icon . By default, the relationship settings are detected from the source.

Review, and if needed, modify the following settings:

Relationship Type

The following types can be specified: inner join, left outer join, right outer join, and full outer join.

Cardinality

The following types can be specified: 1-to-1, 1-to-many, and many-to-1.

Optimization

Use the optimization filters to reduce the number of rows of data that are retrieved when the join is executed. For more information, see [“Join optimization” on page 31](#).

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.

What to do next

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

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

Generating auto-relationships

You can automatically generate new relationships between tables in a data module. The tables can be located in different sources.

This feature has the following benefits:

- Reduces the effort that is required to identify related tables and establish connections between them when [creating relationships manually](#).
- Reveals relationships that users might not have originally considered.
- Enhances the overall efficiency and depth of relationship discovery.

Procedure

1. In the **Relationships** diagram or the data tree, select one or multiple tables, and from their context menu, click **Auto-relationships**.

- If you initiate auto-relationships for one table, when the relationship discovery is successfully completed, the toast message shows the number of generated relationships, and the new relationships are visible in the relationship diagram.
- If you initiate auto-relationships for two or more tables, a dialog box is displayed where you can select or clear the **Include other tables in the data module** checkbox to limit the discovery of relationships to the tables that you selected or include all tables in the data module (default). Then, click **Generate**.

After the discovery is successfully completed, the toast message shows the number of generated relationships, and the new relationships are visible in the relationship diagram.

Tip: You can use the undo  and redo  buttons in the application bar to revert or restore your changes.

2. Save the data module.

Join operators

Join operators are used to specify the type of match between columns that are joined in a relationship.

By default, values are compared by using the equal (=) operator. When you use an operator other than equal (=), you create joins that are based on a range of values.

The following join operators are supported:

Equal (=)

Values in the left and right columns are identical or similar. The values are considered similar when they have a different data type, accent, or case. For example, the following values are considered similar:

- Renee, RENEE, and Renée
- String "123" and integer 123

Less than (<)

Values in the left column are less than values in the right column.

Greater than (>)

Values in the left column are greater than values in the right column.

Less than or equal (<=)

Values in the left column are less than or equal to values in the right column.

Greater than or equal (>=)

Values in the left column are greater than or equal to values in the right column.

Less than and greater than (< >)

Values in the left and right columns are different.

=N

Values in the left and right columns are equal, even if both values are null.

The join operators are used when creating or editing relationships. For more information, see “[Creating a relationship manually](#)” on page 28.

Join optimization

You can optimize joins by applying filters to them.

A report might need a query that requires a relational join across multiple data sources. For example, a transaction database might be used to locate a set of customer details that are then joined to a corporate sales warehouse. Joins across different relational data sources can be performed through local query execution.

In IBM Cognos Analytics, you can optimize how these joins are executed by using filters. The query that is driving the join is executed, and a set of key values is gathered and then added to the query that is executed against the other data source. By extending the predicates (filter criteria) to the data source, the amount of local data processing that the join must perform is reduced. As a result, performance is improved significantly.

The following join filters are available:

No filtering

The optimization is turned off.

Unique values

Values from the table with 1 cardinality are used to filter values from the table with *many* cardinality. The filter uses a single IN expression. For a 1-to-1 relationship, the filtering is applied to the second table.

Range of values

Values from the table with 1 cardinality are used to filter values from the table with *many* cardinality. The filter uses a single BETWEEN expression, using the minimum and maximum values. For a 1-to-1 relationship, the filtering is applied to the second table.

Unique values in a subquery

Values from the table with 1 cardinality are used to filter values from the table with *many* cardinality. The filter is generated in a subquery. For a 1-to-1 relationship, the filtering is applied to the second table.

Unique or range of values

The cardinality is ignored, and values from the table on the left are used to filter values from the table on the right. This filter uses either IN or BETWEEN predicate. An error message is displayed if the optimization can't be applied.

If the data server types are different, ensure that the data types of the matched columns are compatible. Otherwise, you might need to edit the join expression to explicitly cast the data types.

In Cognos Analytics versions 11.1.6 and 11.1.7, a filter optimization error might occur. For more information, see “[XQE-PLN-0355 filter join optimization error](#)” in the *Troubleshooting Guide*.

Custom tables

Custom tables are created from other tables in the data module. These tables exist only in the data module. They are not added to the data sources that the module is based on.

By adding custom tables, you create a more abstract, business-oriented view of data in your data module. For more information, see “[Creating custom tables](#)” on page 33.

Custom tables function in the same way as other tables in the data module. You can use them to create relationships, calculations, filters, and other custom tables.

The **Custom tables** tab is used for creating, viewing, and managing custom tables in a data module. The tab is displayed even if the module doesn't contain any custom tables. To start creating a new table, click **Create custom table**.

If the data module already contains custom tables, the table names are listed when you open the tab. In the following example, the data module contains two custom tables: Product Sales and Product Returns.

Select a custom table to view its definition, or create a new table.

Create custom table

Product Sales

Product Returns

The different types of custom tables are identified by different icons, as shown in the following table:

View table	
Alias table	
Shortcut table	
Join table	
Union table	
Except table	
Intersect table	

From the custom table context menu , you can access the standard table options for creating a relationship, calculation, filter, and so on. Clicking the **View definition** option or the table name opens the table query flow diagram where you can view the data module tables that the custom table is built from. For example, the following diagram of a custom table named Branch_returns shows that the table is a view of three tables: Branches, Returns, and Products.



The diagram is synchronized with the data tree. You can perform the same actions from the table context menu in the diagram and in the data tree.

Creating custom tables

You create a custom table by merging, copying, or referencing selected tables and columns in the data module. These tables are not added to the data sources that the module is based on.

Custom tables can be used to create new custom tables.

About this task

If a custom table is based on multiple tables, the properties of the table that was selected first are applied to the custom table. For example, the **Usage** or **Data cache** properties on the custom table are inherited from the first table. The columns in the custom table also inherit their properties, such as **Label** or **Aggregate**, from the columns in the first table.

Procedure

1. In the data tree, from the data module, table (or multiple tables), or package context menu , click **New > Table**.

You can also start creating a custom table from the **Custom tables** tab by clicking **Create custom table**.

2. Select the tables and columns to add to your custom table.

You can click the add icon to add new tables, or the remove icon to remove tables from the list. These options are not available for a package.

- If you initiated the process for an individual table or multiple tables, the table names appear in the **Create new table** dialog box, in the **Selected tables** pane. The table that was selected first is at the top of the list.

- If you initiated the process from the data module context menu  , no table names appear in the **Selected tables** pane initially. Click **Select tables** to add tables that are already in the data module.
3. Depending on your selection of tables in the previous step, one or more of the following options are available to create the new table. Select one option.

View of tables

This option is available when one or more tables are selected. This is the only option that is available for package tables. The columns in the base tables are referenced in the view. The column properties in the view are independent from the base table. After the view is created, you can select or deselect columns to include in it.

Shortcut to a table

This option is available when one table is selected. The shortcut is a reference to the selected table, and has the following characteristics:

- The shortcut retains relationships of the base table. However, if you use the shortcut table in a new relationship, the new relationship overwrites the relationship inherited from the base table.
- The shortcut column properties depend on the base table, and are read-only.
- Calculations can be added to the shortcut.
- The shortcut doesn't inherit navigation paths from its base table. However, navigation paths can be added to the shortcut.
- Shortcuts can be used to create star schema grouping presentations. For more information, see "Model design and presentation" in the *IBM Cognos Analytics Metadata Modeling Guidelines*.

Alias of a table

This option is available when one table is selected. The alias is a reference to the selected table, and has the following characteristics:

- The alias is independent of its base table and doesn't retain the base table relationships. You can use the alias table in a new relationship.
- The column properties in the alias are dependent on the base table, and are read-only.
- Calculations and embedded filters can be added to the alias.
- The alias doesn't inherit navigation paths from its base table. However, navigation paths can be added to the alias.

Copy of a table

This option is available when one table is selected. You can copy all columns from the table, or only the columns that you select. The new table is disassociated from the base table so changes in one table aren't reflected in the other table. When you copy a non-custom table, the result is a non-custom table. When you copy a custom table, the result is a custom table.

Joined view

This option is available when two tables are selected. You can select or deselect columns to include in your new table.

Proceed to create a join between the two tables. For more information, see ["Creating a relationship manually" on page 28](#).

Union of tables

This option is available when two or more tables are selected. All selected tables must have the same number of columns. The columns must be in the same order, and their data types must be compatible. The new table includes all unique rows from all selected tables.

Tip: The help icon in the product provides information about incompatible data types when columns with such data types are discovered.

Intersect of tables

This option is available when two tables are selected. Both tables must have the same number of columns. The columns must be in the same order, and their data types must match. The new table includes only the unique rows that are shared between the two tables.

Except of tables

This option is available when two tables are selected. Both tables must have the same number of columns. The columns must be in the same order, and their data types must match. The new table includes only the unique rows from the first table.

Note: By default, duplicate rows of data are removed from the union, except, and intersect custom tables. To include the duplicate data, from the table context menu , click **Properties**, and under **Advanced**, set the **Duplicates** property to **Preserve**.

4. Proceed with the option-specific steps, and then click **Finish**.

The custom table diagram appears on the **Custom tables** tab.

Also, the table name is added to the list view on the **Custom tables** tab, and at the top of the data tree.

Note: Copies of non-custom tables are not considered custom tables and don't have query flow diagrams.

5. Save the data module.

What to do next

To view the custom table diagram later, from the table context menu  in the data tree, select **View definition**. The same option is available for the table from the list view on the **Custom tables** tab.

To edit some types of custom tables, select the related **Edit table** option from the table context menu . For more information, see [“Editing the view custom tables” on page 35](#) and [“Editing the union, except, and intersect custom tables” on page 36](#).

To change the custom table name, open the table **Properties** panel, and on the **General** tab, modify the **Label** property.

To delete a custom table, locate the table in the **Data module** panel, and from the table context menu , click **Remove**.

Editing the view custom tables

You can edit an existing view custom table by adding or removing tables and columns.

Procedure

1. In the data tree, the **Relationships** diagram, or on the **Custom tables** tab, locate the view custom table that you want to modify.
2. From the view context menu, click the **Edit table** (or **Edit table view**) option.
3. To add or remove columns from the view, follow these steps:
 - a) In the **Selected tables** panel, ensure that the view is switched to **Reference tables**.

Edit a view of tables

Select items to be included in the table.

Table name: Table - View

- b) Expand the tables, and select the columns to add or remove.
 - c) Click **Next**, then click **Finish**, and save the data module.
4. To add or remove tables from the view, follow these steps:
- a) Click the **Previous** button.
 - b) In the **Selected tables** panel, ensure that the view is switched to **All tables**.

Edit table

Create a custom table in the data module. This table is not added to your data source.

- c) Select the tables to add or remove.
- d) Click **Next**, then click **Finish**, and save the data module.

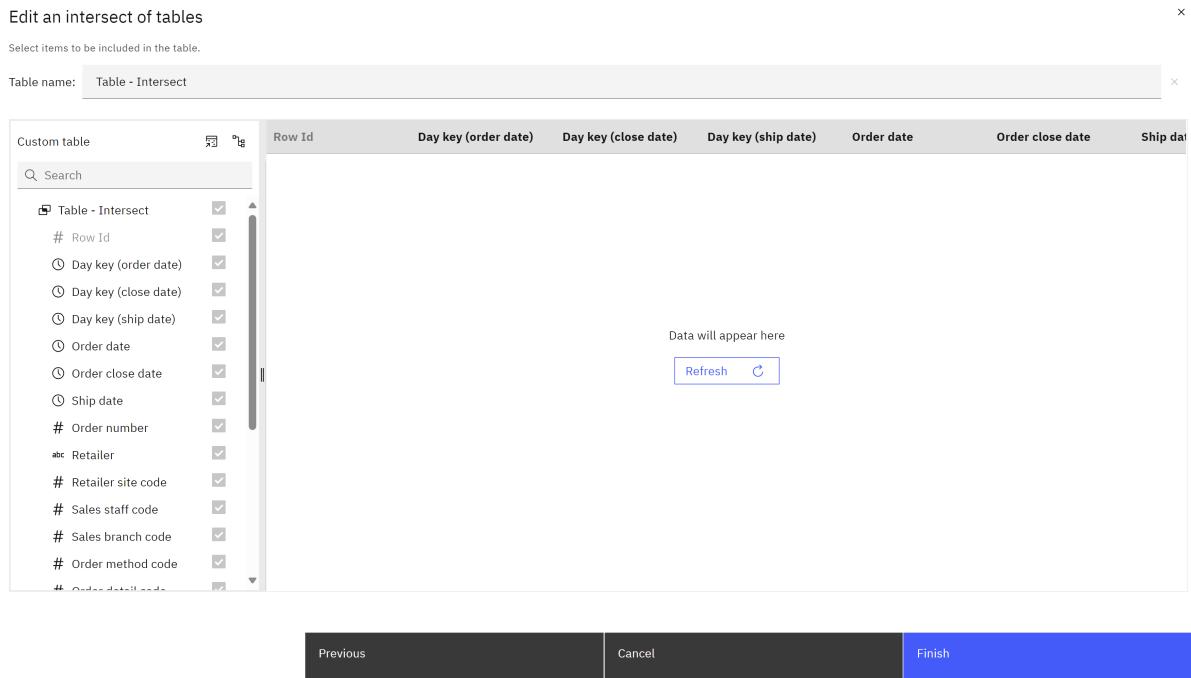
Editing the union, except, and intersect custom tables

You can add and remove tables from an existing union, except, or intersect custom table. You can also update columns in these three types of custom tables based on column changes in their underlying reference tables.

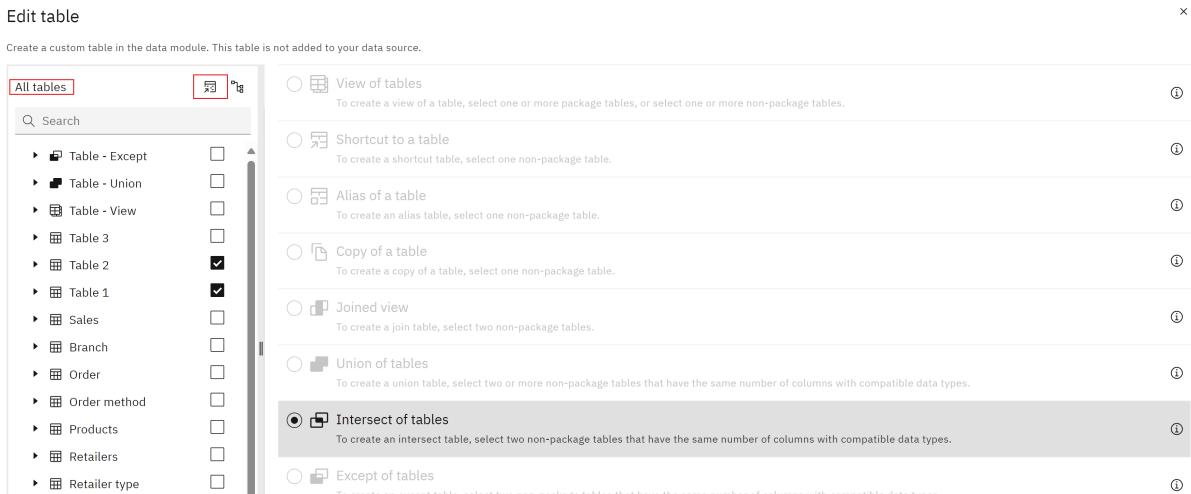
Adding and removing tables

Use the following steps to add or remove tables from a union, except, or intersect table.

1. In the data tree, the **Relationships** diagram, or on the **Custom tables** tab, locate the union, except, or intersect custom table that you want to modify.
2. From the union, except, or intersect table context menu, select the **Edit table** option.
3. In the table editor, click the **Previous** button to navigate to the **Selected tables** panel.



4. In the **Selected tables** panel, select or deselect the tables to add or remove from the custom table. You might need to toggle from **Selected tables** to **All tables** if the table that you want to add is not shown.

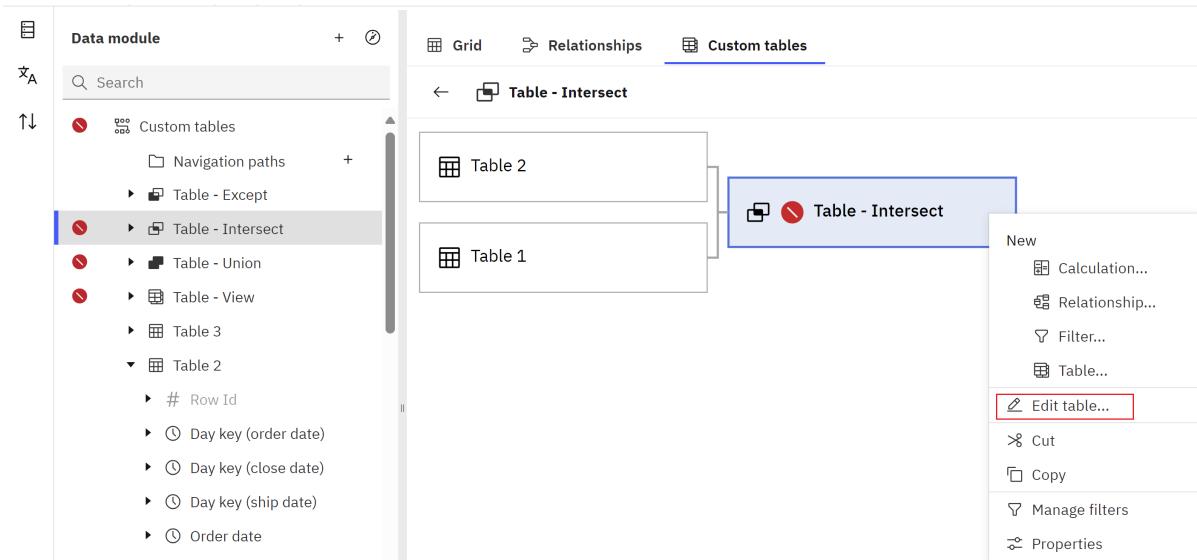


5. Click **Next**, then click **Finish** (in the preview panel), and save the data module.

Updating columns

You cannot add, remove, or modify columns directly in the union, except, or intersect custom table. However, you can modify the columns in its reference tables, and then update the columns in the custom table.

When the columns in the union, except, or intersect custom table differ from the columns in its reference tables, validation errors might be generated in the custom table.



To update the columns in the union, except, or intersect custom table, follow these steps:

1. In the applicable reference tables, add, remove, or modify columns as required.
2. In the data tree, the **Relationships** diagram, or on the **Custom tables** tab, for the union, except, or intersect table, click the **Edit table** option.

In the table editor that is displayed, the table is shown with its updated (added, removed, or modified) columns. Verify if the updates are correct.

3. Click **Finish**.

The errors are cleared, and the columns in the custom table are updated.

Note: If the reference tables do not adhere to the rules that are required to create a union, except, or intersect custom table, the **Select tables** panel is displayed so that the user can add or remove tables from the custom table. For more information, see [“Creating custom tables” on page 33](#).

Creating tables using SQL

You can create new tables in a data module that are based on a custom SQL syntax. The SQL is executed against a source that is already in the data module.

If the SQL validation is successful, the table is populated with a set of projected column names and rows of data.

The supported types of SQL include Cognos SQL, native SQL, and pass-through SQL. For more information, see [“Supported SQL types ” on page 61](#).

Procedure

1. From the data module context menu , select **Create table using SQL**.
2. In the table editor, type the table name.
3. From the **SQL type** drop-down menu, select the type of SQL to use.
4. From the **Source** drop-down menu, select the source to associate the table with. For data server connections, select the connection name. For other types of sources, select the source location, which is either **Team content** or **My content**.
5. In the **Expression** box, type or paste the SQL syntax for your table. The syntax is executed only against the source that you selected in the previous step.

The expression editor provides the following syntax validation and editing options:

- - Validate the syntax. You can validate the whole statement, or only selected segments of code.
 - - Preview columns and rows in your projected table. If the syntax is not correct, the columns are not displayed.
 - - View descriptions of functions, and examples of their usage.
 - - Insert the cursor anywhere in a line of code and select this button to comment out the entire line. To comment out multiple lines of code, select the lines and select this button. The comment string (--) is added at the beginning of each selected line.
- Tip:** To comment out sections of code, manually enclose the text between the following strings: /* and */
- - Apply formatting to the code.
 - - Use high-contrast mode.
 - Change the font size.
6. Click **OK** to save the table.

You can save the table even if it contains syntax errors, and edit the syntax later. However, you cannot modify any aspect of the SQL table, or view its data in the grid, until the table is successfully validated.

Results

The table name appears at the top of the data module tree. To edit the table SQL, from the table context menu , click **Edit SQL table**. You can also edit a column expression in an SQL-based table. However, subsequent updates to the original SQL statement might overwrite the updated expression.

What to do next

You can use and model SQL-based tables in the same way as other data module tables. For example, you can create relationships between this type of tables and other tables. You can also create calculations and navigation paths that include columns from these tables.

Converting referenced tables

You can convert referenced (linked) tables to different types of tables.

About this task

The **Convert** option is available for tables in data modules that are sourced from other data modules.

You can use the **Convert** option to break links to tables in the source data module. However, converting a referenced table to a shortcut or alias does not break the table link to the source module. For more information, see [“Using other data modules as data module sources” on page 9](#).

Note: In Cognos Analytics 11.2.3 and earlier versions, the **Break link** option was used to break the table link to its source table. Starting with version 11.2.4, the **Break link** option is replaced by the **Convert** option.

Procedure

1. In the data module tree, or on the **Relationships** or **Custom tables** tab, click the table context menu , and select **Convert**.

For example, when you select **Convert** for an alias table, the following conversion options are available:

Convert 'Product'

X

Choose a type of table to convert to.

 Shortcut table

Converts the alias table to a shortcut table.

(i)

 Table

Converts the alias table to the type of table that the alias table references.

 Warnings (1)

 Embedded filters in the table referenced by the alias table are not applied in the new table.

 View table

Converts the alias table to a view table.

(i)

Cancel

Convert

2. Select the type of table to convert to, and click **Convert**.

The icon on the converted table changes accordingly.

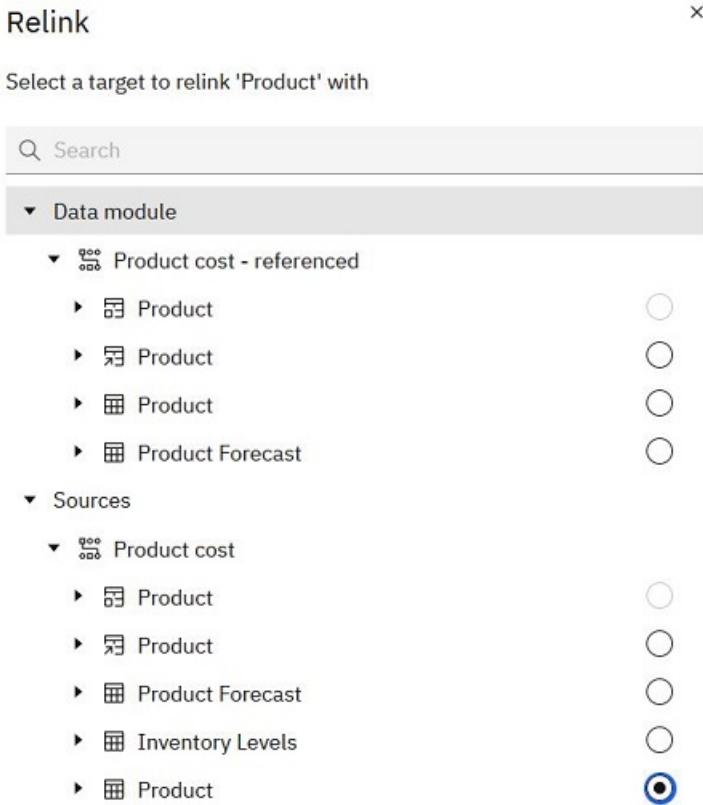
Relinking referenced tables

You can change the references of tables to point to different tables.

