PUBLIC

SAP HANA Platform 2.0 SPS 00
Document Version: 1.0 – 2016-11-30

SAP HANA Modeling Guide

For SAP HANA Web Workbench

Content

1 Introduction to Modeling in SAP HANA

Modeling refers to an activity of refining or slicing data in database tables by creating information views that depict a business scenario. You can use these information views for any reporting and decision making purposes.

Information views consume various combinations of content data (that is, nonmetadata) to model a business use case. You can classify content data as:

- Attributes: Descriptive data - such as customer ID, city, and country.
- Measures: Quantifiable data - such as revenue, quantity sold and counters.

Creating an information view involves simulation of entities (such as customer, product, sales and so on.) and their relationship. Data visualization and analysis applications such as SAP BusinessObjects Explorer and Microsoft Office based reporting tools consume these information views and helps decision makers in their decision process.

SAP HANA Web based modeler provides extensive capabilities that helps you to create and edit information views (content models). It also helps you to create analytic privileges that govern the data access to information views.

You can create the following information view types in using SAP HANA Web based modeler:

- Graphical Calculation Views
- Script-based Calculation Views

Who should read this guide

This guide is intended for data modelers or business analyst or data analysts or application developers building database models or database experts involved in the processes of defining data models and schemas, database tables, information views, primary keys, indexes, partitions and all other aspects of layout and interrelationship of data in SAP HANA.

Related Information

Create Graphical Calculation Views [page 9]
Create Script-Based Calculation Views [page 37]

1.1 Modeling in Web-based Environments

Web-based development environments provide data modelers with tools that enable access to repository and catalog objects from a remote location, for example, using a Web browser.

SAP HANA Web workbench is a Web-browser based Integrated Development Environment that runs on SAP HANA Extended Application Services(SAP HANA XS). It offers data modelers and other user personas the modeling tools to work with SAP HANA content repository and SAP HANA database catalog objects, from a remote location. By using a browser, without requiring them to install any software locally, user can execute the data modeling tasks quickly and easily.

> **Note**
>
> The *SAP HANA Web-based Development Workbench* is available on the SAP HANA XS Web server at the following URL: `http://<WebServerHost>:80<SAPHANAinstance>/sap/hana/ide`. *SAP HANA Web-based Development Workbench* supports Microsoft Internet Explorer (10+), Mozilla Firefox, and Google Chrome Web browsers.

Related Information

Required Roles [page 6]

1.1.1 Required Roles

Before you start using the *SAP HANA Web-based Development Workbench*, the SAP HANA administrator must set up a user account for you in the database and assign the required modeling roles.

If modelers are using the SAP HANA Web-based Development Workbench, the privileges that they require for creating and activating information views as well as tool access are bundled into the following roles:

- sap.hana.xs.ide.roles::EditorDeveloper
- sap.hana.xs.debugger::Debugger

For more information, see SAP HANA Web-Based Development Workbench in the SAP HANA Developer Guide (For Web Workbench)

> **Note**
>
> If you want to grant users with full data access to all information views in your SAP HANA system, then assign the analytic privilege _SYS_BI_CP_ALL to the users, for example, in development systems. If you want to grant only restricted data access to information views, for example, in production systems, then create an analytic privilege with filters by including these information views as secured models, and assign this analytic privilege to the user role. For more information, see Defining Data Access Privileges.

1.2 Attributes and Measures

Attributes and measures form content data that you use for data modeling. The attributes represent the descriptive data, such as region and product. The measures represent quantifiable data such as revenue and quantity sold.

Attributes

Attributes are the non-measurable analytical elements.

Table 1:

Attributes	Description	Example
Simple Attributes	Individual non-measurable analytical elements that are derived from the data sources.	For example, `PRODUCT_ID` and `PRODUCT_NAME` are attributes of product data source.
Calculated Attributes	Derived from one or more existing attributes or constants.	For example, deriving the full name of a customer (first name and last name), assigning a constant value to an attribute that can be used for arithmetic calculations.

Measures

Measures are measurable analytical elements that are derived from calculation views.

Table 2:

Measures	Description	Example
Simple Measures	A simple measure is a measurable analytical element that is derived from the data sources.	For example, `PROFIT`.
Calculated Measures	Calculated measures are defined based on a combination of data from other data sources, arithmetic operators, constants, and functions.	For example, you can use calculated measures to calculate the net profit from revenue and operational cost..
Counters	Counters add a new measure to the calculation view definition to count the distinct occurrences of an attribute.	For example, to count how many times product appears and use this value for reporting purposes.

2 Creating Calculation Views and Previewing its Output

A calculation view is a powerful and flexible information view, which you can use to define more advanced slices on the data available in the SAP HANA database. Calculation views are simple and they mirror the functionality found in both attribute views and analytic views and much more.

However, you use calculation views when your business use case requires advanced logic, which you cannot achieve by creating the previous analytic views or attribute views. For example, you can create calculation views with layers of calculation logic, which includes measures sourced from multiple source tables, or advanced SQL logic, and so on. The data foundation of the calculation view can include any combination of tables, column views, attribute views and analytic views. You can create joins, unions, projections, and aggregation levels on data sources.

Calculation views can include measures and be used for multidimensional reporting or can contain no measures and used for list-type reporting. There are two types of calculation views that you can create:

- Graphical calculation views, which you can create using a graphical editor.
- Script-based calculation views, which can you create by writing SQL scripts.

Characteristics of calculation views:

- Supports both OLAP and OLTP models.
- Supports complex expressions (i.e. IF, Case, Counter).
- Supports reusing Analytic views, Attribute views and other Calculation views (Graphical and Scripted).
- Supports analytic privileges (i.e. restricting a user for a certain cost center).
- Supports SAP ERP specific features (i.e. client handling, language, currency conversion).
- Provides ability to combine facts from multiple tables.
- Provides support for additional data processing operations, (i.e. Union, explicit aggregation).
- Provides ability to leverage specialized languages (i.e. R-Lang).
- Provides ability to leverage both Column and Row tables.
- Performance of certain operations (i.e. star-join, aggregation) are inferior to Analytical Views.

Related Information

Create Graphical Calculation Views [page 9]
Create Script-Based Calculation Views [page 37]
Generate Time Data [page 35]
Preview Calculation View Output [page 41]

2.1 Create Graphical Calculation Views

Create graphical calculation views using a graphical editor to depict a complex business scenario with ease. You can create graphical calculation views to include layers of calculation logic and with measures from multiple data sources.

Context

Creating graphical calculation views is useful in bringing together normalized data that are generally dispersed. You can combine multiple transaction tables and analytic views, while creating a graphical calculation view.

Procedure

1. Open the SAP HANA Web-based Development Workbench Editor.

   The Web-based Editor tool is available on the SAP HANA XS Web server at the following URL: `http://<WebServerHost>:80<SAPHANAinstance>/sap/hana/ide/editor`.
2. In the navigator, select a package where you want to create the new calculation view.
3. In the context menu of the package, select *New* > *Calculation View*
4. Provide name and description.
5. Select the *Graphical* radio button.
6. In the *Data Category* dropdown list, select a data category type.
7. Choose *Create*.

   Modeler launches a new graphical calculation view editor with the semantics node and default aggregation or projection node depending on the data category of the calculation view.
8. Continue modeling the graphical calculation view by dragging and droping the necessary view nodes from the tools pallete.
9. From the editor's tools pallete, drag and drop a view node type to the editor.
10. Add data sources.

    If you want to add data sources to your view node, then

    a. Select a view node.

    b. Choose  .

    c. In the *Find Data Sources* dialog, enter the name of the data source and select it from the list.

       You can add one or more data sources depending on the selected view node.

    d. Choose *OK*.

> **Note**
>
> The find data sources dialog displays only those infomration views that are already activated. In other words, you can add only activated information views as data sources.

11. Define output columns.
    a. Select a view node.
    b. If you want add selected columns from the data source to the output, then in the *Mapping* tab, select the columns you want to add to the output of the view node.
    c. If you want to add all columns in the data source to the output, then in the *Mapping* tab, select the data source.
    d. Choose *Add to Output*.

> **Note**
>
> If you want to retrieve columns of aggregation or projection nodes to the result set even if you do not request it in your query. In other words, if you want to include those columns into the SQL `group by` clause even if you do not select them in the query, then:
>
> 1. Select *Semantics* node.
> 2. In the *Columns* tab select the *Keep Flag* checkbox for the required attributes.

12. Define attributes and measures.

    If you are creating a calculation view with data category as cube, then to successfully activate the information view, you have to specify at least one column as a measure.

    a. Select the *Semantics* node.
    b. In the *Columns* tab, select a column value.
    c. In the *Type* dropdown list, select *Measure* or *Attribute*.

       If the value is set to *Cube*, an additional *Aggregation* column is available to specify the aggregation type for measures.

> **Note**
>
> If you are using any attribute view as a data source to model the calculation view, the *Shared* section displays attributes from the attribute views that are used in the calculation view.

13. Activate the calculation view.
    a. If you want to activate the calculation view, in the menu bar, choose *Save*.
    b. If you want to activate the calculation view along with all objects, in the menu bar, choose *Save All*.
14. Assign changes.
    a. In the navigator toolbar, choose the  icon or the keyboard shortcut `Ctrl` + `S` to save and activate your calculation view.
    b. In the open change list, select a change ID that you want to use to assign your changes.

> **Note**
>
> The change list appears only if you have enabled change tracking for your SAP HANA system.

c. Choose *Assign*.

Next Steps

After creating a graphical calculation view, you can perform certain additional tasks to obtain the desired output. The table below lists the additional tasks that you can perform to enrich the calculation view.

Table 3: Working With View Nodes

Requirement	Task to Perform
If you want to query data from two data sources and combine records from both the data sources based on a join condition or to obtain language specific data.	Create Joins
If you want to query data from database tables that contains spatial data.	Create Spatial Joins
If you want to validate joins and identify whether you have maintained the referential integrity.	Validate Joins
If you want to combine the results of two more data sources.	Create Unions
If you want to partition the data for a set of partition columns, and perform an order by SQL operation on the partitioned data.	Create Rank Nodes
If you want to filter the output of projection or aggregation view nodes.	Filter Output of Aggregation or Projection View Nodes.

Table 4: Working With Attributes and Measures

Requirement	Task to perform
If you want to count the count the number of distinct values for a set of attribute columns.	Create Counters
If you want to create new output columns and calculate its values at runtime using an expression.	Create Calculated Columns
If you want to restrict measure values based on attribute restrictions.	Create Restricted Columns
If you want to assign semantic types to provide more meaning to attributes and measures in calculation views.	Assign Semantics
If you want to parameterize calculation views and execute them based on the values users provide at query runtime.	Create Input Parameters
If you want to, for example, filter the results based on the values that users provide to attributes at runtime.	Assign Variables
If you want associate measures with currency codes and perform currency conversions.	Associate Measures with Currency
If you want associate measures with unit of measures and perform unit conversions.	Associate Measures with Unit of Measure
If you want to create level hierarchies to organize data in reporting tools.	Create Level Hierarchies
If you want to create parent-child hierarchies to organize data in reporting tools.	Create Parent-Child Hierarchies
If you want to group related measures together in a folder.	Group Related Measures.

Table 5: Working With Calculation View Properties

Requirement	Task to perform
If you want to filter the view data either using a fixed client value or using a session client set for the user.	Filter Data for Specific Clients
If you want to execute time travel queries on calculation views.	Enable Calculation Views for Time Travel Queries
If you want to invalidate or remove data from the cache after specific time intervals.	Invalidate Cached Content

Related Information

Use Table Functions as a Data Source [page 14]
Supported View Nodes for Modeling Graphical Calculation Views [page 14]
Working With View Nodes [page 15]
Preview Calculation View Output [page 41]
Working With Attributes and Measures [page 44]
Working With Calculation View Properties [page 84]

2.1.1 Create Graphical Calculation Views with Star Joins

Create graphical calculation view with star joins to join multiple dimensions with a single fact table. In other words, you use star joins to join a central entity to multiple entities that are logically related.

Context

Star joins in calculation views help you to join a fact table with dimensional data. The fact table contains data that represent business facts such price, discount values, number of units sold and so on. Dimension tables represent different ways to organize data, such as geography, time intervals, contact names and so on.

Procedure

1. Open the SAP HANA Web-based Development Workbench Editor.
   The Web-based Editor tool is available on the SAP HANA XS Web server at the following URL: `http://<WebServerHost>:80<SAPHANAinstance>/sap/hana/ide/editor`.
2. In the navigator, select a package where you want to create the new calculation view.
3. In the context menu of the package, select *New* > *Calculation View*
4. Provide name and description.

5. Select the *Graphical* radio button.
6. Select *With Star Join.*

   > **Note**
   >
   > You can create star join with data category as cube only.

7. Choose *Create*.

   Modeler launches a new graphical calculation view editor with the semantics node and the star join node.
8. Add data sources.
   a. Select the *Star Join* node.
   b. Choose  .
   c. In *Find Data Sources* dialog, enter the name of the calculation view with descriptive data.

      > **Note**
      >
      > You can only add calculation views with data category types as dimension or blank as a data source in star join node.

   d. Choose *OK*.
9. Add inputs to star join node.

   Continue modeling the graphical calculation view with a cube structure, which includes attributes and measures. The input to the star join node must provide the central fact table.
10. Maintain star join properties.
    a. Select the *Star Join* node.
    b. In the *Join Definition* tab, create joins by selecting a column from one data source, holding the mouse button down and dragging to a column in the central fact table.
    c. Select the join.
    d. In the *Properties* section, define necessary join properties.
11. Activate the calculation view.
    a. If you want to activate the calculation view, in the menu bar, choose *Save*.
    b. If you want to activate the calculation view along with all objects, in the menu bar, choose *Save All*.
12. Assign changes.
    a. In the open change list, select a change ID that you want to use to assign your changes.

       > **Note**
       >
       > The change list appears only if you have enabled change tracking for your SAP HANA system.

    b. Choose *Assign*.

Related Information

Join Properties [page 25]

Supported Join Types [page 26]

2.1.2 Use Table Functions as a Data Source

Table functions as a data source in graphical calculation views helps build calculation views for complex calculation scenarios. Using table functions in graphical calculation views are an alternate to script-based calculation views, which you generally use for building complex calculation views.

Procedure

1. Open the graphical calculation view in the view editor.
2. Select a view node.
3. Choose  .
4. In the *Find Data Sources* dialog, enter the name of the table function and select it from the list.
5. Choose *OK.*

2.1.3 Supported View Nodes for Modeling Graphical Calculation Views

Considering the different business scenario and reporting uses cases, SAP HANA modeler offers different view nodes to model graphical calculation views.

The table below lists the view nodes and its description.

Table 6:

View Node	Description	Icon	Example
Union	Use union node to combine the result set of two or more data sources. Union nodes have two or more inputs.		For example, for retrieving the names of all employees of a store, which has different branches and each branch maintaining its own employee records table.
Join	Use join node to query data from two or more data sources, based on a specified condition. Join nodes have two inputs.		For example, for retrieving customer details and location based on the postal code column present in the two tables CUSTOMER and GEOGRAPHY. The CUSTOMER table has columns – Customer_ID, Customer_Name, Postal_Code, and GEOGRAPHY table has columns – Postal_Code, Region, Country.

View Node	Description	Icon	Example
Projection	Use projection node to filter or obtain a subset of required columns of a table or an information view. Projection nodes have one input.	Proje...	For example, for selecting the employee name and employee department from a table consisting of many other columns.
Aggregation	Use aggregation node to summarize data for a group of row values, by calculating values in a column. Aggregation nodes have one input.	Aggr...	For example, for retrieving total sales of a product in a month. The supported aggregation types are sum, min, and max.
Rank	Use rank node to partition the data for a set of partition columns, and performs an order by operation on the partitioned data. Rank nodes have one input.	Rank	For example, consider a TRANSACTION table with two columns PRODUCT and SALES. If you want to retrieve the top five products based on its sales, then use a rank node.

Note

You can use data sources, union, join, projection, or aggregation view nodes and the inputs to union, join, projection, and aggregation view nodes.

Related Information

Create Graphical Calculation Views [page 9]

2.1.4 Working With View Nodes

View nodes form the buidling blocks of information views. These view nodes help you build complex, flexible and robust analytic models, and each view node type possess specialized capabilities that triggers advanced features in the database.

This section describes the different views nodes that you can use within graphical calculation views, its functionalities and examples on how you can use these views nodes to build calculation views and obtain the desired output.

Related Information

Create Unions [page 26]
Create Rank Nodes [page 30]

Filter Output of Aggregation or Projection View Nodes [page 33]

2.1.4.1 Create Joins

Use join nodes in calculation views to query data from two data sources. The join nodes help limit the number of records or to combine records from both the data sources, so that they appears as one record in the query results.

Procedure

1. Open the required calculation view in view editor.
2. From the editor's tools palette, drag and drop a join node to the editor.
3. Add data sources.
   a. Select the join node.
   b. Choose  .
   c. In the *Find Data Sources* dialog, enter the name of the data source and select it from the list.
   d. Choose *OK*.
4. Define output columns.
   a. Select the *Mapping* tab.
   b. Select the columns you want to add to the output of the join node.
   c. In the context menu, choose *Add To Output*.

   > **Note**
   >
   > If you want to add all columns of a data source to the output, in the context menu of the data source, choose *Add To Output*.

5. Create a join.
   a. In the *Join Definition* tab, create a join by selecting a column from one data source, holding the mouse button down and dragging to a column in the other data source.
   b. Edit join properties.

      In the *Properties* section, define necessary join properties.

Related Information

Create Spatial Joins [page 17]
Text Joins [page 20]
Dynamic Joins [page 21]
Optimize Join Execution [page 24]

Validate Joins [page 24]
Join Properties [page 25]
Supported Join Types [page 26]

2.1.4.1.1 Create Spatial Joins

Create spatial joins by using the join nodes in graphical calculation views to query data from data sources that have spatial data.

Procedure

1. Open the required graphical calculation view in view editor.
2. From the editor's tools pallete, drag and drop a join node to the editor.
3. Add data sources.
   a. Select the join node.
   b. Choose  .
   c. In the *Find Data Sources* dialog, enter the name of the data source with spatial data and select it from the list.
   d. Choose *OK*.
4. Define output columns.
   a. Select the *Mapping* tab.
   b. Select the columns you want to add to the output of the join node.
   c. In the context menu, choose *Add To Output*.

   > **Note**
   >
   > If you want to add all columns from the data source to the output, in the context menu of the data source, choose *Add To Output*.

5. Create a join.
   a. In the *Join Definition* tab, create a join by selecting a column from one data source, holding the mouse button down and dragging to a column in the other data source.

      For spatial joins, you join the two databases tables on columns of spatial data types.
6. Define spatial join properties.

   In the *Spatial Join* section, define the spatial join properties

   a. In the *Predicates* dropdown list, select a predicate value.

   > **Note**
   >
   > If you select *Relate* as the predicate value, in the *Intersection Matrix* value help, select a value. Similarly, if you select *Within Distance* as the predicate value, in the *Distance* value help, select a

value. You can use a fixed value or an input parameter to provide the intersection matrix or the distance values to modeler at runtime.

b. If you want to execute the spatial join only if predicate condition evaluates to true, then in the dropdown list, select *True*.

Related Information

Supported Spatial Predicates [page 18]
Supported Spatial Data Types [page 19]
Join Properties [page 25]

2.1.4.1.1.1 Supported Spatial Predicates

SAP HANA modeler supports the following spatial predicates.

Table 7:

Method	Type	Description
ST_Contains	ST_Geometry	Tests if a geometry value spatially contains another geometry value.
ST_CoveredBy	ST_Geometry	Tests if a geometry value is spatially covered by another geometry value.
ST_Covers	ST_Geometry	Tests if a geometry value spatially covers another geometry value.
ST_Crosses	ST_Geometry	Tests if a geometry value crosses another geometry value.
ST_Disjoint	ST_Geometry	Test if a geometry value is spatially disjoint from another value.
ST_Equals	ST_Geometry	Tests if an ST_Geometry value is spatially equal to another ST_Geometry value.
ST_Intersects	ST_Geometry	Test if a geometry value spatially intersects another value.
ST_Overlaps	ST_Geometry	Tests if a geometry value overlaps another geometry value.

Method	Type	Description
ST_Relate	ST_Geometry	Tests if a geometry value is spatially related to another geometry value as specified by the intersection matrix. The ST_Relate method uses a 9-character string from the Dimensionally Extended 9 Intersection Model (DE-9IM) to describe the pair-wise relationship between two spatial data items. For example, the ST_Relate method determines if an intersection occurs between the geometries, and the geometry of the resulting intersection, if it exists.
ST_Touches	ST_Geometry	Tests if a geometry value spatially touches another geometry value.
ST_Within	ST_Geometry	Tests if a geometry value is spatially contained within another geometry value.
ST_WithinDistance	ST_Geometry	Test if two geometries are within a specified distance of each other.

2.1.4.1.1.2 Supported Spatial Data Types

SAP HANA modeler offers spatial data types in its data model and query language for storing and accessing geospatial data.

SAP HANA modeler supports the following spatial data types::

Table 8:

Data Type	Description
Geometries	The term geometry means the overarching type for objects such as points, linestrings, and polygons. The geometry type is the supertype for all supported spatial data types.
Points	A point defines a single location in space. A point geometry does not have length or area. A point always has an X and Y coordinate. ST_Dimension returns 0 for non-empty points. In GIS data, points are typically used to represent locations such as addresses, or geographic features such as a mountain.
Multipoints	A multipoint is a collection of individual points. In GIS data, multipoints are typically used to represent a set of locations.

