PUBLIC

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

SAP HANA Modeling Guide

For SAP HANA XS Advanced Model

Content

1 Creating a Graphical Calculation View

A calculation view allows users to define more advanced slices on the data available in the SAP HANA database.

Calculation views are mainly used for analyzing operational data marts or running multidimensional reports on revenue, profitability, and so on. Calculation views consume various combinations of content data (that is, non-metadata) 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.

Calculation views simulate entities (such as customer, product, sales, and so on) and their relationships. Data visualization and analysis applications such as SAP BusinessObjects Explorer and Microsoft Office based reporting tools consume these calculation views and help decision makers in their decision process.

You can create calculation views with layers of calculation logic, which include measures sourced from multiple source tables, or advanced SQL logic, and much more. The data sources in a calculation view can include any combination of tables and calculation views. You can create joins, unions, projections, and aggregations 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. Calculation views can do the following:

- Support both OLAP and OLTP models.
- Support complex expressions (for example, IF, Case, Counter).
- Support analytic privileges (for example, restricting a user for a certain cost center).
- Support SAP ERP specific features (for example, client handling, language, currency conversion).
- Combine facts from multiple tables.
- Support additional data processing operations, (for example, Union, explicit aggregation).
- Leverage both Column and Row tables.

Related Information

Attributes and Measures [page 5]
Create Graphical Calculation Views [page 6]
Preview Calculation View Output [page 45]

1.1 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.

Related Information

Working With Attributes and Measures [page 47]

1.2 Create Graphical Calculation Views

Create graphical calculation views using a graphical editor to depict a complex business scenario. You can also create graphical calculation views to include layers of calculation logic.

Context

Graphical calculation views can bring together normalized data that are generally dispersed. You can combine multiple transaction tables while creating a graphical calculation view.

Procedure

1. Start the SAP Web IDE for SAP HANA tool in a browser.
2. If you want to create a new project for the calculation view, do the following:
   a. In the SAP Web IDE for SAP HANA, choose *File* > *New* > *Project from Template*.
   b. Choose the project template type.

      Currently, there is only one type of project template available, namely: *Multi-Target Application Project*. Select *Multi-Target Application Project* and choose *Next*.
   c. Type a name for the new MTA project (for example, `myApp` and choose *Next* to confirm.
   d. Specify details of the new MTA project and choose *Next* to confirm.
   e. Create the new MTA project; choose *Finish*.
3. Select the HDB module in which you want to create the calculation view.

   > **Tip**
   >
   > If you do not already have a database module, right-click the root folder of your new MTA project and, in the context menu, choose *New* > *HDB Module*.

4. Browse to the *src* folder, right-click it and choose *New* > *Calculation View*.
5. Enter details for the new calculation view.
   a. In the *Name* field, enter the name of the calculation view.
   b. In the *Data Category* dropdown list, select a value.
6. Choose *Create*.

   The tool launches a new graphical calculation view editor with a semantics node and default aggregation, or projection node depending on the data category that you select for the calculation view.
7. Continue modeling the graphical calculation view by dragging and dropping the necessary view nodes from the tools palette.
8. Add data sources.

   If you want to add data sources to the view node,

a. Select a view node.

b. Choose  .

c. In the *Find Data Sources* dialog, select the type of the data source.

d. 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.

9. Choose *Ok*.

> **Note**
>
> Supported data sources in view nodes in the current version.
>
> The *Find Data Sources* dialog box displays multiple objects types in the search results. But, depending on the selected view node, you can only add activated catalog tables, views, CDS entities, virtual tables, and table functions as data sources in the view nodes.

10. Define output columns.

    a. Select a view node.

    b. On the *Mapping* tab, select the column you want to add to the output.

    c. In the context menu, choose *Add To Output*.

    d. If you want to add all columns in a data source to the output, then from the context menu of the data source, select *Add To Output.*

> **Note**
>
> Using keep flag property. The *Keep Flag* property helps retrieve columns from the view node to the result set even when you don't, request it in your query. In other words, if you want to include those columns into the SQL `GROUP BY` clause, even when you don't select them in the query,
>
> 1. Select the view node.
> 2. On the *Mapping* tab, select an output column.
> 3. In the *Properties* section, set the value of *Keep Flag* property to *True*.