Procedure

1. In the data module tree, or on the **Relationships** or **Custom tables** tab, click the table context menu , and select **Relink**.

If the data module is based on another data module, the **Relink** dialog box includes the current data module and the source data module, as shown in the following image:



If the data module is based on another type of data source, such as data server or uploaded file, only the data module tree is available in the **Relink** dialog box.

2. Expand the data module tree, and if applicable, the source tree.
3. Select the table that you want to be the new target table, and click **Relink**.

The table reference is changed.

Column dependencies

Use the column dependency feature to clarify data granularity in a table or view to avoid double-counting of repeated values when data is aggregated.

Column dependencies are created automatically when the source tables are added to the data module. When a calculated column, which is based on columns that exist in the automatically generated column dependency, is created, the column is added to the generated column dependency. However, if a new column is pulled into the data module from the source data tree, the column is not added to the automatically generated column dependencies, and causes a validation error. In this case, the column dependency for the data module must be updated manually.

Note: Column dependencies are not inherited for custom tables. Any table object is considered independent, and if necessary, requires its own explicitly defined column dependencies to prevent double counting.

Column dependency is an equivalent of determinants in Framework Manager. However, column dependency provides more flexibility because you can specify more than one hierarchy per table, view, query subject, or data set. For more information, see "Determinants" in the *IBM Cognos Framework Manager* guide.

There are three common scenarios, described in the following sections, where double-counting can occur. In each scenario, you need to specify column dependencies to avoid double-counting. Remember to always check or review your column dependencies when you test for expected results.

Note: The terms *table*, *view*, *query subject*, and *data set* in this document all represent the same concept: a collection of data. Going forward, the term *table* is used in discussions about the implementation of column dependencies.

Scenario 1

In this scenario, a table contains replicated data (denormalized table).

For example, in the following table that contains Revenue at the day level and Sales target at the month level, the values for Sales target are repeated for each day of a month.

Year	Quarter key	Quarter (caption)	Month key	Month (caption)	Day key	Date	Revenue	Sales target
2010	20101	Q1 2010	201001	January 2010	20100112	Jan 12, 2010	39,036,796.4	57,628,100
2010	20101	Q1 2010	201001	January 2010	20100113	Jan 13, 2010	11,488,458.59	57,628,100
2010	20101	Q1 2010	201001	January 2010	20100114	Jan 14, 2010	3,232,160.48	57,628,100
2010	20101	Q1 2010	201001	January 2010	20100115	Jan 15, 2010	3,080,441.44	57,628,100
2010	20101	Q1 2010	201001	January 2010	20100116	Jan 16, 2010	1,976,896.69	57,628,100

Figure 1. Denormalized table that includes facts at different levels of granularity

Before column dependencies are applied, the Sales target total value is shown as 288,140,500, which is incorrect.

Year	Quarter key	Quarter (caption)	Month key	Month (caption)	Day key	Date	Revenue	Sales target
2010	20101	Q1 2010	201001	January 2010	20100112	Jan 12, 2010	39,036,796.4	57,628,100
2010	20101	Q1 2010	201001	January 2010	20100113	Jan 13, 2010	11,488,458.59	57,628,100
2010	20101	Q1 2010	201001	January 2010	20100114	Jan 14, 2010	3,232,160.48	57,628,100
2010	20101	Q1 2010	201001	January 2010	20100115	Jan 15, 2010	3,080,441.44	57,628,100
2010	20101	Q1 2010	201001	January 2010	20100116	Jan 16, 2010	1,976,896.69	57,628,100
Overall - Total							58,814,753.6	288,140,500

Figure 2. Total value before column dependencies are applied

After column dependencies are applied, the Sales target total value is shown as 57,628,100, which is correct.

Year	Quarter key	Quarter (caption)	Month key	Month (caption)	Day key	Date	Revenue	Sales target
2010	20101	Q1 2010	201001	January 2010	20100112	Jan 12, 2010	39,036,796.4	57,628,100
2010	20101	Q1 2010	201001	January 2010	20100113	Jan 13, 2010	11,488,458.59	57,628,100
2010	20101	Q1 2010	201001	January 2010	20100114	Jan 14, 2010	3,232,160.48	57,628,100
2010	20101	Q1 2010	201001	January 2010	20100115	Jan 15, 2010	3,080,441.44	57,628,100
2010	20101	Q1 2010	201001	January 2010	20100116	Jan 16, 2010	1,976,896.69	57,628,100
Overall - Total							58,814,753.6	57,628,100

Figure 3. Total value after column dependencies are applied

Scenario 2

This scenario involves joining on a key at a higher level of granularity in dimension tables. A dimension table on the 1..1 or 0..1 side of a relationship, containing attributes, is joined to a fact table on the 1..n or 0..n side of a relationship. The columns in the dimension table have repeating values.

In the following example, a Time dimension table that contains data for each calendar date is joined, by using the Month Key, to a fact table that contains the Sales Target data at the month level.

Year	Quarter Key	Quarter En	Month Key	Month En	Day Key	Day Date	Month Key	Sales Target
2010	20101	Q1	201001	January	20100101	Jan 1, 2010 12:00:00 AM		
2010	20101	Q1	201001	January	20100102	Jan 2, 2010 12:00:00 AM		
2010	20101	Q1	201001	January	20100103	Jan 3, 2010 12:00:00 AM		
2010	20101	Q1	201001	January	20100104	Jan 4, 2010 12:00:00 AM		
2010	20101	Q1	201001	January	20100105	Jan 5, 2010 12:00:00 AM		

Figure 4. Time table and Sales target table joined on the Month Key

When a query is created that displays the calendar date values together with the Sales Target data, the values for each month are returned for every date value. The result is an aggregated Sales Target summary value of 288,140,500, which is inflated by a factor equal to the number of days in the month.

Year	Quarter En	Month En	Day Date	Sales Target
2010	Q1	January	Jan 12, 2010 12:00:00 AM	57,628,100
2010	Q1	January	Jan 13, 2010 12:00:00 AM	57,628,100
2010	Q1	January	Jan 14, 2010 12:00:00 AM	57,628,100
2010	Q1	January	Jan 15, 2010 12:00:00 AM	57,628,100
2010	Q1	January	Jan 16, 2010 12:00:00 AM	57,628,100
Overall - Summary				288,140,500

Figure 5. Summary value before column dependencies are applied

After column dependencies are applied, the Sales Target summary value is shown as 57,628,100, which is the expected value.

Year	Quarter En	Month En	Day Date	Sales Target
2010	Q1	January	Jan 12, 2010 12:00:00 AM	57,628,100
2010	Q1	January	Jan 13, 2010 12:00:00 AM	57,628,100
2010	Q1	January	Jan 14, 2010 12:00:00 AM	57,628,100
2010	Q1	January	Jan 15, 2010 12:00:00 AM	57,628,100
2010	Q1	January	Jan 16, 2010 12:00:00 AM	57,628,100
Overall - Summary				57,628,100

Figure 6. Summary value after column dependencies are applied

Scenario 3

In this scenario, measures in a dimension table are involved.

In the following example, the table Employee Training dim (on the left) contains the measures Course Cost and Course Days. The same measures exist in the Employee Training Fact table. The Employee Training dim table is joined to the Employee Training Fact table on the Training Key field. The Employee Dim table (on the right), which introduces more granularity, is joined to the Employee Training Fact table on the Employee Key field. When queries are created that are based on these three tables, there is a danger of double-counting the measures in the Employee Training dim table.

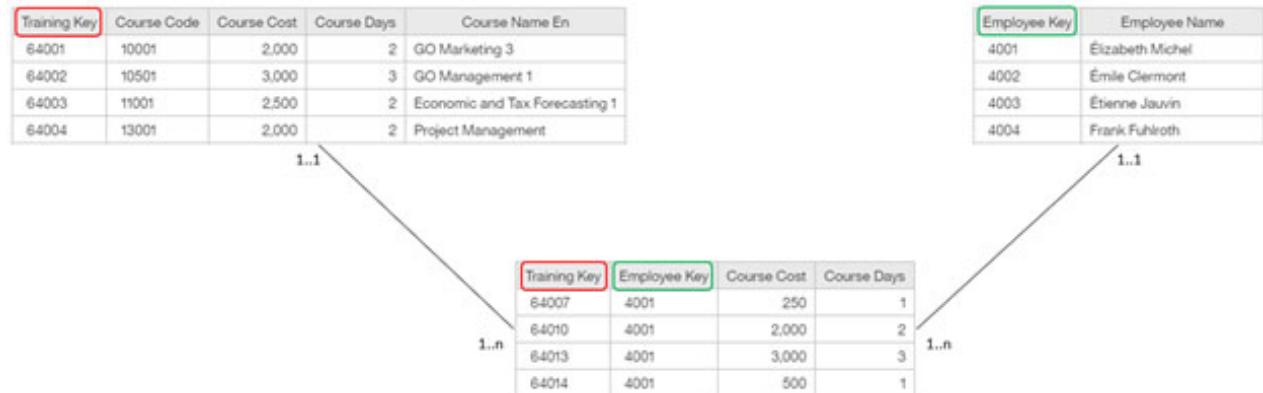


Figure 7. Relationships between the dimension and fact tables

When querying across all three tables, you might want to see the Course Cost and Course Days measures aggregated for both the course granularity and the employee granularity. The following report output shows how you can achieve this effect by using the column dependencies feature.

Course Name En	Employee Name	Course Cost	Course Days	Course Cost	Course Days
GO Communication	Aaltje Hansen	500	1	500	1
	Bayard Lopes	500	1	500	1
	Chiyo Yoshida	500	1	500	1
GO Communication - Summary		500	1	1,500	3
GO Orientation	Aaltje Hansen	250	1	250	1
	Bayard Lopes	250	1	250	1
	Chiyo Yoshida	250	1	250	1
GO Orientation - Summary		250	1	750	3
Overall - Summary		750	2	2,250	6

Figure 8. Report output with measures that are aggregated at different granularity levels

With the column dependencies defined, the first Course Cost and Course Days measures are not double-counted in the Course Name En summary row. However, the second instances of these measures, from the Employee Training Fact table, are aggregated for each employee within the Course Name En grouping. In the Overall - Summary row, the Course Cost for the two courses is shown as 750, and the revenue that was generated by all the students who took the course as 2,250.

Defining column dependencies

Define column dependencies to ensure that the fact data is aggregated correctly based on the keys or attributes of those keys that are used in the query.

The column dependency groups are related to each other in a hierarchy group in an order from coarse to fine granularity.

Tip: An attribute is a column that has the **Usage** property set to **Attribute** or **Identifier**. A fact is a column that has the **Usage** property set to **Measure**.

About this task

You do not need to specify column dependencies for all tables. Do it only when double-counting would take place. Your decision to specify column dependencies affects other Cognos Analytics components, such as reports or dashboards.

Procedure

1. Open an existing data module or create a new module that is based on a source that contains tables with repeated data at different levels of granularity.
2. Identify the attribute and fact columns that might cause double-counting. For example, the table can contain data at the month, quarter, and year level.
Verify whether column properties and data formats are properly specified. For example, change the **Usage** property to **Attribute**, or assign the data format of **Currency** to fact (measure) columns. Save the data module.
3. To see how the data is aggregated before column dependency is specified, you can create a report that is based on your saved data module. Later, you can use this report to verify the effect of applying column dependency on data aggregation.
4. From the table context (right-click) menu ..., select **Specify column dependencies**.

The **Column dependencies**  view is opened.

5. Drag the attribute columns that you identified in step 2, such as Year, Quarter, Month, and Day from the **Data module** panel to the **Column dependencies** view.
6. Click the group icon , and draw a line from the highest level attribute to the left of the next level attribute. Group the columns in a logical order to create a hierarchy group.
Repeat this action for each level until the hierarchy is complete from coarse to fine granularity.
7. Drag any related attributes or measures, such as Quarter (caption), Month (caption), Sales target, and Date and Revenue, inside the related attribute area.

Note: Each column from the table must be in one group. Otherwise, validation warnings are shown.

You can view the groups in the **Horizontal view** or **Vertical view**.

8. Verify, and if needed, modify the column dependency settings. For more information, see [“Configuring column dependencies” on page 47](#).
9. Save the data module.

Results

The following scenario 1 example shows how to group hierarchy columns and add their attributes in a denormalized table that includes facts at different levels of granularity.



Figure 9. Example of column dependencies for scenario 1

The following scenario 2 example shows column dependencies in a Time dimension table that can relate to multiple fact tables at different levels of granularity. You can see how columns in the Time dimension can be grouped, and their attributes added. In this case, the Time dimension is joined to the Sales Target fact table on the Month Key, and to the Sales fact table on the Day Key. The hierarchy for the Time dimension table is specified and configured to prevent double counting when a query includes the Day Key level and the Sales Target fact, which would repeat for every day in the month.

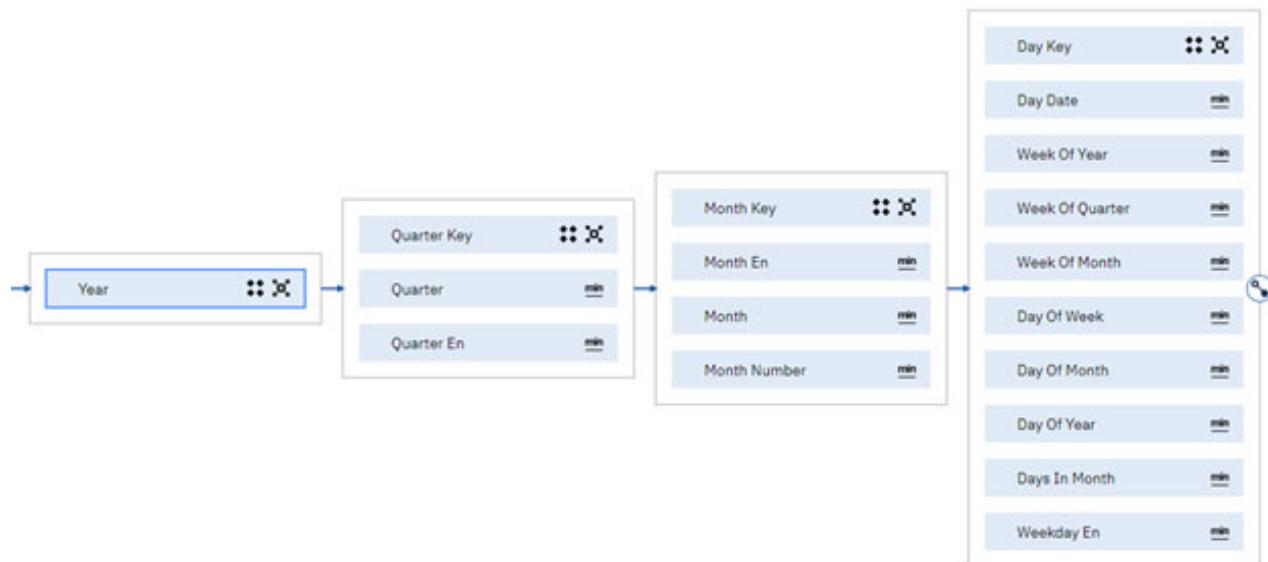


Figure 10. Example of column dependencies for scenario 2

The following scenario 3 example shows how column dependencies can be defined when a dimension table contains measures. The column dependency is built around the unique Training Key column. All other columns are nested underneath as attributes of the Training Key column. In this case, there is no hierarchy in the dimension data so only one dependency group is defined.

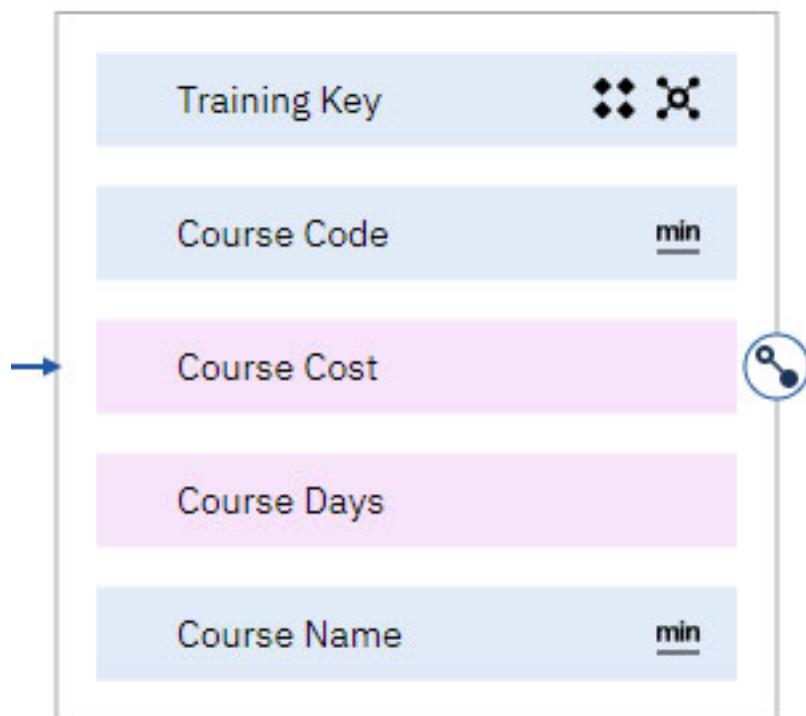


Figure 11. Example of column dependencies for scenario 3

Configuring column dependencies

After you define a column dependency group and a hierarchy group, you can configure the column dependency settings for individual columns.

The following configuration settings are available:

- **Unique** or **Repeating**

This setting specifies whether each row value is unique or repeating. Typically, all levels except for the lowest level in a hierarchy have repeating keys. Unique means that the key doesn't repeat for any row in the data.

- **Dependent** or **Independent**

This setting specifies whether the parent level value is required to identify the key of the current level. For example, a month key that is defined as a number in the range 1 - 12, requires the parent level keys to identify which year and quarter the key belongs to. Conversely, a month key that is defined as 20190101 doesn't require the parent keys to identify it because the month (01), quarter (01), and year (2019) values are included in the key.

- **Minimum** , **Maximum** , and **Group by** or **Average**

This setting specifies if the SQL must be generated with the Min, Max, Avg, or Group by clause when aggregating the data. **Minimum** is the default setting. Use **Minimum**, **Maximum**, or **Average** for data attributes where there is more than one value for a particular key. For example, the key value of YOW might have the airport name values of Macdonald Cartier Airport, Ottawa International Airport, Ottawa/Macdonald-Cartier International Airport, or Macdonald-Cartier International Airport. In this case, select the **Minimum** or **Maximum** setting to prevent double-counting.

When the data attributes don't repeat, which means that they are consistent throughout the data for each key, the **Group by** setting can be used. This setting doesn't apply to measures.

For measures, you might want to use the **Average** setting when the numeric values are similar. For example, when the values are 1000001, 1000002, and 1000003.

If the column dependency is set to **Minimum** or **Maximum**, changing the column **Usage** property from identifier or attribute to measure, or the opposite, doesn't affect the column dependency. However, if the column dependency is set to **Group by** (identifiers or attributes) or **Average** (measures) changing this property sets the column dependency to **Minimum**.

Procedure

1. Open the table **Column dependencies** view.
2. Inside the columns, click the icons that represent the different column dependency settings, and adjust the settings as required. For example, click the **Unique** or **Repeating** setting icon.
3. Save the data module.
4. Optional: To see how the data is aggregated after column dependency is specified, you can create a report that is based on your saved data module. Compare the aggregated results with the report from step 3 in the topic [“Defining column dependencies”](#) on page 44.

Results

The following [scenario 1](#) example shows how to configure columns in a denormalized table. In this case, all keys have repeating values except for Day Key, which has a unique value for every row of data. Each key is independent , and doesn't require the parent level key to identify it. Finally, all attributes for each column dependency group are set to **Group by** because the values are consistent.

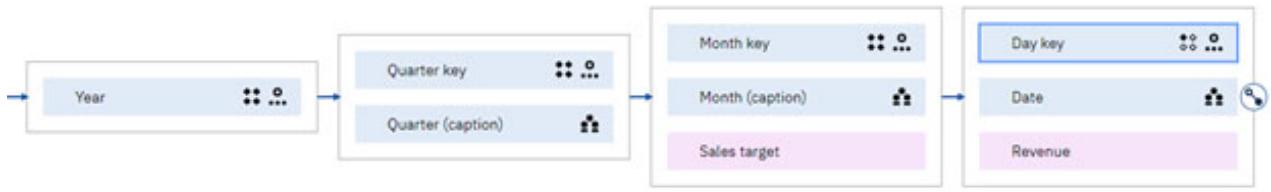


Figure 12. Example of column dependencies configuration for scenario 1

The following [scenario 2](#) example shows how columns in a Time dimension can be configured. In this case, all keys have repeating values except the Day Key which is unique for every row in the data. The Quarter Key and Month Key values can't be identified without the Year key level. Quarters and months are represented by numbers in the range 1 - 4 and 1 - 12. Therefore, these columns must be set to **Dependent**. Year and Day Key are set to **Independent** because their values can identify them. All the attribute values are consistent except Month En so they are set to **Group by**. However, Month En has values such as August, Aug., Aug, and August 08 so it must be set to **Minimum**.



Figure 13. Example of column dependencies configuration for scenario 2

The following [scenario 3](#) example shows how to configure columns in a dimension table that contains measures. The Training Key is unique and independent. The Course Code and Course Name values are all consistent, and can be set to **Group by**.

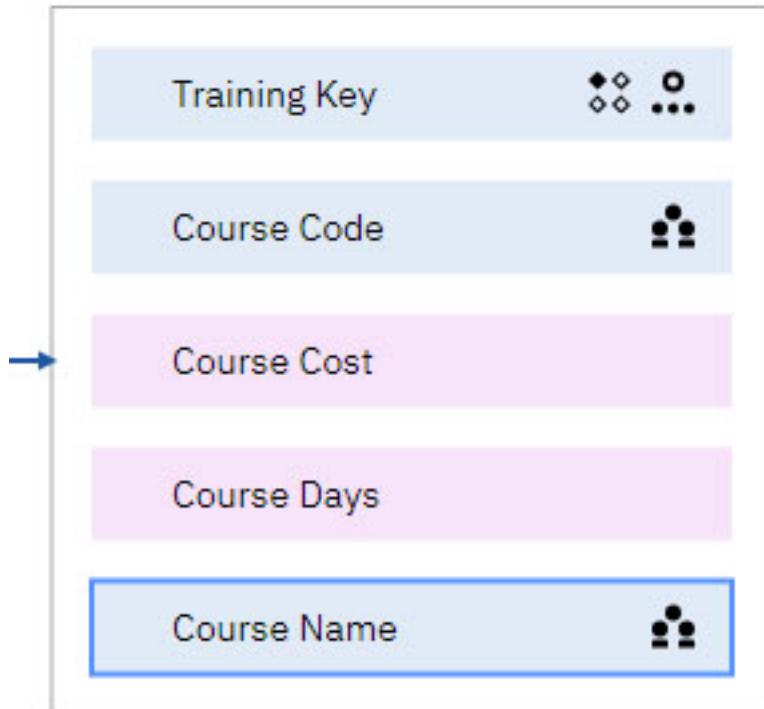


Figure 14. Example of column dependencies configuration for scenario 3

Calculations

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

Data module calculations are available to all Cognos Analytics components, explorations, dashboards, stories, or reports, that use the data module as a source.