Data Type	Description
Linestrings	A linestring is geometry with a length, but without any area. ST_Dimension returns 1 for non-empty linestrings. Linestrings can be characterized by whether they are simple or not simple, closed or not closed. **Simple** means a linestring that does not cross itself. **Closed** means a linestring that starts and ends at the same point. For example, a ring is an example of simple, closed linestring. In GIS data, linestrings are typically used to represent rivers, roads, or delivery routes.
Multilinestring	A multilinestring is a collection of linestrings. In GIS data, multilinestrings are often used to represent geographic features like rivers or a highway network.
Polygons	A polygon defines a region of space. A polygon is constructed from one exterior bounding ring that defines the outside of the region and zero or more interior rings, which define holes in the region. A polygon has an associated area but no length. ST_Dimension returns 2 for non-empty polygons. In GIS data, polygons are typically used to represent territories (counties, towns, states, and so on), lakes, and large geographic features such as parks.
Multipolygons	A multipolygon is a collection of zero or more polygons. In GIS data, multipolygons are often used to represent territories made up of multiple regions (for example a state with islands), or geographic features such as a system of lakes.

2.1.4.1.2 Text Joins

A text join helps obtain language-specific data. It retrieves columns from a text table based on the user's session language.

The text tables contain description for a column value in different languages. For example, consider a `PRODUCT` table that contains `PRODCUT_ID` and a text table `PRODUCT_TEXT` that contains the columns `PRODUCT_ID`, `DESCRIPTION`, and `LANGUAGE`.

PRODUCT

PRODUCT_ID	SALES
1	1000
2	2000
3	4000

PRODUCT_TEXT

PRODUCT_ID	LANGUAGE	DESC
1	E	Description in English.
1	D	Description in German.
2	E	Description in English.
3	E	Description in English.

Create a text join to join the two tables and retrieve language-specific data using the language column `LANGUAGE`. For example, if your session language is E and if you have added all columns to the output of the join node, the output of the text join is:

PRODUCT_ID	LANGUAGE	DESC	SALES
1	E	Description in English.	1000
2	E	Description in English.	2000
3	E	Description in English.	4000

> **Note**
>
> For text joins, always add the text table as the right table.

2.1.4.1.3 Dynamic Joins

After creating a join between two data sources, you can define the join property as dynamic. Dynamic joins improves the join execution process and help reduce the number of records that join node process at run time.

If you define a join as dynamic, engine dynamically defines the join columns based on the columns requested by the client query.

> **Note**
>
> You can set the *Dynamic Join* property only if the two data sources are joined on multiple columns.

The behavior of dynamic joins depends on the client query. This means that, you can improve the join execution process using the dynamic join property if at least one of the join elements is requested by the client query.

Static Join Versus Dynamic Joins

- In static joins, the join condition isn't changed, irrespective of the client query.
- In a dynamic join, if the client query to the join doesn't request a join column, a query run time error occurs. This behavior of dynamic join is different from the static joins.

- Dynamic joins enforces aggregation before executing the join, but for static joins the aggregation happens after the join. This means that, for dynamic joins, if a join column is not requested by the client query, its value is first aggregated, and later the join condition is executed based on columns requested in the client query.

Related Information

Example: Dynamic Joins [page 22]

2.1.4.1.3.1 Example: Dynamic Joins

Consider that you want to evaluate the sales of a product and also calculate the sales share of each product using the below data sources.

SALES

REGION	COUNTRY	SALES
APJ	IND	10
APJ	IND	10
APJ	CHN	20
APJ	CHN	50
EUR	DE	50
EUR	DE	100
EUR	UK	20
EUR	UK	30

PRODUCT

REGION	COUNTRY	PRODUCT
APJ	IND	PROD1
APJ	IND	PROD2
APJ	CHN	PROD1
APJ	CHN	PROD2
EUR	DE	PROD1

REGION	COUNTRY	PRODUCT
EUR	DE	PROD2
EUR	UK	PROD1
EUR	UK	PROD2

So you use a calculation view to join the above two data sources via two different aggregation nodes as inputs to the join node. The aggregation node with the data source `SALES` does not have the `PRODUCT` column but contains total sales for a given region or country.

Now assume that the two aggregation nodes are joined dynamically on the columns, REGION and COUNTRY. The outputs of the join node are columns REGION, PRODUCT, SALES and the calculated columns, TOT_SALES, and SALES_SHARE.

When you execute a client query on the calculation view to calculate the sales share of a product at a region level, the output from the dynamic join and static join is different:

Dynamic Join

REGION	PRODUCT	SALES	TOT_SALES	SALES_SHARE
APJ	PROD1	30	90	.33
APJ	PROD2	60	90	.66
EUR	PROD1	70	200	.35
EUR	PROD2	130	200	.65

Static Join

REGION	PRODUCT	SALES	TOT_SALES	SALES_SHARE
APJ	PROD1	30	90	.78
APJ	PROD2	60	90	1.21
EUR	PROD1	70	200	.73
EUR	PROD2	130	200	1.26

The dynamic join calculates the sales share at the region level by aggregating the sales values before joining the data sources. The static join, on the other hand, first calculates the sales share at the region level and the country level (because the join condition contains both region and country), and then aggregates the resulting sales share after the join is executed.

2.1.4.1.4 Optimize Join Execution

While executing the join, by default, the query retrieves join columns from the database even if you don't specify it in the query. The query automatically includes the join columns into the SQL `GROUP BY` clause without you selecting them in the query.

You can avoid this behavior by using the join property *Optimizing Join Columns*. When this property for a join is set to *True*, only the columns specified in the query are retrieved from the database.

> **Note**
>
> Optimizing join columns is supported only for left outer joins, or text joins (with cardinality 1:1 or N:1), and right outer joins (with cardinality 1:1 or 1:N).

The join optimizer cannot remove attributes of static filters if the filters are defined on join columns for which you have enabled *Optimize Join Columns*. In this case, you can optimize the join column by introducing a dummy projection node between the join and the input node with static filters.

2.1.4.1.5 Validate Joins

SAP HANA modeler allows you to validate the join cardinality and identify whether you have maintained the referential integrity for the join tables.

Prerequisites

You have SELECT privileges on the catalog tables participating in the join to view the join validation status. If the participating catalog tables are virtual tables, then you can view the join validation status only if the user has SELECT privileges on the virtual table and also if the user credential to remote source has SELECT privileges on the remote table.

Context

While defining a join, you can validate the cardinality and identify whether you have maintained referential integrity for the join tables. If you have chosen a cardinality, which is not optimal for the join tables, modeler recommends to you cardinality after analyzing the data in the participating join tables.

> **Note**
>
> Choosing a valid cardinality for your data sources is necessary to avoid incorrect results from the engine, and to achieve better performance. If you are not aware of the optimal cardinality for your join, then it is recommended not to provide any cardinality value.

Procedure

1. Open the graphical calculation view in the view editor.
2. Select the *Join* node.
3. In the *Join Definition* tab, choose the *Validate Join* icon .

Results

After analyzing the participating tables in your join definition, modeler recommends to you an optimal cardinality and also specifies whether you have maintained referential integrity for the join tables. You can choose to modify the join definition based on this recommendation.

> **Note**
>
> The cardinality that SAP HANA modeler recommends is applicable only to the current state of the system. It becomes invalid if you perform any changes to the data or if you transport your calculation view to another system with a different data set.

2.1.4.1.6 Join Properties

After creating a join, define its properties to obtain a desired output when you execute the join.

SAP HANA modeler allows you to define the following join properties.

Join Properties	Description
Join Type	The value of this property specifies the join type used for creating a join. For more information, see Supported Join Types.
Cardinality	The value of this property specifies the cardinality used for creating a join. By default, the cardinality of the join is empty. If you are not sure about the right cardinality for the join tables, it is recommended to not specify any cardinality. Modeler determines the cardinality when executing the join.
Language Column	The value of this property specifies the langue column that modeler must use for executing text joins. For more information, see Text Joins.
Dynamic Join	The value of this property determines whether modeler must dynamically define the columns of the join condition based on the client query. For more information, see Dynamic Joins.

Join Properties	Description
Optimize Join Columns	The value of this property determines whether modeler must retrieve the columns that are not specified in the query from the database. For more information, see Optimize Join Execution.

2.1.4.1.7 Supported Join Types

When creating a join between two tables, you specify the join type. The below table lists the supported join types in SAP HANA modeler.

Table 9:

Join Type	Description
Inner	This join type returns all rows when there is at least one match in both the database tables.
Left Outer	This join type returns all rows from the left table, and the matched rows from the right table.
Right Outer	This join type returns all rows from the right table, and the matched rows from the left table.
Referential	This join type is similar to inner join type, but assumes referential intergrity is maintained for the join tables.
Text Join	This join type is used to obtain language-specific data from the text tables using a language column.
Full Outer Joins	This join type displays results from both left and right outer joins and returns all (matched or unmatched) rows from the tables on both sides of the join clause.

> **Note**
>
> Full outer join type is supported only in new calculation views created with SPS 11 version onwards.

2.1.4.2 Create Unions

Use union nodes in graphical calculation views to combine the results of two or more data sources.

Context

A union node combines multiple data sources, which can have multiple columns. You can manage the output of a union node by mapping the source columns to the output columns or by creating a target output column with constant values.

For a source column that does not have a mapping with any of the output columns, you can create a target output column and map it to the unmapped source columns.

Procedure

1. Open the required graphical calculation view in view editor.
2. From the editor's tools pallete, drag and drop a union node to the editor.
3. Add data sources.
   a. Select the union node.
   b. Choose  .
   c. In the *Find Data Sources* dialog, enter the name of the data source and select it from the list.
   d. Choose *OK*.
4. Define output columns.
   a. Select the *Mapping* pane.
   b. Select the columns you want to add to the output of the union node.
   c. In the context menu, choose *Add To Output*.

   > **Note**
   >
   > If you want to add all columns from the data source to the output, in the context menu of the data source, choose *Add To Output.*

5. Assign constant value.

   This helps to denote the underlying data of the source columns with constant values in the output.

   If you want to assign a constant value to any of the target columns, then

   a. In the *Output Columns* section, select an output column.
   b. In the context menu, choose *Manage Mappings.*
   c. In the *Manage Mappings* section, set the *Source Column* value as blank.
   d. In the *Contant Value* field, enter a constant value.
   e. Choose *OK*.
6. Create a constant output column.

   If you want to create a new output column and assign a constant value to it, then

   a. In the *Output Columns* section, choose  .
   b. In the *Create Target and Manage Mapping* dialog, provide name and data type for the new output column.
   c. Choose *OK*.

   > **Note**
   >
   > By default, the value of the constant output column in null.

Related Information

Example: Constant Columns [page 28]

2.1.4.2.1 Example: Constant Columns

Constant output columns help denote the underlying data from the source columns with constant values in the output. You can also map the unmapped source columns to a constant output column based on the business requirement.

For example, consider that you want to compare the planned sales of each quantity with its actual sales using two data sources with similar structures, `ACTUALSALES` and `PLANNEDSALES`.

ACTUALSALES

SALES_QUANTITY	PRODUCT_ID
5000	P1
3000	P2

PLANNEDSALES

SALES_QUANTITY	PRODUCT_ID
4000	P1
4000	P2

When you use a union node to combine the results of the two data sources, you cannot differentiate the data from these data source.

SALES_QUANTITY	PRODUCT_ID
5000	P1
3000	P2
4000	P1
4000	P2

In such cases, create a constant output column `PLANNED_OR_ACTUAL` and assign the constant value ACTUAL to ACTUALSALES and the constant value PLANNED to PLANNEDSALES.

SALES_QUANTITY	PRODUCT_ID	PLANNEDORACTUAL
5000	P1	ACTUAL

SALES_QUANTITY	PRODUCT_ID	PLANNEDORACTUAL
3000	P2	ACTUAL
4000	P1	PLANNED
4000	P2	PLANNED

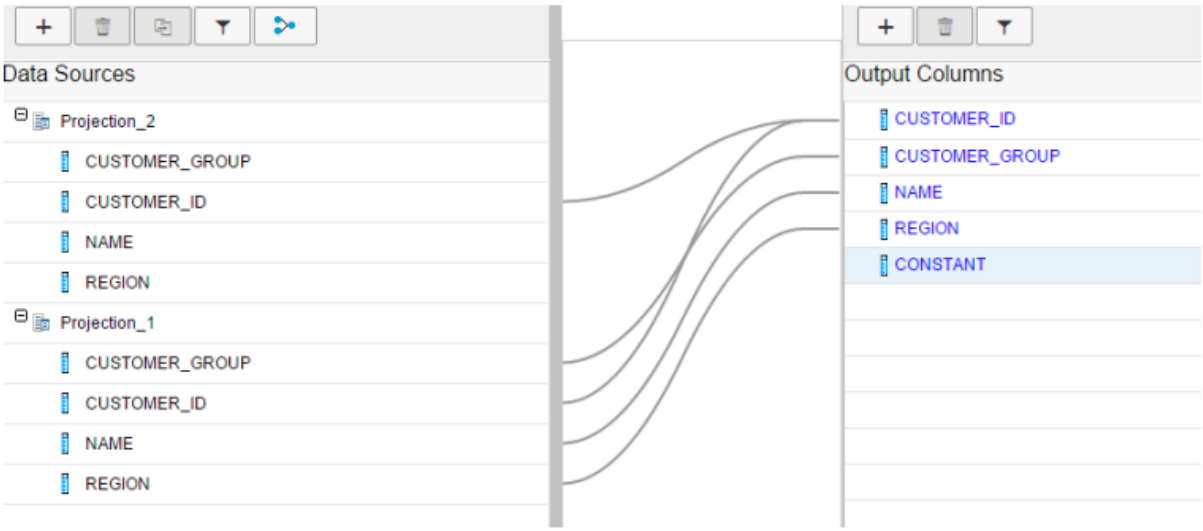
Now, you can identify the data source and its underlying data.

2.1.4.2.2 Empty Union Behavior

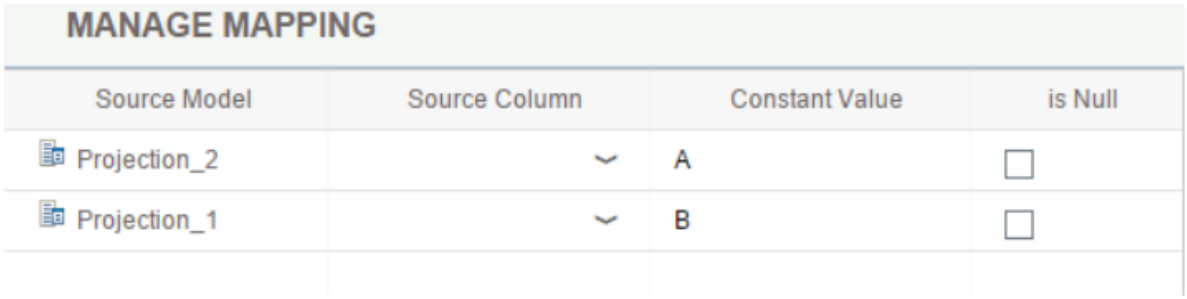
The value of the *Empty Union Behavior* property for a data source in the union node, helps modeler determine whether any queries on union nodes with constant output columns shall return values if no other columns from the data source is queried.

This property is useful, for example, for value help queries in applications. You can select either *No Row* or *Row with Constant* as values for the *Empty Union Behavior* property. Select the data source in the mapping definition and in the *Properties* tab define the values for this property based on your business requirement.

For understanding the *Empty Union Behavior* property and how the value this property determines the output data, consider the following mapping definition in a union node:



Constant values A and B are defined for Projection_1 and Projection_2 using the constant column CONSTANT.



When you execute a query on calculation view with this union node, and if the column CUSTOMER_ID is not queried, the *Empty Union Behavior* property for the Projection_2 data source determines whether the constant column CONSTANT returns the constant value A for Projection_2 in the output.

- If the *Empty Union Behavior* property is set to *No Row*, no data from *Projection _2* appears in the output data. In the other words, only data from *Projection_1* appears in the output data.
- If the *Empty Union Behavior* property is set to *Row with Constant*, then the output data includes one record from *Projection _2*. In this one record, the constant value A appears for the CONSTANT column and values for all other columns appears as null.

2.1.4.3 Create Rank Nodes

Use rank nodes in calculation views to partition the data for a set of partition columns, and perform an `ORDER BY` SQL operation on the partitioned data.

Context

For example, consider a `TRANSACTION` table with two columns `PRODUCT` and `SALES`. If you want to retrieve the top five products based on sales, use a rank node. The rank node first partitions the `TRANSACTION` table with the `PRODUCT` as the partition column, and performs an order by operation on the partitioned table using the `SALES` column to retrieve the top five products based on sales.

Procedure

1. Open the required calculation view in the view editor.
2. From the editor's tools palette, drag the rank node to the editor.
3. Add data source.
   a. Select the rank node.
   b. Choose  .
   c. In the *Find Data Sources* dialog, enter the name of the data source and select it from the list.
   d. Choose *OK*.
4. Define output columns.
   a. On the *Mapping* tab, select the columns you want to add to the output of the rank node.
   b. In the context menu, choose *Add To Output*.

> **Note**
>
> If you want to add all columns from the data source to the output, in the context menu of the data source, choose *Add To Output*.

5. Select the *Definition* tab.
6. Define sort direction.
   a. In the *Sort Direction* dropdown list, select a sort direction.

   Table 10:

Sort Direction	Description
Descending (Top N)	Retrieves top N values from the ordered set where N is the threshold value that you define.
Ascending (Bottom N)	Retrieves bottom N values from the ordered set where N is the threshold value that you define

7. Define a threshold value.

   Use a *Fixed* value or an *Input Parameter* as the threshold value. It helps the tool identify the number of records from the ordered set.

   a. In the *Threshold Value* dropdown list, select a threshold value type and provide the threshold value accordingly.
8. In the *Order By* dropdown list, select a column that modeler must use to perform the order by operation.
9. Partition the data.

   a. In the *Partition By* section, choose  .
   b. In the *Partition By Column* dropdown list, select a partition column that the tool must use to partition the data.

   > **Note**
   >
   > You can partition the data using more than one partition column.

10. If you want to partition the data using only the partition by columns that query requests for processing rank nodes, select the *Dynamic Partition Elements* checkbox.

    > **Note**
    >
    > If you do not select this checkbox, the tool partitions the data with all partition columns that you have added in the *Partition By* section, even if these columns are not requested in the query.

11. Generate a rank column.

    If you want generate an additional output column for the rank node to store the rank values, select the *Generate Rank Column* checkbox.

2.1.4.3.1 Example: Rank Nodes

Rank nodes help you partition the data source for a set of partition columns and then perform an order by SQL operation on the partitioned set.

For example, consider a business scenario where you want to retrieve the top five sales for each product using the SALES_TRANSACTION data source.

SALES_TRANSACTIONS

CUSTOMER_ID	PRODUCT	PRODUCT_SALES
C1	P1	1000
C7	P5	1000
C1	P1	3000
C2	P2	3000
C2	P1	2500
C2	P4	1900
C3	P3	2100
C4	P1	2000
C5	P1	3050
C5	P4	3100
C6	P2	4000

Add SALES_TRANSACTION as a data source within the rank node and define the following:

- Threshold Value = 5
- Partition By Column = PRODUCT
- Order By Column = PRODUCT_SALES

When you execute the rank node, the output of the rank node is:

CUSTOMER_ID	PRODUCT	PRODUCT_SALES
C7	P5	1000
C5	P4	3100
C2	P4	1900
C3	P3	2100
C6	P2	4000

CUSTOMER_ID	PRODUCT	PRODUCT_SALES
C2	P2	3000
C5	P1	3050
C1	P1	3000
C2	P1	2500
C4	P1	2000
C1	P1	1000

2.1.4.4 Filter Output of Aggregation or Projection View Nodes

Apply filters on columns of projection or aggregation view nodes (except the default aggregation or projection node) to filter thier output.

Context

For example, you could apply filters to retrieve sales of a product where (revenue >= 100 AND region = India) OR (revenue >=50 AND region = Germany). You can also define filters using nested or complex expressions.

Filters on columns are equivalent to the `HAVING` clause of SQL. At run time, the system executes the filters after performing all the operations that you have defined in the aggregation, or projection nodes. You can also use input parameters to provide values to filters at run time.

> **Note**
>
> Use the filter expression editor to define filters on columns. You can use the column engine language or the SQL language to define the filter expression.

Procedure

1. Open the required calculation view in the view editor.
2. Select a projection or aggregation node.

   > **Note**
   >
   > You cannot apply filter on columns of the default projection or the default aggregation nodes of calculation views.

3. In the *Filter Expression* tab, select the expression language in the dropdown list.
4. In the expression editor, enter the expression that defines the filter conditions.

   For example,

   ```
   (revenue >= 100 AND region = India) OR (revenue >=50 AND region = Germany)
   ```

   .

2.1.4.4.1 Supported Operators for Filters

The table below lists the operators and its meanings, which you can use while defining filter conditions.

Table 11:

Filter Operator	Description
Equal	To filter and show data corresponding to the filter value
Not Equal	To filter and show data other than the filter value
Between	To filter and show data for a particular range specified in the From Value and To Value
List of Values	To filter and show data for a specific list of values separated by comma
Not in list	To filter data and show data for the values other than the ones specified. You can provide a list of values to be excluded using comma.
Is NULL	To filter and show row data having NULL values
Is not NULL	To filter and show data of all the rows that have non NULL values
Less than	To filter and show data with values less than the one specified as filter value
Less than or Equal to	To filter and show data with values less than or equal to the one specified as filter value
Greater than	To filter and show data with values greater than the one specified as filter value
Greater than or Equal to	To filter and show data with values greater than or equal to the one specified as filter value
Contains Pattern	To filter and show data that matches the pattern specified in the filter value. You can use '?' question mark to substitute a single character, and '*' asterik to substitute many. For example, to filter data for continents that start with letter A, use Contains Pattern filter with value A*. This would show the data for all the continents that start with A like Asia and Africa. The filter `Contains Pattern` in expression editor is converted as `match`. Hence, for the given example the corresponding filter expression is `(match("CONTINENT",'A*'))`.

2.2 Generate Time Data

Generate time data into default time-related tables present in the _SYS_BI schema and use these tables in calculation views to add a time dimension.

Context

For modeling business scenarios that require time dimension, you generate time data in default time-related tables available in the _SYS_BI schema. You can select the calendar type and granularity and generate the time data for a specific time span.

Procedure

1. Open the SAP HANA Web-based Development Workbench Editor.

   The Web-based Editor tool is available on the SAP HANA XS Web server at the following URL: `http://<WebServerHost>:80<SAPHANAinstance>/sap/hana//ide/editor`.