11. Define attributes and measures.

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

    a. Select the *Semantics* node.

    b. On the *Columns* tab, select a column value.

    c. In the *Type* dropdown list, select *Measure* or *Attribute*.

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

12. Choose *Save* on the menu bar to save your calculation view.

13. Build an HDB module.

    The build process uses the design-time database artifacts to generate the corresponding runtime objects in the database catalog.

    a. From the module context menu, choose *Build*.

Next Steps

After creating a graphical calculation view, you can modify the output to your needs. The table below shows how you can modify the calculation view.

Table 3: Working With View Nodes

Requirement	Task to Perform
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
Combine the results of two or more data sources.	Create Unions
Partition the data for a set of partition columns, and perform an order by SQL operation on the partitioned data.	Create Rank Nodes
Execute any of the available graph operations on the graph workspace.	Create Graph Nodes
Filter the output of projection or aggregation view nodes.	Filter Output of Aggregation or Projection View Nodes.

Table 4: Working With Columns

Requirement	Task to perform
Count the number of distinct values for a set of attribute columns.	Create Counters
Create new output columns and calculate their values at run time using an expression.	Create Calculated Columns
Assign semantic types to provide more meaning to attributes and measures in calculation views.	Assign Semantics
Parameterize calculation views and execute them based on the values users provide at query run time.	Create Input Parameters
Filter the results based on the values that users provide to attributes at run time.	Assign Variables
Create level hierarchies to organize data in reporting tools.	Create Level Hierarchies
Create parent-child hierarchies to organize data in reporting tools.	Create Parent-Child Hierarchies
Associate measures with currency codes and perform currency conversions.	Associate Measures with Currency
Associate measures with unit of measures and perform unit conversions.	Associate Measures with Unit of Measure
Group related measures together in a folder.	Group Related Measures

Table 5: Working With Calculation View Properties

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

Related Information

Working With View Nodes [page 16]
Preview Calculation View Output [page 45]
Working With Attributes and Measures [page 47]
Working With Calculation View Properties [page 85]
Additional Functionality for Calculation Views [page 100]
Defining Data Access Privileges [page 93]

1.2.1 Create Graphical Calculation Views with Star Joins

Star joins connect a central data entity to multiple entities that are logically related. You can create a graphical calculation view with star joins that join multiple dimensions to a single fact table.

Context

Star joins in calculation views help join a fact table with dimensional data. The fact table contains data that represent business facts such as, 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 more.

Procedure

1. Start the SAP HANA Web IDE for SAP HANA tool in a web browser.
2. If you want to create a new project for the calculation view, do the following:
   a. In the SAP Web IDE for SAP HANA, choose *File* > *New* > *Project from Template*.
   b. Choose the project template type.

      Currently, there is only one type of project template available, namely: *Multi-Target Application Project*. Select *Multi-Target Application Project* and choose *Next*.
   c. Type a name for the new MTA project (for example, **myApp** and choose *Next* to confirm.
   d. Specify details of the new MTA project and choose *Next* to confirm.
   e. Create the new MTA project; choose *Finish*.
3. Select the HDB module in which you want to create the calculation view.

   > **Tip**
   >
   > If you do not already have a database module, right-click the root folder of your new MTA project and, in the context menu, choose *New* > *HDB Module*.

4. Browse to the *src* folder, right-click it and choose *New* > *Calculation View*.

5. Enter details for the new calculation view.
   a. In the *Name* field, enter the name of the calculation view.
   b. In the *Data Category* dropdown list, select *CUBE*.

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

   c. Select the *With Star Join* checkbox.
6. Choose *Create*.

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

      > **Note**
      >
      > When adding calculation views as a data source in a star join node, you can use only calculation views with these data category types: dimension or blank.

   d. Choose *OK*.
8. Add inputs to the 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.
9. Maintain star join properties.
   a. Select the *Star Join* node.
   b. On the *Join Definition* tab, create joins by dragging a column from the shared calculation view to a column in the central fact table.
   c. Select the join.
   d. In the *Properties* section, define necessary join properties.
10. Choose *Save* in the menu bar to save your calculation view.
11. Build an HDB module.

    The build process uses the design-time database artifacts to generate the corresponding actual objects in the database catalog.
    a. From the module context menu, choose *Build*.