Data modules support the following types of calculations:

- Stand-alone calculations

These types of calculations, also referred to as global calculations, reside outside of a table in a data module. Stand-alone calculations can refer to columns in any table in the module. You can move a stand-alone calculation into a folder to better organize items in the data module.

- Embedded calculations

These types of calculations, also referred to as table calculations, reside within a table in a data module. Embedded calculations can refer only to columns in the same table.

You can create basic arithmetic calculations, and more complex, custom calculations.

Creating basic calculations

You can create basic arithmetic calculations for columns with numeric data types.

About this task

The expression for these calculations is predefined, and you need only to select the proper mathematical operator, and in some cases, a constant value. For example, you can create a column Revenue by multiplying values for Quantity and Unit price.

Procedure

1. In the data module tree, right-click one or two columns that you want to use in the calculation.

For calculations that are based on two columns, use control-click to select the columns. The columns can be in two different tables. The column that was selected first is used as the first operand in the calculation.

You can also create the calculations at the folder level.

2. Click **Create calculation**.
3. Specify the calculation name by overwriting the automatically generated name.
4. For **Expression**, if your calculation is based on one column, choose the required operator and specify the constant value.

If your calculation is based on two columns, ensure that the order of columns is correct, and choose the required operator. If the order of columns is incorrect, click **Cancel**, and start the calculation again by selecting the columns in the proper order.

5. Leave clear or select the **Calculate after aggregation** checkbox.

When you leave this checkbox clear, the calculation is performed on the column values before they are aggregated. If you select this checkbox, the calculation is performed after the values are aggregated. The calculation results might be different in each case.

For more information, see the "Aggregation and the order of operations" in the *IBM Cognos Analytics Metadata Modeling Guidelines*.

6. Click **OK** to finish the calculation, or click **Use calculation editor** to view the calculation expression, or to modify it.

Results

A calculation based on one column is created as an embedded calculation. The calculation is added at the top of the list of columns in the table. You can move it to a folder inside the same table.

A calculation based on two columns can be created as an embedded or stand-alone calculation. If the columns are from the same table, an embedded calculation is created. It is added at the top of the list of columns in the table. If the columns are from different tables, a stand-alone calculation is created. It is added at the top of the data module tree, outside of any table. You can move this calculation to a module-level folder.

What to do next

The calculated column can be used in the same way as other columns in the table. From its context menu , you can access the different actions to edit the calculation, format its data, or remove it.

The calculation can be used in reports, dashboards, explorations, and other Cognos Analytics content.

Creating custom calculations

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

The expression editor provides functions and operators to create your expressions. It also includes function examples and documentation, and validates the expressions. For more information about using the expression editor, see [this blog](#) in the product community.

About this task

Custom calculations can be based on one column or multiple columns from different tables.

Procedure

1. In the data module tree, right-click the data module name, a table name, or a folder name, and click **Calculation**.

Note: You can also create calculations from the **Relationships** or **Custom tables** view.

- In the expression editor that is displayed, specify the calculation name by overwriting the automatically generated name.
- In the **Expression** box, define the expression for your calculation by using the expression editor capabilities.
 - 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. For example, the following expression can be used to concatenate a person's first and last name and create a calculated column called Name:

```
concat ( [Sales target (query)].[Sales staff].[First name],  
[Sales target (query)].[Sales staff].[Last name] )
```

- To add table columns to your expression, drag-and-drop one or more columns from the data module tree to the **Expression** box. 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.

- Click **Validate** to check if the expression is valid. The calculation will be created even if the expression is not valid, but it won't be usable.
- Leave clear or select the **Calculate after aggregation** checkbox.

When you leave this checkbox clear, the calculation is performed on the column values before they are aggregated. If you select this checkbox, the calculation is performed after the values are aggregated. The calculation results might be different in each case.

For more information, see the "Aggregation and the order of operations" in the *IBM Cognos Analytics Metadata Modeling Guidelines*.

- Click **OK**.

Results

A calculation based on one column is created as an embedded calculation. The calculation is added at the top of the list of columns in the table. You can move it to a folder inside the same table.

A calculation based on two or more columns can be created as an embedded or stand-alone calculation. If the columns are from the same table, an embedded calculation is created. It is added at the top of the list of columns in the table. If the columns are from different tables, a stand-alone calculation is created. It is added at the top of the data module tree, outside of any table. You can move this calculation to a module-level folder.

What to do next

The calculated column can be used in the same way as other columns in the table. From its context menu , you can access the different actions to edit the calculation, format its data, or remove it.

The calculation can be used in reports, dashboards, explorations, and other Cognos Analytics content.

Filters

A filter specifies the conditions that rows must meet to be retrieved from tables when the data module is used with reports, dashboards, explorations, and other Cognos Analytics content.

Data modules support the following types of filters:

- Embedded filters.

These filters are always applied to a table when the data module is used in dashboards, reports, explorations, and so on.

- Selectable filters.

These filters are created as selectable items in the data module tree, inside a table or at the root of the data module. The users can decide whether to apply these filters to dashboards, reports, explorations, and so on.

Security filters are a different type of filters that are specified at the source level. For more information, see [“Securing data” on page 24](#).

Creating embedded filters

Create embedded filters to customize the view of data in the data module for specific use cases.

For example, you can filter out data that is irrelevant for certain geographies, time periods, product lines, and so on.

About this task

Embedded filters can be created for a single column, or for multiple columns in a table.

Procedure

1. In the data module tree, locate the table that you want to add filters to, and choose one of the following options:
 - To create a filter for a single column in the table, from the column context menu , click **Filter**.
 - To create filters for multiple columns in a table, from the table context menu, click **Manage filters**. On the **Filters** tab, select the column for which you want to create the filter, and click **Add a filter**.
The option **via expression editor** allows you to create a filter by using the expression editor. Specify the filter name, type its expression, and click **OK**. Next, you can either continue adding filters by using this option, or proceed to step 2.
2. Specify the filter values. The options to select values depend on the column data type.
 - For columns with integer and decimal data types, you have two options to specify the values:
Range and **Individual items**. When you choose **Range**, you can use the slider to specify the range values, or type the range start and end values. When you choose **Individual items**, select the check boxes that are associated with the values.
Tip: You can enter decimal values only for columns with the decimal data type. When you try to enter a decimal value for a column with the integer data type, the decimal separator is cleared.
 - 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.
 - For columns with text data types, select the check boxes that are associated with the values.To select values that are outside the range that you specified, click **Invert**.
3. Click **OK** to save the filter.
Please note that the effect of the filter is not automatically shown for the members in the data tree. To see that effect, click the **Refresh members** action on the related column in the data tree.
4. If you used the **Manage filters** option, you can continue creating filters for other columns in the table. The filters that you created are listed on the **Filters** tab.

Results

After an embedded filter is created, the filter icon  appears next to the column and table names in the data module panel, diagram, and expression editor.

The filter icon on a table indicates that the table contains at least one embedded filter. All embedded filters that a table contains are listed in the table properties, on the **Filters** tab.

What to do next

To edit a filter on a single column, from the column context menu  click **Filter**, and modify the filter conditions. To remove the filter, click **Clear all**.

To edit or remove embedded filters for a table, from the table context menu  click **Manage filters**. On the **Filters** tab, view or edit the filters, or remove them.

Creating selectable filters

Create selectable filters when you want to give users the choice of applying the filters in dashboards, reports, explorations, and so on.

The filters suggest possible options to filter data in the data module, but the users can also view unfiltered data.

About this task

Selectable filters can be created inside a table or outside a table (at the root of the data module).

Note: You cannot insert data module tables that contain both columns and filters into the report. A warning **Unable to insert an object in the selected location** will be displayed. Instead, select the columns and add them to the report, then select the filters and add them to the report.

Procedure

1. Open a data module and decide what type of a selectable filter you want to create.
 - To create a filter inside a table, from the table context menu  click **Filter**.
The **Filter** option is also available for a folder inside the table, and for the table in the diagram.
 - To create a filter outside a table, from the data module context menu  click **Filter**.
The **Filter** option is also available for a folder at the root of the data module.
The expression editor is displayed.
2. Type the filter name.
3. In the **Expression** panel, provide the filter expression by using the expression editor capabilities.
4. Click **OK**

Results

The filter is added as a stand-alone entry in the data module tree with the filter icon  before the filter name. Filters created at the folder-level are added to the applicable folder.

What to do next

To edit the filter, from the filter context menu  click **Edit filter**. To remove the filter, from the filter context menu  click **Remove**.

Hiding items

You can hide items in a data module, such as tables, columns, uploaded files, and folders.

The hidden items are not visible in the metadata tree in reports, dashboards, stories, and explorations, but are fully functional in the product.

About this task

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

The following read-only items can't be hidden by default:

- Packages.

To work around this issue, create a folder in the data module tree, and move your package with all its content into that folder. Then, hide the folder.

- Tables that are linked to the source data module, which is indicated by the linked table icon .

To work around this issue, in the data module tree, from the table context menu ..., select **Break link**. The link is broken, and you can now hide the table.

Procedure

1. In the data module tree, click the context menu icon  for an item such as a table or column, and click **Hide from users**.

You can also select multiple items to hide at once.

Tip: To show the hidden items, click the context menu  for the hidden item, and click **Show**.

2. Save the data module.

Results

The labels on the hidden items are grayed out in the data module tree and in the diagram. Also, on the **General** tab of the item properties, the **Hide from users** property is selected.

The hidden items are not visible in the metadata tree in reports, dashboards, stories, and explorations. In addition, hidden items can't be selected for use in new visualizations, but are fully functional in existing visualizations that utilize them.

Creating data groups

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

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.

About this task

Depending on the column data type, you can create data groups by using the following styles:

- Numeric style.

Each data group includes a range of values. By default, this style is available for columns with numeric data types. However, while you create the data group, you can switch to the text style, and continue switching between the two styles until you save the group.

- Text style.

Each data group includes individual values. By default, this style is available for columns with text data types. For columns with numeric data types, you can switch to this style when you create a data group.

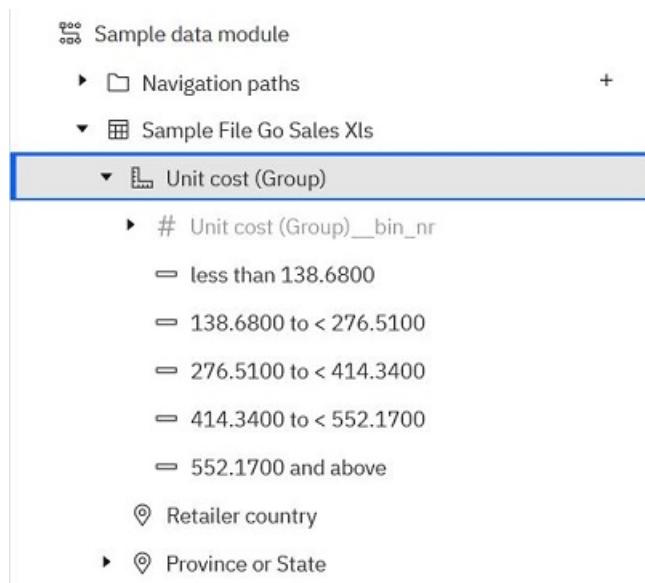
For stand-alone (global) calculations, data grouping is available only when the **Aggregate** property on the calculated column is set to **None**, and the data type is numeric or string.

Note: For columns that contain large number of items, it might be more efficient to define conditional groups manually by using the expression editor.

Procedure

1. In the data module tree, right-click the column where you want to group data, and click **Create data group**.
2. If you selected a numeric column, set the groups in the following way:
 - a) In the **Name** field, specify a name for the grouped column.
 - b) From the **Groups** menu, select the number of groups that you want to create. Each group is automatically assigned an equal number of values. When you change the number of groups, the values are dynamically redistributed, and the range values are set.
 - c) In the **Group names** column, replace the automatically generated labels with meaningful names. However, if you change the number of groups, the custom labels are cleared, and the automatically generated labels are restored for each group.
 - d) If needed, manually lock the **Range border values** for each group. The **Higher** and **Lower** range border values can be changed to numeric inputs. To return to the equal distribution of values, click the **Reset distribution**  icon.
 - e) To create a group for NULL values, select the **Group NULL values as** checkbox, and type a name for the group.
 - f) If you want to switch to the text style, click **Create a data group (text style)** and proceed to step 3. You can continue switching between the **Create a data group (numeric style)** and **Create a data group (text style)** dialog boxes until you click **Create**.
 - g) Click **Create**.

The data group appears in the data tree, at the top of the list of columns in the table. In the following example, it's the `Unit cost(Group)` column.

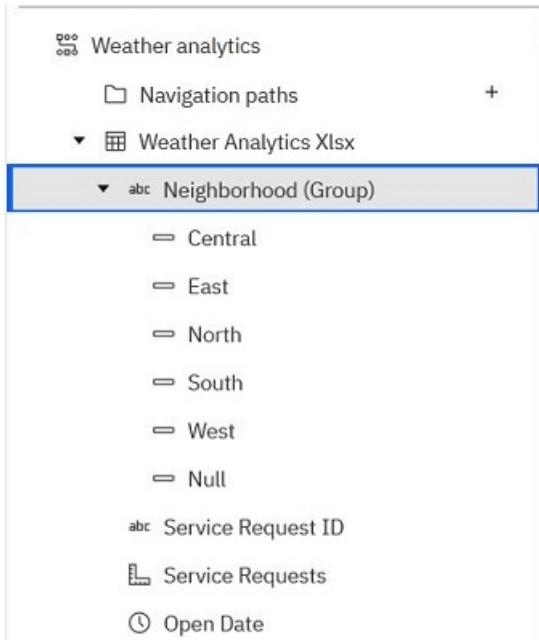


The data group view in the data tree also includes a column that represents the number of groups (`Unit cost(Group)_bin_nr`). This column is hidden by default, but you can edit its properties, including the label and expression. If you change the label on this column (for example, to `Unit cost - groups`), and then edit the data group through the **Edit data group** option, the column label is changed back to its default value (`Unit cost(Group)_bin_nr`).

If you selected the **Replace the existing column** checkbox, the grouped column replaces the original column in the table.

3. If you selected a text column, set the groups in the following way:
 - a) In the **Name** field, specify a name for the grouped column.
 - b) In the **Groups** box, click **New group**, and type a group name.
 - c) In the **Remaining items in column** box, control-select the values to include in the group, and click the **Add to group** icon →. The values are moved to the **Group items** box.
Note: The maximum number of items that can be preloaded in the **Remaining items in column** box is 32000. You can select from these items to add them to the groups that you define.
 - d) Repeat steps b to c to create more groups.
 - e) Optional: To create a group that contains all of the values that aren't already included in any group, select the **Group remaining and future values in** checkbox, and type a name for the group.
 - f) Optional: To replace the original column in the table with the grouped column, select the **Replace the existing column** checkbox.
 - g) Click **OK**.

The data group appears in the data tree, at the top of the list of columns in the table. In the following example, it's the Neighborhood (Group) column.



If you selected the **Replace the existing column** checkbox, the grouped column replaces the original column in the table.

What to do next

To edit the data group, click its context menu icon ⋮, and select **Edit data group**. If you need to modify the group expression, go to **Properties**, and for the **Expression** property, click the **View or edit** link. After you modify the expression and save it, the **Edit data group** option is no longer available from the data group context menu. The only way to edit the data group in the future is through the **View or edit** link.

Note: You can't change the data group style (from numeric to text, or from text to numeric) by editing the data group.

Cleaning data

Data is often messy and inconsistent. You might want to clean 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

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**.

If you replace all column values with **Null**, the column data type remains unchanged. You need to change the data type manually to avoid errors when the values are used in expressions. For more information, see [step 4](#).

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**.

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**.

The "cleaned" values are displayed in the data module. This is the final step for most **Clean** options.

The expression editor automatically creates an expression for the modified columns. To view the expression, open the column properties panel, and for the **Expression** property, click **View or edit**.

4. If you used the **Replace this value with NULL** option to replace all column values, perform the following steps to change the column data type:

- a) Open the column properties panel, and for the **Expression** property, click **View or edit**.

- b) View the expression for the cleaned column. The expression is similar to this one:

```
nullif (
    i2000_YR2000
    ;
)
```

The column **Data type** property in this example is **Text**.

- c) Use the **cast** function on the expression to convert the values in the column to the required data type, as shown in the following example:

```
cast(nullif (
    i2000_YR2000
    ;
),
integer)
```

Tip: To view the documentation about the **cast** function, click the function name in the expression, and then click the information icon  in the toolbar. The function description is displayed in the **Information** panel.

- d) Click **OK** to save the expression.

The column **Data type** property is now changed to the required type. In the example provided in this step, it's the **Integer** data type.

The values in each column now show **Null**.

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 replace **NULL** with the empty string.

Creating navigation paths

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

When a data module contains navigation paths, the dashboard or report 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 path by either following the order of columns in the navigation path, or by choosing the column to which they want to proceed.

A data module can have multiple navigation paths. The navigation paths are saved in the **Navigation paths** folder at the top of the data module tree. This folder is created by default when the data module is created.

Note: Navigation paths in a data module are not available when creating a union, except, intersect, join, or custom SQL query in Reporting. For the navigation paths to be available, these types of queries must be created directly in the data module. For more information, see “[Custom tables](#)” on page 31 and “[Creating tables using SQL](#)” on page 38.

About this task

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

Columns from different tables can be added to a navigation path. The same column can be added to multiple navigation paths.

Procedure

- At the top of the data module tree, find the **Navigation paths** folder, and click the plus icon  next to it.

The **Create navigation path** dialog box is displayed.

- From the data module tree, select the columns that you want to include in the navigation path. Drag or double-click the columns to add them to the navigation path.

- To add different columns, drag the columns from the data module tree to the navigation path. 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 path name, overwrite the existing name.

The default name of the navigation path includes names of the first and last columns in the path. The default name reacts to the changed order of columns. You can change this name. If you overwrite the default name, it does no longer change when you modify the navigation path definition. The name cannot be blank.

- Click **OK**.

The new navigation path is added within the **Navigation paths** folder.

- To modify a navigation path, expand the **Navigation paths** folder, and locate the navigation path that you want to edit.

- From its context menu , click **Edit navigation path**.
- Make the required changes, and click **OK**.
- To remove a navigation patch, click the **Remove** action from the context menu .

- Save the data module to preserve the navigation path that you added.

Results

The navigation path is added to the **Navigation paths** folder in the data module tree. The following screen capture shows two navigation path **Retailer codes** and **Retailer address**.

The screenshot shows the Data module interface. At the top, there are tabs for 'Grid' (which is selected) and 'Relationships'. Below the tabs is a search bar with the placeholder 'Search'. The main area displays a tree view of navigation paths. Under 'Navigation paths', there are several items: 'Retailer codes', 'Retailer address', 'Retailer Site', 'Retailer', and 'Retailer Site Mb'. A context menu is open over the 'Retailer codes' item, containing three options: 'Edit navigation path...', 'Remove', and 'Properties'. The 'Edit navigation path...' option is highlighted with a blue border.

You can expand each navigation path to view the columns that it contains.

If you select the icon **Identify navigation path members**  in the data module toolbar, the columns that are members of any navigation path are underlined.

The navigation path is also added to the **Navigation paths** tab in the data module properties. This is the alternative user interface to work with the navigation paths.

The screenshot shows the 'Properties' dialog box. At the top, there is a close button 'X'. Below it, there are three tabs: 'General', 'Relationships', and 'Navigation paths' (which is selected). The 'Navigation paths' tab lists two navigation paths: 'Navigation path (5 columns)' (underlined text 'Retailer codes') and 'Navigation path (3 columns)' (underlined text 'Retailer address'). Each entry has a minus sign '-' to its right. At the bottom of the list is a button 'Add a navigation path' with a plus sign '+'. The entire dialog has a light gray background with white text.

Formatting data

The column data format specifies how column values are displayed in dashboards, stories, explorations, or reports. You can choose a format type such as text, percent, or currency.

Uploaded Microsoft Excel spreadsheets retain the column data formats that were defined in Excel. These formats are set as default data formats in the base data modules that are created from these spreadsheets.

Each format type contains properties that further specify how the data is displayed. Initially, the properties are set based on the format options that are returned from the source. If no option is returned from the source, the property is set to the data module default.

About this task

Some characters are language-sensitive and display properly only when your locale supports the applicable font. For example, for Japanese currency symbols to display correctly, your locale must be set to Japanese.

You can define the format type for several columns at the same time if the columns contain the same type of data.

Procedure

1. In the data module tree, from a column context menu , click **Format data**.
2. In the **Data format** dialog box, select the appropriate **Format type**.

For example, you can select **Date**, **Time**, **Date/Time**, or other type.

If the data was not formatted in the source, the **Unformatted** type is assigned in the data module. If the source supplies a format, the format is reflected in the module.

The **Custom** format is an advanced option where you can supply your own format string and use it to format the data.

3. Specify properties for the selected format type.

For example, for the **Date** type, you can specify the **Date separator**, the **Date style**, and **Date ordering** properties. For the **Currency** type, you can specify the **Number of decimal places** property.

The **Default** setting is specific to the property that it refers to. You can view the default for each property by clicking the information icon  next to the property. For example, for the thousands separator the default value is inherited from the user's content language. For the decimal places default, if the property is not set, the number of decimal places vary depending on the number rendered.

Click **Advanced options** to view and specify more data format properties. Click **Reset properties** if you want to reset the properties to default values.

4. Click **OK** to apply the formatting.

SQL in Cognos Analytics

SQL is the industry-standard language for creating, updating, and querying relational database management systems.

Supported SQL types

IBM Cognos Analytics supports three types of SQL: Cognos SQL, native SQL, and pass-through SQL.

Cognos SQL

Cognos SQL is an implementation of the standard SQL. It works with all relational and tabular data sources. This is the optimal type of SQL for use with Cognos Analytics applications. Using Cognos SQL is preferable because you can have fewer database restrictions. Cognos SQL improves table performance by, for example, removing unused elements at query time.

Cognos SQL does not support non-standard SQL.

Native SQL

Native SQL uses vendor-specific SQL constructs. Use native SQL to pass the SQL statement that you enter to the database. Cognos Analytics might add statements to what you enter. A native SQL statement can reference only one data source.

This type of SQL must be completely self-contained. It can't reference anything outside that SQL, such as database prompts, variables, or native formatting that would normally be supplied by the calling application.

If you are comfortable working with a native SQL version, you can use keywords that are not available in Cognos SQL, and copy and paste SQL statements from other applications to Cognos Analytics.

This type of SQL might not work with a different data source.

Pass-Through SQL

Use pass-through SQL when the SQL statement that you enter is not valid inside a derived table, such as in a `With` or `OrderBy` clause. Generally, you should use pass-through SQL only if you must create a table that contains source-specific constructs.