2. In the workbench menu bar, choose the *Menu* icon .
3. In the menu options, choose *Tools* > *Generate Time Data* .
4. In the *Calendar Type* dropdown list, select a calendar type.
5. In the *From Year* and *To Year* textboxes, enter the time range for which you want to generate time data into time-related tables.
6. If you have selected the *Gregorian* calendar type, in the *Granularity* dropdown list select the required granularity.

   > **Note**
   >
   > For the granularity level *Week*, you need to specify the first day of the week.

7. If you have selected the *Fiscal* calendar type,
   a. In *Variant Schema* dropdown list, select a variant schema that contains tables having variant data.

      > **Note**
      >
      > Tables T009 and T009B contain variant data.

   b. Select the required variant.

      The variant specifies the number of periods along with the start and end dates.

8. Choose *Create*.

   > **Note**
   >
   > For the *Gregorian* calendar type, modeler generates time dimension data into M_TIME_DIMENSION_YEAR, M_TIME_DIMENSION_MONTH, M_TIME_DIMENSION_WEEK,

M_TIME_DIMENSION tables and for the *Fiscal* calendar type, the modeler populates the generated time dimension data into the M_FISCAL_CALENDAR table. These tables are present in _SYS_BI schema.

Related Information

Time Range to Generate Time Data [page 36]
Supported Calendar Types to Generate Time Data [page 37]

2.2.1 Time Range to Generate Time Data

For the Gregorian calendar type, based on the granularity you choose, modeler defines certain restrictions on the time range for which you can generate time dimension data.

For each granularity levels, the table below displays the time range for which you can generate time dimension data.

Granularity	Range
Seconds	<= 5 years
Minutes	<= 15 years
Hour	<= 30 years
Day	<= 50 years
Week	<= 50 years
Month	<= 50 years
Year	<= 50 years

Note

The following restrictions are applicable for generating time dimension data:

- Minimum start year: 1900
- Maximum end year: 2200
- Maximum years generated: 50

Related Information

Generate Time Data [page 35]

2.2.2 Supported Calendar Types to Generate Time Data

SAP HANA modeler supports generating time dimension data using the Fiscal or Gregorian calendar types.

The table below provides more information on each of the calendar types.

Table 12:

Calendar Type	Description
Gregorian	Use the Gregorian calendar type, if your financial year is same as the calendar year, for example, January to December.
Fiscal	Use the Fiscal calendar type, if your financial year is not same as the calendar year, for example, March to April.

Related Information

Generate Time Data [page 35]

2.3 Create Script-Based Calculation Views

Create script-based calculation views to depict complex calculation scenarios by writing SQL script statements. It is a viable alternative to depict complex business scenarios, which you cannot achieve by creating graphical calculation views.

Context

For example, if you want to create calculation views that require certain SQL functions (i.e. window), or predictive functions (i.e. R-Lang), then you use script-based calculation views. Sufficient knowledge of SQL scripting including the behavior and optimization characteristics of the different data models is a prerequisite for creating script-based calculation views.

Procedure

1. Open the SAP HANA Web-based Development Workbench Editor.

   The Web-based Editor tool is available on the SAP HANA XS Web server at the following URL: `http://<WebServerHost>:80<SAPHANAinstance>/sap/hana/ide/editor`.
2. In the navigator, select a package where you want to create the new calculation view.

3. In the context menu of the package, select *New* > *Calculation View*.
4. Provide a name and description.
5. Select the *Script* radio button.
6. Choose *Create*.
7. Select default schema
   a. Select the *Semantics* node.
   b. Choose the *View Properties* tab.
   c. In the *Default Schema* dropdown list, select the default schema.

   > **Note**
   >
   > If you do not select a default schema while scripting, then you need to provide fully qualified names of the objects used.

8. If you want to create and add new columns to the output, then in the *Columns* pane, create new columns by typing the column name and its data type using the syntax col_name>:<data_type>.
   a. If you want to edit or define additional properties to these columns, then choose the *Columns* tab.

      For example, you can mark columns as attributes or measures, assign semantics, assign variables and so on.
   b. Define required additional properties.
9. If you want to create or add new input parameters of type *Direct* to the output, then in the *Parameters* pane, create new parameters using the syntax <parameter_name>:<data_type>.
   a. If you want to edit or define additional properties on these parameters, then choose *Parameters/ Variables* tab.

      For example, you can create new parameters or variables, manage parameter mappings, choose parameter or variable types, assign semantics and so on.
   b. Define additional required properties.

   > **Note**
   >
   > The defined order and data types of columns and parameters must match with the order and data types of the columns and parameters in the select query, which is assigned to the output function var_out.

10. If you want to add multiple columns that are part of existing information views or catalog tables or table functions to the output structure of script-based calculation views, then:
    a. In the *Columns* pane, choose *Add Columns From*.
    b. In the *Available Columns* section, choose *Add*.
    c. Enter the name of the object that contains the columns you want to add to the output.
    d. Select one or more objects from the dropdown list.
    e. Choose *OK*.
    f. In the *Available Columns* pane, select the columns that you want to add to the output.
    g. If you want to add all columns of an object to the output, then select the object.
    h. Choose .

> **Note**
>
> For all duplicate column names in the *Select Columns* pane, the modeler displays an error. You cannot add two columns with the same name to your output. If you want to retain both the columns, then change the name of columns in the *Target* pane before you add them to the output.

i. If you want to override the existing output structure, select *Replace existing output columns in the Output*.
j. Choose *OK*.

11. Write the SQL Script statements to fill the output columns.
12. Activate the calculation view.
    a. If you want to activate the calculation view, in the menu bar, choose *Save*.
    b. If you want to activate the calculation view along with all objects, in the menu bar, choose *Save All*.
13. Assign changes.
    a. In the navigator toolbar, choose the  icon or the keyboard shortcut `Ctrl` + `S` to save and activate your calculation view.
    b. In the open change list, select a change ID that you want to use to assign your changes.

> **Note**
>
> The change list appears only if you have enabled change tracking for your SAP HANA system.

    c. Choose *Assign*.

Next Steps

After creating a script-based calculation view, you can perform certain additional tasks to obtain the desired output. The table below lists the additional tasks that you can perform to enrich the calculation view.

Table 13: Working With Attributes and Measures

Requirement	Task to perform
If you want to assign semantic types to provide more meaning to attributes and measures in calculation views.	Assign Semantics
If you want to parameterize calculation views and execute them based on the values users provide at query runtime.	Create Input Parameters
If you want to, for example, filter the results based on the values that users provide to attributes at runtime.	Assign Variables
If you want associate measures with currency codes and perform currency conversions.	Associate Measures with Currency
If you want associate measures with unit of measures and perform unit conversions.	Associate Measures with Unit of Measure
If you want to create level hierarchies to organize data in reporting tools.	Create Level Hierarchies
If you want to create parent-child hierarchies to organize data in reporting tools.	Create Parent-Child Hierarchies

Requirement	Task to perform
If you want to group related measures together in a folder.	Group Related Measures.

Table 14: Working With Calculation View Properties

Requirement	Task to perform
If you want to filter the view data either using a fixed client value or using a session client set for the user.	Filter Data for Specific Clients
If you want to execute time travel queries on calculation views.	Enable Calculation Views for Time Travel Queries
If you want to invalidate or remove data from the cache after specific time intervals.	Invalidate Cached Content

Related Information

Assign Variables [page 57]
Create Input Parameters [page 60]
Create Level Hierarchies [page 67]
Create Parent-Child Hierarchies [page 70]
Group Related Measures [page 82]
Filter Data for Specific Clients [page 84]
Enable Calculation Views for Time Travel Queries [page 85]
Invalidate Cached Content [page 86]

2.4 Supported Data Categories for Information Views

SAP HANA modeler supports two types of data categories to classify calculation views. The table below provides more information on each of these data category types.

Table 15:

Data Category	Description
Cube	Calculation views with data categories as Cube are visible to the reporting tools and supports data analysis with multidimensional reporting. If the data categories for graphical calculation views are set to *Cube*, modeler provides aggregation as the default view node. Also, an additional aggregation column behavior is available that you can use to specify the aggregation types for measures.

Data Category	Description
Dimension	Calculation views with data categories as Dimension are not visible to the reporting tools and does not support data analysis. However, you can use these information views as data sources in other calculation views, which have data category as Cube, for any multidimensional reporting purposes. If the data category is Dimension, you cannot create measures. You can only consume them with SQL. For example, you use such calculation views to fill simple list user interfaces, where recurring attribute values are not a problem, but are desired. The output node offers only attributes of numerical data types. If the data categories for graphical calculation views are set to *Dimension*, modeler provides projection as the default view node. You cannot create script-based calculation views with data category as Dimension.
<blank>	Calculation views with data categories as <blank>, or if the calculation views are not classified as cube or dimension, then they are not visible to the reporting tools and do not support multidimensional reporting. However, you can use these calculation views as data sources in other information views, which have data category as Cube, for any multidimensional reporting purposes. If the graphical calculation views are not classified as cube or dimension, modeler provides projection as the default view node.

Related Information

Create Script-Based Calculation Views [page 37]
Create Graphical Calculation Views [page 9]

2.5 Preview Calculation View Output

After modeling calculation views based on your requirements, deploy them within SAP HANA modeler to preview and analyze the output data. You can also view the SQL query that modeler generates for the deployed calculation view.

Context

Data preview refers to visualizing the output of calculation views in graphical or tabular format. You can preview output of information views within SAP HANA modeler using any of the following preview options:

- Preview raw data output and the SQL query

- Preview output in SAP BusinessObjects Design Studio

Procedure

1. Open the SAP HANA Web-based Development Workbench Editor.

   The Web-based Editor tool is available on the SAP HANA XS Web server at the following URL: `http://<WebServerHost>:80<SAPHANAinstance>/sap/hana/ide/editor`.
2. Open the required calculation view in the view editor.
3. Preview raw data.

   a. In the view editor menu bar, choose  .

      If you have not used any input parameters or variables, this operation opens a new editor and displays the raw data output of all attributes along with its data in a simple table format.

   b. In the menu bar, choose the icon  dropdown.
   c. Choose *Open Content* to view raw data output.
4. Provide values to input parameters and variables.

   If you have defined any variables or input parameters, provide your values.
5. Apply filters.

   a. If you want to apply filters and view filtered data, choose  .
   b. Choose *Add Filters*.
   c. Choose a column and define filter conditions.
6. Export output data.

   If you want to export the raw data output to a .csv file, choose  .
7. View SQL query for the calculation view.

   a. If you want to view the SQL query that modeler generates for the deployed calculation view, choose  .
   b. If you want to modify the SQL query and view output accordingly, modify the query in SQL console and choose  to see output for the modified query.
8. Preview output in SAP BusinessObjects Design Studio.

   Launch SAP BusinessObjects Design Studio, which is integrated within the SAP HANA Web-based Development Workbench Editor to preview and analyze the output of your calculation view.

   a. In the view editor menu bar, choose  .

      This launches SAP BusinessObjects Design Studio in a new tab of your browser.

> **Note**
>
> If there are inconsistencies in runtime information (that is, calculation views in catalog or in tables related to runtime) of an calculation view, you get invalidated view error. In such cases, you need to redeploy the view in order to correct the inconsistencies of runtime information.

Related Information

Create Script-Based Calculation Views [page 37]
Create Graphical Calculation Views [page 9]
SAP BusinessObjects Design Studio [page 43]

2.5.1 SAP BusinessObjects Design Studio

Use SAP BusinessObjects Design Studio to view output data with rich visualizations, and to perform advanced analysis using the labels and value axis. For example, to analyze the sales on a country basis, add Country to the labels axis and Sales to the value axis.

In SAP BusinessObjects Design Studio you can preview calculation view output in different charts such as the column charts, pie charts, and so on. You can also apply filters on attributes and visualize filtered data.

For more information on SAP BusinessObjects Design Studio, see SAP BusinessObjects Design Studio documentation.

3 Working With Attributes and Measures

Attributes and measures form content data that you use for data modeling. The attributes represent the descriptive data such as city and country and the measures represent quantifiable data such as revenue and quantity sold.

Information views can contain two types of columns, the measures and the attributes. Measures are columns for which you define an aggregation. If information views are used in SQL statements, then you have to aggregate the measures, for example, using the SQL functions SUM(<column name>), MIN(<column name>), or MAX(<column name>. Attributes can be handled as regular columns as they do not need to be aggregated.

This section describes the different operations you can perform using the attributes and measures. For example, you can create calculated attributes or calculated measures.

Related Information

Working With Attributes and Measures [page 44]
Using Hierarchies for Reporting [page 66]

3.1 Create Counters

If you want to count the number of distinct values of one or more attribute columns, then you create counters, which are special type of columns that displays the distinct count of attribute columns.

Context

You can create counters for multiple attribute columns at a time. For example, if you create a counter for two columns, then the counter displays the count of distinct combinations of both the columns.

> **Note**
>
> You can create counters for attribute columns in the default star join node or in the default aggregation view node only.

If you want to create counters for columns in your information view, then perform the following sub-steps:

Procedure

1. Open the calculation view in the view editor.
2. Select the default aggregation node or the default star join node.
3. Choose the *Calculated Column/Counters* tab.
4. Choose *Add* > *Counters*
5. In the *General* section, provide counter name and description.
6. If you want to hide the counter for data preview, then select the *Hidden* checkbox.
7. In the *Counter* section, choose *Add*.
8. In the dropdown list, choose an attribute.

Related Information

Counter Properties [page 45]
Example: Counters [page 46]

3.1.1 Counter Properties

After creating a counter, you can view its properties or change them based on your business requirement.

Modeler displays the following properties for counter in the *Semantics* node.

Table 16:

Properties	Description
Data Type	The value of this property specifies the data type of the counter.
Semantic Type	The value of this property specifies the semantics assigned to the counter. For more information, see Assign Semantics [page 51]
Hidden	The value of this property determines whether the counter column is hidden in reporting tools.
Exception Aggregation Type	The value of this property specifies the exception aggregation type used for creating counters. SAP HANA modeler supports only the COUNT_DISTINCT exception aggregation type for counters. This exception aggregation type counts the distinct occurrences of values for a set off attribute columns.

Properties	Description
Display Folder	If the counter is grouped in any of the display folder, the value of this property specifies the display folder that was used to group related measures. For more information, see Group Related Measures [page 82].
Columns	The attribute columns used in the counter. Modeler counts the distinct combinations of these columns in the data source.

3.1.2 Example: Counters

Counters help you count the number of distinct values for one or more of attribute columns.

For example, consider a business scenario where you want count the distinct products in each region.

Consider the sales transaction table, SALES_TRANSACTION with columns PRODUCT_ID and REGION.

SALES_TRANSACTION

PRODUCT_ID	REGION
P1	R1
P2	R1
P3	R2
P4	R3
P5	R4
P6	R4
P7	R1
P8	R1
P9	R2
P10	R3
P11	R4
P12	R4

Create a counter, DISTINCT_PRODUCTS using the attributes REGION and PRODUCT_ID within an aggregation node.

After creating the counter, add the columns PRODUCT_ID and REGION to the output of the aggregation node. When you execute the aggregation node, the output is:

REGION	DISTINCT_COUNT
R1	4
R1	2
R2	2
R3	4

3.2 Create Calculated Columns

Create new output columns and calculate its values at runtime based on the result of an expression. You can use other column values, functions, input parameters or constants in the expression.

Context

For example, you can create a calculated column DISCOUNT using the expression if("PRODUCT" = 'NOTEBOOK', "DISCOUNT" * 0.10, "DISCOUNT"). In this sample expression, you use the function if(), the column PRODUCT and operator * to obtain values for the calculated column DISCOUNT.

Procedure

1. Open the required graphical calculation view in the view editor.
2. Select the view node in which you want to create the calculated column.
3. Choose the *Calculated Columns* tab.
4. Choose  .
5. In the *General* section, enter a name and description for the calculated column.
6. In the *Data Type* dropdown list, select the data type of the calculated column.
7. Enter length and scale based on the selected data type.
8. Select a column type.

   You can create calculated attributes or calculated measures using attributes or measures respectively.

   a. In the *Column Type* dropdown list, select a value.

> **Note**
>
> If you want to create a calculated measure and enable client side aggregation for the calculated measure, select the *Enable client side aggregation* checkbox.
>
> This allows you to propose the aggregation that client needs to perform on calculated measures.

9. If you want to hide the calculated column in reporting tools, select the *Hidden* checkbox.
10. Assign semantics to the calculated column.
    a. In the *Semantic Type* dropdown list, select a semantic value.
11. Provide an expression.
    a. In the *Expression* section, choose the *Expression Editor*.
    b. In the expression editor enter a valid expression.

       Modeler computes this expression at runtime to obtain values of calculated columns.

       For example, if("PRODUCT" = 'NOTEBOOK', "DISCOUNT" * 0.10, "DISCOUNT") which is equivalent to, if attribute PRODUCT equals the string 'ABC' then DISCOUNT equals to DISCOUNT multiplied by 0.10 should be returned. Else use the original value of the attribute DISCOUNT.

       > **Note**
       >
       > You can also create an expression by dragging and dropping the expression elements, operators and functions from the menus to the expression editor.

    c. Choose *Validate Syntax* to validate your expression.
    d. Choose *Back*.

> **Note**
>
> By default, modeler allows you to drilldown calculated attributes in reporting tools. If you want to disable the drilldown property, in the *Drill Down* dropdown list, set the value to blank.

Related Information

Calculated Column Properties [page 48]
Using Functions in Expressions [page 111]
Example: Calculated Measures [page 49]
Example: Calculated Attributes [page 50]

3.2.1 Calculated Column Properties

After creating a calculated attribute or a calculated measure, you can view its properties or change them based on your business requirements.

Modeler displays the following properties for calculated columns in the *Semantics* node.

Table 17:

Properties	Description
Data Type	The value of this property specifies the data type of the calculated attributes or calculated measures.
Semantic Type	The value of this property specifies the semantics assigned to the calculated attributes or calculated measures. For more information, see Assign Semantics [page 51].
Hidden	The value of this property determines whether the calculated column is hidden in reporting tools.
Drill Down Enablement	The value of this property determines whether the calculated attribute is enabled for drill down in reporting tools. If it is enabled, the value of this property specifies the drill down type. For more information, see Enable or Disable Attributes for Drilldown in Reporting Tools [page 80].
Display Folder	If the calculated measure is grouped in any of the display folder, the value of this property specifies the display folder that was used to group related measures. For more information, see Group Related Measures [page 82].

3.2.2 Example: Calculated Measures

Create a new measure column and calculate its value at runtime based on the result of an expression.

For example, consider a business scenario where you want to create a new calculated measure column, PRODUCT_PROFIT_PERCENT within an aggregation node. This measure column stores the profit of a product in percentage.

Consider the sales transaction table, SALES_TRANSACTION with columns, PRODUCT_ID, PRODUCT_COST_PRICE and PRODUCT_SALES_PRICE.

SALES_TRANSACTION

PRODUCT_ID	PRODUCT_COST_PRICE	PRODUCT_SALES_PRICE
P1	30000	32000
P2	32000	24000
P3	40000	41000
P4	10000	11000
P5	14000	13800
P6	18000	17000

Create a new calculated measure, PRODUCT_PROFIT_PERCENT using the expression:

Sample Code

```
(("PRODUCT_SALES_PRICE" - "PRODUCT_COST_PRICE")/"PRODUCT_COST_PRICE")*100
```

Add the columns, PRODUCT_ID, PRODUCT_COST_PRICE, PRODUCT_SALES_PRICE and PRODUCT_PROFIT_PERCENT to the output of aggregation view node. When you execute the node, the output data is:

PRODUCT_ID	PRODUCT_COST_PRICE	PRODUCT_SALES_PRICE	PRODUCT_PROFIT_PERCENTAGE
P1	30000	32000	6.67
P2	32000	24000	-4.00
P3	40000	41000	2.50
P4	10000	11000	10.00
P5	14000	13800	-1.42
P6	18000	17000	-.56

3.2.3 Example: Calculated Attributes

Create a new attribute column and calculate its value at runtime based on the results of an expression.

For example, consider a business scenario where you want to create a new calculated attribute column, PRODUCT_SALES_RATING within an aggregation node. This attribute column stores the rating for sales of a product as either Good Sales or Poor Sales or Average Sales based on the product quantity sold.

Consider the sales transaction table, SALES_TRANSACTION with columns PRODUCT_ID, PRODUCT_QUANTITY_SOLD.

SALES_TRANSACTION

Table 18:

PRODUCT_ID	PRODUCT_QUANTITY_SOLD
P1	50
P2	30
P3	20
P4	25
P5	40

PRODUCT_ID	PRODUCT_QUANTITY_SOLD
P6	10

Create a new calculated attribute, PRODUCT_SALES_RATING using the expression

Sample Code

```
if("PRODUCT_QUANTITY_SOLD" <= 10,'Poor Sales',if("PRODUCT_QUANTITY_SOLD"
<30,'Average Sales','Good Sales'))
```

Add the columns, PRODUCT_SALES, PRODUCT_QUANTITY_SOLD and PRODUCT_SALES_RATING to the output of aggregation view node. When you execute the node, the output data is:

PRODUCT_ID	PRODUCT_QUANTITY_SOLD	PRODUCT_SALES_RATING
P1	50	Good Sales
P2	30	Good Sales
P3	20	Average Sales
P4	25	Average Sales
P5	40	Good Sales
P6	10	Poor Sales

3.3 Assign Semantics

Assigning semantics to attributes and measures helps define the output structure of an information view, as they provide meaning to attributes and measures of an information view.

Context

For example, if you use an attribute PRICE, which contains price data of product, and another attribute CODE which contains currency format, then you can assign semantic type *Currency Code* to the attribute CODE and assign the semantic type *Amount with Currency Code* to the attribute PRICE.

Procedure

1. Open a calculation view in the view editor.
2. Select the *Semantics* node.
3. Choose the *Columns* tab.
4. Select a measure or attribute.
5. Choose the  icon.
6. In the *Semantic Type* dropdown list, choose an appropriate semantic type to assign it to an attribute or measure.