1.2.2 Create Graphical Calculation Views with Time Dimension

Add time dimension to a calculation view by using the standard time-related tables as data sources in the calculation view.

Prerequisites

You have synonyms to the standard time-related tables, (`M_TIME_DIMENSION_YEAR`, `M_TIME_DIMENSION_MONTH`, `M_TIME_DIMENSION_WEEK`, `M_TIME_DIMENSION`, `M_FISCAL_CALENDAR`) in the same HDB module in which you are creating the calculation view. The standard time-related tables are available in the `_SYS_BI` schema.

Procedure

1. Start the SAP HANA Web IDE for SAP HANA tool in a web browser.
2. If you want to create a new project for the calculation view, do the following:
   a. In the SAP Web IDE for SAP HANA, choose *File* > *New* > *Project from Template*.
   b. Choose the project template type.

      Currently, there is only one type of project template available, namely: *Multi-Target Application Project*. Select *Multi-Target Application Project* and choose *Next*.
   c. Type a name for the new MTA project (for example, **myApp** and choose *Next* to confirm.
   d. Specify details of the new MTA project and choose *Next* to confirm.
   e. Create the new MTA project; choose *Finish*.
3. Select the HDB module in which you want to create the graphical calculation view.

   > **Tip**
   >
   > If you do not already have a database module, right-click the root folder of your new MTA project and, in the context menu, choose *New* > *HDB Module*.

4. Browse to the *src* folder, right-click it and choose *New* > *Calculation View*.
5. Enter details for the new calculation view.
   a. In the *Name* field, enter the name of the calculation view.
   b. In the *Data Category* dropdown list, select *Dimension*.
   c. In the *Type* dropdown list, select *TIME*.
   d. Select the *Auto Create* checkbox if you want the tool to automatically include the time-related tables as data sources in the calculation view, and also to define filters and hierarchies in the view based on the selected calendar type and the granularity.
   e. In the *Calendar* dropdown list, select the required calendar type.
   f. If you have selected the *Gregorian* calendar type, in *Granularity* dropdown list, select the required granularity.

g. Based on the *Calendar* type and the *Granularity* type, in the *Table* field, use the *Find Data Sources* dialog to search select the synonym relevant to the time-related table.

6. Choose *Create*.

   The tool opens a new calculation view editor. If you have not selected *Auto Create*, you need to manually include the required time-related tables as data sources and define filters, hierarchies based on your requirement.

   > **Note**
   >
   > For the *Fiscal* calendar type, *Auto Create* only includes the `M_FISCAL_CALENDAR` table as data source in the calculation view. It does not automatically define any filters or hierarchies in the view.

7. Choose *Save* in the menu bar to save your calculation view.
8. Build an HDB module.

   The build process uses the design-time database artifacts to generate the corresponding actual objects in the database catalog.

   a. From the module context menu, choose *Build*.

Related Information

Generate Time Data [page 12]
Supported Calendar Types For Generating Time Data [page 14]
Supported Time Range for Generating Time Data [page 14]

1.2.2.1 Generate Time Data

Generate time data into the standard time-related tables that are available in the `_SYS_BI` schema. After generating the time data, you can use the standard time-related tables as data sources in the calculation view to add a time dimension to the view.

Prerequisites

You have `INSERT` privileges on the time-related tables. You can use the SAP HANA database explorer to add a *SYSTEM* database user.

Context

You use the SAP HANA database explorer to generate time related data. While generating the time data, you can specify the calendar type and the granularity.

Procedure

1. In SAP Web IDE for SAP HANA, choose *Tools* > *Database Explorer*.
2. Select the HDI service for which the user has `INSERT` privileges on the time-related tables.
3. In the menu bar, choose
4. Choose *Time Data* > *Generate*
5. In the *Calendar Type* dropdown list, select the required calendar type.
6. In the *From Year* and *To Year* text fields, enter the start year and end year for which you want to generate time data into the standard time-related tables.
7. 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.

8. If you have selected the *Fiscal* calendar type,
   a. In the *Schema* text field, enter the name of the variant schema that contains tables having variant data.

      > **Note**
      >
      > SAP tables T009 and T009B are typically used to store the variant data.

   b. In the *Variant* text field, enter the required variant.