Pass-through SQL lets you use native SQL without any of the restrictions that the data source imposes on subqueries (pass-through SQL tables are not processed as subqueries). Instead, the SQL for each table is sent directly to the data source where the query results are generated.

Performance is slower because each table is sent to the source as a separate statement rather than being optimized by Cognos Analytics. Therefore, when choosing between native SQL and pass-through SQL, you must decide which is more important: performance or using SQL that is not permitted in a subquery.

Pass-through SQL must be completely self-contained. It must not reference anything outside of that SQL, such as database prompts, variables, or native formatting that would normally be supplied by the calling application.

A pass-through SQL statement might not work with a different data source.

Showing the query information

Modelers can view and copy the query information (SQL) that was used to generate tables and relationships for SQL-based data sources.

The source types that this feature applies to include data servers, packages, and data modules that are based on data servers and packages.

The following types of query information are supported:

- **Cognos SQL**

An SQL statement that uses the IBM Cognos implementation of standard SQL.

- **Native SQL**

An SQL statement that uses vendor-specific SQL constructs.

- **Query response**

The complete response that contains all messages that were generated when the query was processed.

For more information, see [“Supported SQL types” on page 61](#).

Procedure

1. To view the table SQL, in the data tree, from the table context menu  select **Show query information**.
To view the relationship SQL, on the **Relationships** tab, right-click the relationship join in the diagram, and select **Show query information**.
2. By default, the query information is generated as **Cognos SQL**. From the **Query information type** menu, you can change the viewing options to **Native SQL** or **Query response**.

3. To copy the SQL to the clipboard, click the copy  icon.

Generating the query SQL

You can specify how Cognos Analytics generates the SQL that retrieves data from tables when queries run.

Use the **Item list** property on tables to specify the SQL generation type. Depending on the setting that you specify for this property, the generated query SQL includes all or only selected columns.

Procedure

1. In the table **Properties**, on the **General** tab, the **Advanced** section, locate the **Item list** property.
2. Select one of the following settings:

All

The generated SQL contains all columns in the table.

Used

The generated SQL contains only the columns that are used by the query directly, and the columns that are needed by the join.

Automatic

This is the default setting. It works identically as the **All** setting.

3. Save the data module.

Results

The query generation settings that you specify apply to queries in dashboards, reports, stories, and explorations.

Validating data modules

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

Validation identifies the following issues:

- 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.

About this task

Automatic and manual data module validation is available. Automatic validation is turned on by default. You can disable automatic validation, and manually start the validation when required.

With automatic validation, the results are refreshed after each update of the data module.

Procedure

1. In the application bar, or in the **Data module** panel, click the validation icon . You can also click **Validate** from the data module context menu.

The validation pane opens.

2. To enable or disable automatic validation, click the **Automatic** toggle switch. To validate the data module manually, click **Validate**.
3. Check for validation errors.

The failed validation icon  is displayed in the data module tree and in the diagram, next to the column where the error is discovered.

4. Click the error icons to view the number of errors and the details about the errors.

The errors are listed in the **Validation issues** panel.

5. View the error messages by clicking the **Details** link for each issue.

To copy the error messages to the clipboard, click the copy  icon.

6. Using the validation messages, try to resolve the errors.

You can save the data module with validation errors, and resolve the errors later.

Object properties

You can view and modify the data module, table, column, and folder properties in the modeling interface.

The properties can be accessed from the data module, table, column, or folder context menu , or by using the **Properties** icon  in the application bar.

Some properties, such as **Label** and **Identifier**, are available for all objects in the data module, while other properties are available only for certain types of objects. For example, **Member display list** is available only for data modules, **Item list** only for tables, and **Aggregate** only for columns.

You can view and modify the following properties on the **General** tab of the **Properties** pane.

Label

Specifies the item's name that is displayed in the user interface. This property applies to all items. You can't change the label for members. To change the label for a data module, save the module by using the **Save as** option.

Hide from users

Use this property to hide items, such as tables, columns, packages, or folders, in a data module. The hidden items are grayed out in the modeling interface, and not visible in other interfaces, such as reporting or dashboarding. For more information, see [“Hiding items” on page 53](#).

Expression

Shows the underlying expression for a column. Click **View or edit** to open the expression editor where you can modify the expression.

Usage

This property applies to columns, and specifies the intended use for the column data.

The initial property value is based on the type of data that the column represents in the source. You need to verify that the property is set correctly. For example, if you import a numeric column that participates in a relationship, the **Usage** property is set to **Identifier**. You can change this property.

The following **Usage** types are supported:

Identifier

Represents a column that is used to group or summarize data in a **Measure** column with which it has a relationship. It can also represent an index, date, or time column type. For example, **Invoice number**, or **Invoice date**.

Measure

Represents a column that contains numeric data that can be grouped or summarized, such as **Product Cost**.

Attribute

Represents a column that is not an **Identifier** or a **Measure**, such as **Description**.

Aggregate

This property applies to columns, and defines the type of aggregation that is applied to a summary column in a report or dashboard. For example, if the **Aggregate** property value of the **Quantity** column is **Total**, and the column is grouped by **Product Name** in a report, the **Quantity** column in the report shows the total quantity of each product. Aggregated data improves query performance and helps to retrieve data faster.

The default type of aggregation is inherited from the source. When modifying this property, you can select values that the source does not provide, such as average or maximum. To know which aggregate value is required, you must understand what your data represents. For example, if you aggregate **Part number**, the aggregate values that apply are count, count distinct, maximum, and minimum.

The following aggregation types are supported:

- None (no aggregation is set up for a column)
- Average
- Count
- Count distinct
- Maximum
- Minimum
- Total

Data type

The column data type is inherited from the source and can't be modified in the data module.

Represents

Use this property to specify whether a column includes the date or time, or geographic location type of data. This information is used in the reporting and dashboarding environments to suggest the most appropriate default visualizations, among other possibilities.

Geographic location

The values include **Continent**, **Sub Continent**, **Country**, **Region**, **State Province**, **County**, **City**, **Postal code**, **Street Address**, **Position**, **Latitude**, and **Longitude**.

Time

The values include **Date**, **Year**, **Quarter**, **Season**, **Month**, **Week**, **Day**, **Hour**, **Minute**, and **Second**.

Lookup reference

This column property is used to create a data module for relative date analysis. For more information, see [“Creating a data module for relative date analysis” on page 81](#).

Members

Use this column property to enable or disable displaying relational members in the data tree. The following settings are available:

Automatic

Members can be expanded in the data tree. Sorting is enabled, and members are sorted by the current column. Sort order is **Ascending**. This is the default setting.

Enabled

Members can be expanded in the data tree. You can select the column to sort by, and set the sort order to **Ascending** or **Descending**. The members that are shown in the data tree don't dynamically adjust to the changed sort order. Use the **Refresh members** action from the column context menu  for the sort order to be reflected in the data tree.

Disabled

Members can't be expanded in the data tree. Previously shown members are removed, and new members can't be loaded for the column.

For more information, see [“Displaying relational members” on page 72](#).

Members display limit

This data module property is used to specify the maximum number of members to load in the data tree nodes for each fetch request. For more information, see [“Setting the members display limits” on page 71](#).

Description

Use this property to specify optional information about the data module, table, column, or folder. The description is available in reports and dashboards that are based on the data module.

Comments

Use this property to specify optional information about the data module, table, column, or folder. Applies to all items in the data module. The comment is available only within the data module environment.

Screen tip

Use this property to specify an optional short description of the table or column. Applies to all items in the data module. The screen tip appears when you pause your pointer over the table or column name in the modeling, reporting, or dashboards environment.

Advanced properties

The following properties are specified on the **General** tab, **Advanced** section, of the **Properties** pane:

Identifier

This property uniquely identifies objects. It is used, either by itself or in conjunction with parent object identifiers, to generate SQL queries in expressions, reports, dashboards, and other objects. The property is created automatically for a data module and all its objects. For tables and columns, the property value is inherited from the data source. The property can be modified for all objects except for folders and the data module itself. To change the data module identifier, save the data module under a different name.

When changing this property, ensure that:

- The first character is a letter or an underscore (_).
- The subsequent characters are letters, digits, or underscores (_), without spaces.

On the source tables, you cannot change the column identifier because at least one instance of that identifier is needed as a reference to the data source. While tables capture this reference in the background, columns do not. If you want to change the identifier for a column, for example if you are trying to create a more cryptic identifier, the recommended approach is to make copies of the columns, hide the original columns, and rename the identifiers of the copies.

Technical data type

This property reflects how the column is defined in the database. For example, for a column with the **Data type** of Text, the **Technical data type** might be char(5), nvarchar(200), or varchar(10).

Usage

This property applies to tables. It controls how the query engine should understand and process the table, and its child objects, in a query. The **Usage** property has the following settings:

Automatic

This setting informs the query engine that the table is an ordinary table, and requires no special processing. This is the default setting.

Bridge

The bridge table is used to remove the many-to-many relationships between tables by setting the many side of the relationship to the bridge table. By default, the Cognos Analytics query engine understands tables that are at the many end of relationships to be fact tables. The bridge table is

not a fact table. So for the query engine to understand the role of the table and properly generate the query, the bridge table **Usage** property must be set to **Bridge**.

The bridge table can be defined either in the database or in the data module (or Framework Manager model). However, it is preferable to create bridge tables in the database.

Summary

This setting summarizes the values in the table. When the table is used in a report or dashboard, the data retrieved from the table is already summarized, columns with the **Usage** property set to **Measure** are aggregated, and all other columns are used as grouping columns.

Summary tables, such as unions, joined views, excepts, intersects, and SQL-based tables can be modeled in reports and data modules. It is more efficient to model these tables in the data module because they can be available for all reports and dashboards. The tables are modeled once, there is only one possible point of failure, and the generated numbers are consistent.

Item list

Use this property to specify the SQL generation type for a table. Depending on the setting for this property, the generated query SQL includes all or only selected columns. This property applies to tables only. For more information, see [“Generating the query SQL” on page 63](#).

Data cache

Use this property to enable data caching and specify the cache expiry options for tables in the data module. For more information, see [“Setting up data caching” on page 22](#).

Source

This property applies to all objects in the data module. For a table or column, it shows the source name and path.

Supports NULL values

Specifies whether a column supports null values. By default, this property value is inherited from the source. You can change this value.

Refreshing properties

Some column or calculation property values are derived from the property values of another column or calculation that they depend on. After a column or calculation property changes, you can batch-update the property values of its dependent columns and calculations.

Use the **Refresh properties** functionality to apply this type of updates. The following column or calculation properties can be affected by this update: **Data type**, **Technical data type**, **Aggregate**, and **Usage**.

For example, a data module contains a column that is named `Target_date` with the **Data type** of Date. This column is used in multiple calculations and custom tables, such as views or unions. Later, the data type for this column is changed to Timestamp in the underlying schema. After the [schema metadata is reloaded](#), the `Target_date` data type is set to Timestamp. Columns and calculations that reference the newly updated `Target_date` will still use its previous data type of Date until you update them by using the **Refresh properties** functionality. As a result, the metadata in the data module is synchronized with changes in the schema.

About this task

You can refresh the column and calculation properties in the whole data module, or in selected tables and folders. Also, individual columns and calculations can be refreshed.

If the data module is based on another data module, you need to refresh the properties of the source module first.

Procedure

1. In the data module tree, select the item or items for which you want to refresh properties.
You can select the data module itself, or select one or multiple tables, columns, calculations, or folders (outside of a table).
2. Click the item (or items) context menu , and select **Refresh properties**.
The refresh functionality determines which property values need to be updated within the selected scope, and then proceeds with the update. A message is displayed that informs whether the update was successful.

Chapter 4. Members in the data tree

The data module tree shows the content of relational and dimensional sources. In both relational and dimensional sources, members are shown in the data tree.

To view the content of a dimensional source, expand the package that contains the source items. Dimensional members are the nodes of the **Members** folder.

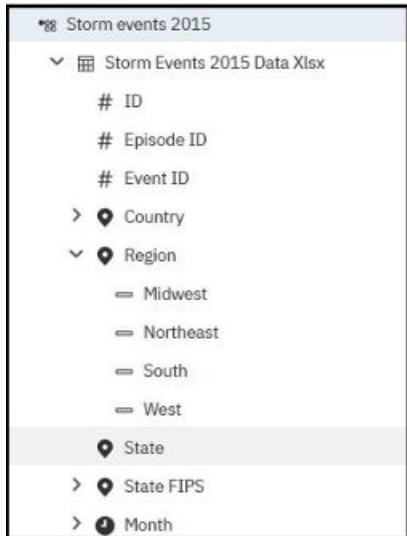
Relational members are the nodes of columns in relational sources.

You can initiate the search for members from the data tree, from the context menu *** of hierarchies, levels, members, or columns. For more information, see [“Searching for members” on page 71](#).

Relational sources

For relational sources, each unique value in a column is shown as a member in the data tree. These types of members are referred to as relational members.

In the following sample data module, the Region column contains the following members: Midwest, Northeast, South, and West.

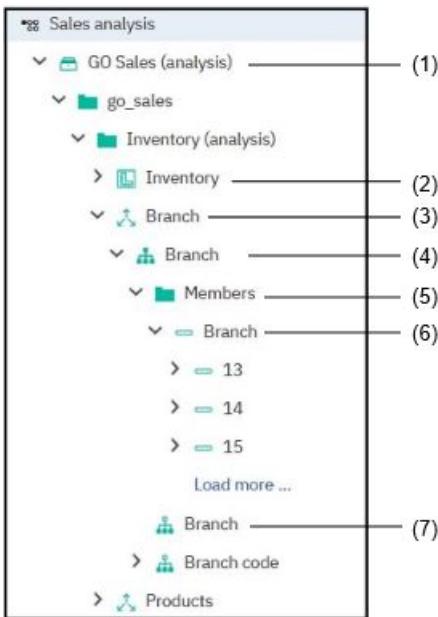


The modeler can disable showing relational members for a particular column. For more information, see [“Displaying relational members” on page 72](#).

Dimensional sources

Dimensional data is available through packages. Expand the package to view its content. The content can include dimensions, hierarchies, levels, members, and folders.

The following graphic shows an example of a dimensional data tree in a data module:



The data tree includes the following items:

1. Package

Packages are subsets of Framework Manager models containing items that can be used to create data modules, data sets, reports, dashboards, and explorations. Packages can also be containers for dimensional sources, such as Planning Analytics cubes and PowerCubes.

2. Measure dimension

Measure dimensions are collections of facts. They are composed of only quantitative items.

3. Dimension

Dimensions are broad groupings of descriptive data about a major aspect of a business, such as products, dates, or markets.

4. Hierarchy

Hierarchies are groupings of specific data within a dimension.

5. Members folder

The **Members** folders contain the available members for a hierarchy or level.

6. Member

Members are unique items within a hierarchy. A member can be a container for other members.

7. Level

Levels are positions within the dimensional hierarchy that contain information at the same order of detail and have attributes in common. Multiple levels can exist within a level hierarchy, beginning with a root level.

The modeler can restrict the number of members that are loaded in the data tree when first expanding a node that contains members, and when clicking the **Load more** link. Setting the limit of members returned in one fetch is done at the data module level, for all member fetches. For more information, see “[Setting the members display limits](#)” on page 71.

Modelers can’t create joins between dimensional data items, and can’t view content in the grid.

Searching for members

Search for members returns members only. This type of search is separate from the metadata search, which doesn't return members.

Search for members works identically in data modules, dashboards, stories, and explorations.

Each member in the search results includes the member unique name (MUN), which is a unique identifier for a member in Cognos Analytics. For more information about MUNs, search the *IBM Cognos Analytics Framework Manager User Guide* or *IBM Cognos Analytics Transformer User Guide*.

About this task

Search for members can be invoked from the context menu of a hierarchy, level, member or column. The search looks for all children of the item from which it was invoked. For levels, the search includes only the immediate child-members. A relational member has no children so the search menu is not available for it.

The search results are limited to members that the users have permissions to view.

The system limit for search results is 50. If there are more than 50 matches, you can click **Load more** to load the next 50 matches found by the search.

If a member is not found, you might need to refine the search text, or navigate to a lower level in the data tree and start the search from there.

You can't invoke the search for members from a **Members** folder.

Procedure

1. Expand the data tree to view its nodes.
2. From the context menu  of a hierarchy, level, member, or column, select **Search for members**.
3. In the search bar, type the search text.

The matching members are listed in the search results.

If the number of returned members is higher than 50, the **Load more** link is added below the set of retrieved results. When this link is clicked, another set of matching members is loaded.

Each member in the search results includes the member unique name (MUN). Hover the cursor over the displayed MUN to see its entire value. Here is an example of a MUN:

```
M1.[automation_Great_Outdoors_Company.mdc].[Products].[Products].[Product line]->:[PC].[@MEMBER].[1]
```

If the search doesn't return any members, type a different text. The search is reset every time you type a new text.

4. To exit the search for members, click **Cancel** in the search bar.

You are back in the data tree.

What to do next

In dashboards, stories, and explorations, you can perform operations on members that are found by the search.

Setting the members display limits

The members display limit specifies the maximum number of members to load in the data tree nodes for each fetch request.

A fetch request occurs when first expanding a node with members, or clicking **Load more**.

For dimensional sources, members are found in the **Members** folder, or as child items of a Member node. For relational sources, members are child items of a column.

About this task

The members display limits are specified at the data module level and apply to members of all sources that the data module is comprised of.

Procedure

1. Open the module properties by clicking the properties icon  in the application bar.
2. On the **General** tab, specify the **Members display limit** property by using one of the following options:
 - **Set the limit**

Set the maximum number of members to display in data tree nodes (hierarchies, levels, members, and columns) for each fetch request.
Enter an integer number between 1 and 3000 in the **Members limit** field. The default number is 15.
For **More members**, specify how to display members over the specified limit. Choose one of the following options:

 - **Load more**

The **Load more** link is added to the data tree node below the members that are already loaded. When the link is clicked, the next set of members is loaded. The number of members in the set is equal to the number that is specified as the members limit.
 - **Search**

The **Search** link is added to the data tree node below the members that are already loaded. Users can use this link to search for members.
 - **System limit**

The system limit is to load up to 3000 members per one fetch request.
 - **Search link only**

No members are displayed in the data tree. Instead, the **Search** link is added to the data tree that the users can use to search for members.
3. Save the data module.

Results

The members display limit affects members in the data module, as well as dashboards, explorations, and stories that use the data module as their source.

Displaying relational members

For relational data sources, you can show or hide members in the data module tree for a column.

Also, you can set the sort order for members when they are displayed in the data module tree, or in visualizations in dashboards, stories, explorations, and reports that use the data module as their source. By default, IBM Cognos Analytics loads items in the order that was defined in the data source.

You can control whether the column values are sorted, the sort order, what rule defines a sort order, and where nulls are placed.

Starting with IBM Cognos Analytics 12.0.1, automatic sorting and custom sorting is available.

About this task

Members display properties are specified at the column level, for non-measure columns.

Procedure

1. Select one or multiple non-measure columns in the same or different tables.
2. In the application bar, click the properties icon  to open the properties panel.
3. On the **General** tab, locate the **Members display** property, and select one of the following **Display options**:
 - **Automatic**

Members can be expanded in the data tree. Sorting is enabled, and members are sorted by the current column. Sort order is **Ascending**. This is the default setting.
 - **Show members**

Members can be expanded in the data tree. You can select the column to sort by, and set the sort order to **Ascending** or **Descending**. The members that are shown in the data tree don't dynamically adjust to the changed sort order. Use the **Refresh members** action from the column context menu for the sort order to be reflected in the data tree.

The column to sort by should be related one-for-one to the current column. If you map the sort-by column to a column that is less unique than the current column, the output will be sorted randomly and grouping of the data results will be misaligned.
 - **Hide members**

Members can't be expanded in the data tree. Previously shown members are removed, and new members can't be loaded for the column.
 - **Custom**

Sorting is enabled, and members are sorted according to the custom sort order that is automatically specified or manually defined in the data module.
4. Save the data module.

Results

The members display that is specified in the data module is used as the default display in the metadata tree of dashboards, stories, and explorations. The members sort order can be overwritten in visualizations.

Automatic sorting in month, weekday, and season columns

Starting with IBM Cognos Analytics 12.0.1, in newly loaded database schemas and uploaded files, members in the month, weekday, and season columns are automatically sorted in chronological (custom) order.

For example, in the month column, the sort order is January, February, March, and so on.

In a data module or a dashboard, the **Members display** property on the column is automatically set to **Custom**.

The screenshot shows the IBM Cognos Analytics interface. On the left, there's a navigation pane with icons for Home, Recent, Dashboards, and Data. Below that is a tree view of 'Selected sources / automatic_sorting_month.xlsx'. Under this tree, there's a folder named 'automatic_s..._month.xlsx' which contains a single item labeled 'Month'. This 'Month' item has a list of twelve members: January, February, March, April, May, June, July, August, September, October, November, and December. To the right of the tree view is a large panel titled 'Properties'. This panel is divided into several sections: 'Usage' (Identifier), 'Aggregate' (Count Distinct), 'Data type' (Text), 'Represents' (Time, Month), 'Members display' (with a 'Display options' dropdown), and 'Sort members by' (which is currently set to 'Custom'). A red box highlights the 'Custom' option in the 'Display options' dropdown.

Values that are not included in the automatic custom sort are appended at the end of the list of sorted items.

Note: In dashboards, the automatically generated custom sorting is different than the **Custom** sorting that is available in visualizations.

You can modify the auto-generated sorting definitions by applying custom sort manually. For more information, see [“Defining custom sort order” on page 74](#).

Multilingual considerations

Auto-generated sorting definitions are created for the locale of the user who uploaded the file or loaded the schema. For example, if a German user uploads a file that has the month columns for English, German, and French, the custom sorting is applied to the German month column only. The custom sorting definition that is shown in the data module, dashboard, or report, which use this file as their source, has values of the German months. The English and French columns do not have custom sorting defined.

Defining custom sort order

You can manually sort column values in a custom order.

The custom sorting that is defined in data modules can be overwritten in dashboards or explorations that are based on this module.

About this task

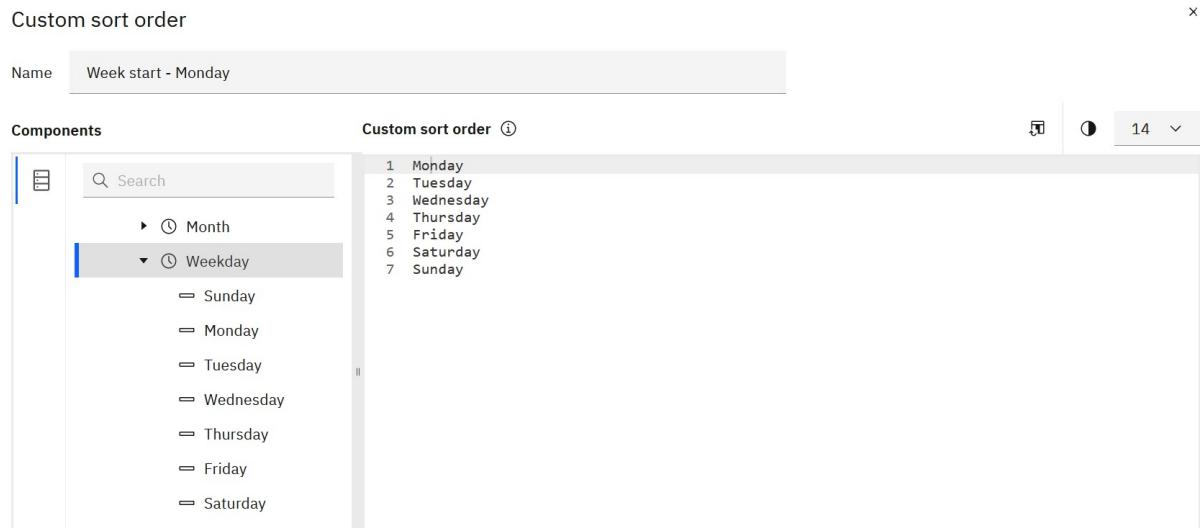
Custom sorting is automatically generated for month, weekday, and season columns when new schemas and files are loaded. You can modify the automatic sorting by using the manual custom sorting. For example, in the automatically sorted weekday column Sunday is the first day of the week. If you want Monday to be the first day of the week, edit the custom sorting definition. For more information, see [“Automatic sorting in month, weekday, and season columns” on page 73](#).