3.3.1 Extract and Copy Semantics From Underlying Data Sources

Defining semantics for calculation views includes defining the output columns of the calculation views (its label, its label column, its aggregation type, and its semantic type) and the hierarchies. While defining the semantics for a calculation view, you can extract and copy the semantic definitions of columns and hierarchies from their underlying data sources.

Context

For example, consider that you are modeling a complex calculation view with multiple underlying data sources and these data source have their own semantic definitions for its columns and hierarchies. In such cases, you can extract and copy the semantic definitions of columns and hierarchies from their underlying data sources to define the semantics of the calculation view. This way of extracting and copying the semantic definitions helps you save the effort of manually defining the semantics of the calculation view.

Procedure

1. Open the required graphical calculation view in view editor.
2. Select the *Semantics* node.
3. Choose the *Columns* tab.
4. Choose  .

   In the *Extract Semantics* dialog, modeler displays the output columns and hierarchies of underlying data sources.
5. Select columns and columns properties.

   If you want to extract and copy semantic definition of columns from thier underlying data sources,

a. In the *Columns* tab, select the columns available in the underlying data sources.

> **Note**
>
> If the same column is available in two or more data sources specify the data source that modeler must use to extract and copy the semantic definition. In the *Data Sources* dropdown list, select the data source.

b. Select the checkbox of those column properties (Label, Label column, Aggregation Type and Semantic Type) that you want to extract and copy to the semantic definition of the calculation view.

6. Select hierarchies.

   If you want to extract and copy hierarchies defined in the underlying views to the semantic definition,

   a. Select the *Hierarchies* tab.

   > **Note**
   >
   > You can extract and copy hierarchies only if the nodes in the hierarchies are available as output columns of the calculation view.

   b. Select the hierarchies defined in the underlying views.
   c. If hierarchies with same name already exist in the calculation view, in the *New Name* field, provide a different name.

7. If you want to override the existing semantic definition of the calculation view with the extracted semantics, in the *Columns* tab, select the *Overwrite semantics already defined* checkbox.
8. Choose *OK*.

3.3.2 Propagate Columns to Semantics

Propagate columns from underlying view nodes to the semantics node and to other view nodes that are in the joined path. In other words, you can reuse the output columns of underlying view nodes in other view nodes up to the semantic node.

Context

Modeler allows to you propagate columns from an underlying view node to all nodes in the joined path up to the semantics node. This helps you to avoid defining the output columns of each node if the same columns are available in its underlying node and you also require them as output columns in the above nodes up to the semantic node. Propagating columns are useful in complex calculation views with many levels of view nodes.

Procedure

1. Open the required graphical calculation view in view editor.

2. Select a view node.

> **Note**
>
> You cannot select the default view node and propagate columns to the semantics node.

3. In the *Mappings* tab, select an output column that you want to propagate to the semantics node.

> **Note**
>
> You can select more than one column using the CTRL key.

4. In the context menu, choose *Propagate to Semantics*.

Results

The modeler propagates the columns you select to all view nodes and up to the semantics node. If a column is already present in any of the view node in the propagated path, the columns are not propagated.

3.3.3 Supported Semantic Types for Attributes

Client tools use semantic types to represent data in appropriate format. The system supports the following semantic types for attributes.

- Amount with Currency Code
- Quantity with Unit of Measures
- Currency Code
- Unit of Measure
- Date
- Date – Business Date From
- Date – Business Date To
- Geo Location - Longitude
- Geo Location - Latitude
- Geo Location - Carto ID
- Geo Location – Normalized Name

3.3.4 Supported Semantic Types for Measures

The following semantic types are supported for measures.

- Amount with Currency Code
- Quantity with Unit of Measures

3.4 Create Restricted Columns

Create restricted columns to restrict values of measures based on attribute restrictions. For example, you can choose to restrict the value for the REVENUE column only for REGION = APJ, and YEAR = 2012.

Context

You can apply restrictions on measures defined in the semantics node by using any of the below approaches:

- Apply restrictions on attribute values by using values from other attribute columns.
- Apply restriction on attribute values using expressions.

> **Note**
>
> For restricted columns, modeler applies the aggregation type of the base column, and you can create restricted columns in the default aggregation view node or star join node only.

Procedure

1. 1. Open the required graphical calculation view in the editor.
2. Select the default aggregation node.

   > **Note**
   >
   > You can also create restricted columns in star join nodes.

3. Choose the *Restricted Columns* tab.
4. Choose  .
5. In the *General* section, provide a name and description to the restricted column.
6. In the *Base Measure* dropdown list, select a measure on which you want to apply restrictions.
7. If you want to hide the restricted column for data preview, select the *Hidden* checkbox.
8. Apply restrictions using column values.

   You can define a condition using attribute columns to apply restrictions on the base measure.

   a. Select *Columns*.
   b. Choose  .
   c. In the *Column* dropdown list, selecte an attribute column.
   d. In the *Operator* dropdown list, select a required operation to define the condition.
   e. In the *Value* field, select a value from the value help.
   f. If you want to apply restrictions only for the defined conditions, choose *Include*.

> **Note**
>
> You can apply restrictions using more than one attribute column.

9. Apply restrictions using expressions.

   If you want to use an expression to apply restrictions on the base measure, then:

   a. Select *Expression*.
   b. Choose *Edit Expression*.
   c. In the expression editor, enter your expression.

   > **Note**
   >
   > You can also use input parameters in your expressions to create restricted columns.

Related Information

Using Functions in Expressions [page 111]
Restricted Column Properties [page 56]
Example: Restricted Columns [page 57]

3.4.1 Restricted Column Properties

After creating a restricted column, you can view its properties or change them based on your business requirements.

Modeler displays the following properties for restricted columns in the *Semantics* node.

Table 19:

Properties	Description
Data Type	The value of this property specifies the data type of the restricted column.
Hidden	The value of this property determines whether the restricted column is hidden in reporting tools.
Display Folder	If the restricted measure is grouped in any of the display folder, the value of this property specifies the display folder that was used to group related measures. For more information, see Group Related Measures [page 82].

3.4.2 Example: Restricted Columns

Restricted columns help you restrict measure values based on attribute restrictions.

For example, consider a business scenario where you want to create a new restricted measure column, REGION_SALES within an aggregation node. This restricted column is used to restrict values of the measure, QUANTITY_SOLD using the attribute, REGION.

Consider the sales transaction table, SALES_TRANSACTION with columns PRODUCT_ID, REGION, COUNTRY and QUANTITY_SOLD.

SALES_TRANSACTION

PRODUCT_ID	REGION	COUNTRY	QUANTITY_SOLD
P1	Europe	DE	3000
P1	Europe	UK	4000
P1	Europe	GR	5000
P1	APJ	India	2000
P1	APJ	CHN	2500
P1	APJ	JAP	3500

Create a restricted column for the measure QUANTITY_SOLD using the attribute restriction, REGION=Europe

After creating the restricted column, add columns, PRODUCT_ID, REGION, QUANTITY_SOLD and the restricted measure, REGION_SALES to the output of the aggregation node. When you execute the aggregation node, the output is:

PRODUCT_ID	REGION	QUANTITY_SOLD	SALES_REGION
P1	Europe	12000	12000
P1	APJ	8000	?

3.5 Assign Variables

Calculation views contain variables that are bound to specific attributes within the calculation view. Variables are runtime filters that help to filter attributes, based on values that users provide.

Context

You assign variables to attributes in calculation views to. for example, filter the results. At run time, you can provide values to variables by manually entering a value or by selecting them from the value help dialog.

Procedure

1. Maintain variable details.
   a. Select the *Semantics* node.
   b. Choose the *Parameters* tab.
   c. Choose  .
   d. Choose the *Variable* menu option.
   e. Provide a name and description for your variable.
2. Define a variable type.
   a. In the *Selection Type* dropdown list, select a variable type.
   b. If you want to configure the variable to mandatorily accept a value at run time, select the *Is Mandatory* checkbox.

      > **Note**
      >
      > If you do not provide a value to variable at run time and if you have not selected the *Is Mandatory* checkbox, then the tool displays unfiltered data.

   c. If you want to configure the variables to accept multiple values from client tools at run time, select the *Multiple Entries* checkbox.

      For example, you can assign variables to identify the revenue for the period 2000 to 2005 and 2012, at run time.
3. Define the value list for value help dialog box.
   a. In the *Reference Column* dropdown list, choose an attribute value. The tool uses this attribute data to provide values in the value help dialog box at run time.

      > **Note**
      >
      > If you want to use attribute data from another calculation view as the reference column, in *View/Table Value Help* dropdown list, select the calculation view that contains the required attribute.

   b. In reporting tools, if you want use a hierarchy to organize the filtered data, then in the *Hierarchy* dropdown list, select a hierarchy.

      > **Note**
      >
      > The hierarchy must contain the variable's reference column at the leaf level (in level hierarchies) or as a parent attribute (in parent-child hierarchies).
4. Provide a default value.

   Provide a default value that modeler must consider as the variable value when you do not provide any value to the variable.

   a. In the *Default Value* section, provide default values using constant values or expressions.

Table 20:

Default Value	Meaning
Constant	If you want to use a constant as the default variable value, do the following: 1. In the *Type* dropdown list, select *Constant*. 2. Provide values for *From Value* or both *From Value* and *To Value* depending on the variable type and the operator. For example, if you are using variable type *Single Value* and operator *Equal*, provide just the *From* value.
Expression	If you want to provide the result of an expression as the default value, do the following: 1. In the *Type* dropdown list, select *Expression*. You use the column engine language or the SQL language to provide the expression. 2. Provide values for *From Value* or both *From Value* and *To Value* depending on the variable type and the operator. For example, if you are using variable type *Single Value* and operator *Equal*, then provide just the *From* value. 3. In the *From Value* field or *To Value* field, choose the value help icon to open the expression editor. 4. In the *Expression Editor*, provide a valid expression. 5. Choose *Validate Syntax* to identify any inconsistencies in the expression. 6. Choose *Back*.

> **Note**
>
> Providing multiple default values.
>
> If you have configured the variable to accept multiple values at the run time by selecting the *Multiple Entries* checkbox, then you can provide multiple default values to the variable. In the *Default Value* section, choose  to add multiple default values. You can select these values from the selection screen when executing the calculation view.

5. Assign variables to attributes.

   Assign the variable to an attribute to filter its data at run time.

   a. In the *Apply Filter* section, choose add icon to add an attribute.
   b. In the *Attribute* value help, select an attribute.

3.5.1 Supported Variable Types

SAP HANA modeling environment supports the following variable types:

Type	Description
Single Value	Use this to filter and view data based on a single attribute value. For example, to view the sales of a product where the month is equal to January.

Type	Description
Interval	Use this to filter and view a specific set of data. For example, to view the expenditure of a company from March to April.
Range	Use this to filter view data based on the conditions that involve operators such as "="(equal to), ">" (greater than), "<" (less than), ">=" (greater than or equal to), and "<=" (less than or equal to). For example, to view the sales of all products in a month where the quantity sold is >= 100..

3.6 Create Input Parameters

Input parameters helps you parameterize calculation views and execute them based on the values you provide to the input parameters at query runtime. The engine considers input parameters as the PLACEHOLDER clause of the SQL statement.

Context

You create an input parameter at design time (while creating your calculation views), and provide value to the engine at runtime and execute calculation views accordingly. For example, if you want your information view to provide data for a specific region, then REGION is a possible input parameter. You can provide value to REGION at runtime.

Procedure

1. If you want to create an input parameter from the *Semantics* node, then
   a. Select the *Semantincs* node.
   b. Choose the *Parameters/Variables* tab.
   c. Choose the icon  dropdown menu.
   d. Choose *Input Parameter* menu option.
2. 2. If you want create an input parameter at view node other than the Semantics node,
   a. Select the view node.
   b. Choose the *Parameters* tab.
   c. Choose  .
3. In the *General* section, provide a name and description to your input parameter.
4. Maintain input parameter details
   a. If you want to configure the input parameter to mandatorily accept a value at runtime, then select the *Is Mandatory* checkbox.
   b. If you want to configure the input parameter to accept multiple values at runtime, then select the *Multiple Entries* checkbox.

For example, you can create input parameter to identify the revenue for the period 2000 to 2005 and 2012, at runtime.

> **Note**
>
> You cannot configure input parameters of type Derived from table and Derived from Procedure/ Scalar functions to mandatorily accept a value or to accept multiple values at runtime.

5. Define input parameter type.
   a. In the *Parameter Type* dropdown list, selct an input parameter type.

Table 21:

Input Parameter Type	Description	Next Steps
Column	At runtime, modeler provides a value help with attribute data. You can choose a value from the attribute data as an input parameter value. You can also choose a hierarchy from the calculation view to organize the data in reporting tools. But, only if the hierarchy contains the variable's reference column at the leaf level (in level hierarchies) or as a parent attribute (in parent-child hierarchies).	a. In the *Reference Column* dropdown list, select an attribute. b. If you want to use attribute data from another calculation view as the reference column, in the *View/Table for value help* dropdown list, select the information view that contains the required attribute. c. If you want use a hierarchy to organize the data in reporting tools, in *Hierarchy* dropdown list, select a hierarchy.
Derived from table	At runtime, modeler uses the value from the table's return column as the input parameter value. This means that, you need not provide any values to the inputparameter at runtime. Input parameters of this type are typically used to evaluate a formula. For example, you calculate a discount for specific clients by creating an input parameter, which is derived from the SALES table and return column REVENUE with a filter set on the CLIENT_ID.	a. In the *Table Name* dropdown list, select a table. b. For the table you select, in the *Return Column* dropdown list, select a column value. c. In the *Filters* section, define filter conditions to filter the values of return column.

Input Parameter Type	Description	Next Steps
Direct	Specify the data type and length and scale of the input parameter value that you want to use at runtime. You can also define an input parameter with semantic type as Currency or Unit of Measure or Date. For example, in currency conversions, you can specify the target currency value at run time by creating an input parameter of type Direct with semantic type as Currency.	b. In the *Data Type* dropdown list, select the data type. c. Provide the *Length* and *Scale* for the data type you choose. a. Optionally, In the *Semantic Type* dropdown list, specify the semantic type for you input parameter.
Static List	At runtime, modeler provides a value help with the static list. You can choose a value from this list as an input parameter value.	a. In the *Data Type* dropdown list, select the data type for the list values. b. Provide the *Length* and *Scale* for the data type you choose. c. In the *List of Values* section, choose the add icon to provide the list values.
Derived from Procedure/Scalar functions	At runtime, modeler uses the value returned from the procedure or scalar function as the input parameter value.	a..In *Procedure/ Scalar Function* textbox, provide the name of procedure or scalar function.

> **Note**
>
> For input parameter of type *Derived from Procedure/Scalar functions* or *Derived From Table*, if you want to provide a different value to the parameter at runtime (override the default value) and do want modeler to automatically use the value returned by the procedure or scalar function or the table as the input parameter, then select the *Input Enabled* checkbox. If this checkbox is enabled, then at runtime modeler displays the value returned by the procedure or scalar function as the default value but, you can override this value based on your requirement.

6. Provide default values

   Provide a default value that modeler must consider as the input parameter value if you do not provide any value to the input parameter at runtime.

   a. In the *Default Value* section, provide default values using constant values or expressions.

Table 22:

Default Value	Meaning
Constant	If you want to use a constant value as the default input parameter value, 1. In the *Default Value* section, choose the add icon. 2. In *Type* dropdown list section, select *Constant*. 3. In Value field, provide a constant value.
Expression	If you want to use the result of an expression as the default input parameter value: 1. In the *Default Value* section, choose the add icon. 2. In *Type* dropdown list section, select *Expression*. 3. In the *Value* field, choose the value help to open the expression editor. 4. In the *Expression Editor*, enter a valid expression. 5. Choose *Validate Syntax*. 6. Choose *Back.*. For example, you can evaluate the expression date(Now()), and use the result as the default input parameter value at runtime.

> **Note**
>
> Providing multiple default constant values. If you have configured the input parameter to accept multiple values at the runtime by selecting the *Multiple Entries* checkbox, then you can provide multiple default constant values to the input parameter. In the *Default Value* section, choose the add icon to add multiple default constant values. These values appear on the selection screen when you execute the calculation view.
>
> You cannot use a combination of expressions and constants as default values for input parameters.

Related Information

Input Parameters [page 65]
Using Functions in Expressions [page 111]

3.6.1 Map Input Parameters or Variables

If you are creating a calculation view by using other calculation views, attribute views or analytic views, which have input parameters or variables defined on it, then you can map the input parameters or variables of the underlying data sources with the input parameters or variables of the calculation view that you are creating.

Context

Similarly, if you are creating a calculation view, which consists of other external views as value help references in its variables or input parameters, then you can map the parameters/ variable of external views with the parameters or variables of the calculation view that you are creating.

> **Note**
>
> SAP HANA modeler for web workbench does not support mapping input paramters of attribute views with the input parameters of calculation views.

Mapping parameters of the current view to the parameters of the underlying data sources, moves the filters down to the underlying data sources during runtime, which reduces the amount of data transferred across them. For value-helps from external views, in addition to parameters, you could also map variables from current view to the external views.

> **Note**
>
> Only if the input parameters are used in the dependent data sources, they are available for mapping.

Procedure

1. Open the calculation view in the editor.
2. Select *Semantics* node.
3. Choose the *Parameters/Variables* tab.
4. Choose *Manage Mapping*.
5. If you are using other data sources in your calculation view and if you want map input parameters of these data sources with the input parameters of the calculation view, then in the *Select Type* dropdown filter, choose *Data Sources*.
6. If you are using input parameters or variables, which refer to external views for value help references and if you want to map input parameters or variables of external views with the input parameters or variables of the calculation view, then then in the *Select Type* dropdown filter, choose *Views for value help for variables/input parameters*.
7. Manage mappings for the source and target's input parameters or variables by selecting a value from the source, holding the mouse button down and dragging to a value in the target.
8. If you want auto-map based on the source and target input parameter or variable names, then:

a. In the toolbar, choose » .
b. Choose *Auto Map*.

> **Note**
>
> If you are choosing *Auto Map*, then for mapping all unmapped parameters or variables at the source, the system creates an input parameter or variable of the same name at the target.

9. If you want to create a constant value at the target calculation view, then:
   a. Select *Create Constant*.
   b. Enter constant value.
   c. Choose *OK*.

> **Note**
>
> If you want to map input parameters of type derived from table or derived from procedure or scalar function, which are input enabled, then you can only map them to a constant value of the target calculation view.

3.6.2 Input Parameters

You use input parameters to define internal parameterization of the view to obtain a desired functionality when you run the view.

This means that the engine needs to know and use the parameter value, for example, calculate a formula for a calculated measure. The parameter value is passed to the engine through the PLACEHOLDER clause of the SQL statement. Normally, a parameter can only have a single value, for example, for currency conversion. However, when working with the **in() function** in filter expressions of the calculation views, you can pass several values as an **IN List**. The quoting must be followed as shown here:

For numerical type parameters

The filter expression of a calculation view CV1 is defined as follows:

```
in("attr", $$param$$)
```

Then you need to pass several values as:

```
select ... from CV1( 'PLACEHOLDER' = ('$$var$$' = 'VAL1,VAL2,VAL3')
```

For string type parameters

The filter expression of a calculation view CV1 is defined as:

```
in("attr", $$param$$)
```

Then you need to pass several values (with double quotes) as:

```
select ... from CV1( 'PLACEHOLDER' = ('$$var$$' = '''VAL1'',''VAL2'',''VAL3''')
```

You use input parameters as placeholders during currency conversion, unit of measure conversion, or in calculated column expressions. When used in formulas, the calculation of the formula is based on the input that you provide at run time during data preview.

The expected behavior of the input parameter when a value at run time is not provided is as follows:

Table 23:

Default Value	Expected Behavior
Yes	Calculates the formula based on the default value
No	Results in error

The table implies that it is mandatory to provide a value for the input parameter at run time, or assign a default value while creating the view, to avoid errors.

3.7 Using Hierarchies for Reporting

SAP HANA modeler helps create hierarchies to organize data in a tree structure for multidimensional reporting. Each hierarchy comprises of a set of levels having many-to-one relationships between each other and collectively these levels make up the hierarchical structure.

For example, a time hierarchy comprises of levels such as Fiscal Year, Fiscal Quarter, Fiscal Month, and so on. You can create the following two types of hierarchies in SAP HANA Modeler:

- Level Hierarchies
- Parent-child Hierarchies

> **Note**
>
> Hierarchies in attribute views are not available in a calculation view that reuses the attribute view.

Additional information is available for using Multidimensional Expressions (MDX) in the *SAP HANA Developer Guide (for SAP HANA Web Workbench)*. The link is included in *Related Information*.

Related Information

Create Level Hierarchies [page 67]
Create Parent-Child Hierarchies [page 70]
SAP HANA Developer Guide (for SAP HANA Web Workbench)

3.7.1 Create Level Hierarchies

In level hierarchies each level represents a position in the hierarchy. For example, a time dimension can have a hierarchy that represents data at the month, quarter, and year levels.

Context

Level hierarchies consist of one or more levels of aggregation. Attributes roll up to the next higher level in a many-to-one relationship and members at this higher level roll up into the next higher level, and so on, until they reach the highest level. A hierarchy typically comprises of several levels, and you can include a single level in more than one hierarchy. A level hierarchy is rigid in nature, and you can access the root and child node in a defined order only.

Procedure

1. Open the SAP HANA Web-based Development Workbench Editor.

   The Web-based Editor tool is available on the SAP HANA XS Web server at the following URL: `http://<WebServerHost>:80<SAPHANAinstance>/sap/hana/ide/editor`.
2. Open the required graphical calculation view in the view editor.
3. Select the *Semantics* node.
4. Choose the *Hierarchies* tab.
5. Choose the icon  dropdown.
6. Choose the *Level Hierarchy* menu option.
7. In the *General* section, provide hierarchy name and description.
8. Define node style

   The node style determines the node ID for the level hierarchy.

   a. In the *Node Style* dropdown list, select a value.
9. Create levels.

   a. In the *Nodes* section, choose  to create a level.
   b. In the *Column* dropdown list, select a column value for each level.
   c. In *Level Type* dropdown list, select a required level type.