      The variant specifies the number of periods along with the start and end dates.
9. Choose *Create*.

   > **Note**
   >
   > For the *Gregorian* calendar type, the tool generates the time 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 tool generates the time data into the `M_FISCAL_CALENDAR` table. These tables are present in `_SYS_BI` schema.

Related Information

Create Graphical Calculation Views with Time Dimension [page 11]
Supported Calendar Types For Generating Time Data [page 14]
Supported Time Range for Generating Time Data [page 14]

1.2.2.2 Supported Calendar Types For Generating Time Data

In SAP Web IDE for SAP HANA the following calendar types are supported for generating time data.

Table 6:

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.

1.2.2.3 Supported Time Range for Generating Time Data

For the Gregorian calendar type, based on the granularity you choose, the tool provides restrictions on the time range for which you can generate time data.

For each granularity level, the table below displays the time range that the tool supports for generating time 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

1.2.3 Supported View Nodes for Modeling Calculation Views

The SAP Web IDE for SAP HANA graphical calculation view editor offers various view nodes to model calculation views and obtain the desired output.

The table below lists shows the various view nodes.

Table 7:

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.	Union	For retrieving the names of all employees of a store, which has different branches, with each branch maintaining its own employee records table.
Join	Use *Join* node to query data from two data sources, based on a specified condition. Join nodes have two inputs.	Join	For retrieving customer details and location based on the postal code columns in the `CUSTOMER` and `GEOGRAPHY` tables. The `CUSTOMER` table has columns `Customer_ID`, `Customer_Name` and `Postal_Code`, and the `GEOGRAPHY` table has columns `Customer_ID`, `Postal_Code`, `Region` and `Country`.
Graph	Use *Graph* node to execute any of the available graph operations or actions on the graph workspace. A graph node is always the leaf node only.		Execute graph actions such as the shortest path or the strongest connection between components in the graph workspace. The graph workspace includes the definition of the vertex table and edge table that are required to execute the action.
Projection	Use *Projection* node to filter or obtain a subset of required columns of a data source (tables, views, table functions, and so on.). Projection nodes have one input.	Proje...	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 retrieving total sales of a product in a month. The supported aggregation types are SUM, MIN, VAR, STDDEV, MAX, COUNT, AVG.
Rank	Use *Rank* node to partition the data for a set of partition columns, and to perform an order by operation on the partitioned data.		Retrieving the top five products, based on sales, from a `TRANSACTION` table with columns `PRODUCT` and `SALES`.

> **Note**
>
> You can add data sources, unions, joins, projections or aggregations as inputs to view nodes.

Related Information

Working With View Nodes [page 16]

1.2.4 Working With View Nodes

View nodes are the building blocks of calculation views. They help you build complex and flexible analytic models, and each view node type possess specialized capabilities that can trigger advanced features in the database.

This section describes the different views nodes that you can use within graphical calculation views, along with examples of how you can use them to model calculation views.

Related Information

Create Joins [page 16]
Create Temporal Joins [page 18]
Create Unions [page 28]
Create Rank Nodes [page 35]
Create Graph Nodes [page 36]
Filter Output of Aggregation or Projection View Nodes [page 43]

1.2.4.1 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 combine records from both the data sources, so that they appears as a single record in the query results.

Procedure

1. Open the required calculation view in the view editor.
2. From the editor's tools palette, drag 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. On the *Mapping* tab, select the columns you want to add to the output of the join node.
   b. 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. On the *Join Definition* tab, create a join by dragging a column from one data source to a column in the other data source.
6. Edit join properties.
   a. Select the join.
   b. In the *Properties* section, define necessary join properties.
   c. If you want to switch the left tables and right tables, in the *Properties* section, choose 
7. (Optional) Use propose join cardinality.

   Modeler analyses the data of tables participating in the join and proposes an optimal cardinality that you can use while defining the join properties.

   a. Select the join.
   b. In the *Properties* section, choose .

      The tool can propose an optimal cardinality. Choose *Yes* to use the proposed cardinality

> **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, it is recommended not to provide any cardinality value.

Related Information

Create Temporal Joins [page 18]
Create Spatial Joins [page 20]
Text Joins [page 23]
Dynamic Joins [page 24]
Optimize Join Execution [page 27]
Join Properties [page 27]
Supported Join Types [page 28]

1.2.4.1.1 Create Temporal Joins

Temporal joins let you join the transaction data (fact table) with the master data, based on temporal column values from the transaction data and the time validity from the master data.