There is a hard-coded limit of 100 members for the custom sorting editor. If you have more than 100 members you would need to use a sort index with the **Sort members by** property. For more information, see [“Displaying relational members” on page 72](#)

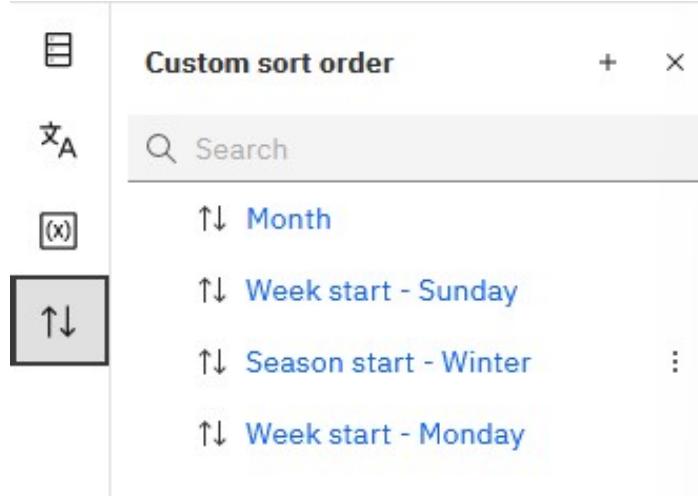
Procedure

1. Click the vertical arrows icon  to open the **Custom sort order** panel.
2. In the **Custom sort order** panel, click the add icon .
3. In the **Custom sort order** dialog box, select the column for which you want to specify the custom sort, and type a name for the custom sort.
4. Move (drag, click, or type) the column members from the data tree to the expression area, and sort them in the desired order. Then click **OK**.

In the following example, custom sort order **Week start - Monday** is defined for the **Weekday** column.



The **Week start - Monday** definition appears in the **Custom sort order** panel.



Results

The new **Custom sort order** is now available with the **Custom** sort option for the column property **Members display**.

Properties

General **Navigation paths**

Members display

Display options	Custom
Sort members by	Weekday
Members order	Ascending
NULL values	<input type="radio"/> First <input checked="" type="radio"/> Last
Custom sort order	Week start - Mond

The custom sorting is stored as part of the metadata of the uploaded file or schema. As such, it is available to all data modules which consume the file or schema, or in the case of an uploaded file, all dashboards which consume it directly.

Chapter 5. Relative date analysis

With the relative dates feature, you can analyze measures filtered by date periods that are relative to a particular date. Examples of relative date filters include current quarter, last quarter, quarter-to-date, or month-to-date.

When using relative dates, you can create reports and dashboards that show date-filtered results in different visualizations, crosstabs, and so on. By default, the date-filtered measures in the data use today's date as the reference date in the analysis.

The implementation of this feature uses a subset of Cognos Analytics base samples that include the sample calendar data modules, **Gregorian Calendar** and **Fiscal Calendar**. Ensure that the samples are available in your Cognos Analytics installation before you start other, related tasks. For more information, see ["Sample calendars" on page 77](#).

Tip: The base samples are installed with the Cognos Analytics server, and an administrator imports the sample deployment **Samples_for_Install_12_x_x** to the content store. For more information, see "Importing the base samples" in the *IBM Cognos Analytics Samples Guide*.

To enable relative date analysis in Cognos Analytics, you must create a data module that maps your data to a calendar. This data module can then be used as a source for relative date analysis in reports and dashboards. For more information, see ["Creating a data module for relative date analysis" on page 81](#).

If you want to customize the reference date for relative date analysis based on user roles, use the **_as_of_date** global parameter. For more information, see ["Customizing the reference date" on page 92](#).

After the Cognos Analytics environment is set up for relative date analysis, users can use the relative date filters and measures to perform analysis of data in reports and dashboards. For more information, see "Relative date analysis" in the *IBM Cognos Analytics Reporting Guide*.

Video

Here's a video that captures the process of creating a data module for relative date analysis: [video](#) (<https://www.youtube.com/watch?v=sIaU4xXJo-c>).

Sample calendars

The IBM Cognos Analytics base samples include a set of sample calendars that you need to set up relative date analysis.

The sample calendar data modules and their sources can be found in the **Team content > Calendars** folder.

The following samples are available in this folder:

- **Gregorian calendar** data module.

Contains dates from January 1, 1950 to December 31, 2050.

- **Fiscal calendar** data module.

Contains dates from March 1, 1950 to February 28, 2050.

- **Fiscal calendars** folder.

This folder contains 12 sample calendar data modules. Each of these calendars covers 100 years (1950 to 2050), but the dates in each calendar start in a different month. The month in the calendar name indicates the start of a fiscal year. For example, the **02. February 1** data module is the sample calendar for the fiscal year starting on February 1.

The **03. March 1** calendar is the same as the **Fiscal calendar**, and the **01. January 1** calendar is the same as the **Gregorian calendar**.

- **Retail 4-5-4 calendar** data module.

Contains dates from January 31, 2016 to January 30, 2023. The retail calendar is based on the National Retail Federation (NRF) 4-5-4 Calendar. You can also generate a custom retail calendar. For more information, see “[Creating a custom retail calendar](#)” on page 79.

- **Source files** folder.

This folder contains the source .csv files for the calendars data modules. The files contain dates for the associated calendars.

Columns and dates in the sample calendar data modules

The dates are listed in the following columns:

TheDate

The main reference date for each row.

PD_TheDate

Previous day from **TheDate**. The dates in this column are one day prior to **TheDate**.

ND_TheDate

Next day from **TheDate**. The dates in this column are one day after **TheDate**.

dYear

The date that is the beginning of the year to which **TheDate** belongs.

Tip: The fiscal calendars are denoted by the year in which the calendar ends.

PY_TheDate

Previous year for **TheDate**. The dates in this column are one year prior to **TheDate**.

NY_TheDate

Next year for **TheDate**. The dates in this column are one year after **TheDate**.

dQuarter

The date that is the beginning of the quarter to which **TheDate** belongs.

PQ_TheDate

Previous quarter for **TheDate**. The dates in this column are one quarter prior to **TheDate**.

NQ_TheDate

Next quarter for **TheDate**. The dates in this column are one quarter after **TheDate**.

dMonth

The date that is the beginning of the month to which **TheDate** belongs.

PM_TheDate

Previous month for **TheDate**. The dates in this column are one month prior to **TheDate**.

NM_TheDate

Next month for **TheDate**. The dates in this column are one month after **TheDate**.

Important: Do not modify the column names in the sample data modules and .csv files because that would break the relative date filters in the sample calendars.

Predefined filters in the sample calendar data modules

The sample calendars data modules contain a set of predefined date filters. These filters can be used to perform relative date analysis in different types of visualizations.

The following filters are predefined in the sample calendars:

- Prior week
- Prior month (not available in the retail calendar)
- Prior quarter

- Prior year
- Current week
- Current month
- Current quarter
- Current year
- WTD (week to date)
- MTD (month to date)
- QTD (quarter to date)
- YTD (year to date)
- Prior WTD
- Prior MTD
- Prior QTD
- Prior YTD
- Same week last year
- Same month last quarter
- Same month last year
- Same quarter last year
- Same MTD last quarter
- Same MTD last year
- Same QTD last year

You can view the expression associated with each filter by clicking **Edit filter** from its context menu.

You can also create your own custom filters. For more information, see [“Creating relative date filters ” on page 83.](#)

Creating a custom retail calendar

If the out-of-the box **Retail 4-5-4 calendar** sample data module is not sufficient for your reporting or dashboarding needs, you can create your own retail calendar.

About this task

Use the **Retail 4-5-4 calendar generator** data module that is included with the Cognos Analytics samples to generate the custom retail calendar. You can modify the start and end years of the calendar, or restate which years have only 52 weeks.

Use the National Retail Federation (NRF) [4-5-4 Calendar](#) as a reference.

Procedure

1. In the **Team content > Calendars > Tools** folder, locate the **Retail 4-5-4 calendar generator** data module.
2. Using the **Save as** option, save the calendar generator data module under a different name to the location where the other calendar data modules are located, which is **Team content > Calendars > Tools**.

Use this copy of the calendar generator to continue with your edits.

3. In the **Data module** panel, from the table context menu , select **Edit SQL table**.

The table SQL is shown in the expression editor.

4. In the **Name** field, change the table name so that it reflects the new date range. For example, type **Retail 454 Calendar 2016-2023**.

5. In the **Expression** box, modify the table SQL as required.

Follow the steps in the comments to modify the code. For example, to add years to the retail calendar, follow steps 1 to 5.

- **Step 1:** Set the number of years with 364 days and number of years with 371 days.

In the 2016-2022 calendar, years 2016, 2018, 2019, 2020, 2021, 2022 have 364 days, and year 2017 has 371 days, which is reflected in the following select statement:

```
select R + 1 from gen_rows where (6 * 364 + 1 * 371) >= R
```

If changing start or end years, you must specify a proper number of years with 364 days and years with 371 days, as defined in the NRF calendar. For example, to extend the calendar to include NRF year 2023 that has 371 days, use the following select statement:

```
select R + 1 from gen_rows where (6 * 364 + 2 * 371) >= R
```

The new statement reflects that years 2016, 2018, 2019, 2020, 2021, 2022 have 364 days, and years 2017 and 2023 have 371 days.

- **Step 2:** Specify the beginning date for the calendar.

In the 2016-2022 calendar, the January 31, 2016 is the first day of retail year 2016, which is reflected in the following select statement:

```
select _add_days ( date '2016-01-31' , R ) from gen_rows
```

If you want to use a different start date, use the date as defined in the NRF calendar. To extend the calendar to include NRF year 2023, the start date remains unchanged.

- **Step 3:** Specify dYear (the beginning date of the current retail calendar year).

In the following case statement, each when clause represents the start date of one retail calendar year. To extend the calendar to include NRF year 2023, a new when clause is added, as highlighted in bold font in the code below:

```
case when D >= '2023-01-29' then cast ('2023-01-29' as date)
      when D >= '2022-01-30' then cast ('2022-01-30' as date)
          when D >= '2021-01-31' then cast ('2021-01-31' as date)
          when D >= '2020-02-02' then cast ('2020-02-02' as date)
          when D >= '2019-02-03' then cast ('2019-02-03' as date)
          when D >= '2018-02-04' then cast ('2018-02-04' as date)
          when D >= '2017-01-29' then cast ('2017-01-29' as date)
          when D >= '2016-01-31' then cast ('2016-01-31' as date)
      else null end as dYear,
```

- **Step 4:** Specify dyear_PY (the beginning date of the previous retail calendar year).

In the following case statement, each when clause represents the beginning date of the previous calendar year, and add_days represents the negative number of days in the previous retail calendar year. The bolded when clause defines the start date for the previous year (NRF 2022) for all days in NRF year 2023.

```
case when D >= '2023-01-29' then _add_days ( cast( '2023-01-29' as date), -364 )
      when D >= '2022-01-30' then _add_days ( cast( '2022-01-30' as date), -364 )
      when D >= '2021-01-31' then _add_days ( cast( '2021-01-31' as date), -364 )
      when D >= '2020-02-02' then _add_days ( cast( '2020-02-02' as date), -364 )
      when D >= '2019-02-03' then _add_days ( cast( '2019-02-03' as date), -364 )
      when D >= '2018-02-04' then _add_days ( cast( '2018-02-04' as date), -371 )
      when D >= '2017-01-29' then _add_days ( cast( '2017-01-29' as date), -364 )
      when D >= '2016-01-31' then _add_days ( cast( '2016-01-31' as date), -364 )
      else null end as dYear_PY,
```

- **Step 5:** Specify dyear_NY (the beginning date of the next retail calendar year).

In the following case statement, each when clause represents the beginning date of the next retail calendar year, and add_days represents the number of days in the current retail calendar year. The

bolded when clause defines the start date for the next year (NRF 2024). That date is 371 days after 2023-01-29` since NRF year 2023 has 371 days.

```
case when D >= '2023-01-29' then _add_days ( cast( '2023-01-29' as date), 371 )
      when D >= '2022-01-30' then _add_days ( cast( '2022-01-30' as date), 364 )
      when D >= '2021-01-31' then _add_days ( cast( '2021-01-31' as date), 364 )
      when D >= '2020-02-02' then _add_days ( cast( '2020-02-02' as date),
      364 )
      when D >= '2019-02-03' then _add_days ( cast( '2019-02-03' as date),
      364 )
      when D >= '2018-02-04' then _add_days ( cast( '2018-02-04' as date),
      364 )
      when D >= '2017-01-29' then _add_days ( cast( '2017-01-29' as date),
      371 )
      when D >= '2016-01-31' then _add_days ( cast( '2016-01-31' as date),
      364 )
      else null end as dYear_NY
```

6. Click **OK** to save the expression. Then click **Save** to save the data module.

You created a new retail calendar generator data module.

7. Create a CSV file from your retail calendar generator data module in the following way:

- Using the new retail calendar generator data module as a source, create a list report in Cognos Analytics Reporting.
 - Drag all columns from the custom calendar table into the list.
 - Optional: Sort the **TheDate** column as **Ascending**.
 - Run the report using the **Run CSV** option to generate the CSV output.
 - Save the report .csv file to the location where the other calendar .csv files are saved. Current calendar source files are located in the **Team content > Calendars > Source files** folder.
8. Replace the data in the **Retail 4-5-4 calendar** data module with the data from the new retail calendar in the following way:
 - From **Team content > Calendars**, open the **Retail 4-5-4 calendar** data module.
 - Expand the **Source view** .
 - In the **Sources** panel, right-click the module .csv file, and select **Relink**.
 - Select the .csv file for the custom retail calendar to use as the new source.
 9. Save the updated retail calendar under a different name either in **Team content > Calendars**, or in a different location.

Creating a data module for relative date analysis

To enable relative date analysis, you need to create a data module where business data is associated to a calendar.

In this data module, at least one date column must be associated to a calendar, and at least one measure column must be associated to the date column. This association is done by using the column property **Lookup reference**.

Before you begin

The sample calendars must be available.

About this task

You can create a new data module from scratch, or add relative date capabilities to an existing data module.

Tip: The **Team content > Samples > By feature > Relative dates** folder contains the **Relative dates on a report** and **Relative dates on a dashboard** samples that illustrate the implementation of this feature in a report and dashboard. You can use these samples as a reference when you create your data module.

Procedure

1. Create a data module, or open an existing data module.
2. Verify that your business data sources contain at least one date column and one measure column.
 - a) From the date column menu, select **Properties > General**. Ensure that the **Data type** property of the column is set to Date.

If the **Data type** property is set to **Timestamp**, you can change the type to **Date** by using the **cast** function in the expression editor.

If the data source is an Excel file or a CSV file, dates in the date column must be formatted with the ISO 8601 notation *yyyy-mm-dd*.
 - b) From the measure column menu, select **Properties > General**. Ensure that the **Usage** property of the column is set to **Measure**.
3. If the **Usage** property is set to **Identifier**, you can change the property to **Measure**.

Tip: If your data module source is linked to its source, which is indicated by the link icon , you need to break the link. Otherwise, the data module is read-only, and you can't modify its properties. To break the link, select the **Break link** option from the data module menu. However, do not break links in any of the sample calendar data modules.

3. In the **Data module** panel, click the **Add sources and tables**  icon to add a calendar source, which can be one of the following sources:
 - The sample **Gregorian calendar** data module in the **Team content > Calendars** folder.
 - The sample **Fiscal calendar** data module in the **Team content > Calendars** folder.
 - One of the sample data modules in the **Team content > Calendars > Fiscal calendars** folder.
 - The sample **Retail calendar_454_2016_2022** data module in the **Team content > Calendars** folder.
4. In your business data source that you specified in steps 1 and 2, associate at least one date column to the calendar, and at least one measure column to the date column.
 - a) For the date column that you want to associate to the calendar, open **Properties**, and locate the **Lookup reference** property. From the **Lookup reference** drop-down menu, select the name of the calendar source that you added to the data module. If needed, repeat this step for other date columns.

The relative date filters, such as **Prior year**, **Prior month**, **MTD**, and so on, appear under the date column. To view the full list of filters, see “[Sample calendars](#)” on page 77.
 - b) For the measure column that you want to associate to the date, open **Properties**, and locate the **Lookup reference** property. From the **Lookup reference** drop-down menu, select the date column to reference. If you defined **Lookup reference** for multiple date columns, choose the date column that is appropriate for this measure. If needed, repeat this step for other measure columns.
5. Save the data module to a folder in **Team content**.

Tip: If you add or remove a filter from a calendar data module, the data modules that reference this calendar through the **Lookup reference** property don't reflect the change until you close and reopen them.

Results

The data module can now be used to create dashboards and reports.

Creating relative date filters

A relative date filter specifies a range of dates that are relative to the `_as_of_date` global parameter.

The sample Gregorian and Fiscal calendars already contain a number of predefined relative date filters. If you need custom filters, you can add them to these calendars.

Before you begin

1. Know the columns in the sample calendar.

New filters are added to the sample Gregorian or Fiscal calendar data modules. To understand how these calendars are structured, look at the columns, and view dates in different columns in the same row. For example, in the Gregorian calendar data module, for the column **TheDate** with the value of September 30, 2018, the related values for columns **dYear**, **PY_TheDate**, and **dMonth** are shown in the following table:

TheDate	dYear	PY_TheDate	dMonth
2018-09-30	2018-01-01	2017-09-30	2018-09-01

For more information, see [“Sample calendars” on page 77](#).

2. Conceptualize the filter lower bound and upper bound in relation to the `_as_of_date` global parameter.

A relative date filter defines a range of dates between the filter lower bound (range start) and upper bound (range end) dates. The lower and upper bounds are set against a reference date that is the `_as_of_date` parameter value.

For example, for a year-to-date (YTD) filter, the lower bound date is the first day of the first month in the year that contains the `_as_of_date` date. The upper bound date is the date that is the `_as_of_date` parameter value. If the `_as_of_date` parameter is December 19, 2018 (**TheDate**), the lower bound date is January 1, 2018, and the upper bound date is December 19, 2018.

By default, the `_as_of_date` parameter has a value of today. However, it can be set to a different date. For more information, see [“Customizing the reference date” on page 92](#)

3. Build the filter expression.

The critical element of creating a relative date filter is the filter expression. Familiarize yourself with the expression syntax and variables before you start entering the code in the expression editor. For more information, see [“Creating filter expressions” on page 84](#).

Procedure

1. From the **Team content > Calendars** folder, open the sample calendar data module where you plan to add the new filter.

The data module contains one table with a number of existing filters. Add your new filter to this table.

2. From the table context menu , click **Filter**.
3. In the filter editor that is displayed, type the new filter name.
4. In the **Expression** pane, type or paste the filter expression.

For example, to create the [Last 12 months](#) filter, enter the following expression:

```
// validate: 1 = 1
#$_this.parent.idForExpression# >=
    #queryValue($_this.parent.split.ref + '.dMonth',
    $_this.parent.split.ref + '.TheDate = ' +
        queryValue($_this.parent.split.ref + '.PY_theDate' ,
```

```

        $_this.parent.split.ref + '.TheDate = ' + $_as_of_date)#
AND
#$_this.parent.idForExpression# <
    #queryValue($_this.parent.split.ref + '.dMonth',
    $_this.parent.split.ref + '.TheDate = ' + $_as_of_date)#

```

Another filter example that you can use is [Next 4 months](#).

For more information, see “[Creating filter expressions](#)” on page 84.

5. Validate the expression.

Validation of date filter expressions must be done manually because the validate button  in the expression editor doesn't validate macro expressions. So you can only visually confirm that the following elements are correct:

- The expression is preceded with the // validate: 1 = 1 comment.
- The outer block of the queryValue macro function is enclosed within hash marks (#).
- Each queryValue has matching round brackets () for its two arguments.

Tip: You can debug filter expressions in Reporting. To do so, open a report that contains the relative date filters, and set the validation option in the report to **Information**.

6. Click **OK**.

The new filter is added to the calendar table at the top of the list of filters. You can drag the filter to change its position in the list. The filter is created even if the expression contains errors. To modify the filter, from its context menu , click **Edit filter**.

Results

The new filter is now available to the data modules that reference this calendar through the **Lookup reference** property, and can be used for relative date analysis.

Tip: The new filter, as other filters in the calendar, should remain hidden.

Creating filter expressions

A relative date filter is based on an expression. The expression defines the filter lower and upper bounds, and the timeline between the bounds. The timeline is mapped to the queryValue macro.

When you [create a new filter](#), you enter the expression in the expression editor.

Use the following syntax to create the filter expression:

```

// validate: 1 = 1
#$_this.parent.idForExpression# >= lower_bound_date expression#
AND
#$_this.parent.idForExpression# <= upper_bound_date expression#

```

For example, the year-to-date (YTD) filter that is available in the sample calendars data modules uses the following expression:

```

// validate: 1 = 1
#$_this.parent.idForExpression# >=
    #queryValue($_this.parent.split.ref + '.dYear',
    $_this.parent.split.ref + '.TheDate = ' + $_as_of_date)#
AND
#$_this.parent.idForExpression# <= #$_as_of_date# 1

```

In this filter, expression 1 is the filter *lower_bound_date expression*. Expression 2 is the filter *upper_bound_date expression*.

The lower bound and upper bound code blocks are combined by using the AND operator.

Tip: The comment // validate: 1 = 1 must always be included at the beginning of the expression.

The *lower_bound_date expression* and *upper_bound_date expression* are the elements that you must define for your filter. The remaining part of the expression remains unchanged for all filters.

For a description of variables that are used in relative date filter expressions, see “[Expression variables](#)” on page 86.

To define the lower bound and upper bound expressions, you need to complete the following tasks:

- [Identify the move intervals for the filter timeline](#)
- [Map each move interval to one queryValue macro](#)

Identify the move intervals for the filter timeline

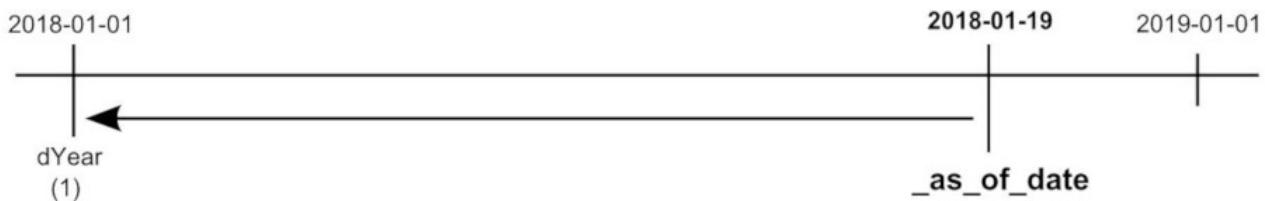
A timeline starts from the **_as_of_date** date, and then uses one or more move intervals (units of time) to reach the lower bound or upper bound date.