      The level type specifies the semantics for the level attributes. For example, level type *TIMEMONTHS* indicates that the attributes are months such as, "January", February, and similarly level type *REGULAR* indicates that the level does not require any special formatting.
   d. In the *Order BY* dropdown list, select a column value that modeler must use to order the hierarchy members.
   a. In the *Sort Direction* dropdown list, select a value that modeler must use to sort and display the hierarchy members.

10. Define level hierarchy properties.
    a. In the *Properties* section, select *Aggregate All Nodes* only if you want to include the values of intermediate nodes of the hierarchy to the total value of the hierarchy's root node (Level 1). If you do not select *Aggregate All Nodes*, the modeler does not roll-up the values of intermediate nodes to the root node (Level 1).

    > **Note**
    >
    > The value of *Aggregate All Nodes* property is interpreted only by the SAP HANA MDX engine. In the BW OLAP engine, the modeler always counts the node values. Whether you want to select this property depends on the business requirement. If you are sure that there is no data posted on aggregate nodes, you should set the option to false. The engine then executes the hierarchy faster.

    b. If you want the level hierarchy to support multiple parents for its elements, select the *Multiple Parent* checkbox.
    c. In the *Default Member* textbox, enter a value for the default member.

       This value helps modeler identify the default member of the hierarchy. If you do not provide any value, all members of hierarchy are default members.
    d. In the *Root Node Visibility* dropdown list, select a value.

       The value helps modeler know if it needs to add an additional root node to the hierarchy.
    e. In the *Orphan Nodes* dropdown list, select a value.

       This value helps modeler know how to handle orphan nodes in the hierarchy.

> **Note**
>
> If you select *Step Parent* option to handle orphan nodes, enter a value (node ID) for the step parent node. The step parent node must already exist in the hierarchy at the root level and you must enter the node ID according to the node style that you select for the hierarchy. For example if you select node style *Level Name*, the stepparent node ID can be [Level2].[B2]. The modeler assigns all orphan nodes under this node.

Related Information

Level Hierarchy Properties [page 68]
Node Style [page 69]
Root Node Visibility [page 73]
Orphan Nodes [page 73]

3.7.1.1 Level Hierarchy Properties

Based on your business requirements, you can define certain properties of level hierarchies. The value of these properties determines the characteristics of the hierarchy at runtime.

Open a level hierarchy to view the following properties in the *Properties* section.

Table 24:

Properties	Description
Aggregate All Nodes	The value of this property determines whether modeler must roll-up the value of intermediate nodes of the hierarchy to the root node of the hierarchy. If the value is set to True, modeler rolls-up the value of intermediate nodes to the total value of the hierarchy's root node.
Default Member (English)	This value of this property helps modeler identify the default member of the hierarchy. If you do not provide any value, all members of hierarchy are default members.
Root Node Visibility	The value of this property helps modeler know if it needs to add an additional root node to the hierarchy. For more information, see Root Node Visibility [page 73].
Node Style	The value of this property specifies the node ID format of the level hierarchy. For more information, see Node Style [page 69]

3.7.1.2 Node Style

Node style is applicable for level hierarchies, and helps modeler identify the format the node ID. For example, if the node ID must comprise of the level name and the node name in the reporting tools.

Table 25:

Node Style	Description
Level Name	For this node style, the node ID comprises of the level name and the node name. For example, for a fiscal hierarchy, the Level Name node style implies: MONTH.JAN
Name Only	For this node style, the node ID comprises of the level name only. For example, for a fiscal hierarchy, the Name Only node style implies: JAN
Name Path	For this node style, the node ID comprises of the node name and the names of all ancestors apart from the (single physical) root node. For example, for a fiscal hierarchy, the Level Name node style implies: FISCAL_2015.QUARTER_1.JAN

3.7.2 Create Parent-Child Hierarchies

In parent-child hierarchies, you use a parent attribute that determines the relationship among the view attributes. Parent-child hierarchies have elements of the same type and do not contain named levels.

Context

Parent-child hierarchies are value-based hierarchies, and you create a parent-child hierarchy from a single parent attribute. You can also define multiple parent-child pairs to support the compound node IDs. For example, you can create a compound parent-child hierarchy that uniquely identifies cost centers with the following two parent-child pairs:

- CostCenter and ParentCostCenter and
- ControllingArea and ParentControllingArea,

A parent-child hierarchy is always based on two table columns and these columns define the hierarchical relationships amongst its elements. Others examples of parent-child hierarchies are bill of materials hierarchy (parent and child) or employee master (employee and manager) hierarchy.

Procedure

1. Open the SAP HANA Web-based Development Workbench Editor.

   The Web-based Editor tool is available on the SAP HANA XS Web server at the following URL: `http://<WebServerHost>:80<SAPHANAinstance>/sap/hana/ide/editor`.
2. Open the required graphical calculation view in the view editor.
3. Select the *Semantics* node.
4. Choose the *Hierarchies* tab.
5. Choose the icon  dropdown.
6. Choose the *Parent Child Hierarchy* menu option.
7. In the *General* section, provide hierarchy name and description.
8. Create parent-child elements

   a. In the *Nodes* section, choose  .
   b. In the *Child column* dropdown list, select a column value as the child attribute.
   c. In the *Parent column* dropdown list, select a column value as a parent attribute for the child column that you have selected.
   d. If you want to place orphan nodes in the hierarchy under a step parent node, then in the *Step Parent* column dropdown list, enter a value (node ID) for the step parent node.
   e. If you want to place the parent-child hierarchies under a root node, in the *Root Node* value help, select a value.
9. Define parent-child hierarchy properties.

a. a. In the *Properties* section, select *Aggregate All Nodes* only if you want to include the values of intermediate nodes of the hierarchy to the total value of the hierarchy's root node. If you do not select *Aggregate All Nodes*, modeler does not roll-up the values of intermediate nodes to the root node.

> **Note**
>
> The value of *Aggregate All Nodes* property is interpreted only by the SAP HANA MDX engine. In the BW OLAP engine, the modeler always counts the node values. Whether you want to select this property depends on the business requirement. If you are sure that there is no data posted on aggregate nodes, you should set the option to false. The engine then executes the hierarchy faster.

b. If you want the parent-child hierarchy to support multiple parents for its elements, select the *Multiple Parent* checkbox.
c. In the *Default Member* textbox, enter a value for the default member.

   This value helps modeler identify the default member of the hierarchy. If you do not provide any value, all members of hierarchy are default members.
d. In the *Root Node Visibility* dropdown list, select a value.

   The value helps modeler know if it needs to add an additional root node to the hierarchy.
e. In the *Orphan Nodes* dropdown list, select a value.

   This value helps modeler know how to handle orphan nodes in the hierarchy.

> **Note**
>
> If you select Step Parent option to handle orphan nodes, enter a value (node ID) for the step parent node. The step parent node must already exist in the hierarchy at the root level.

10. If you want to add additional attributes to execute the hierarchy, then
    a. In *Additional Attributes* section, choose the add icon.
    b. In the dropdown list, select an attribute value.
11. Order and sort hierarchy elements.

    If you want to order and sort elements of a parent child hierarchy based on a column value,

    a. In the *Order By* section, choose the add icon.
    b. In the *Order By* dropdown list, select a column value that modeler must use to order the hierarchy members.
    c. In *Sort Direction* dropdown list, select a value that modeler must use to sort and display the hierarchy members.
12. Enable hierarchy for time dependency

    If elements in your hierarchy are changing elements (time-dependent elements), you can enable your parent-child hierarchy as a time dependent hierarchy. In other words, if you are creating hierarchies that are relevant for specific time period, then enable time dependency for such hierarchies. This helps you display different versions on the hierarchy at runtime.

> **Note**
>
> Not all reporting tools support time dependent hierarchies. For example, time dependent hierarchies does not work with BI clients such as MDX or Design Studio.

a. In the *Time Dependency* section, select the *Enable Time Dependency* checkbox.
b. In the *Valid From Column* dropdown list, select a column value.
c. In the *Valid To Column* dropdown list, select a column value.

   SAP HANA modeler uses *Valid From Column* and *Valid To Column* values as the validity time for the time dependent hierarchies.

13. If you want to use an input parameter to specify the validity of the time dependent hierarchy at runtime,
    a. Select *Interval*.
    b. In the *From Date Parameter* dropdown list, select an input parameter that you want to use to provide the valid from date at runtime.
    c. In the *To Date Parameter* dropdown list, select an input parameter that you want to use to provide the valid to date at run time.
14. If you want to use an input parameter to specify the key date at run time,
    a. Select *Key Date*.
    b. In the *Key Date Parameter* dropdown list, select an input parameter value that you want to use to provide key date value at runtime.

Related Information

Parent-Child Hierarchy Properties [page 72]
Root Node Visibility [page 73]
Orphan Nodes [page 73]

3.7.2.1 Parent-Child Hierarchy Properties

Based on your business requirements, you can define certain properties of parent-child hierarchies. The value of these properties determines the characteristics of the hierarchy at runtime.

Open a parent-child hierarchy to view the following properties in the *Properties* section.

Table 26:

Properties	Description
Aggregate All Nodes	The value of this property determines whether modeler must roll-up the value of intermediate nodes of the hierarchy to the root node of the hierarchy. If the value is set to True, modeler rolls-up the value of intermediate nodes to the total value of the hierarchy's root node.
Default Member (English)	This value of this property helps modeler identify the default member of the hierarchy. If you do not provide any value, all members of hierarchy are default members.
Root Node Visibility	The value of this property helps modeler know whether it needs to add an additional root node to the hierarchy. For more information, see Root Node Visibility [page 73].

3.7.3 Root Node Visibility

Based on your business requirement, choose to add an additional root node to the hierarchy and place all other nodes as its descendants.

Table 27:

Root Node Visibility	Description
Add Root Node If Defined	This applicable only for parent-child hierarchies. Modeler adds a root node only if you have defined a root node value, while creating the parent child hierarchy.
Add Root node	The modeler adds an additional root node to the hierarchy and all other nodes are placed as descendants to this node. Select this value if your hierarchy does not have a root node, but needs one for reporting purposes. Modeler creates a root node with the technical name ALL.
Do Not Add Root Node	The modeler does not add an additional root node to the hierarchy.

3.7.4 Orphan Nodes

For orphan nodes in a hierarchy, SAP HANA modeler provides different options to handle them. For example, you can treat orphan nodes as root nodes or treat them as errors.

Table 28:

Options	Description
Root Node	Treat orphan nodes as root nodes.
Error	Stop processing and hierarchy and show an error.
Ignore	Ignore orphan nodes.
Step Parent	Put orphan nodes under a step parent node.

3.8 Using Currency and Unit of Measure Conversions

If measures in your calculation views represent currency or unit values, associate them with currency codes or unit of measures. This helps you display the measure values along with currency codes or unit of measures at data preview or in reporting tools.

Associating measures with currency code or unit of measure is also necessary for currency conversion or unit conversions respectively.

Modeler performs currency conversions based on the source currency value, target currency value, exchange rate, and date of conversion. Similarly, it performs unit conversions based on the source unit and target unit.

Use input parameters in currency conversion and unit conversion to provide the target currency value, the exchange rate, the date of conversion or the target unit value at runtime.

3.8.1 Associate Measures with Currency

If measures in your calculation views or analytic views represent currency or unit values, associate them with currency codes or unit of measures. This helps you display the measure values along with currency codes or unit of measures at data preview or in reporting tools.

Prerequisites

You have imported the currency tables TCURC, TCURF, TCURN, TCURR, TCURT, TCURV, TCURW, and TCURX.

Context

Associating measures with currency codes is also necessary for currency conversions. For example, consider that you want to generate a sales report for a region in a particular currency code and you have the sales data in the database table with a different currency code. In such cases, create a calculation view by using the table column containing the sales data in different currency as a measure and associate the measure with your desired currency to perform currency conversion. Activate the calculation view to generate required reports.

Procedure

1. Open the required graphical calculation view in the view editor.
2. Select the *Semantics* node.
3. In the *Columns* tab, select a measure to associate it with currency code.
4. Choose the icon  dropdown list.
5. Choose the *Assign Semantics* menu option.
6. In the *Semantic Type* dropdown list, select *Amount with Currency Code*.
7. Select a display currency code.

   Modeler displays the measure values with this currency code in reporting tools.

   a. In the *Currency* field, choose the value help.
   b. In the *Type* dropdown list, select a value.

Table 29:

Value	Description
Fixed	Associate the measure with a currency code available in the currency table TCURC.
Column	Associate the measure with an attribute column available in the information view.

c. Provide values based on the selected currency type.
d. Choose *OK*.

8. Enabling decimal shift.

   By default the precision of all values is 2 digits in SAP ERP tables. As some currencies require accuracy in value, modeler shifts the decimal points according to the settings in the TCURX currency table. For example, if the source currency has 0 valid digits, then each value needs to be multiplied by 100 because in SAP ERP systems values are stored using 2 digits.

   a. If you want to enable a decimal shift for the source currency that you select, select the *Decimal shift* checkbox.

9. Enabling conversion.
   a. If you want to convert the measure value to another currency, select the *Conversion* checkbox.
10. Enabling rounding.
   a. If you want to round the result value after currency conversion to the number of digits of the target currency, select the *Rounding* checkbox.

> **Note**
>
> You should use this step carefully if subsequent aggregations will take place on the number as rounding errors could accumulate.

11. Enabling decimal shift back.

   Decimal shift back is necessary if the result of the calculation views are interpreted in ABAP. The ABAP layer, by default, always executes the decimal shift. In such cases, decimal shift back helps avoid wrong numbers due to a double shift.

   a. If you want to shift back the result of a currency conversion according to the decimal places that you use for the target currency, select *Decimal shift back*.

12. If you have enabled conversion to convert a measure value to another, provide details for conversion.
   a. In the value help of *Schema for currency conversion*, select the required schema that has the currency tables necessary for conversion.
   b. In the value help of *Client for currency conversion*, select the required value that modeler must use for currency conversion rates.

   Table 30:

Value	Description
Fixed/ Session Client	Fixed client value or to select a session client for currency conversions. Provide the required value in the value help.
Column	Attribute column available in the calculation view to provide the client value. Select the required value from the value help.

Value	Description
Input Parameter	Input parameter to provide the client value to modeler at runtime. Select the required input parameter from the value help.

c. Specify the source currency.

In the value help of *Source Currency*, select the required value.

Table 31:

Value	Description
Fixed	Select the source currency from the currency table TCURC. Provide the required value in the value help.
Column	Attribute column available in the calculation view to provide the source currency value. Select the required value from the value help.

d. Specify the target currency.

In the value help of *Target Currency*, select the required value.

Table 32:

Value	Description
Fixed	Select the target currency from the currency table TCURC. Provide the required value in the value help.
Column	Attribute column available in the calculation view to provide the target currency value. Select the required value from the value help.
Input Parameter	Input parameter to provide the target currency value to modeler at runtime. Select the required input parameter from the value help.

e. Specify the exchange rate type.

In the value help of *Exchange Type*, select the required value.

Table 33:

Value	Description
Fixed	Select the exchange rate from the currency table TCURC. Provide the required value in the value help.
Column	Attribute column available in the calculation view to provide the exchange rate value. Select the required value from the value help.
Input Parameter	Input parameter to provide the exchange rate value to modeler at runtime. Select the required input parameter from the value help.

f. Specify the date for currency conversion.

In the value help of *Conversion Date*, select the required value.

Table 34:

Value	Description
Fixed	Select the fixed conversion date from the calendar.

Value	Description
Column	Attribute column available in the calculation view to provide the date for currency conversion. Select the required value from the value help.
Input Parameter	Input parameter to provide the date for currency conversion to modeler at runtime. Select the required input parameter from the value help.

g. If you want to use value from a column in the information view to specify the exchange rate, in *Exchange Rate* value help, select a column value.

13. Provide the data type of value after currency conversion.
    a. In the *Data Type* dropdown list, select the required data type.
    b. Provide the length and scale for the selected data type.
14. Generate result currency column.
    a. If you want the modeler to generate a column to store the result currency conversion values, select the *Generate result currency column* checkbox.

> **Note**
>
> The result currency column is not available in reporting tools. You can only consume them using other calculation views to perform any calculations.

15. Error handling.

    In the *Upon Conversion Failure* dropdown list, select the required value that specifies how modeler must populate data if conversion fails.

    Table 35:

Value	Description
Fail	Modeler displays error for conversion failures at data preview.
Set to NULL	Modeler sets the values for corresponding records to NULL at data preview.
Ignore	Modeler displays unconverted value for the corresponding records at data preview.

16. Choose *OK*.

3.8.2 Associate Measures with Unit of Measure

If measures in calculation views represent unit values, associate the measures with a unit of measure. This helps you display the measure values along with the unit of measures at data preview or in reporting tools.

Prerequisites

You have imported the unit tables T006, T006D, and T006A.

Context

Associating measures with unit of measures is also necessary for unit conversions. For example, if you want to convert a unit of a measure from cubic meters to barrels to perform volume calculations, then associate the unit of measure with the sematic type Quantity with Unit of Measure and perform unit conversions.

Procedure

1. Open the required graphical calculation view in the view editor.
2. Select the *Semantics* node.
3. In the *Columns* tab, select a measure to associate it with currency code.
4. Choose the icon  dropdown list.
5. Choose the *Assign Semantics* menu option.
6. In the *Semantic Type* dropdown list, select *Quantity with Unit Of Measure*.
7. Select a unit.

   Modeler displays the measure values with this unit in reporting tools.

   a. In the *Display Unit* dropdown list, select a value.

   Table 36:

Value	Description
Fixed	Associate the measure with a unit of measure available in the unit tables T006, T006A or T006D.
Column	Associate the measure with an attribute column available in the calculation view.

   b. In the value help, select the required value based on the selected unit type.
   c. Choose *OK*.

8. If you want to convert the unit value to another unit, select the *Enable for Conversion* checkbox.
   a. In the value help of *Schema for Unit Conversion*, select the required schema that has the unit tables necessary for conversion.
   b. In the *Client for Currency Conversion* dropdown list, select the required value that modeler must use for unit conversion factors.

   Table 37:

Value	Description
Fixed	Fixed client value or to select a session client for unit conversion factors. Provide the required value in the value help.
Column	Attribute column available in the calculation view to provide the client value. Select the required value from the value help.

Value	Description
Input Parameter	Input parameter to provide the client value to modeler at runtime. Select the required input parameter from the value help.

c. Specify the source unit.

In the *Source Unit* dropdown list, select the required value.

Table 38:

Value	Description
Fixed	Select the source unit from the unit tables T006, T006A or T006D. Provide the required value in the value help.
Column	Attribute column available in the calculation view to provide the source unit value. Select the required value from the value help.

d. Specify the target unit.

In the *Target Unit* dropdown list, select the required value.

Table 39:

Value	Description
Fixed	Select the target unit from the unit tables T006, T006A or T006D. Provide the required value in the value help.
Column	Attribute column available in the calculation view to provide the target unit value. Select the required value from the value help.
Input Parameter	Input parameter to provide the target unit value to modeler at runtime. Select the required input parameter from the value help.

9. Generate result unit column.

a. If you want the modeler to generate a column to store the result unit conversion values, select the *Generate result unit column* checkbox.

> **Note**
>
> The result unit column is not available in reporting tools. You can only consume them using other calculation views to perform any calculations.

10. Error handling.

In the *Upon failure* dropdown list, select the required value that specifies how modeler must populate data if conversion fails.

Table 40:

Value	Description
Fail	Modeler displays error for conversion failures at data preview.
Set to NULL	Modeler sets the values for corresponding records to NULL at data preview.
Ignore	Modeler displays unconverted value for the corresponding records at data preview.

11. Choose *OK*.

3.9 Enable or Disable Attributes for Drilldown in Reporting Tools

By default, the tool lets you drilldown the attributes or calculated attributes in the reporting tools. You can disable this behavior for selected attributes.

Procedure

1. Open the required calculation view in the view editor.
2. Select the *Semantics* node.
3. Select an attribute.
4. In the *Drill Down Enablement* dropdown list, select a value.

   Set the value to blank if you want to disable drilldown for the selected attribute.

3.9.1 Supported Drilldown Types for Attributes

Enable attributes in informatio views for drilldown or disable them for drilldown in reporting tools.

SAP HANA modeler supports the following drilldown types for attributes in calculation views.

Table 41:

Drilldown Type	Description
<blank>	Attributes are not available for drilldown operations and the tool does not generate an additional flat hierarchy.
Drill Down	Attributes appear as a separate dimension in the reporting tools and is available for drilldown operations.
Drill Down with flat hierarchy (MDX)	This drilldown option is available for the attributes in calculation views (with data category as dimension) and also for the attributes in attribute views. The attributes are available for drilldown in reporting tools and the tool generates an additional flat hierarchy for this attribute. In this hierarchy, all the distinct attribute values make up the first and the only level of the hierarchy. The hierarchy enables the attribute for drilldown in MDX based tools.

3.10 Assign Value Help for Attributes

If you are using attribute data to provide values to variables and input parameters at runtime, you can assign a value help to that attribute in order to use values from other attributes, which are available within the same calculation view or in other tables or other calculation views.

Context

For example, consider you have defined an input parameter in calculation view CV1 using the attribute CUSTOMER_ID. If you want to provide values to the input parameter using the attribute CUSTOMER_ID of calculation view CV2, then assign the value to attribute in CV1 with the reference column CUSTOMER_ID of CV2.

Procedure

1. Select the *Semantics* node.
2. Choose the *Columns* tab.
3. Select an attribute.
4. Choose the icon  dropdown.
5. Choose the *Assign Value Help* menu option.
6. Select a calculation view or table.
   a. In the *View/Table Value Help* field, select the calculation view or table that you want to use for providing values.
7. Select an attribute.

   Modeler displays attributes that are available in the selected table or calculation view.

   a. a. In the *Reference Column* dropdown list, select an attribute.

Results

At runtime, modeler provides a value help that has values from the selected attribute. You can use these values for input parameters and variables.

3.11 Add Descriptions to Attributes

In an information view, you can associate an attribute or a column having texts, as a label column to another attribute or column.

Context

Based on user settings like KEY, TEXT, KEY(TEXT), TEXT(KEY), some of the reporting tools displays attribute or dimension values in combination with their texts, For such scenarios, in your information view, you can associate an attribute having texts, for example, PRODUCT TEXT, as a label column to another attribute, for example, PRODUCT. In data preview, the attribute column and its label column, which contains its descriptions, appear next to each other.

Procedure

1. Open the information view in the editor.
2. Select the *Semantics* node.
3. In the *Columns* tab, select an attribute for which you want to associate label column.
4. In the *Label Column* dropdown list, select an attribute from the information view that contains necessary descriptions.

   If you have created an object using the old editor (which supported the old style of description mapping), and if you try to open it using the new editor, then you will see a new column <attribute>.description (as an attribute), which is hidden. You can rename the value and use it like other attributes based on your requirements.

3.12 Group Related Measures

In your analytic views or calculation views, if you are using multiple measures and if you want to organize them, for example, to segregate the planned measures with the actual measures, then you can create a folder and group all related measures within this folder.

Context

SAP HANA modeler helps you to create a *Display Folder*, which essentially is a folder that you can use to group related measures of attribute views and calculation views.

> **Note**
>
> A measure can exist in multiple display folders.

Procedure

1. Open a calculation view in the editor.
2. Select the *Semantics* node.
3. In the *Columns* tab, select a measure.
4. If you want to group the measure in just one folder, then in the *Display Folder* column, enter the folder name.

   For example, Folder1
5. If you want to create a hierarchy structure of folders and group the measure, then in the *Display Folder* column, provide values for more than one display folder separated by back slashes.

   For example: Folder1/Folder1.1/Folder1.1
6. If you want to associate a measure with multiple *Display Folders*, then in the *Display Folder* column, provide values for more than one display folder separated by semi-colon (;).

   For example: Folder1; Folder2; Folder3.....

   > **Note**
   >
   > For each measure, the system creates a display folder with a name based on the value that you provide in the *Display Folder* column.

4 Working With Calculation View Properties

SAP HANA modeler allows you to define certain properties for calculation views. The modeler refers to the values of these properties, for example, to access the data from the database or identify how to execute the calculation view.

This section describe the different calculation view properties, the possible values for each property and how these values help modeler determine the activation or execution behavior of the calculation view.

For defining the view properties, select the *Semantics* node and define the properties in the *View Properties* tab.

Related Information

Filter Data for Specific Clients [page 84]
Enable Calculation Views for Time Travel Queries [page 85]
Invalidate Cached Content [page 86]
Quick Reference: Calculation View Properties [page 88]

4.1 Filter Data for Specific Clients

Obtain data from all clients or filter the calculation view data either with fixed client value or with session client value set for the user.

Context

In SAP Web IDE for SAP HANA, for filtering data for specific clients, you have to explicitly specify the client filter column for each data source in the calculation view. Select the required data source in the calculation view, and in the *Mapping* tab, use the *Client Column* value help list to explicitly specify the client filter value for the selected data source.

Procedure

1. Open the calculation view in the view editor

2. Select the *Semantics* node.
3. Choose the *View Properties* tab.
4. In the *Default Client* dropdown list, select a value.

Related Information

Default Client Values [page 85]

4.1.1 Default Client Values

Assign a default client to an calculation view and filter data at runtime based on the default client value. The table below lists the default client value types you can assign and their description.

Table 42:

Default Client Value	Description
Session Client	If you use session client as the default client value, then at run time, the tool filters the table data according to the value you specify as the session client in the user profile.
Cross Client	If you use cross client as the default client value, the tool does not filter the table data against any client and you see values relevant to all clients.
Fixed Client	If you want to use a fixed client value, for example, 001, then the tool filters the table data for this client value.

4.2 Enable Calculation Views for Time Travel Queries

Time travel queries are queries against the historical states of the database. When you execute a time travel query on your information view, you can query the data at a specified time in past.

Context

If you have enabled time travel for information views, you can view data for a specific time in the past using the AS OF SQL extension. For example, you can execute the following SQL statement on information views as a timestamp query:

```
select * from <information_view> AS OF TIMESTAMP <timestamp>
```

SAP HANA supports creating history tables, which allows you to track changes made to other database tables. These tables help you associate time related information to your data. For example, you can use HISTORY

table to track changes performed to the CUSTOMER table. When you use history tables as data sources in calculation views, specify a parameter that you can use to provide the timestamp at runtime, and execute time travel queries on calculation views with history tables.

Procedure

1. Open your information view in the view editor.
2. In the *View Properties* tab, select the *Enable History* checkbox.
3. In the *History Input Parameter* dropdown list, choose an input parameter.

   You use input parameters to specify the timestamp in time travel queries.

   > **Note**
   >
   > You must use input parameters with data type DATE or SECONDDATE or TIMESTAMP or VARCHAR(8) of semantic type DATE to specify the timestamp.

4.3 Invalidate Cached Content

In order to maintain the significance of data cached for your calculation views, the tool supports time-based cache invalidation and transaction-based cache invalidation.

Prerequisites

You have enabled support for cache invalidation for the requried calculation view.

Context

In time-based cache invalidation, the system invalidates or removes the data from the cache after specific time intervals. Time-based cache invalidation is necessary to refresh data after every specific time period. By default, the cache invalidation period is null. This means, the result of the complex query that you execute resides in the cache until you execute the next query. Similarly, if you set your cache invalidation period as one hour, the result of the query resides in the cache for one hour, and system does not clear the cache for all other queries that you execute until this time period.

In transaction-based cache invalidation, the system invalidates the cache whenever the underlying data is modified.

> **Note**
>
> Cache invalidation is applicable only to complex SQL queries that are executed on top of your calculation views.

Procedure

1. Open the calculation view in the editor.
2. Choose *View Properties* tab.
3. In the *Cache Invalidation* dropdown list, select a time interval.

4.3.1 Enable Support for Cache Invalidation

Enable cache invalidation for your SAP HANA system to invalidate or remove data from the cache after specific time intervals or when underlying data is modified.

Context

You enable support for cache invalidation on your SAP HANA system. This acton, by default, enables cache invalidation support for all views in the system.

Procedure

1. In *Systems* view, double click your SAP HANA system.
2. Under *Configuration* tab, navigate to *indexserver.ini* > *cache* .
3. Set the property *resultcache_enabled* to*yes*.

> **Note**
>
> You can also enable cache invalidation support for specific information views. Open the information view in the view editor, and in the *View Properties* tab, select the *Cache* checkbox.

4.4 Quick Reference: Calculation View Properties

Considering different business scenarios, SAP HANA modeler allows you to define certain properties for calculation views. The value of these properties determines the characteristics of the view at runtime.

When you are modeling your calculation views, in the *View Properties* tab of the *Semantics* node, SAP HANA modeler allows you to define the following properties.

Graphical Calculation View Properties

Table 43:

Properties	Description
Data Category	The value of this property determines whether your calculation view supports analysis with multidimensional reporting. For more information, see Supported Data Categories for Information Views [page 40].
Default Client	The value of this property determines whether modeler must filter data for a fixed client or a session client or a cross client (does not filter data). For more information, see Filter Data for Specific Clients [page 84].
Apply Privileges	The value of this property specifies the analytic privilege type selected for data access restrictions on the calculation view. For more information, see Defining Data Access Privileges [page 90].
Default Schema	The value of this property helps modeler identify the default schema, which contains the tables necessary for currency or unit conversions. For more information, see Using Currency and Unit of Measure Conversions [page 73].
Default Member	This value of this property helps modeler identify the default member for all hierarchies in the calculation views.
Enable History	The value of this property determines whether your calculation view supports time travel queries. For more information see, Enable Calculation Views for Time Travel Queries [page 85].
History Input Parameter	Input parameter used to specify the timestamp in time travel queries.
Execute In	The value of this property impacts the output data. It determines whether modeler must execute the calculation view in SQL engine or column engine. For more information see SAP Note 1857202.
Cache Invalidation Period	The value of this property impacts the output data. It determines the time interval to invalidate or remove the cached content. For more information, see Invalidate Cached Content [page 86].

Properties	Description
Execution Hints	Execution hints impacts the output data. SAP HANA provides different execution hints. Execution hints are name-value pairs, which helps the optimizer to optimize the execution process dynamically based on specific criteria. In other words, the hints instruct the optimizer on the access paths to execute the calculation view.

Script-based Calculation View Properties

Table 44:

Properties	Description
Data Category	The value of this property determines whether your calculation view supports analysis with multidimensional reporting. For script-based calculation views, modeler supports only data category of type cube or blank. For more information, see Supported Data Categories for Information Views [page 40].
Default Client	The value of this property determines whether modeler must filter data for a fixed client or a session client or a cross client (does not filter data). For more information, see Filter Data for Specific Clients [page 84].
Apply Privileges	The value of this property specifies the analytic privilege type selected for data access restrictions on the calculation view. For more information, see Defining Data Access Privileges [page 90].
Default Schema	This value of this property helps modeler identify the default schema, which contains the tables used in the script-based calculation views.
Enable History	The value of this property determines whether your calculation view supports time travel queries. For more information, see Enable Calculation Views for Time Travel Queries [page 85] .
History Input Parameter	Input parameter used to specify the timestamp in time travel queries.
Run With	The value of this property helps modeler identify the authorization to use for selecting the data from the database, and for executing the calculation view or procedure. If the property is set to Definer's rights, then modeler uses the authorizations of the user who defines the view or procedure. Similarly, if the property is set to Invoker's right, modeler uses the authorizations of the current user to access data from the database.
Cache Invalidation Period	The value of this property impacts the output data. It determines the time interval to invalidate or remove the cached content. For more information, see Invalidate Cached Content [page 86].

5 Defining Data Access Privileges

This section describes how to create analytic privileges and assign them to different users to provide selective data access control to activated information views.

Analytic privileges grant different users access to different portions of data in the same view based on their business role. Within the definition of an analytic privilege, the conditions that control which data users see is either contained in an XML document or defined using SQL.

Standard object privileges (SELECT, ALTER, DROP, and so on) implement coarse-grained authorization at object level only. Users either have access to an object, such as a table, view or procedure, or they don't. While this is often sufficient, there are cases when access to data in an object depends on certain values or combinations of values. Analytic privileges are used in the SAP HANA database to provide such fine-grained control at row level of which data individual users can see within the same view.

> **Example**
>
> Sales data for all regions are contained within one analytic view. However, regional sales managers should only see the data for their region. In this case, an analytic privilege could be modeled so that they can all query the view, but only the data that each user is authorized to see is returned.

SAP HANA modeler supports creating the following two types of analytic privileges, the classical XML-based analytic privileges and the SQL analytic privileges.

XML- Versus SQL-Based Analytic Privileges

Before you implement row-level authorization using analytic privileges, you need to decide which type of analytic privilege is suitable for your scenario. In general, SQL-based analytic privileges allow you to more easily formulate complex filter conditions that might be cumbersome to model using XML-based analytic privileges.

The following are the main differences between XML-based and SQL-based analytic privileges:

Table 45:

Feature	SQL-Based Analytic Privileges	Classical XML-Based Analytic Privileges
Control of read-only access to SAP HANA information models: • Attribute views • Analytic views • Calculation views	Yes	Yes
Control of read-only access to SQL views	Yes	No
Control of read-only access to database tables	No	No

Feature	SQL-Based Analytic Privileges	Classical XML-Based Analytic Privileges
Design-time modeling in the *Editor* tool of the SAP HANA Web Workbench	Yes	Yes
Design-time modeling in the *SAP HANA Modeler* perspective of the SAP HANA studio	Yes	Yes
Transportable	Yes	Yes
Complex filtering	Yes	No

Enabling an Authorization Check Based on Analytic Privileges

All column views modeled and activated in the SAP HANA modeler and the SAP HANA Web-based Development Workbench automatically enforce an authorization check based on analytic privileges. XML-based analytic privileges are selected by default, but you can switch to SQL-based analytic privileges.

Column views created using SQL must be explicitly registered for such a check by passing the relevant parameter:

- `REGISTERVIEWFORAPCHECK` for a check based on XML-based analytic privileges
- `STRUCTURED PRIVILEGE CHECK` for a check based on SQL-based analytic privileges

SQL views must always be explicitly registered for an authorization check based analytic privileges by passing the STRUCTURED PRIVILEGE CHECK parameter.

> **Note**
>
> It is not possible to enforce an authorization check on the same view using both XML-based and SQL-based analytic privileges. However, it is possible to build views with different authorization checks on each other.

Related Information

Create Classical XML-Based Analytic Privileges [page 92]

5.1 Create Classical XML-Based Analytic Privileges

Create analytic privileges for information views and assign them to different users to provide selective access that are based on certain combinations of data.

Prerequisites

If you want to use a classical XML-based analytic privilege to apply data access restrictions on information views, set the *Apply Privileges* property for the information view to *Classical Analytic Privileges*.

1. Open the calculation view in the view editor.
2. Select the *Semantics* node.
3. Choose the *View Properties* tab.
4. In the *Apply Privileges* dropdown list, select *Classical Analytic Privileges*.

Context

Analytic privileges help restrict data access to information views based on attributes or procedures. You can create and apply analytic privileges for a selected group of models or apply them to all models across packages.

After you create analytic privileges, assign it to users. This restricts users to access data only for certain combinations of dimension attributes.

Procedure

1. Open the SAP HANA Web-based Development Workbench Editor.

   The Web-based Editor tool is available on the SAP HANA XS Web server at the following URL: `http://<WebServerHost>:80<SAPHANAinstance>/sap/hana/ide/editor`.
2. In the navigator pane, select a package where you want to create the new analytic privilege.
3. In the context menu of the package, select *New* > *Analytic Privilege*.
4. Provide a name and description.
5. In the *Type* dropdown list, select *Classical*.
6. Choose *Create*.
7. Define validity for the analytic privilege.

   In the *Privilege Validity* section, specify the time period for which the analytic privilege is valid.
   a. Choose *Add*.
   b. Select a required operator.
   c. Provide the validity period based on the selected operator.

8. Define scope of the analytic privilege.

   In the *Secured Models* section, select the models for which the analytic privileges restrictions are applicable.

   a. Choose *Add*.
   b. If you want to create an analytic privilege and apply the data access restrictions for selected list of models, in the *Find Data Sources* dialog, select the models for which you want apply the analytic privilege restrictions.
   c. Choose *OK*.
   d. If you want to create an analytic privilege and apply the data access restrictions for all models, select the *Apply to all information models* checkbox.

9. Select attributes and define restrictions.

   Use attributes from the secured models to define data access restrictions.

   a. In the *Associated Attributes Restrictions* section, choose *Add*.
   b. In the *Attributes* dialog, select the attributes.

      > **Note**
      >
      > Select a model if you want to use all attributes from the model to define restrictions.

   c. Choose *OK*.
   d. Choose 

      to define restrictions on the selected attributes.

      Modeler uses the restrictions defined on the attributes to restrict data access. Each attribute restriction is associated with only one attribute, but can contain multiple value filters. You can create more than one attribute restrictions.
   e. In the *Restriction Type* dropdown list, select a restriction type.
   f. Select the required operator and provide a value using the value help.
   g. For catalog procedure or repository procedure, you can also provide values using the syntax <schema name>::<procedure name> or <package name>::<procedure name> respectively.

10. Activate analytic privileges.

    a. If you want to activate the analytic privilege, in the menu bar, choose *Save*.
    b. If you want to activate the analytic privilege along with all objects, in the menu bar, choose *Save All*.

       > **Note**
       >
       > Activate the analytic privilege only if you have defined at least one restriction on attributes in the *Associated Attributes Restrictions* section.

11. Assign privileges to a user.

    If you want to assign privileges to an authorization role, execute the following steps:

    a. In the menu bar, choose  .
    b. Select the *Security* menu option.

       This opens a new tab in the browser where you can assign the analytic privileges to users.
    c. Expand *Users*.
    d. Select a user.

e. In the *Analytic Privileges* tab page, choose the add icon to add the privilege.
f. In the editor toolbar, choose *Activate*.

Related Information

Supported Restriction Types in Analytic Privileges [page 94]

5.1.1 Supported Restriction Types in Analytic Privileges

Define data access restrictions on information views using fixed restrictions or dynamic restrictions.

Table 46:

Restriction Type	Description	Example
Fixed Value	A fixed value or static value filter consists of an operator and either a list of values as the filter operands or a single value as the filter operand. All data types are supported except those for LOB data types (CLOB, BLOB, and NCLOB).	For example, a value filter (EQ 2006) can be defined for an attribute YEAR in a dimension restriction to filter accessible data using the condition YEAR=2006 for potential users.
Catalog Procedure or Repository Procedure.	Catalog Procedures or Repository Procedures are dynamic value filters, which consists of an operator and a stored procedure call that determines the operand value at runtime.	For example, a value filter (IN (GET_MATERIAL_NUMBER_FOR_CURRENT_USER())) is defined for the attribute MATERIAL_NUMBER. This filter indicates that a user with this analytic privilege is only allowed to access material data with the numbers returned by the procedure GET_MATERIAL_NUMBER_FOR_CURRENT_USER.

5.2 Create SQL Analytic Privileges

SQL based analytic privileges provides you the flexibility to create analytic privileges within the familiar SQL environment. You can create and apply SQL analytic privileges for a selected group of models or apply them to all models across packages.

Prerequisites

If you want to use a SQL analytic privilege to apply data access restrictions on information views, set the *Apply Privileges* property for the information view to *SQL Analytic Privileges*.

1. Open the information view in the view editor.

2. Select the *Semantics* node.
3. Choose the *View Properties* tab.
4. In the *Apply Privileges* dropdown list, select *SQL Analytic Privileges*.

Context

SAP HANA modeler support types SQL analytic privileges, the static SQL analytic privileges with predefined static filter conditions, and dynamic SQL analytic privileges with filter conditions determined dynamically at runtime using a database procedure.

Procedure

1. Open the SAP HANA Web-based Development Workbench Editor.

   The Web-based Editor tool is available on the SAP HANA XS Web server at the following URL: `http://<WebServerHost>:80<SAPHANAinstance>/sap/hana/ide/editor`.
2. In the navigator pane, select a package where you want to create the new analytic privilege.
3. In the context menu of the package, select *New* > *Analytic Privilege*.
4. Provide a name and description.
5. In the *Type* dropdown list, select *SQL*.
6. Choose *Create*.
7. In the header region, select *SQL Editor*.
8. Select information models.

   If you want to create an analytic privilege and apply the data access restrictions for selected list of models, in the *Secured Models* section,

   a. Choose *Add*.
   b. In the *Find Data Sources* dialog, select the models for which you want apply the analytic privilege restrictions.
   c. Choose *OK*.
9. Defining static SQL analytic privileges.

   If you want to define static SQL analytic privileges, then

   a. In the SQL editor, provide the attribute restrictions and its validity.

      For example,

```
(("REGION" = 'EAST') OR ("REGION" = 'NORTH')) AND (("CUSTOMER_ID" =
'SAP')) AND ((CURRENT_DATE BETWEEN 2015-05-15 00:00:00.000 AND 2015-05-15
23:59:59.999))
```

> **Note**
>
> You can also use hierarchy nodes in SQL analytic privilege definitions. For example, "SalesRepHierarchyNode" = "MAJESTIX".

10. Defining dynamic SQL analytic privileges.

    Dynamic SQL analytic privileges determine the filter condition string at runtime. If you want to define dynamic SQL analytic privileges,

    a. In the SQL editor, specify the procedure within the CONDITION PROVIDER clause.

       For example, CONDITION PROVIDER schema_name.procedure_name.

11. Activate analytic privileges.

    a. If you want to activate the analytic privilege, in the menu bar, choose *Save*.
    b. If you want to activate the analytic privilege along with all objects, in the menu bar, choose *Save All*.

    > **Note**
    >
    > Activate the analytic privilege only if you have defined at least one restriction on attributes in the *Associated Attributes Restrictions* section.

12. Assign privileges to a user.

    If you want to assign privileges to an authorization role, execute the following steps:

    a. In the menu bar, choose  .
    b. Select the *Security* menu option.

       This opens a new tab in the browser where you can assign the analytic privileges to users.
    c. Expand *Users*.
    d. Select a user.
    e. In the *Analytic Privileges* tab page, choose the add icon to add the privilege.
    f. In the editor toolbar, choose *Activate*.

    > **Note**
    >
    > For SQL analytic privileges, you can also define the attribute restrictions and dynamic restrictions without using the SQL editors. If you want to define attribute restrictions, select the *Attributes* radio button and define the attribute restrictions. Similarly, if you want to define dynamic analytic privileges, select the *Dynamic* radio button and use a repository procedure or a catalog procedure, which dynamically determines the restrictions on information models.

5.2.1 Static SQL Analytic Privileges

Static SQL analytic privileges or fixed analytic privileges allows you to combine one or multiple filter conditions on the same attribute or different attributes using the logical AND or OR operators.

Static SQL analytic privileges conditions typically have the following structure, <attribute> <operator> <scalar_operands_or_subquery>. For example, "country IN (scalar_operands_or_subquery) AND product = (scalar_operands_or_subquery)." The supported operator types are IN, LIKE, BETWEEN, <=, >=, <, >.

If you want to create static SQL analytic privileges using subqueries, then the user creating the analytic privileges must have corresponding privileges on the database objects (tables/views) involved in the subqueries.

5.2.2 Dynamic SQL Analytic Privileges

In dynamic analytic privileges, you use a database procedure to dynamically obtain the filter condition string at runtime. You can provide the database procedure value within the CONDITION PROVIDER clause.

You can use only procedures, which achieve the following conditions to define dynamic SQL analytic privileges.

- DEFINER procedures.
- Read-only procedures.
- Procedure with no input parameters
- Procedure with only one output parameter of type VARCHAR or NVARCHAR for the filter condition string.
- Procedures executable by _SYS_REPO. This means that, _SYS_REPO is either the owner of the procedure or the owner of the procedure has all privileges on the underlying tables/views with GRANT OPTION and has granted the EXECUTE privilege on the procedure to the _SYS_REPO user.

> **Note**
>
> Modeler supports only simple filter conditions in dynamic SQL analytic privileges and you cannot use subqueries for dynamic analytic privileges.