The sample calendars support the following move intervals: day, month, quarter, and year.

A move interval is expressed by using the following calendar columns:

- **PD_TheDate** - move to the previous day
- **ND_TheDate** - move to the next day
- **dYear** - move back to the first day of the year
- **PY_TheDate** - move back to the same or equivalent date in the previous year
- **NY_TheDate** - move forward to the same or equivalent date in the next year
- **dQuarter** - move back to the first day of the quarter
- **PQ_TheDate** - move back to the same or equivalent date in the previous quarter
- **NQ_TheDate** - move forward to the same or equivalent date in the next quarter
- **dMonth** - move back to the first day of the month
- **PM_TheDate** - move back to the same or equivalent date in the previous month
- **NM_TheDate** - move forward to the same or equivalent date in the next month

The type of filter implies which columns are used to express the timeline. For example, in the year-to-date (YTD) filter, the **dYear** column is the lower bound move interval, as shown in the following graphic. There is no upper bound move interval for this filter.



Map the move intervals to queryValue macros

After you identify the move intervals for the filter lower and upper bounds, you need to map each move interval to one queryValue macro.

The queryValue uses the following syntax.

```
#queryValue($_this.parent.split.ref + move_interval ,  
$_this.parent.split.ref + '.TheDate =' + date)#+$#
```

For example, here is how the move interval **dYear** is mapped to the queryValue macro (parts of the code in bold font) in the year-to-date (YTD) filter expression:

```
// validate: 1 = 1  
#$#_this.parent.idForExpression#+=
```

```

    #queryValue($_this.parent.split.ref + '.dYear',
                $_this.parent.split.ref + '.TheDate = ' + $_as_of_date)#
AND
#$_this.parent.idForExpression# <= #$_as_of_date#

```

Depending on the type of date filter, your expression might include multiple, nested `queryValue` macros, such as in the following **Prior YTD** filter:

```

// validate: 1 = 1
#$_this.parent.idForExpression# >=
    #queryValue($_this.parent.split.ref + '.dYear',
                $_this.parent.split.ref + '.TheDate = ' +
                    queryValue($_this.parent.split.ref + '.PY_TheDate',
                                $_this.parent.split.ref + '.TheDate = ' + $_as_of_date)
    )#
AND
#$_this.parent.idForExpression# <=
    #queryValue($_this.parent.split.ref + '.PY_TheDate',
                $_this.parent.split.ref + '.TheDate = ' + $_as_of_date)#

```

You can find another example of nested `queryValue` macros in the [Next 4 months](#) filter.

Expression variables

The relative date filter expression uses a set of variables to define the filter conditions. The variables evaluate to specific values when the filter is applied in visualizations.

Use the information in this topic when [creating relative date filter expressions](#).

The following **Prior Year** filter expression from the sample Gregorian calendar can be used as an illustration when reading descriptions of the variables:

```

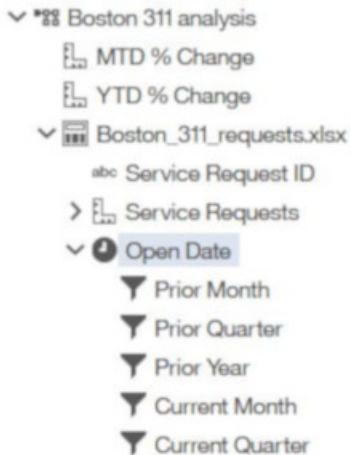
// validate: 1 = 1
#$_this.parent.idForExpression# >=
    #queryValue($_this.parent.split.ref + '.dYear',
                $_this.parent.split.ref + '.TheDate = ' +
                    queryValue($_this.parent.split.ref + '.PY_TheDate',
                                $_this.parent.split.ref + '.TheDate = ' + $_as_of_date)
    )#
AND
#$_this.parent.idForExpression# <
    #queryValue($_this.parent.split.ref + '.dYear',
                $_this.parent.split.ref + '.TheDate = ' + $_as_of_date)#

```

`$_this.parent`

This variable refers to a table column of type Date whose **Lookup reference** property is set to the calendar that you are referencing. The table with the Date column is in the data module that is used for relative date analysis. For more information, see “[Creating a data module for relative date analysis](#)” on [page 81](#).

For example, in the following data module, `$_this.parent` refers to the Open Date column.



The following two variables function within the context of the Date column:

- `$_this.parent.idForExpression`

This variable evaluates to the `idForExpression` for the Date column. The `idForExpression` is the full identifier that uniquely identifies the Date column within the data module. This identifier is not viewable from the user interface.

- `$_this.parent.split.ref`

This variable evaluates to the calendar that is referenced by the **Lookup reference** property of the Date column.

All date filters in the calendar, including the new ones that you add, are accessible as child filters of the Date column in the data module that references this calendar through the **Lookup reference** property. The filters are used for relative date analysis in reports and dashboards.

queryValue

`queryValue` is one of the macro functions that Cognos Analytics provides.

Tip: To view the description of the `queryValue` macro, click the functions tab in the expression editor and search for the macro. The description is shown in the **Information** pane.

Within the context of relative date filters, `queryValue` returns the date value from the specified Date column, at the specified date. The specified Date column is the first parameter to the `queryValue` function. The second parameter is the specified date.

In the following example from the year-to-date (YTD) filter, the `queryValue` returns, from the calendar column `dYear`, the date where the calendar `TheDate` date equals to the `_as_of_date` date.

```
#queryValue($_this.parent.split.ref + '.dYear',
           $_this.parent.split.ref + '.TheDate = ' + $_as_of_date)#+
```

Example filter: Last 12 months

This topic provides an expression for a date filter that includes the last 12 months relative to the `_as_of_date` parameter.

Paste this expression in the filter editor to create the Last 12 months date filter.

```
// validate: 1 = 1
#$_this.parent.idForExpression# >=
    #queryValue($_this.parent.split.ref + '.dMonth',
               $_this.parent.split.ref + '.TheDate = ' +
                   queryValue($_this.parent.split.ref + '.PY_TheDate',
                             $_this.parent.split.ref + '.TheDate = ' + $_as_of_date)
    )#
AND
#$_this.parent.idForExpression# <
```

```
#queryValue($_this.parent.split.ref + '.dMonth',
$_this.parent.split.ref + '.TheDate = ' + $_as_of_date)#{
```

Here are the steps that were used to build this expression:

1. Identify the calendar columns to use.

The filter uses the calendar columns **TheDate**, **dMonth**, and **PY_TheDate** from the sample **Gregorian calendar** data module.

For more information, see “[Sample calendars](#)” on page 77.

2. Define the filter lower and upper bounds.

The filter lower bound is the first day of the month that is 12 months prior to the month containing the date that is represented by the **_as_of_date** parameter. The filter upper bound is the last day of the last complete month, relative to the date that is represented by the **_as_of_date** parameter.

The following table shows the move intervals for the lower bound date when the **_as_of_date** date (**TheDate**) is January 19, 2019.

TheDate	PY_TheDate	dMonth
2019-01-19	2018-01-19	
2018-01-19		2018-01-01

The filter lower bound date is 2018-01-01. The filter upper bound date is 2018-12-31.

3. Define the move intervals for the lower bound and upper bound timelines.

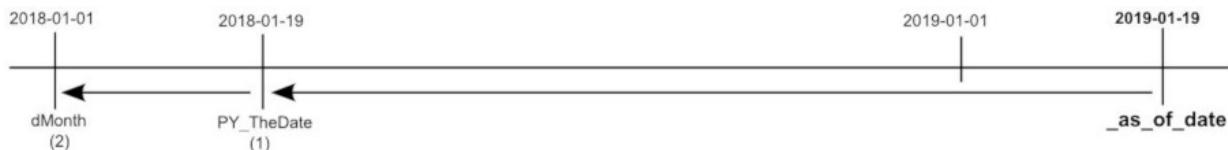
The timeline consists of move intervals that are based on the columns **PY_TheDate** and **dMonth**.

The following move intervals exist for the lower bound timeline:

- Move interval 1: **PY_TheDate**
- Move interval 2: **dMonth**

The move interval for the upper bound timeline is **dMonth**.

Here is a graphical representation of the move intervals for the lower bound timeline when January 19, 2019 is the **_as_of_date** date.



4. Map each move interval to one `queryValue` macro.

The expression of the lower bound consists of two `queryValue` macros. Each `queryValue` maps to one move interval within the lower bound expression. The initial move interval (**PY_TheDate**) is nested within the second move interval (**dMonth**), as shown below:

```
#$$_this.parent.idForExpression# >=
#queryValue($_this.parent.split.ref + '.dMonth',
$_this.parent.split.ref + '.TheDate = ' +
queryValue($_this.parent.split.ref + '.PY_TheDate' ,
$_this.parent.split.ref + '.TheDate = ' + $_as_of_date)
)#{
```

The expression of the upper bound consists of one `queryValue` macro, as shown below:

```
#$$_this.parent.idForExpression# <
#queryValue($_this.parent.split.ref + '.dMonth',
```

```
$_this.parent.split.ref + '.TheDate = ' + $_as_of_date)#{
```

This expression uses the less than (<) sign because the filter includes only dates prior to the upper bound, and not equal to the upper bound itself.

Example filter: Next 4 months

This topic provides an expression for a date filter that includes the next 4 months relative to the **_as_of_date** parameter.

You can paste this expression in the filter editor to create the Next 4 months date filter.

```
// validate: 1 = 1
#$$_this.parent.idForExpression# >=
    #queryValue($_this.parent.split.ref + '.dMonth',
    $_this.parent.split.ref + '.TheDate = ' + $_as_of_date)#
AND
#$$_this.parent.idForExpression# <
    #queryValue($_this.parent.split.ref + '.dMonth',
    $_this.parent.split.ref + '.TheDate = ' +
    #queryValue($_this.parent.split.ref + 'NM_The_Date',
    $_this.parent.split.ref + '.TheDate = ' +
    #queryValue($_this.parent.split.ref + 'NM_The_Date',
    $_this.parent.split.ref + '.TheDate = ' +
    #queryValue($_this.parent.split.ref + 'NM_The_Date',
    $_this.parent.split.ref + '.TheDate = ' +
    #queryValue($_this.parent.split.ref + 'NM_The_Date',
    $_this.parent.split.ref + '.TheDate = ' +
    #queryValue($_this.parent.split.ref + 'NM_The_Date',
    $_this.parent.split.ref + '.TheDate = ' + $_as_of_date)
)
)
)
)#{
```

Here are the steps that were used to build this expression:

1. Identify the calendar columns to use.

The expression uses the **TheDate**, **dMonth**, and **NM_TheDate** columns from the sample **Gregorian calendar**.

For more information, see [“Sample calendars” on page 77](#).

2. Define the filter lower and upper bounds.

The filter lower bound is the first day of the month containing the date that is represented by the **_as_of_date** parameter. The upper bound is the last day of the month that is 3 months after the month containing the **_as_of_date** date.

The following table shows the move intervals for the upper bound dates when the **_as_of_date** date (**TheDate**) is December 19, 2018.

TheDate	NM_TheDate	dMonth
2018-12-19	2019-01-19	2018-12-01
2019-01-19	2019-02-19	
2019-02-19	2019-03-19	
2019-03-19	2019-04-19	
2019-04-19		2019-04-01

3. Define the move intervals for the lower bound and upper bound.

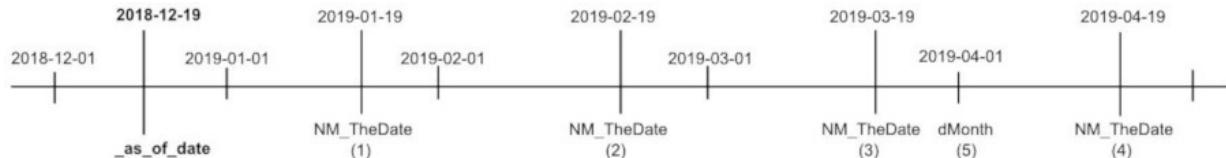
The filter timeline consists of move intervals that are based on the columns **NM_TheDate** and **dMonth**.

The move interval for the lower bound timeline is **dMonth**.

The upper bound timeline includes the following move intervals:

- Move interval 1: **NM_TheDate**
 - Move interval 2: **NM_TheDate**
 - Move interval 3: **NM_TheDate**
 - Move interval 4: **NM_TheDate**
 - Move interval 5: **dMonth**

Here is a graphical representation of the timeline when the **_as_of_date** date is December 19, 2018.



4. Map each move interval to one `queryValue` macro.

The lower bound expression consists of one `queryValue` macro, as shown below:

```
##$_this.parent.idForExpression# >=
    #queryValue($_this.parent.split.ref + '.dMonth',
    $_this.parent.split.ref + '.TheDate = ' + $_as_of_date)##
```

The upper bound expression consists of 5 `queryValue` macros, nested within each other. Each `queryValue` maps to one move interval. The earlier move intervals are nested within the later move intervals, as shown below:

```

##$_this.parent.idForExpression# <
    #queryValue($_this.parent.split.ref + '.dMonth',
                $_this.parent.split.ref + '.TheDate = ' +
        #queryValue($_this.parent.split.ref + 'NM_The_Date',
                    $_this.parent.split.ref + '.TheDate = ' +
            #queryValue($_this.parent.split.ref + 'NM_The_Date',
                        $_this.parent.split.ref + '.TheDate = ' +
                #queryValue($_this.parent.split.ref + 'NM_The_Date',
                            $_this.parent.split.ref + '.TheDate = ' +
                    #queryValue($_this.parent.split.ref + 'NM_The_Date',
                                $_this.parent.split.ref + '.TheDate = ' + $_as_of_date)
    )
)
)#

```

This expression uses the less than (<) sign because the filter includes only dates prior to the upper bound, and not equal to the upper bound itself.

Other examples of relative date filters

This topic provides examples of relative date filter expressions.

Copy and paste the selected expression in the filter editor when creating the filter.

Note: To pass validation, the line // validate: 1 = 1 must remain in the expression as a comment.

The following examples are available:

- “Last 12 complete months” on page 91
 - “Prior month last year” on page 91
 - “Prior YTD 2 years ago” on page 91
 - “Prior year yesterday” on page 91
 - “Prior year 2 years ago” on page 92

- “Yesterday” on page 92
- “Last 7 days” on page 92

Last 12 complete months

```
// validate: 1 = 1
#${_this.parent.idForExpression# >=
    #queryValue($_this.parent.split.ref + '.dMonth',
        $_this.parent.split.ref + '.TheDate = ' +
            queryValue($_this.parent.split.ref + '.PY_TheDate',
                $_this.parent.split.ref + '.TheDate = ' + $_as_of_date)
)#
AND
#${_this.parent.idForExpression# <
    #queryValue($_this.parent.split.ref + '.dMonth',
        $_this.parent.split.ref + '.TheDate = ' + $_as_of_date)#
```

Prior month last year

```
// validate: 1 = 1
#${_this.parent.idForExpression# >=
    #queryValue($_this.parent.split.ref + '.PM_TheDate',
        $_this.parent.split.ref + '.TheDate = ' +
            queryValue($_this.parent.split.ref + '.dMonth',
                $_this.parent.split.ref + '.TheDate = ' +
                    queryValue($_this.parent.split.ref + '.PY_TheDate',
                        $_this.parent.split.ref + '.TheDate = ' + $_as_of_date)
)#
AND
#${_this.parent.idForExpression# <
    #
        #queryValue($_this.parent.split.ref + '.dMonth',
            $_this.parent.split.ref + '.TheDate = ' +
                queryValue($_this.parent.split.ref + '.PY_TheDate',
                    $_this.parent.split.ref + '.TheDate = ' +
                        $_as_of_date)
)#
$as_of_date)
```

Prior YTD 2 years ago

```
// validate: 1 = 1
#${_this.parent.idForExpression# >=
    #queryValue($_this.parent.split.ref + '.dYear',
        $_this.parent.split.ref + '.TheDate = ' +
            queryValue($_this.parent.split.ref + '.PY_TheDate',
                $_this.parent.split.ref + '.TheDate = ' +
                    queryValue($_this.parent.split.ref + '.PY_TheDate',
                        $_this.parent.split.ref + '.TheDate = ' +
                            $_as_of_date)
            )
)#
AND
#${_this.parent.idForExpression# <=
    #queryValue($_this.parent.split.ref + '.PY_TheDate',
        $_this.parent.split.ref + '.TheDate = ' +
            queryValue($_this.parent.split.ref + '.PY_TheDate',
                $_this.parent.split.ref + '.TheDate = ' + $_as_of_date)
)#
$as_of_date)
```

Prior year yesterday

```
// validate: 1 = 1
#${_this.parent.idForExpression# =
    _add_days ( #queryValue($_this.parent.split.ref + '.PY_TheDate',
        $_this.parent.split.ref + '.TheDate = ' + $_as_of_date)#, -1 )
```

Prior year 2 years ago

```
// validate: 1 = 1
#$_this.parent.idForExpression# >=
    #queryValue($_this.parent.split.ref + '.dYear',
        $_this.parent.split.ref + '.TheDate = ' +
            queryValue($_this.parent.split.ref + '.PY_TheDate',
                $_this.parent.split.ref + '.TheDate = ' +
                    queryValue($_this.parent.split.ref + '.PY_TheDate',
                        $_this.parent.split.ref + '.TheDate = ' +
                            $_as_of_date)
            )
    )#
AND
#$_this.parent.idForExpression# <
    #queryValue($_this.parent.split.ref + '.PY_TheDate',
        $_this.parent.split.ref + '.TheDate = ' +
            queryValue($_this.parent.split.ref + '.dYear',
                $_this.parent.split.ref + '.TheDate = ' + $_as_of_date)
    )#
```

Yesterday

```
// validate: 1 = 1
#$_this.parent.idForExpression# = _add_days ( #$_as_of_date#, -1 )
```

Last 7 days

```
// validate: 1 = 1
#$_this.parent.idForExpression# > #_add_days ( $_as_of_date, -7)#
AND
#$_this.parent.idForExpression# <= #$_as_of_date#
```

Customizing the reference date

The global parameter **_as_of_date** is used for relative date analysis. The parameter allows you to change the date that your relative date periods are based on.

By default, the relative date periods are based on the current date. For example, when the current date is July 15, 2018, the YTD (year-to-date) filter includes data from January 1 to July 15, 2018, and the prior month filter includes data from June 1 to June 30, 2018. When you set a specific date as a value for the **_as_of_date** parameter, your analysis is done as of that date.

Before you begin

After the **_as_of_date** parameter is set up by the administrator, log out, and log back in to Cognos Analytics for this parameter to be displayed for you.

If this parameter is not set up, see [“Setting the **_as_of_date** global parameter” on page 93](#) for more information.

Procedure

1. In the Cognos Analytics welcome page, select the **My parameters** icon  in the application bar.
2. For the **_as_of_date** parameter, select a new date by using the calendar picker, and click **Apply**.

Tip: The parameter can have different names, depending on the label that was specified for it by the administrator.

3. Re-run the reports and dashboards that use relative dates.

Results

The data in the reports and dashboards is updated based on the new reference date.

Setting the `_as_of_date` global parameter

You can set up the global parameter `_as_of_date`, and make it available to all system and tenant roles. The on-premises administrators can customize this parameter for specific user roles.

Procedure

1. Go to **Manage > Customization**, and select the **Parameters** tab.
2. Depending on the version of Cognos Analytics, perform one of the following steps:
 - In version 11.1.4 and later, click the **New** link, and type `_as_of_date` in the space provided. Press **Enter** on the keyboard. Alternatively, click the **Import** link, and import the `_as_of_date` parameter from the sample "Global parameter date picker" report. This report is located in **Team content > Calendars > Tools**.
 - In version 11.1.3 and earlier, click the **Import** link, and import the `_as_of_date` parameter from the sample "Global parameter date picker" report. This report is located in **Team content > Samples > Relative dates > Tools**.
3. From the `_as_of_date` parameter context menu , click **Properties**.
4. Specify a custom label for the parameter. To specify a language-specific label, next to **Languages**, click **Set**. You can also add a description of the parameter, or disable it.
5. Select the **Applied to all roles** checkbox.

When you select this property, all system and tenant user roles can use this parameter.

If you are a Cognos Analytics on-premises user, and want to customize this parameter for specific roles, don't select the **Applied to all roles** checkbox. Instead, proceed to step 6.
6. Customize the `as_of_date` parameter for specific roles in the following way:
 - a) In **Manage > People**, select the **Accounts** tab.
 - b) Locate the role for which you want to customize this parameter, and in the role **Properties** panel, select the **Customization** tab.
 - c) Next to **Parameters**, click **Settings**.
 - d) Select the checkbox next to the `_as_of_date` parameter that you specified in step 2.

Click **OK** to finish setting this parameter without changing the default date, which is the current date. To set a specific date, select the **Set values** link, select the date, and click **Apply**.
 - e) If needed, repeat steps b to d for other roles. The date that you select can be different for different roles.
7. Log out, and log back in.

Results

All users in the system or tenant can now see the **My Parameters** dialog box, and the `_as_of_date` parameter is available to users when they run reports or dashboards that include the relative date filters and measures. The users can customize this parameter for their needs. For more information, see ["Customizing the reference date" on page 92](#).

Setting a dynamic `_as_of_date` parameter

To set a dynamic `_as_of_date` parameter, you also need to set the **Time Perspective** global parameter.

About this task

The global parameter approach is sufficient for most use cases, but there are circumstances where additional user interactions are required. For example, many customers use data warehouses that load based on yesterday's data. When these customers run their reports today, they must manually select

yesterday's date in the **_as_of_date** parameter to display any data. For such customers, using the dynamic **_as_of_date** parameter saves time and effort.

You can set any date you choose as the global parameter that Cognos Analytics will remember. For example, you could set use `yesterday` or use `last Sunday`, which are common use cases.

The Gregorian calendar (**Team content > Calendars > Gregorian calendar**) includes the optional specification of the **Time Perspective** global parameter, which also employs the **_as_of_date** global parameter. When users run reports or dashboards that include the relative date filters and measures from the **Gregorian calendar** data module, they can customize these parameters for their needs.

Procedure

1. Go to **Manage > Customization**, and select the **Parameters** tab.
2. Click the **Import** link, and import the **_as_of_date** and **Time Perspective** parameters from the sample **Global parameter date picker** report. This report is located in **Team content > Calendars > Tools**.
3. From the **_as_of_date** parameter context menu , click **Properties**.
4. Specify a custom label for the parameter. To specify a language-specific label, next to **Languages**, click **Set**. You can also add a description of the parameter, or disable it.
5. Select the **Applied to all roles** checkbox.

When you select this property, all system and tenant user roles can use this parameter.

If you are a Cognos Analytics on-premises user, and want to customize this parameter for specific roles, don't select the **Applied to all roles** checkbox. Instead, proceed to step 6.
6. Customize the **as_of_date** parameter for specific roles in the following way:
 - a) In **Manage > People**, select the **Accounts** tab.
 - b) Locate the role for which you want to customize this parameter, and in the role **Properties** panel, select the **Customization** tab.
 - c) Next to **Parameters**, click **Settings**.
 - d) Select the checkbox next to the **_as_of_date** parameter that you specified in step 2.