5.2.3 Structure of SQL-Based Analytic Privileges

An analytic privilege consists of a set of restrictions against which user access to a particular attribute view, analytic view, calculation view, or SQL view is verified. In an SQL-based analytic privilege, these restrictions are specified as filter conditions that are fully SQL based.

SQL-based analytic privileges are created in the *Editor* tool of the SAP HANA Web-based Development Workbench (*New* > *Analytic Privilege* > *Type: SQL*) on the basis of the CREATE STRUCTURED PRIVILEGE statement:

```
CREATE STRUCTURED PRVILEGE <privilege_name> FOR <action> ON <view_name>
<filter_condition>
```

The FOR clause is used restrict the type of access (only the SELECT action is supported). The ON clause is used to restrict access to one or more views with the same filter attributes.

The `<filter condition>` parameter is used to restrict the data visible to individual users. The following methods of specifying filter conditions are possible:

- Fixed filter (WHERE) clause
- Dynamically generated filter (CONDITION PROVIDER) clause

Fixed Filter Clauses

A **fixed filter clause** consists of an WHERE clause that is specified in the definition of the analytic privilege itself.

You can express fixed filter conditions freely using SQL, including subqueries.

By incorporating built-in SQL functions into the subqueries, in particular SESSION_USER, you can define an even more flexible filter condition.

Example

country IN (SELECT a.country FROM authorizationtable a WHERE SESSION_USER= a.user_name)

Note

A **calculation view** cannot be secured using an SQL-based analytic privilege that contains a complex filter condition if the view is defined on top of analytic and/or attributes views that themselves are secured with an SQL-based analytic privilege with a complex filter condition.

Remember

If you use a subquery, you (the creating user) must have the required privileges on the database objects (tables and views) involved in the subquery.

Comparative conditions can be nested and combined using AND and OR (with corresponding brackets).

For examples, see *Examples: Securing Views Using SQL-Based Analytic Privileges*.

Dynamically Generated Filter Clauses

With a dynamically generated filter clause, the WHERE clause that specifies the filter condition is generated every time the analytic privilege is evaluated. This is useful in an environment in which the filter clause changes very dynamically. The filter condition is determined by a procedure specified in the CONDITION PROVIDER clause, for example:

Sample Code

```
CREATE STRUCTURED PRVILEGE dynamic_ap FOR SELECT ON schema1.v1 CONDITION PROVIDER
schema2.procedure1;
```

Procedures in the CONDITION PROVIDER clause must have the following properties:

- They must have the security mode DEFINER.
- They must be read-only procedures.
- They must have a predefined signature. Here, the following conditions apply:
  - No input parameter
  - Only one output parameter for the filter condition string
- The procedure may only return conditions expressed with the following operators:
  - =, <=, <, >, >=
  - LIKE
  - BETWEEN
  - IN

  A complex filter condition, that is a subquery, may not be returned.

- The procedure must be executable by _SYS_REPO, that is, either_SYS_REPO must be the owner of the procedure or the owner of the procedure has all privileges on the underlying tables/views with GRANT OPTION and has granted the EXECUTE privilege on the procedure to the _SYS_REPO user.

If errors occur in procedure execution, the user receives a `Not authorized` error, even if he has the analytic privileges that would grant access.

For examples, see *Examples: Securing Views Using SQL-Based Analytic Privileges*.

Related Information

Examples: Securing Views Using SQL-Based Analytic Privileges [page 99]
SAP HANA SQL and System Views Reference

5.2.4 Examples: Securing Views Using SQL-Based Analytic Privileges

Use the CREATE STRUCTURED PRIVILEGE statement to create SQL-based analytic privileges for different scenarios.

Context

The examples provided here take you through the following scenarios:

- Example 1: Securing a column view using an SQL-based analytic privilege with a fixed filter clause [page 100]
- Example 2: Securing an SQL view using an SQL-based analytic privilege with a complex filter clause (subquery) [page 101]
- Example 3: Securing a column view using an SQL-based analytic privilege with a dynamically generated filter clause [page 103]

> **Note**
>
> The analytic privileges in these example are created using the CREATE STRUCTURED PRIVILEGE statement. Under normal circumstances, you create SQL-based analytic privileges using the SAP HANA Web-based Development Workbench. Analytic privileges created using CREATE STRUCTURED PRIVILEGE are not owned by the user _SYS_REPO. They can be granted and revoked only by the actual database user who creates them.

Example 1: Secure a Column View Using an SQL-Based Analytic Privilege with a Fixed Filter Clause

Prerequisites

The database user TABLEOWNER has set up a calculation scenario based on the table SALES_TABLE, which contains the data to be protected.

Context

All sales data is contained in a single view. You want to restrict user access so that sales managers can see only information about the product "car" in the sales region UK and Germany. You want to do this by creating an analytic privilege with a fixed filter clause.

A fixed filter clause consists of an SQL WHERE clause that is specified in the definition of the analytic privilege itself.

Procedure

1. Create the view containing the sales data:

```
CREATE COLUMN VIEW "TABLEOWNER"."VIEW_SALES" TYPE CALCULATION WITH PARAMETERS
('PARENTCALCINDEXSCHEMA'='TABLEOWNER',
     'PARENTCALCINDEX'='CALCSCEN_SALES',
    'PARENTCALCNODE'='SALES_TABLE',
     'REGISTERVIEWFORAPCHECK'='0') STRUCTURED PRIVILEGE CHECK
;
```

> **Note**
>
> You can see above that the authorization check using XML-based analytic privileges is disabled with `'REGISTERVIEWFORAPCHECK'='0'`, while the authorization check using SQL-based analytic privileges is enabled with `STRUCTURED PRIVILEGE CHECK`. Both checks cannot be enabled at the same time.

2. Create the analytic privilege:

```
CREATE STRUCTURED PRIVILEGE AP_SALES_1 FOR SELECT
     ON TABLEOWNER.VIEW_SALES
WHERE REGION IN ('DE','UK')
OR PRODUCT = 'CAR'
;
```

> **Remember**
>
> When specifying filters, remember the following:
>
> - You can specify only the SELECT action in the FOR clause.

- You can specify one or more views with the same filter attributes in the ON clause
- You can specify comparative conditions between attributes and constant values using only the following operators:
  - =, <=, <, >, >=
  - LIKE
  - BETWEEN
  - IN
- You can create complex filter conditions by including SQL statements as subqueries inside the WHERE clause. Example 2 illustrates how you do this. But remember: A **calculation view** cannot be secured using an SQL-based analytic privilege that contains a complex filter condition if the view is defined on top of analytic and/or attributes views that themselves are secured with an SQL-based analytic privilege with a complex filter condition.
  Also remember that if you use a subquery, you must have the required privileges on the database objects (tables and views) involved in the subquery.

3. Grant the SELECT privilege on the view TABLEOWNER.VIEW_SALES to the relevant users/roles:

```
GRANT SELECT on TABLEOWNER.VIEW_SALES to <SALES_MANAGERS>;
```

> **Remember**
>
> Only the view owner or a user who has the SELECT privilege WITH GRANT OPTION on the view can perform the grant.

4. Grant the analytic privilege to the relevant users/roles:

```
GRANT STRUCTURED PRIVILEGE AP_SALES_1 TO <SALES_MANAGERS>;
```

> **Remember**
>
> Only the owner of the analytic privilege can grant it.

Example 2: Secure an SQL View Using an SQL-Based Analytic Privilege with a Complex Filter Clause (Subquery)

Prerequisites

The database user TABLEOWNER has created a table TABLEOWNER.SALES, which contains the data to be protected.

Context

All sales data is contained in a single view. You want to restrict access of user MILLER so that he can see only product information from the year 2008. You want to do this by creating an analytic privilege with a complex filter clause.

With a complex filter clause, the SQL WHERE clause that specifies the filter condition includes an SQL statement, or a subquery. This allows you to create complex filter conditions to control which data individual users see.

Procedure

1. Create the view containing the sales data which needs to be secured:

```
CREATE VIEW "VIEWOWNER"."ROW_VIEW_SALES_ON_SALES" AS SELECT
     * FROM "TABLEOWNER"."SALES" WITH STRUCTURED PRIVILEGE CHECK
;
```

➡ Remember

The user creating the view must have the SELECT privilege WITH GRANT OPTION on the table TABLEOWNER.SALES.

2. Create the table containing user-specific authorization data:

```
CREATE COLUMN TABLE "VIEWOWNER"."AUTHORIZATION_VALUES"("VALUE" VARCHAR(256),
    "USER_NAME" VARCHAR(20));
```

3. Insert authorization information for user MILLER:

```
INSERT INTO "VIEWOWNER"."AUTHORIZATION_VALUES" VALUES('2008', 'MILLER);
```

4. Create the analytic privilege using a subquery as the condition provider:

```
CREATE STRUCTURED PRIVILEGE AP_ROW_VIEW_SALES_ON_SALES FOR SELECT
     ON "VIEWOWNER"."ROW_VIEW_SALES_ON_SALES"
WHERE (CURRENT_DATE BETWEEN 2015-01-01 AND 2015-01-11) AND YEAR IN (SELECT
VALUE FROM VIEWOWNER.AUTHORIZATION_VALUES WHERE USER_NAME = SESSION_USER)
;
```

➡ Remember

- Subqueries allow you to create complex filter conditions, but remember: A **calculation view** cannot be secured using an SQL-based analytic privilege that contains a complex filter condition if the view is defined on top of analytic and/or attributes views that themselves are secured with an SQL-based analytic privilege with a complex filter condition.
- The user creating the analytic privilege must have the SELECT privilege on the objects involved in the subquery, in this case table VIEWOWNER.AUTHORIZATION_VALUES.
- The session user is the database user who is executing the query to access a secured view. This is therefore the user whose privileges must be checked. For this reason, the table containing the authorization information needs a column to store the user name so that the subquery can filter on this column using the SQL function SESSION_USER.

> **Caution**
>
> Do not map the executing user to the application user. The application user is unreliable because it is controlled by the client application. For example, it may set the application user to a technical user or it may not set it at all. In addition, the trustworthiness of the client application cannot be guaranteed.

5. Grant the SELECT privilege on the view VIEWOWNER.ROW_VIEW_SALES_ON_SALES to user MILLER.

```
GRANT SELECT ON "VIEWOWNER"."ROW_VIEW_SALES_ON_SALES" TO MILLER;
```

> **Remember**
>
> Only the view owner or a user who has the SELECT privilege WITH GRANT OPTION on the view can perform the grant.

6. Grant the analytic privilege to user MILLER.

```
GRANT STRUCTURED PRIVILEGE AP_ROW_SALES_ON_SALES TO MILLER;
```

> **Remember**
>
> Only the owner of the analytic privilege can grant it.

Example 3: Secure a Column View Using an SQL-Based Analytic Privilege with a Dynamically Generated Filter Clause

Prerequisites

The database user TABLEOWNER has set up a calculation scenario based on the table SALES_TABLE, which contains the data to be protected.

Context

All sales data is contained in a single view. You want to restrict access of user ADAMS so that he can see only information about cars bought by customer Company A or bikes sold in 2006. You want to do this by creating an analytic privilege with a dynamically generated filter clause.

With a dynamically generated filter clause, the SQL WHERE clause that specifies the filter condition is generated every time the analytic privilege is evaluated. This is useful in an environment in which the filter clause changes very dynamically.

Procedure

1. Create the view containing the sales data:

```
CREATE COLUMN VIEW "TABLEOWNER"."VIEW_SALES" TYPE CALCULATION WITH PARAMETERS
('PARENTCALCINDEXSCHEMA'='TABLEOWNER',
    'PARENTCALCINDEX'='CALCSCEN_SALES',
    'PARENTCALCNODE'='SALES_TABLE',
    'REGISTERVIEWFORAPCHECK'='0') STRUCTURED PRIVILEGE CHECK
;
```

2. Create a table containing user-specific filter strings:

```
CREATE COLUMN TABLE "AUTHORIZATION"."AUTHORIZATION_FILTERS"("FILTER"
VARCHAR(256),
    "USER_NAME" VARCHAR(20))
;
```

3. Create an authorization filter for user ADAMS:

```
INSERT
INTO "AUTHORIZATION"."AUTHORIZATION_FILTERS" VALUES('(CUSTOMER=''Company A''
AND PRODUCT=''Car'') OR (YEAR=''2006'' AND PRODUCT=''Bike'')',
    'ADAMS')
;
```

> **Remember**
>
> Filters containing comparative conditions must be defined as specified in example 1.

4. Create the database procedure that provides the filter clause for the analytic privilege and grant it to user _SYS_REPO:

```
CREATE PROCEDURE "PROCOWNER"."GET_FILTER_FOR_USER"(OUT OUT_FILTER
VARCHAR(256))
LANGUAGE SQLSCRIPT SQL SECURITY DEFINER READS SQL DATA AS
    v_Filter VARCHAR(256);
    CURSOR v_Cursor FOR SELECT "FILTER" FROM
"PROCOWNER"."AUTHORIZATION_FILTERS" WHERE "USER_NAME" = SESSION_USER;
BEGIN
    OPEN v_Cursor;
    FETCH v_Cursor INTO v_Filter;
    OUT_FILTER := v_Filter;
    CLOSE v_Cursor;
END;
GRANT EXECUTE ON "PROCOWNER"."GET_FILTER_FOR_USER" TO _SYS_REPO;
```

> **Remember**
>
> When using procedures as the condition provider in an SQL-based analytic privilege, remember the following:
>
> - Procedures must have the following properties:
> - They must have the security mode DEFINER.
> - They must be read-only procedures.
> - A procedure with a predefined signature must be used. The following conditions apply:
> - No input parameter
> - Only 1 output parameter of VARCHAR(256) or NVARCHAR(256) type for the filter condition string

- The procedure may **not** return a complex filter condition, that is a subquery.
- The procedure must be executable by _SYS_REPO, that is, either_SYS_REPO must be the owner of the procedure or the owner of the procedure has all privileges on the underlying tables/views with GRANT OPTION and has granted the EXECUTE privilege on the procedure to the _SYS_REPO user.
- The session user is the database user who is executing the query to access a secured view. This is therefore the user whose privileges must be checked. For this reason, the table or view used in the procedure should contain a column to store the user name so that the procedure can filter on this column using the SQL function SESSION_USER.
- If errors occur in procedure execution, the user receives a `Not authorized` error, even if he has the analytic privileges that would grant access.

5. Create the analytic privilege using the procedure as condition provider:

```
CREATE STRUCTURED PRIVILEGE AP_SALES_2 FOR SELECT ON
"TABLEOWNER"."VIEW_SALES" CONDITION PROVIDER
"AUTHORIZATION"."GET_FILTER_FOR_USER";
```

   On evaluation of the analytic privilege for user ADAMS, the WHERE clause (CUSTOMER='Company A' AND PRODUCT='Car') OR (YEAR='2006' AND PRODUCT='Bike'), as provided by the procedure GET_FILTER_FOR_USER, will be used.

6. Grant the SELECT privilege on the view TABLEOWNER.VIEW_SALES to user ADAMS:

```
GRANT SELECT on TABLEOWNER.VIEW_SALES to ADAMS;
```

   Remember

   Only the view owner or a user who has the SELECT privilege WITH GRANT OPTION on the view can perform the grant.

7. Grant the analytic privilege to user ADAMS:

```
GRANT STRUCTURED PRIVILEGE AP_SALES_2 TO ADAMS;
```

   Remember

   Only the owner of the analytic privilege can grant it.

6 Additional Functionality for Calculation Views

After modeling calculation views or at design time you can perform certain additional functions, which helps improve the efficieny of modeling calculation views.

This section describes the different additional functions that SAP HANA modeler offers and how you can use these functions to efficiently model views.

Related Information

Replacing Nodes and Data Sources [page 121]
Manage Calculation Views with Missing Objects [page 119]
Use Aliases for Table Names and Column Names [page 120]

6.1 Performance Analysis

Open calculation views in performance analysis mode and obtain information on the catalog tables. This information helps you analyze the possible performance impacts on information views at runtime. For example, you can obtain information on table partitions, number of rows in tables, and so on.

In performance analysis mode, you obtain following key information and much more depending on data sources in your calculation view.

Identify number of rows in a table

Identify those data sources that have number of rows above a certain threshold value. You can configure this threshold value in the modeling settings.

1. In the menu bar, choose  .
2. In the *Editor setting* dialog, choose the *Modeling* tab.
3. In the *Threshold Value* textbox, provide a threshold value.
4. Choose *Apply*.

Identify Table Partitions and Table Types

If you have partitioned tables, identify the partitioned tables along with partition type (Hash, Range, Round Robin) and columns used for partitioning the tables. In addition, you can also obtain information on the table type. For example, if you are using virtual tables, then modeler provides information on the virtual table properties (remote DB, remote source, remote owner, and remote object) and its values.

> **Note**
>
> If you have enabled performance analysis mode, modeler displays a warning icon across those catalog tables that have number of rows more than the threshold value that you have defined. Similarly, modeler also displays certain indicators across the tables, which helps you to identify the partition type of those particular tables.

6.1.1 Open Calculation Views in Performance Analysis Mode

The number of rows in a data source and table partitions impact the performance of your queries. When you open an calculation view in performance analysis mode, you can obtain information on join tables, table partitions, table types and other useful information, which helps you analyze your calculation view and its possible performance impacts at runtime.

Procedure

1. Open the SAP HANA Web-based Development Workbench Editor.

   The Web-based Editor tool is available on the SAP HANA XS Web server at the following URL: `http://<WebServerHost>:80<SAPHANAinstance>/sap/hana/ide/editor`
2. Open the required graphical calculation view in the view editor.

   > **Note**
   >
   > Performance analysis is not supported for script-based calculation views.
3. Select a view node with catalog tables.

   > **Note**
   >
   > You cannot analyze the performance of the *Semantics* node.
4. In the menu bar, choose the  icon to enable performance analysis mode.

   The modeler displays the following information in the *Performance Analysis* tab.
   - Join Details
   - Data Source Details

   > **Note**
   >
   > You can choose the same icon to hide performance analysis mode for a calculation view.

   > **Note**
   >
   > If you want to always open a calculation view in performance analysis mode by default, configure the modeling settings.

1. In the menu bar, choose  .
2. In the *Editor setting* dialog, choose the *Modeling* tab.
3. Select *Always open Calculation Views* in performance analysis mode.
4. Choose *Apply*.

6.1.2 Join Details

If you have enabled the performance analysis mode by default, open an information view and select a join node with catalog tables to obtain join details.

For a selected join node, the *JOIN DETAILS* section in *Performance Analysis* tab provides the following information:

- Catalog tables participating in the join.
- The cardinality and join type that you have selected for each join.
- The cardinality that modeler recommends after analyzing the participating catalog tables in the join.
- Information on whether you have maintained the referential integrity for the join table.
- The join validation status. The join validation status is in warning if the cardinality that modeler proposes is different from the cardinality that you select and also if you have not maintained referential integrity.

> **Note**
>
> Only users with SELECT privileges on the catalog tables participating in the join can view join validation status. If the participating catalog tables are virtual tables, then you can view the join validation status only if the user has SELECT privileges on the virtual table and also if the user credential to remote source has SELECT privileges on the remote table.

6.1.3 Data Source Details

If you have enabled the performance analysis mode, open an information view and select a view node with catalog tables to obtain data source details.

For a selected view node, the *DATA SOURCE DETAILS* section in *Performance Analysis* tab provides the following information:

- The catalog tables available in a selected view node.
- The catalog table type.
- If catalog table is partitioned, then modeler provides details on the partition type (Hash, Range, Round Robin).
- If multiple levels of partitions exist for the catalog table, then select the data source to view all levels of partition details.
- Number of rows in the catalog table. Also, modeler displays a warning icon for catalog tables with number of rows more than the threshold value that you have defined.

- If you are using virtual catalog tables, modeler provides details on the virtual table properties and its values. The virtual table properties include the remote DB, remote source, remote owner, and remote object.
- If you are using scale-out architecture with multiple nodes connected to an SAP HANA system, modeler provides information on the table group name, and the table group type and its subtype.

> **Note**
>
> Only users with system privilege INIFILE ADMIN can idenitfy whether a system is using a scale-out architecture.

6.2 Maintain Comments for Modeler Objects

When you are modeling a calculation view, you can also maintain comments for the view or for its objects such as parameters, hierarchies, view nodes and so on. The comments can include, for example, information that provides more clarity on the calculation view or its objects for data modelers accessing the same view or its objects.

Context

Maintaining comments helps your store more information related to the calculation view or to store and provide reference information for other data modelers working on the same calculation view. You can also use the comments for documentation purposes.

- Columns in the semantics node
- View Node
- Input Parameters and Variables
- Hierarchies
- Calculated columns in underlying view nodes

> **Note**
>
> You cannot translate comments that you maintain for the modeler objects.

Procedure

1. Open the SAP HANA Web-based Development Workbench Editor.

   The Web-based Editor tool is available on the SAP HANA XS Web server at the following URL: http://<WebServerHost>:80<SAPHANAinstance>/sap/hana/ide/editor
2. Open the graphical calculation view for which you want to maintain comments in view editor.

3. Maintain comments for the information view at the view level.
   a. Select the *Semantics* node.
   b. Choose the *View Properties* tab.
   c. In the *General* section, choose .
   d. Enter a new comment or edit an existing comment.
4. Maintain comments for columns in the semantics node.
   a. Select the *Semantics* node.
   b. Choose the *Columns* tab.
   c. Select the column for which you want to maintain comments.
   d. Choose .
   e. Enter a new comment or edit an existing comment.
5. Maintain comments for view nodes.
   a. Select the view node other than the *Semantics* node and the default view node.
   b. Choose .
   c. Enter a new comment or edit an existing comment.
6. Maintain comments for input parameters and variables.
   a. Select the *Semantics* node.
   b. Choose the *Parameters/Variables* tab.
   c. Select an input parameter or variable for which you want to maintain comments.
   d. Choose .
   e. Enter a new comment or edit an existing comment.
7. Maintain comments for hierarchies.
   a. Select the *Semantics* node.
   b. Choose the *Hierarchies* tab.
   c. Select a hierarchy for which you want to maintain comments.
   d. Choose .
   e. Enter a new comment or edit an existing comment.
8. Maintain comments for calculated columns.
   a. Select a view node with a calculated column.
   b. Choose the *Calculated Columns* tab.
   c. Select a calculated column for which you want to maintain comments.
   d. Choose .
   e. Enter a new comment or edit an existing comment.
9. Maintain comments for restricted columns.
   a. Select a default aggregation node.
   b. Choose the *Restricted Columns* tab.
   c. Select a restricted column for which you want to maintain comments.
   d. Choose .
   e. Enter a new comment or edit an existing comment.

6.3 Using Functions in Expressions

This section describes the functions, which you can use while creating expressions for calculated attributes and calculated measures.

You can create expressions, for example in calculated columns using the column engine (CS) language or the SQL language.

> **Note**
>
> Related SAP Notes. The SAP Note 2252224 describes the differences between the CS and SQL string expression with respect to Unicode or multi-byte encoding. The SAP Note 1857202 describes the SQL execution of calculation views.

Related Information

String Functions [page 113]
Conversion Functions [page 111]
Mathematical Functions [page 116]
Date Functions [page 116]
Miscellaneous Functions [page 118]
Using Functions in Expressions [page 111]

6.3.1 Conversion Functions

Data type conversion functions are used to convert arguments from one data type to another, or to test whether a conversion is possible.

Function	Syntax	Purpose	Example
int	int int(arg)	convert arg to int type	int(2)
float	float float(arg)	convert arg to float type	float(3.0)
double	double double (arg)	convert arg to double type	double(3)
sdfloat	sdfloat sdfloat (arg)	convert arg to sdfloat type	
decfloat	decfloat decfloat (arg)	convert arg to decfloat type	
fixed	fixed fixed (arg, int, int)	arg2 and arg3 are the intDigits and fractdigits parameters, respectively. Convert arg to a fixed type of	fixed(3.2, 8, 2) + fixed(2.3, 8, 3)

Function	Syntax	Purpose	Example
		either 8, 12, or 16 byte length, depending on intDigits and fractDigits	
string	string string (arg)	convert arg to string type	
raw	raw raw (arg)	convert arg to raw type	
date	date date(stringarg) date date(fixedarg) date date(int, int) date date(int, int, int) date date(int, int, int, int) date date(int, int, int, int, int) date date(int, int, int, int, int, int)	convert arg to date type. The first version parses a string in the format "yyyy-mm-dd hh:mi:ss" where trailing components except for the year may be omitted. The version with one fixed number arg strips digits behind the comma and tries to make a date from the rest. The other versions accept the individual components to be set.	date(2009) -> date('2009') date(2009, 1, 2) -> date('2009-01-02') date(fixed(20000203135026.1234567, 10, 4)) -> date('2000-02-03 13:50:26')
longdate	longdate longdate(stringarg) longdate longdate(fixedarg) longdate longdate(int, int, int) longdate longdate(int, int, int, int, int) longdate longdate(int, int, int, int, int, int) longdate longdate(int, int, int, int, int, int, int)	convert arg to longdate type, similar to date function above.	longdate(fixed(20000203135026.1234567, 10, 5)) -> longdate('2000-02-03 13:50:26.1234500') longdate(2011, 3, 16, 9, 48, 12, 1234567) -> longdate('2011-03-16 09:48:12.1234567')
time	time time(stringarg) time time(fixedarg) time time(int, int) time time(int, int, int)	convert arg to time type, similar to date function above	
seconddate	seconddate(string) seconddate(int, int, int, int, int, int)	Convert to seconddate. One stringargs is a string with default parsing; two stringargs is a format string in the second arg, numeric args are the date components.	
secondtime	secondtime(string)	Converto to secondtime.	

Function	Syntax	Purpose	Example
	secondtime(string, string) secondtime(int, int, int)		

6.3.2 String Functions

String functions are scalar functions that perform an operation on a string input value and return a string or numeric value.

Function	Syntax	Purpose
strlen	int strlen(string)	Returns the length of a string in bytes, as an integer number.
midstr	string midstr(string, int, int)	Returns a part of the string starting at arg2, arg3 bytes long. arg2 is counted from 1 (not 0).
midstru	string midstru(string, int) string midstru(string, int, int)	Returns a part of the string starting at character or surrugate arg2, arg3 characters or surrogates long.
leftstr	string leftstr(string, int)	Returns arg2 bytes from the left of the arg1. If arg1 is shorter than the value of arg2, the complete string will be returned.
rightstr	string rightstr(string, int)	Returns arg2 bytes from the right of the arg1. If arg1 is shorter than the value of arg2, the complete string will be returned.
rightsru	string rightstru(string, int)	return arg2 characters from the right of string. If arg1 is shorter than arg2 characters, the complete string will be returned.
instr	int instr(string, string)	Returns the position of the first occurrence of the second string within the first string (> = 1) or 0, if the second string is not contained in the first.
instru	int instru(string, string)	return the character position of the first occurrence of the second string within the first string (>= 1) or 0, if the

Function	Syntax	Purpose
		second string is not contained in the first. This assumes inputs to be unicode expressed in utf-8, if the input is not encoded this way, the result will be less meaningful.
hextoraw	string hextoraw(string)	Convert a hexadecimal representation of bytes to a string of bytes. The hexadecimal string may contain 0-9, upper or lowercase a-f and no spaces between the two digits of a byte; spaces between bytes are allowed.
rawtohex	string rawtohex(string)	convert a string of bytes to its hexadecimal representation. The output will contain only 0-9 and (upper case) A-F, no spaces and is twice as many bytes as the original string.
ltrim	string ltrim(string) string ltrim(string, string)	removes a whitespace prefix from a string. The Whitespace characters may be specified in an optional argument. This functions operates on raw bytes of the UTF8-string and has no knowledge of multi byte codes (you may not specify multi byte whitespace characters).
rtrim	string rtrim(string) string rtrim(string, string)	removes trailing whitespace from a string. The Whitespace characters may be specified in an optional argument. This functions operates on raw bytes of the UTF8-string and has no knowledge of multi byte codes (you may not specify multi byte whitespace characters).
trim	string trim(string) string trim(string, string)	removes whitespace from the beginning and end of a string.
lpad	string lpad(string, int) string lpad(string, int, string)	add whitespace to the left of a string. A second string argument specifies the whitespace which will be added repeatedly until the string has reached the intended length. If no second string argument is specified, chr(32) (' ') gets added. This function operated on UTF-8 bytes and has no knowledge of unicode characters (neither for the

Function	Syntax	Purpose
		whitespace string nor for length computation).
rpad	string rpad(string, int) string rpad(string, int, string)	add whitespace to the end of a string. A second string argument specifies the whitespace which will be added repeatedly until the string has reached the intended length. If no second string argument is specified, chr(32) (' ') gets added. This function operated on UTF-8 bytes and has no knowledge of unicode characters (neither for the whitespace string nor for length computation).
replace	string replace(string, string, string)	replace every occurrence of arg2 in arg1 with arg3 and return the resulting string
upper	string upper(string)	return an all upper case version of the string. Unlike most other string functions, this also attempts to convert unicode characters in CESU encoding beside the usual a-z.
lower	string lower(string)	return an all lower case version of the string. Unlike most other string functions, this also attempts to convert unicode characters in CESU encoding beside the usual A-Z.
rightstru	string rightstru(string, int)	return arg2 characters from the right of string. If arg1 is shorter than arg2 characters, the complete string will be returned.
chars	chars(string)	return the number of characters in a string. This returns the number of characters in a UTF-8 encoded string. In a CESU-8 encoded string, it will return the number of 16-bit words of that string if it were encoded if UTF-16.
charpos	charpos(string, int)	return the position of the nth character in a string (n starting with 1). The string is interpreted as UTF-8 as it is in the chars() function above.

6.3.3 Mathematical Functions

Scalar math functions perform a calculation, usually based on input values that are provided as arguments, and return a numeric value.

Function	Syntax	Purpose	Example
sign	int sign(double) int sign(time) int sign(date)	Sign returns -1, 0 or 1 depending on the sign of its argument. Sign is implemented for all numeric types, date and time.	
abs	double abs(double) decfloat abs(decfloat) decfloat abs(decfloat) time abs(time)	Abs returns arg, if arg is positive or zero, -arg else. Abs is implemented for all numeric types and time.	
round	double round(double, int)	Round does rounding of absolute values toward zero while the sign is retained	round(123.456, 0) = 123 round(123.456, 1) = 123.5 round(-123.456, 1) = -123.5 round(123.456, -1) = 120
rounddown	double rounddown(double, int)	Rounddown rounds toward negative infinity making rounddown(-1.1, 0) = -2	rounddown(123.456, -1) = 120 rounddown(-123.456, -1) = -130

6.3.4 Date Functions

Date and time functions are scalar functions that perform an operation on a date and time input value and returns either a string, numeric, or date and time value.

Function	Syntax	Purpose
utctolocal	utctolocal(datearg, timezonearg)	Interprets datearg (a date, without timezone) as utc and convert it to the timezone named by timezonearg (a string)
localtoutc	localtoutc(datearg, timezonearg)	Converts the local datetime datearg to the timezone specified by the string timezonearg, return as a date
weekday	weekday(date)	Returns the weekday as an integer in the range 0..6, 0 is Monday.

Function	Syntax	Purpose
now	now()	Returns the current date and time (localtime of the server timezone) as date
daysbetween	daysbetween(date1, date2) daysbetween(daydate1, daydate2) daysbetween(seconddate1, seconddate2) daysbetween(longdate1, longdate2)	Returns the number of days (integer) between date1 and date2. The first version is an alternative to date2 - date1. Instead of rounding or checking for exactly 24 hours distance, this truncates both date values today precision and subtract the resulting day numbers, meaning that if arg2 is not the calendar day following arg1, daysbetween returns 1 regardless of the time components of arg1 and arg2.
secondsbetween	secondsbetween(seconddate1, seconddate2) secondsbetween(longdate1, longdate2)	Returns the number of seconds the first to the second arg, as a fixed point number. The returned value is positive if the first argument is less than the second. The return values are fixed18.0 in both cases (note that it may prove more useful to use fixed11.7 in case of longdate arguments).
component	component(date, int)	The int argument may be int the range 1..6, the values mean year, month, day, hour, minute, second, respectively. If a component is not set in the date, the component function returns a default value, 1 for the month or the day, 0 for other components. You can also apply the component function to longdate and time types.
addseconds	addseconds(date, int) addseconds(seconddate, decfloat) addseconds(longdate, decfloat)	Return a date plus a number of seconds. Fractional seconds is used in case of longdate. Null handling is (in opposition to the default done with adds) to return null if any argument is null.
adddays	adddays(date, int) adddays(daydate, int) adddays(seconddate, int) adddays(longdate, int)	Return a date plus a number of days. Null handling is (in opposition to the default done with adds) to return null if any argument is null.
quarter	quarter(date)	Return a string 'yyyy-Qn', yyyy being the year of the quarter and n the

Function	Syntax	Purpose
	quarter(date, month)	quarter of the year. An optional start month (of the fiscal year) may be supplied. For example, quarter(date('2011-01-01'), 6) is '2010-Q3' and quarter(date('2011-06-01'), 6) is '2011-Q1'.
format	format(longdate, string)	Date values may be used together with format strings, as described elsewhere in the NewDb documentation (look for descriptions of the TO_DATE and TO_CHAR SQL functions) . For example, format(longdate('2011-06-09 20:20:13.1234567'), 'YYYY/MM/DD"T"HH24:MI:SS.FF7')

6.3.5 Miscellaneous Functions

The table below lists the miscellaneous functions that you can use while creating expressions.

Function	Syntax	Purpose	Example
if	if(intarg, arg2, arg3)	return arg2 if intarg is considered true (not equal to zero), else return arg3. Currently, no shortcut evaluation is implemented, meaning that both arg2 and arg3 are evaluated in any case. This means that you cannot use if to avoid a divide by zero error which has the side effect of terminating expression evaluation when it occurs.	if("NETWR"<=500000,'A', if("NETWR"<=1000000,'B','C'))
jf	if(intarg, arg2, arg3)	The function jf behaves similar to if, only with SQL semantic. While if will return NULL if the predicate (first argument) is NULL (undefined), jf will use the else-value (arg3) in that case.	
in	in(arg1, ...)	return 1 (= true) if arg1 is equal to any of the remaining args, return 0 else	

Function	Syntax	Purpose	Example
case	case(arg1, default) case(arg1, cmp1, value1, cmp2, value2, ..., default)	return value1 if arg1 == cmp1, value2 if arg1 == cmp2 and so on, default if there no match	case("CATEGORY", 'A', 'LV', 'B', 'MV', 'HV')
box		The function box behaves similar to case, only with SQL semantic. While case will return NULL if arg1 is NULL, box will return the default in that case.	
isnull	isnull(arg1)	return 1 (= true), if arg1 is set to null and null checking is on during Evaluator run (EVALUATOR_MAY_RETURN_NULL)	
max	max(arg1, arg2, arg3, ...)	return the maximum value of the passed arguments list. An arbitrary number of arguments is allowed. Arguments must be at least convertible into a common type.	max(0, 5, 3, 1)
min	min(arg1, arg2, arg3, ...)	return the minimum value of the passed arguments list. An arbitrary number of arguments is allowed. Arguments must be at least convertible into a common type.	min(1, 2, 3, 4)
sqladd	sqladd(arg1, arg2)	sqladd behaves like the operator '+', with NULL handling changed to SQL standard. While the operator '+' returns the other argument when one argument is NULL, sqladd will return NULL if any of its arguments is NULL.	sqladd(if("VAL_B" = 0, int(null), "VAL_C") / "VAL_B", -1)

6.4 Manage Calculation Views with Missing Objects

If objects within a calculation view are missing, for example, if the objects or its references are deleted, then such calculation views are referred to as broken models. By using proxies, SAP HANA modeler helps you work with broken models and fix inconsistencies.

When you open broken models, modeler displays red decorators for all missing objects that are essential to successfully activate the calculation view.

Example:

If you have defined a calculation view CV1 on table T1 (C1, C2, C3) such that Attributes A1, A2, A3 is defined on columns C1, C2, C3 respectively. Now, if you remove column C3 from the table T1, then the attribute A3 becomes inconsistent. In such cases, modeler injects proxies for C3, and when you open the calculation view CV1 in the editor, modeler displays a red decorator for C3 and an error marker for A3 to indicate that it is inconsistent.

> **Note**
>
> If the connection to SAP HANA system is not available, and if you try to open a calculation view, then modeler uses proxies for all required objects and opens the calculation view in read-only mode. But, since the model is not broken, the red decorators and the error markers are not shown.

You can resolve inconsistencies in analytic views or attribute views or calculation views by performing one of the following:

- Deleting those missing objects that the calculation view requires. This action clears all references of missing object.
- Adjusting mappings of inconsistent objects.
- Deleting inconsistent objects.

6.5 Use Aliases for Table Names and Column Names

You can assign one or more alternate names (or aliases) to tables. For example, if you want to improve the readability of a table name or if you want to add the same table again to your data foundation node, then you can use aliases to avoid name conflicts.

Procedure

1. Aliases for table names.

   If you want to use aliases for table names, then:

   a. Select a node with data source.
   b. In the *Mapping* tab, select a table.
   c. Unhide the *Properties* section.
   d. In the *Alias* text box, provide an alias name for the table.
2. Aliases for column names.

   If you want to use alias names for column names, then:

   a. Select a node with data source.
   b. In the *Mapping* tab, select an output column.
   c. Unhide the *Properties* section.
   d. In the *Name* text box, provide an alias name for the column.

6.6 Replacing Nodes and Data Sources

Replace a view node with any of the other underlying view nodes or replace a data source in view node with other available data sources in the catalog object.

The column view for complex calculation views may contain multiple levels of view nodes. If you manually delete a node in column view (without using the replace view node feature) and add new node, you lose the semantic information of the deleted node. However, if your requirement is to replace the deleted view node with its underlying view node, then you can use the replace feature to replace the view node with its underlying node and retain the semantic information of the changed node. Similarly, you can also replace a data source in a view node with other available data sources in the catalog object.

Related Information

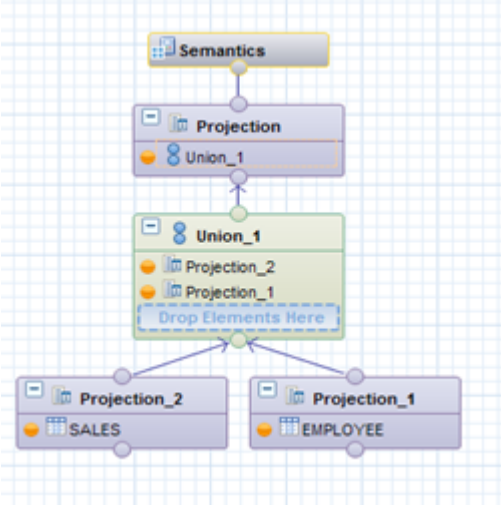
Replace a View Node in Calculation Views [page 121]
Replace a Data Source in Calculation Views [page 122]

6.6.1 Replace a View Node in Calculation Views

Replace a view node in a calculation view with any of its underlying nodes without performing a delete operation, and retain the semantic information of the changed node.

Context

For example, in the below calculation view, if you want to replace the node Union_1 with the node, Projection_1, then you can do it by executing the below procedure.



Procedure

1. Open required calculation view in view editor.
2. Select a node that you want to replace.
3. In the context menu, choose *Replace With Node*.
4. In the *Select New Node* dialog, select a view node that you want use for replacing.
5. Manage the source and target mappings accordingly.

   > **Note**
   >
   > You need to delete all unmapped target columns and references.

6. If you want to remove the node from the column view, select *Delete the node after replace*.

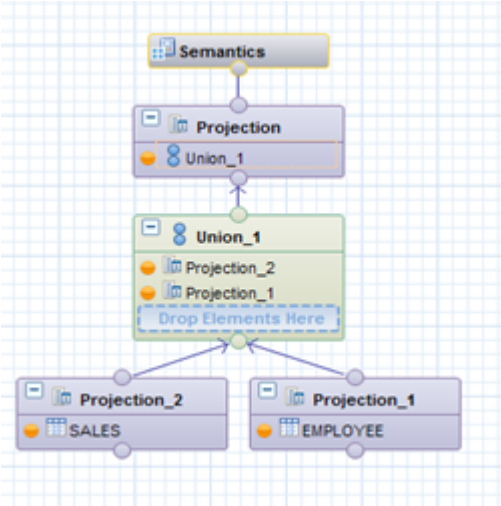
   If you do not select this checkbox, the view node appears as an orphan node in the column view.
7. Choose *Finish*.

6.6.2 Replace a Data Source in Calculation Views

Replace a data source in a calculation view with another data source in the catalog object without performing a delete operation, and retain the semantic information of the changed node.

Context

For example, in the below calculation view, if you want to replace the data source, Projection_1 of Union_1 with another data source in the catalog object, then you can do it by executing the below procedure.



Procedure

1. Open required calculation view in view editor.
2. Select a data source you want to replace.
3. In the context menu of the data source, choose *Replace With Data Source*.
4. In the *Find Data Sources* dialog, enter the name of the new data source.
5. Select the data source from the list.
6. Choose *Next*.
7. Manage the source and target mappings.
8. Choose *Finish*.

Important Disclaimer for Features in SAP HANA Platform, Options and Capabilities

SAP HANA server software and tools can be used for several SAP HANA platform and options scenarios as well as the respective capabilities used in these scenarios. The availability of these is based on the available SAP HANA licenses and the SAP HANA landscape, including the type and version of the back-end systems the SAP HANA administration and development tools are connected to. There are several types of licenses available for SAP HANA. Depending on your SAP HANA installation license type, some of the features and tools described in the SAP HANA platform documentation may only be available in the SAP HANA options and capabilities, which may be released independently of an SAP HANA Platform Support Package Stack (SPS). Although various features included in SAP HANA options and capabilities are cited in the SAP HANA platform documentation, each SAP HANA edition governs the options and capabilities available. Based on this, customers do not necessarily have the right to use features included in SAP HANA options and capabilities. For customers to whom these license restrictions apply, the use of features included in SAP HANA options and capabilities in a production system requires purchasing the corresponding software license(s) from SAP. The documentation for the SAP HANA optional components is available in SAP Help Portal at http://help.sap.com/hana_options. If you have additional questions about what your particular license provides, or wish to discuss licensing features available in SAP HANA options, please contact your SAP account team representative.

Important Disclaimers and Legal Information

Coding Samples

Any software coding and/or code lines / strings ("Code") included in this documentation are only examples and are not intended to be used in a productive system environment. The Code is only intended to better explain and visualize the syntax and phrasing rules of certain coding. SAP does not warrant the correctness and completeness of the Code given herein, and SAP shall not be liable for errors or damages caused by the usage of the Code, unless damages were caused by SAP intentionally or by SAP's gross negligence.

Accessibility

The information contained in the SAP documentation represents SAP's current view of accessibility criteria as of the date of publication; it is in no way intended to be a binding guideline on how to ensure accessibility of software products. SAP in particular disclaims any liability in relation to this document. This disclaimer, however, does not apply in cases of willful misconduct or gross negligence of SAP. Furthermore, this document does not result in any direct or indirect contractual obligations of SAP.

Gender-Neutral Language

As far as possible, SAP documentation is gender neutral. Depending on the context, the reader is addressed directly with "you", or a gender-neutral noun (such as "sales person" or "working days") is used. If when referring to members of both sexes, however, the third-person singular cannot be avoided or a gender-neutral noun does not exist, SAP reserves the right to use the masculine form of the noun and pronoun. This is to ensure that the documentation remains comprehensible.

Internet Hyperlinks

The SAP documentation may contain hyperlinks to the Internet. These hyperlinks are intended to serve as a hint about where to find related information. SAP does not warrant the availability and correctness of this related information or the ability of this information to serve a particular purpose. SAP shall not be liable for any damages caused by the use of related information unless damages have been caused by SAP's gross negligence or willful misconduct. All links are categorized for transparency (see: http://help.sap.com/disclaimer).

go.sap.com/registration/contact.html