Click **OK** to finish setting this parameter without changing the default date, which is the current date. To set a specific date, select the **Set values** link, select the date, and click **Apply**.
7. Repeat steps 3-6 for the **Time Perspective** parameter.
8. Log out, and log back in.
9. In the Cognos Analytics welcome page, select the **My parameters** icon  in the application bar.
10. For the **Time Perspective** parameter, select a reference date from the drop-down menu, and click **Apply**
11. Re-run the reports and dashboards that use relative dates.

Results

All users in the system or tenant can now see the **My Parameters** dialog box, and the **_as_of_date** and **Time Perspective** parameters are available to users when they run reports or dashboards that include the relative date filters and measures from the **Gregorian calendar** data module. You can customize these parameters. For more information, see [“Customizing the reference date” on page 92](#).

Query optimization for relative date measures

Starting with Cognos Analytics 11.2.3, a detail filter is automatically generated for a query that includes relative date measures, based on the filtered measures that are used in the query.

With this additional filter, a data server might be able to further optimize how it executes the query. Without this filter, more rows of data are passed to each relative date measure.

Relative date measures are defined by using the **Lookup reference** (split definition) property. This property links the measure to a table, such as the **Gregorian Calendar** or **Fiscal Calendar** sample table. The filter definitions in the tables are based on macro expressions, and generate SQL fragments, such as in the following example:

```
SUM(  
    CASE  
        WHEN  
            ORDER_HEADER0.ORDER_DATE >= DATE '2006-10-01' AND  
            ORDER_HEADER0.ORDER_DATE <= DATE '2006-12-13'  
        THEN  
            ORDER_DETAILSO.QUANTITY  
    END) AS QTD__Quantity_;
```

This feature is implemented in a query transformation that under the right conditions adds a detail filter to the query in the WHERE-clause.

The following governors control the aspects of this feature:

optimizeFilteredMeasures

The following values are supported:

- **enabled** (default) - apply the optimization with all the conditions, as specified in the "Conditions" section in this document
- **disabled** - the optimization is disabled
- **enabled_verbose** - apply the optimization and add warning or info messages in the validation mode

optimizeFilteredMeasuresBetween

Values: enabled, disabled (default)

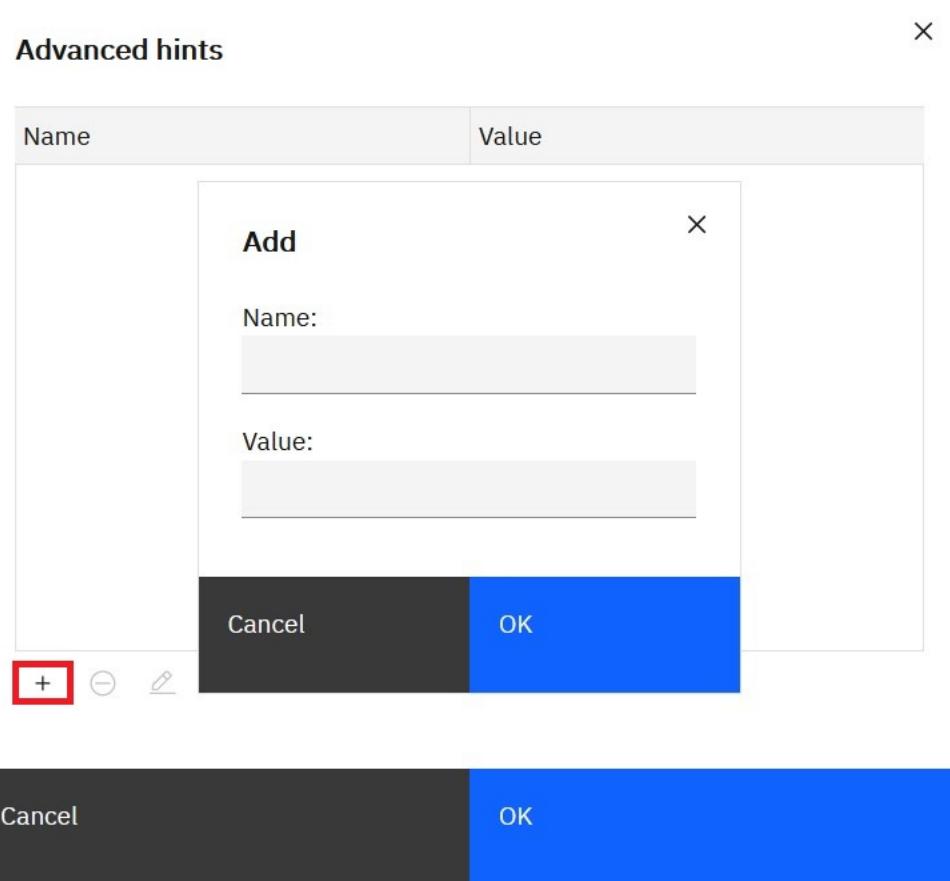
optimizeFilteredMeasuresPredicates

Values: single, multi (default)

No governors are exposed in data modules.

In Cognos Analytics Reporting 11.2.4 and later versions, you can configure this functionality through advanced query hints in the following way:

1. Open the report in edit mode.
 2. From the **Report** menu, click **Queries**, and select the query that you want to configure.
 3. Open the **Properties** panel.
 4. Under **QUERY HINTS**, click the **Advanced hints** property, and then click its context menu ..
- The **Advanced hints** dialog box is opened.
5. Click the plus icon to add a query hint.



6. Specify the hint name and value, and click **OK**.

For more information, see "Optimizing query performance with query hints" in the *IBM Cognos Analytics Reporting Guide*.

Conditions

By default, 2 expression patterns are recognized:

- Case when expr1 (> | >=) literal1 AND expr1 (< | <=) literal2 then expr2 end
- Case when expr1 (> | = | < | >= | <=) literal3 then expr2 end

When setting the governor optimizeFilteredMeasuresBetween=enabled, the case when expr1 between literal1 AND literal2 then expr2 end is also recognized.

The following rules apply:

1. expr1 occurrences must be the same (applies only to the first pattern).
2. The whole case expression must be a child of a summary operation.
3. The items referenced in expr1 must be from a data module and must have a split definition (**Lookup reference**).
4. Adjoining and overlapping ranges based on the same expr1 values are reduced to a single predicate. Adjoining considers the < and <= operands to be the same, as well as > and >=. This may lead to the inclusion of the = literal value for a range that ends with < literal value, and the next range that starts with > literal value.
5. If the computed range is negative-infinity to positive-infinity, the where clause predicate is not generated at all. For example, exp1 >= 2022-01-01 OR exp1 <= 2022-01-01.

6. No additional consideration is given to null values. Using a `>`, `>=`, `<`, `<=` operation with a null operand results in an error. Using a null literal in a between predicate is accepted, but results only in null. A `NONE` is a value expression, not a literal.
7. Setting the governor `optimizeFilteredMeasuresPredicates=single` creates a single, all-encompassing range predicate, causing overfetching for the price of a simpler predicate.
8. The generated predicates are not compared to existing where-clause predicates, which might result in duplicate predicates if filters were specified in other places.
9. The optimization affects the query of each fact-stream.
10. When other measures in the query are not filtered, the transformation is not applied.

When to disable the optimization

You could disable the optimization if you have a date range for which the categorical values are not present. When the optimization is disabled, the category would be part of the output, with a null value for the filtered measure. When the optimization is enabled, the category would not be part of the output.

For example, in the following situation the optimization can be disabled (there is no 'Mountaineering Equipment' for '2004'):

Product Line En	Quantity
Camping equipment	181,822
Golf Equipment	99,400
Mountaineering Equipment	301,958
Outdoor Protection	557,854
Personal Accessories	389,908

There's no filtered measure optimization on the between predicate in the case statement.

Product Line En	Quantity
Camping equipment	181,822
Golf Equipment	18,912
Mountaineering Equipment	
Outdoor Protection	274,650
Personal Accessories	86,322

The operands `>=` and `<=` trigger filtered measure optimization.

Product Line En	Quantity
Camping equipment	181,822
Golf Equipment	18,912
Outdoor Protection	274,650
Personal Accessories	86,322

Chapter 6. Multilingual metadata

When a data module supports multilingual metadata, users can view labels and descriptions of tables, columns, or calculations in different languages. Currently, data modules that support multilingual metadata can be used in Reporting only.

Reports that are based on data modules with multilingual metadata can be viewed without modifications in different languages.

Note: The folder, standalone calculation, and standalone filter metadata is not translated.

Typically, before you enable support for multilingual metadata, the data should already be translated (localized). It does not make sense to use translated labels when the data is not translated. To enable users to view data in Cognos Analytics, in their language, most often macros are used to look up the user's run locale, and return the data in the proper language.

The translation of metadata is driven by the localization requirements. Before you decide to translate metadata, you should know if your users need to see the names of report objects in their language. In some situations, the multilingual support is driven by statutory requirements. For example, in Canada a report that is provided by the federal government is by law required to be either in English or French, with both the data and metadata translated.

Multilingual metadata translation sources are external to the data module. They are stored in uploaded files or database tables. Updates and refinements to translations do not necessarily require editing of the data module.

The way you store the multilingual data and metadata can impact your database design.

Enabling support for multilingual metadata in a data module

To support multilingual metadata, the **Label**, **Description**, and **Screen tip** properties of the data module objects must be available in different languages.

When using a multilingual data module, a report can be created in one language and viewed in any number of other languages without modifications. It is possible to use a data module that was created in one locale to create a report in a second locale, and then run the report in a third locale. The [user locale preferences](#) control the display of the report labels in the appropriate language.

To enable multilingual metadata in a data module, do the following tasks:

- [Export the metadata to an external service](#) to be machine or human-translated to the specified locales.
- Import the translated sources into the data module.
- Add a translation definition for the supported language to the data module.

Exporting metadata for translation

When you export metadata for translation, a CSV file is generated that contains the strings to be translated, and context information for translators.

By default, this file is named `Exported_language.csv`.

For each exported language, the following columns can be created in the CSV file:

Original text

This column is always included. It contains the properties that you chose to export. The properties can be **Label**, **Screen tip**, or **Description**. The values of these properties will be translated.

Property

This column identifies whether the object is a **Label**, **Screen tip**, or **Description**.

Short identifier

This column contains the **Identifier** of the object.

Long identifier

This column is comprised of the object **Identifier** property, and the parent object identifiers that are needed to identify where the object is located. You can also use the long identifier to refine the grain of the translation.

For example, a table includes a column called Expected Volume. The value for **Original text** would be Expected Volume. The value for **Property** would be **Label**. The value for **Short identifier** would be EXPECTED_VOLUME. The value for **Long identifier** would be DIST_PRODUCT_FORECAST_FACT.EXPECTED_VOLUME. The string DIST_PRODUCT_FORECAST_FACT is the table identifier and is necessary to uniquely identify the column.

Note: Folders do not have the long identifier because they are used to organize objects only, and are not relevant when identifying the object path.

Exclude hidden objects | Include hidden objects

Use this option to exclude or include the metadata of hidden objects in the translation. The default is not to include them.

Ignore case sensitivity | Maintain case sensitivity

Use this option to control whether the letter case must be considered when translating the strings. By default, the letter case of strings is ignored.

Exclude external references | Include external references

Use this option to exclude or include objects from another data module in the translation.

The objects in the translation file are sorted alphabetically. You can change the sort order later.

For related information, see “[Object properties](#)” on page 64.

Before you begin

Think about the scope and costs of translating the metadata. It's possible that translating selected items of the data module could cost more than exporting and translating the entire data module.

About this task

You can control whether to export the entire data module or only selected items. If you select the root node of the data module, everything in the data module is exported. If you select a folder, a table, or multiple tables, only the selected items and their contents are exported.

Note: The folder, standalone calculation, and standalone filter metadata is not translated.

Procedure

1. Open the data module that contains the metadata for translation.
2. In the **Data module** tree, identify the items for translation, such as a table, and from the item context menu , select **Export language**.
3. In the **Export language** dialog box, specify the export options, as documented earlier in this topic.

In **Preview**, check if all required columns are created.

4. Click **Export without closing** or **Export and Close**.

When you click **Export without closing**, the Exported language.csv file is created for you, and the **Export language** dialog box remains open.

When you click **Export and Close**, the Exported language.csv file is created for you, and the **Export language** dialog box is closed.

Here is an example of the Exported language.csv file.

	Original text	Property	Short identifier	Long identifier
2	Branch code	Label	Branch_code	page_12.Branch_code
3	City	Label	City	page_12.City
4	Country	Label	Country	page_12.Country
5	Country code	Label	Country_code	page_12.Country_code
6	Date hired	Label	Date_hired	page_12.Date_hired
7	Email	Label	Email	page_12.Email
8	First name	Label	First_name	page_12.First_name
9	Last name	Label	Last_name	page_12.Last_name
10	Manager	Label	Manager	page_12.Manager
11	Position	Label	Position_	page_12.Position_
12	Sales region	Label	Sales_region	page_12.Sales_region
13	Sales region code	Label	Sales_region_code	page_12.Sales_region_code
14	Sales rep	Label	page_12	page_12
15	Sales staff code	Label	Sales_staff_code	page_12.Sales_staff_code
16	Staff name	Label	Staff_name	page_12.Staff_name

What to do next

Send the translation file to a translation service. The translators will translate the strings in the **Original text** column into the requested languages. Later, the sources that contain the translated strings can be imported into the data module.

Translation sources

The metadata for translation is most often stored in a table that contains the **Original text** column, and columns that contain the equivalent, translated values. The long identifier can also be included. The metadata can also be stored in a set of related tables, assuming that a valid query can be generated from them at run time.

The translation sources are separate from the data module. The module needs them only to set up the translation definitions. The translated metadata remains in the translation sources. Updates and refinements to translations do not necessarily require editing of the data module.

An example of a translation source is the sample `Metadata.xlsx` file that is provided by IBM. For more information, see [“Using the sample files to create a multilingual data module” on page 104](#).

You can have any number of translation sources. A source can be used in multiple translations. This flexibility means that you could create many copies of your exported metadata file and give one to each of your translators. Once they return their version of the file, you could upload it and use it to create a [translation definition](#).

An alternative could be to merge the translations into one file. To do that, you would sort the translations by the **Original text** column, which should be common to all of the translated files. You would then copy, for each translation, the **Original text**, **Long identifier** (if you intend to use it), and other translation columns into the proposed translation collection file. You would then use the spreadsheet utilities, such as EXACT, to ensure that each translation has all of the exported metadata. If that's the case, you could then use that file as your translation source. This method is possibly more cumbersome than using multiple translation files, and combining the files manually.

No matter which method you use, you must ensure that no metadata was accidentally removed or altered by the translator.

Creating the translation definition

The translation definition maps the label, description, and screen tip strings in your data module to the translations of those strings.

Before you begin

Your metadata must already be translated and the [translation sources](#) must already be added to the data module.

About this task

Specify the following properties to create the translation definition:

Label

The label is the name of the translation. The default value is the language name as set by the **To language** property. For example, if the **To language** property is French, the label value is French. You can change the label to any word or phrase. If you do this, the synchronization between the **To language** property and the label is lost. To restore the synchronization between the **To language** property and the label, click the Clear label button.

To language

This property defines the locale of the translation. If the [content language](#) of the user matches the **To language** value, the translation is used.

From language

This property represents the locale of the [original text](#) column. The locale of the original text could be different than the **Design language** property of the data module.

Specifying the **From language** locale allows the modeler to override translations from source modules that could be in a different design locale.

Original text field

The original text is the text of the **Label**, **Screen tip**, or **Description** from the data module. This text is the source string for translation. The instances of that string will be translated.

Select the checkbox **Match the case of the original text** if you know that your data source is case-sensitive. Otherwise, labels such as `Retailer country` and `Retailer Country` would be translated identically.

Translated text field

The translated equivalent of the **Original text field**.

Long identifier

This property consists of the identifier of the object whose **Label**, **Screen tip**, or **Description** is being translated, and any qualifying identifiers. This property is optional.

You can use the **Long identifier** to refine the scope of localization so that a string can be translated in different ways for different objects with the same label.

Select the checkbox **Match the case of the long identifier** to further refine of scope of the translation by matching the case of the long identifier. By default, the matching is case-insensitive

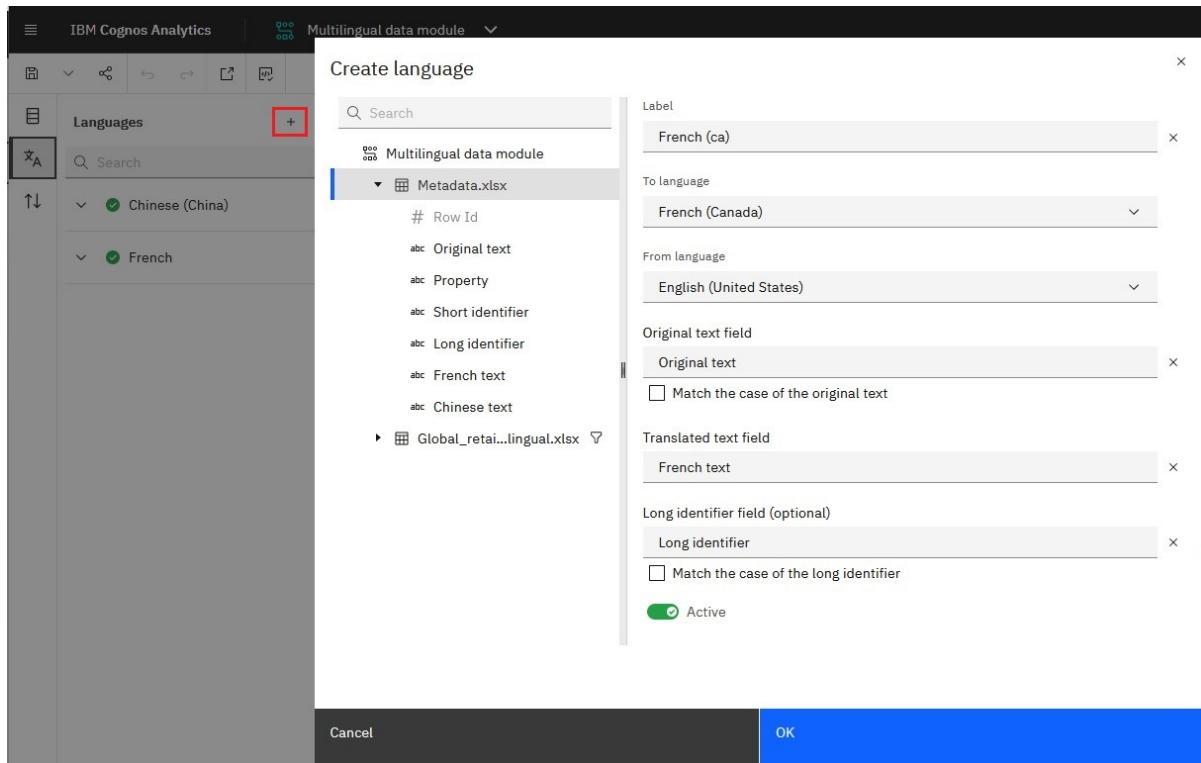
Active

You can have only one translation for a **To language** and **From language** locale pair set as active. However, you can have multiple inactive translations for the same locale pair.

Procedure

1. Click the **Languages** icon  in the vertical toolbar.
2. Click the **Add metadata conversion** icon , and in the **Create language** dialog box, specify the required settings.

The following image shows the translation settings based on the sample data. For more information, see “[Using the sample files to create a multilingual data module](#)” on page 104.



The translation definition is added to the **Languages** pane.

The screenshot shows the 'Languages' pane in the IBM Cognos Analytics interface. It displays two language definitions:

- Chinese (China)
- French

Both entries have a green checkmark and a context menu icon (three dots) to their right.

3. To preview the definition, from its context menu, click **Preview language**.

A new tab is created where you can see the translated metadata. A green border is added around the previewed metadata.

You can save the data module preview with the translated strings in it. However, do not overwrite the source data module with the preview.

4. Save the data module.

5. To test the translated metadata, create a report based on your data module.

In user locale preferences, change the content language to the language to which your metadata is translated. The data tree in the report shows the translated strings in your selected language.



Using the sample files to create a multilingual data module

You can download two sample .xlsx files, provided by IBM, that support multilingual data and multilingual metadata, and use them to create a multilingual data module.

The following sample files are available for download:

Global_retail_sales_multilingual.xlsx

This file contains the business data in multiple languages. The last column named **Product language** specifies the language of each row.

Download location: https://public.dhe.ibm.com/software/data/sw-library/cognos/mobile/C11/data/Global_retail_sales_multilingual.xlsx

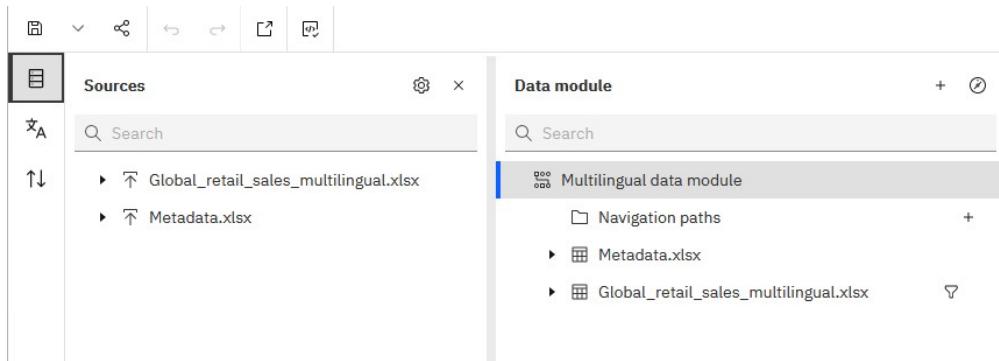
Metadata.xlsx

This file contains all of the text strings, including the names of the two uploaded files, and the table column headings. Each row contains the original English string and the equivalent translated string for French and Chinese. The **Short identifier** and **Long identifier** columns are also included.

Download location: <https://public.dhe.ibm.com/software/data/sw-library/cognos/mobile/C11/data/Metadata.xlsx>

Procedure

1. Upload the Global_retail_sales_multilingual.xlsx and Metadata.xlsx sample files to Cognos Analytics.
2. Create a data module selecting the two uploaded files as the sources.



3. Click the **Languages** icon

4. In the pane, click the **Add metadata translation** icon

5. In the **Create language** dialog box, add the French and Chinese translation definitions.

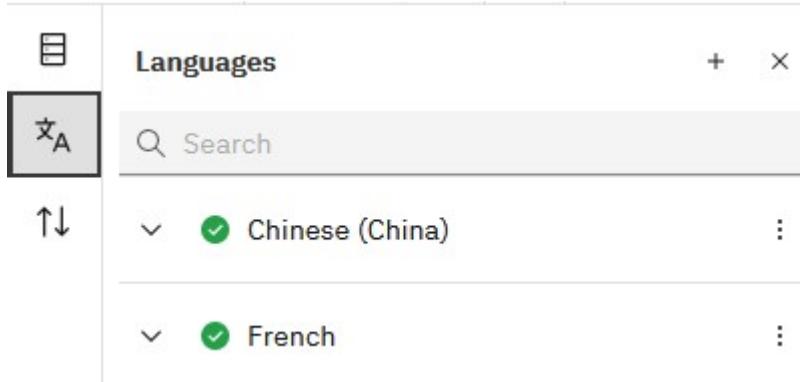
For the French translation, specify the following settings:

- For **Label**, type French.
- For **To language**
- For , select French.
- For **From language**, select English.
- In the data tree, expand Metadata.xlsx, and drag the column **Original text** onto the **Original text field** field.
- Drag the column **French text** onto the **Translated text field** field.
- Optional: Drag the column **Long identifier** onto the **Long identifier field** field.
- Click **OK**.

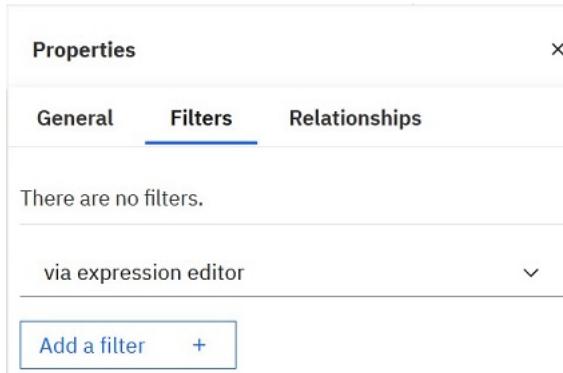
For the Chinese translation, specify the following settings:

- For **Label**, type Chinese (China).
- For **To language**, select Chinese (China).
- For **From language**, select English.
- In the data tree, expand Metadata.xlsx, and drag the column **Original text** onto the **Original text field** field.
- Drag the column **Chinese text** onto the **Translated text field** field.
- Optional: Drag the column **Long identifier** onto the **Long identifier field** field.
- Click **OK**.

The following translation definitions are added to the **Languages** pane.

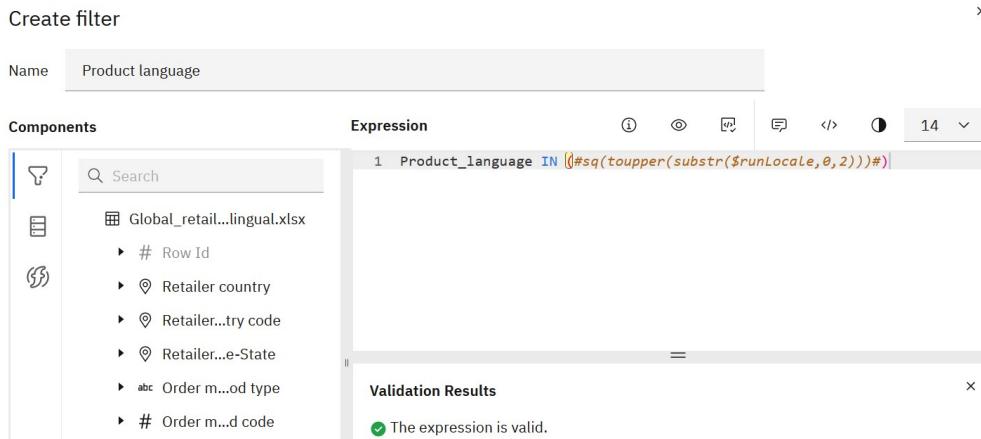


6. Create a filter in the data module that will return only the rows that contain the user's locale, as specified in the **Product language** column in the `Global_retail_sales_multilingual.xlsx` file.
 - a) In the **Data module** pane, select the `Global_retail_sales_multilingual.xlsx` file, and from its context menu, click **Properties**.
 - b) On the **Filters** tab, from the **Select a column** drop-down menu, click **via expression editor**.
 - c) Click **Add a filter**.



- d) In the expression editor, specify the filter name `Product_language`, and enter the following expression:

```
Product_language IN (#sq(toupper(substr($runLocale,0,2)))#)
```



- e) Validate the expression, and click **OK** to save the filter.

In the **Data module** pane, the filter `Product_language` is added as the last column within the `Global_retail_sales_multilingual.xlsx` data tree. The filter is set to English (EN).

7. Save the data module.
8. Test the data module to see if the translated metadata is displayed as expected.
 - a) From the **Personal menu**  in the application bar, click **Profile and settings**.
 - b) On the **Settings** tab, set **Content language** to French, and reload the page.

In the **Data module** pane, the filter `Product_language` is now set to French (FR).

 - c) In the `Global_retail_sales_multilingual.xlsx` data tree, click any column with the text data type. For example, **Product line** or **Product**.

The labels in the grid are now displayed in French and the data uses French locale settings.

 Grid  Relationships  Custom tables

 Product line	Product line code	Product type	Product type code	Product
Matériel de camping	991	Lampes	955	Lecteur de cartes LuckyLux
Matériel de camping	991	Lampes	955	Lecteur de cartes LuckyLux
Matériel de camping	991	Lampes	955	Lampe torche LuckyLux 2
Matériel de camping	991	Lampes	955	Lampe tout usage LuckyLux
Matériel de camping	991	Lampes	955	Lampe tout usage LuckyLux
Matériel de camping	991	Lampes	955	Lampe-tempête PermaLux
Matériel de camping	991	Sacs à dos et autres conteneants	954	Sac d'excursion Atlas
Matériel de camping	991	Sacs à dos et autres	954	Sacoches à vélo Atlas

Appendix A. Supported SQL data types

The IBM Cognos Analytics query service supports the standard relational data types that represent numeric, character, or temporal values.

When data modules and models are built, and queries are planned and executed, the data source is required to describe the column metadata, such as the data type, precision, scale, and nullability, to the query service. This includes columns in tables or views that are returned by a query or passed as parameters to procedures, functions, or query parameters. The query service maps the source column data types to the types that it supports. If the source data type is not supported by the query service, the query service treats it as an unknown type.

The following list shows the data types that are supported by the query service:

Precise and imprecise numeric types

The following precise numeric types are supported: smallint, integer, bigint, decimal, and decfloat.

The following imprecise numeric types are supported: float (real treated as float), and double precision.

When database vendors support numeric data types that are equivalent to the types that the query service supports, the query service easily maps the source data types to the types that it supports.

When database vendors use a general "number" data type, where the range of values that a column or parameter can hold is determined by the column precision and scale, the query service must determine which of its built-in data types to use for the mapping. The query service assigns the data type based on the precision and scale of the metadata. For example, a column in ORACLE that is described as NUMBER(3) is mapped to the smallint type. Columns with higher precision are mapped to larger precise (integer, bigint, or decimal) or imprecise (double precision) data types. For very large numeric values, the query service can use the decfloat data type.

For more information, see [ibmcognos.decfloat](#).

Character types

The following types are supported: char, varchar, clob, national char, national varchar, and national clob.

Character large objects (clob) can hold large strings and impose restrictions on how they can be used in a query. For more information, see the **ibmcognos.maxvarcharsize** parameter in *Managing IBM Cognos Analytics*.

The maximum length of a character string supported by dynamic query is 64 KB.

Datetime types

The following types are supported: date, time, time with time zone, timestamp, and timestamp with time zone.

Interval types

The following types are supported: interval year to month, and interval year to second.

Logical types

The supported type is Boolean.

The query service does not return the Boolean type to reports or dashboards.

Unknown types

The query service might not support a data type that is an equivalent of the source data type. A Framework Manager model or a data module that include columns with such data types show the type as an unknown data type. The query service can't perform any local query processing on values with the unknown data type, and the values can't be displayed in reports and dashboards.

A column of an unknown type can be referenced in expressions (calculations or filters) that are processed by the underlying data source. For example, a table includes a spatial column. A report or model might include a detail filter that the data source uses to evaluate if a customer is located within a distance from the specified spatial value. The data source must evaluate the expression in the filter.

If a table includes a bit string column, the report or model that uses the column can include an expression to convert the bit string to a type, such as integer, that is supported by the query engine. The expression must be supported by the data source.

Some data sources are supported through a vendor JDBC driver. In such cases, it might be possible to automatically convert the type and values of a built-in data type into a type that is supported by the query service. The query service would not be aware of the original data type. For more information about mapping the vendors built-in data types to JDBC data types, see the applicable SQL reference or programming guides from the vendors.

Appendix B. Using the expression editor

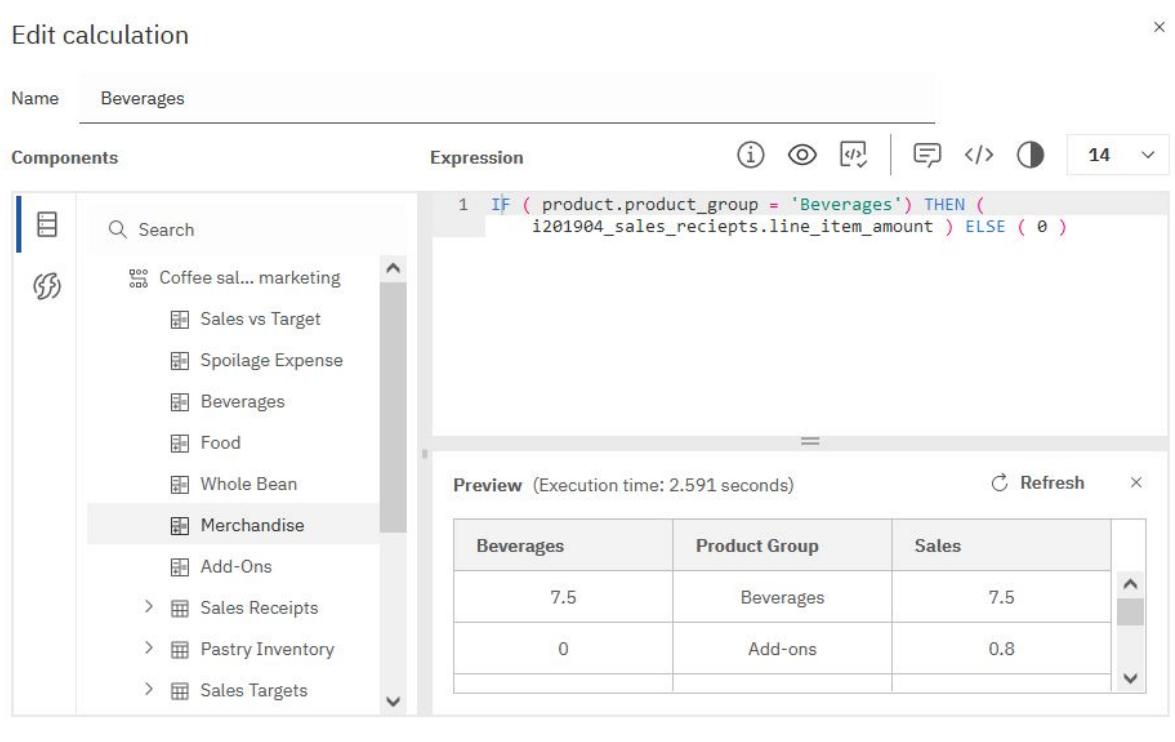
The expression editor in IBM Cognos Analytics is part of the product user interface where you can create and edit calculations, filters, and SQL expressions.

Note: The expression editor in Reporting is not the same. It has a different user interface, and some of the functionality and functions are different.

You access the expression editor when performing the following tasks:

- [Creating or editing calculations](#)
- [Creating or editing filters](#)
- [Creating or editing SQL-based tables](#)
- [Viewing the Expression property on columns](#)

The following screen capture shows the expression editor user interface:



The **Components** pane includes the data tab , and the functions tab .

The data tab contains the data module tree. You can drag tables and columns from the tree into the **Expression** box to create expressions.

The functions tab shows the list of supported functions and operators that you can use in the expressions. You can drag the functions into the **Expression** box to create expressions. For more information, see [“Creating expressions” on page 112](#). The functions tab is not available in all contexts, for example, when creating SQL tables.

The **Expression** area is the primary work area to create, edit, or view expressions. Here you can type or paste the expression or SQL code. Then, you can validate the code, and preview the projected columns or table.

The expression editor provides the following syntax validation and editing options:

-  - Validate the syntax. You can validate the whole statement, or only selected segments of code.
 -  - Preview columns and rows in your projected calculation, filter, or table. If the syntax is not correct, the columns are not displayed.
 -  - View descriptions of functions, and examples of their usage.
 -  - Insert the cursor anywhere in a line of code and select this button to comment out the entire line. To comment out multiple lines of code, select the lines and select this button. The comment string // is added at the beginning of each selected line.
- Tip:** To comment out sections of code, manually enclose the text between the following strings: /* and */
-  - Apply formatting to the code.
 -  - Use high-contrast mode.
 -  - Change the font size.

For more information about working in the expression editor, including examples, see [this article](#) in the IBM Cognos Analytics community.

Creating expressions

An expression is a combination of functions, operators, and values. They are used to create calculations, filters, SQL-based tables, and to modify a column's expression.

The expression editor is used to work with expressions in Data Modules, Dashboards, and Explore. It can be invoked from a number of places within these components, as documented in the topic [Appendix B, "Using the expression editor," on page 111](#).

The expression editor provides the following elements to create expressions:

- 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.

- Summaries

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.

- Constants

Fixed values that you can use in an expression.

- 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.

- Business Date/Time functions

Business functions for performing date and time calculations.

- Period predicates

Time period functions that evaluate to either true or false.

- Vendor Specific Functions

SQL functions that are specific to databases that Cognos Analytics supports.

- Common Functions

Standard SQL functions.

- Macro Functions

Functions that can be used within a macro. A macro is enclosed between the number signs (#) at the beginning and at the end. Everything between the number signs is treated as a macro expression, and is executed at run time. A macro can contain one or more macro functions.

All these elements are documented in the expression editor. For more information, see [“Expression editor documentation” on page 113](#).

Expression editor documentation

The expression editor includes descriptions of functions, as well as syntax and usage examples.

To view the documentation about a specific function that is used in your expression, click the function name in the expression. The documentation is displayed in the **Information** pane, as shown in the following screen capture.

Name Beverages

Components Expression

1 IF (product.product_group = 'Beverages') THEN (i201904_sales_receipts.line_item_amount) ELSE (0)

Information

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

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.

If you want to access the documentation of a specific function outside of the context of an expression, click the functions tab , and find the function in the list. You can use search to find the function. Click the function name, and view its documentation in the **Information** pane.

Create calculation

Name Calculation name

Components Expression 14

Search Operators Summaries Constants Constructs Business Da... Functions

`_add_minutes`

`_add_hours`

`_add_days`

`_add_months`

`_add_seconds`

`_add_years`

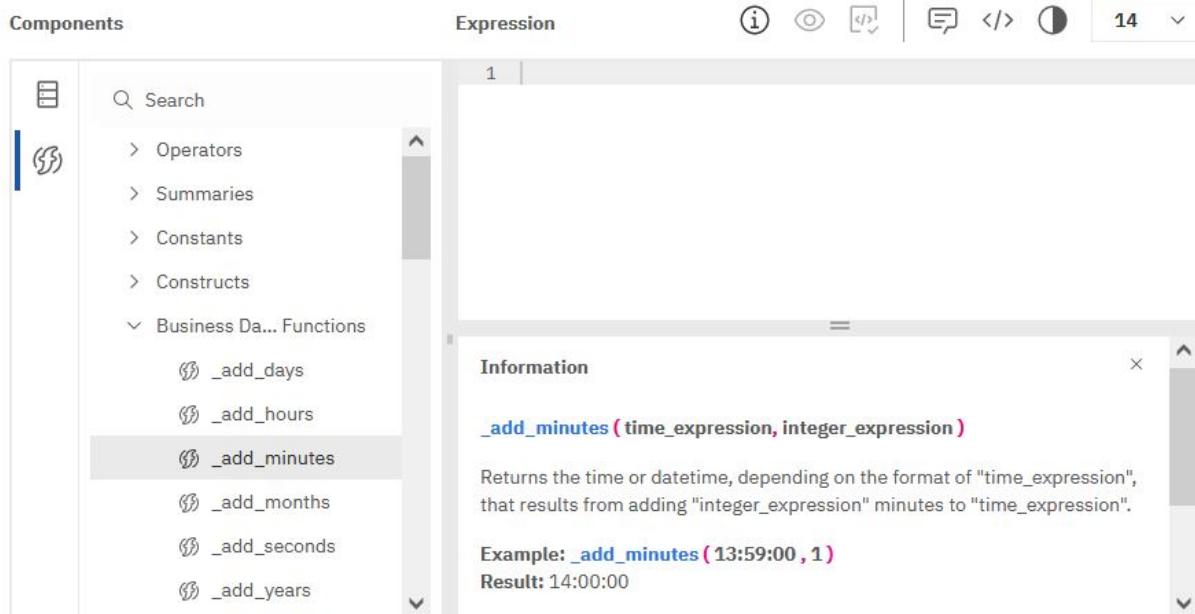
Information

`_add_minutes (time_expression, integer_expression)`

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

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

Result: 14:00:00



Appendix C. Data modules and Framework Manager

Data Modules is the primary metadata modeling environment in IBM Cognos Analytics. However, IBM Cognos Framework Manager, the metadata modeling tool that is associated with older versions (10.2.2 and earlier) of Cognos Analytics, is also supported.

If your organization still uses Framework Manager, you might be interested how the data module capabilities compare with Framework Manager capabilities, and how Framework Manager packages are used with the new versions (11.0.x and later) of Cognos Analytics.

For information about modeling concepts and best practices that are common for both tools, see the *IBM Cognos Analytics Metadata modeling guidelines*.

Framework Manager features not supported by data modules

Long-time IBM Cognos Business Intelligence users who are accustomed to modeling in IBM Cognos Framework Manager might be curious how this modeling environment compares to data modules.

Data modules currently don't support some of the modeling capabilities that Framework Manager provides. The following features are not, or not fully supported by data modules:

Stored procedures

In Framework Manager, you can import database stored procedures as a query object that can accept parameters, and can either retrieve or update data based on the nature of the stored procedure.

User-defined functions

In Framework Manager, you can import user-defined functions (UDF) from a database. Data modules do not support UDFs.

Prompts and parameters user interface

In Framework Manager, you can set the default prompt properties, such as the prompt type, use and display values, or filter item reference, on each query item in the user interface.

This type of user interface currently doesn't exist in data modules. To create new parameters and prompts in data modules, you need to use the macro functions - `prompt()` and `promptmany()`, as well as the `?parameter?` syntax in expressions. The data modules can't generate a preview of data for unresolved parameters, but the parameters work as expected in reports and dashboards.

When a table definition in a data module includes a column with an expression based on a prompt, the user is always prompted when any column from that table is included in a report or dashboard, even if the column with the prompt is not added to that report or dashboard. This behavior provides a consistent view of the data, and applies to both data modules and Framework Manager models.

Object security

In Framework Manager you can specify security on the reporting objects, such as query subjects and query items (tables and columns). The object security determines which objects a user can see in the metadata tree in reports or dashboards.

Parameter maps

In Framework Manager, modelers can dynamically substitute one value for another by using parameter maps. The parameter maps have two columns, one column for the value that you want to pass in via a macro expression that is calling the parameter map, and the other column for the value that you would like to substitute the first value with.

Governors

Governors in a Framework Manager model allow to modify queries at run-time. For example, by using governors, you can specify the maximum number of report tables or the query execution-time limit. Data modules don't have the user interface to specify governors. For more information, see “[Governors and data modules](#)” on page 117.

Relationships (joins) with expressions

In Framework Manager, modelers can create complex, custom join conditions directly in the **Relationship Definition** dialog box.

The **Edit relationship** user interface in data modules doesn't support custom joins.

Publish, change impact, and report dependencies

In Framework Manager, you can find report dependencies on specified objects, as well as a publish impact based on changes made to the model. As a result, the modeler can notify authors that their reports might be affected by the change in the model.

SAP HANA input variables

In Framework Manager, when importing an SAP HANA view that incorporates an input parameter, the parameters are exposed as a tab in the data source query subject. This allows modelers to provide a hard-coded value or dynamic value by using a macro expression.

Model Advisor

Model Advisor in Framework Manager is used to analyze models. It looks for problem areas based on the Cognos Analytics modeling guidelines.

Namespaces

Framework Manager namespaces are containers that organize and uniquely qualify content in a model. As a result, the same name can be used for multiple objects, as long as they reside in different namespaces. For example, you can have a query subject called Time in namespace A, and a query subject called Time in namespace B. The namespace name is part of the object identifier so that the objects can be distinguished from one another.

Star schema grouping presentation

Star schema grouping refers to the presentation of fact tables and their related dimension tables. Framework Manager uses namespaces and shortcuts to allow modelers to present the star schema groupings to authors. Each namespace contains one fact table and its related dimensions. Namespaces with the same dimensions in them are considered shared dimensions, which can be used to query facts from multiple namespaces that contain the shared dimensions.

Star schema grouping with shortcuts is available for data modules as of Cognos Analytics 11.2.3. In previous versions of Cognos Analytics this feature is not available.

Context explorer

Context Explorer in Framework Manager provides a view of the model based on an existing object.

In **Context Explorer**, modelers can view, test, and modify relationships of an existing object. They can also hide an object, change the layout, fit all objects in the window (with zoom-in and zoom-out), print, preview diagrams before printing, and change the page setup. **Context Explorer** is helpful when troubleshooting modeling issues.

The focus mode in data modules offers some similar capabilities, but the functionality is limited.

Model automation

In Framework Manager, model builds can be automated. This functionality is valued by many OEMs who want to automate model builds for their clients.

Creating agents, events, and tasks in Event Agent

Framework Manager packages can be used in IBM Cognos Analytics Event Agent to create agents that monitor your organization's data to detect occurrences of business events. Data modules cannot be used with Event Agent.

Tip: In Cognos Analytics 12.0 and earlier versions, Event Agent was named Event Studio.

Reporting limitations when using data modules

The following limitations apply when data modules are used as sources in Cognos Analytics Reporting:

- Data modules and Framework Manager packages cannot be combined in the same report.
- Data modules that are based on dimensional data sources, such as PowerCubes, dynamic cubes, TM1 data sources, and dimensionally-modeled relational (DMR) data sources, are not supported.
- The **Tree prompt** is not available in reports.

Governors and data modules

Users who are familiar with IBM Cognos Framework Manager might be interested how governors are used with data modules, since data modules don't have the user interface to define or change governors.

The following sections describe the contexts in which governors are used with data modules, and the governor values that can be assumed by queries in those contexts.

What governors do queries use while using a data module?

You can use a package directly from a dashboard. Such queries use the governors defined in the package.

Queries that use data modules use the default governor settings assumed by the dynamic query mode.

For more information, see "Dynamic query mode governors" in the *IBM Cognos Framework Manager User Guide*.

Note: The compatible query mode (CQM) governors are ignored by the dynamic query so these types of governors are not applicable to data modules. For more information, see "Governors" in the *IBM Cognos Framework Manager User Guide*.

What governors do queries use when a data module references Framework Manager packages?

A data module can reference one or more packages. Objects that are referenced from packages are resolved by using queries that use the governors from their package. Objects that are defined in the data module use the governors of the data module, thus the default governor values for the dynamic query mode.

For example, VIEW_ONE can include references to package A, and VIEW_TWO - references to package B. These views might be related, and referenced by another view, VIEW_THREE. In this example, the governors are applied in the following way:

- The query formed for VIEW_ONE uses the governors from package A.
- The query formed for VIEW_TWO uses the governors from package B.
- The query formed for VIEW_THREE uses the governors that were assumed by the data module when the data returned by VIEW_ONE and VIEW_TWO was joined.

For information about joining objects from different packages in a data module, see [“Creating custom tables” on page 33](#).

IBM.[®]