Procedure

1. Open the calculation view in the view editor.
2. Select the *Star Join* node.

   You can create temporal joins in the star join node only. The star join node must contain a calculation view with a *Dimension* data category type as the data source, and the input to the star join node must provide the central fact table.
3. Create a join.

   Create a join by dragging a column from the shared calculation view (master table) to a column in the other data source (fact table).
4. Select the join to view its details.
5. Define join properties.

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

   > **Note**
   >
   > You can create temporal joins in calculation views with the join type *Inner* only.
6. Define the temporal column and the temporal conditions.

   In the *Temporal Properties* section, provide values to create the temporal join.

   a. In the *Temporal Column* dropdown list, select a time column in the calculation view.
   b. In the *Temporal Condition* dropdown list, select a temporal condition.
   c. In the *From Column* and *To Column* dropdown lists, specify the start and end time values from the calculation view to fetch the records.

Related Information

Temporal Joins [page 19]
Temporal Conditions [page 19]
Example: Temporal Joins [page 19]

1.2.4.1.1.1 Temporal Joins

A temporal join lets you set time intervals for which you want to fetch records. It indicates the time interval mapping between the master data and the transaction data.

The temporal join, is based on the date field from the fact table and the time interval (to and from fields) from the master data view. The date field from the fact table is called temporal column.

This means that the tables are joined if the temporal column values in the fact table are within the valid time interval values from the master data view. A time interval is assigned to each record in the results set, and the records are valid for the duration of the interval to which they are assigned.

The supported data types for *Temporal Column*, *From Column*, and *To Column* are timestamp, date, and integers only.

1.2.4.1.1.2 Temporal Conditions

Temporal condition values, which you define in temporal joins, help determine whether to include or exclude the value of the FROM and TO date fields of the master data view, while executing the join condition.

The table below lists the temporal conditions.

Table 8:

Temporal Condition	Meaning
Include To Exclude From	This temporal condition includes the value of the *To Column* field and excludes the value of the *From Column* field while executing the join.
Exclude To Include From	This temporal condition excludes the value of the *To Column* field and includes the value of the *From Column* field while executing the join.
Exclude Both	This temporal condition excludes the value from both the *To Column* field and the *From Column* field while executing the join.
Include Both	This temporal condition includes the value from both the *To Column* field and *From Column* field while executing the join.

1.2.4.1.1.3 Example: Temporal Joins

Create temporal joins to join the master data with the transaction data (fact table), based on a time column value from the transaction data and the time validity columns from the master data.

Consider a dimension calculation view named `PRODUCT` (master data) with attributes `PRODUCT_ID`, `VALID_FROM_DATE`, and `VALID_TO_DATE` and a calculation view of type cube, `SALES` (transactional data) with attributes `PRODUCT_ID`, `DATE`, and `REVENUE`.

Now, you can create a temporal join between the master data view and transaction data using the attribute `PRODUCT_ID` to analyze sales of the product for a particular period.

1.2.4.1.2 Create Spatial Joins

Create spatial joins to query data from data sources that have spatial data.

Context

You use the join node in a calculation view to create spatial joins, and create them by joining two databases tables on columns of spatial data types.

Procedure

1. Open the required calculation view in view editor.
2. From the editor's tools palette, drag 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. On the *Mapping* tab, select the columns you want to add to the output of the join node.
   b. 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. On the *Join Definition* tab, create a join by dragging a column from one data source to a column in the other data source.
   b. In the *Properties* section, define the necessary join properties.
   c. If you want to switch the left tables and right tables, in the *Properties* section, choose 
6. Define spatial join properties.

   In the *Spatial Join* section, define the necessary 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* text field enter the required values. Similarly, if you select *Within Distance* as the predicate value, in the *Distance* text field, enter a fixed value.

b. In the *Execute Join If* dropdown list, select *Predicate Evaluates to True* if you want to execute the spatial join only if the predicate condition evaluates to true.

Related Information

Supported Spatial Predicates [page 21]
Supported Spatial Data Types [page 22]

1.2.4.1.2.1 Supported Spatial Predicates

The following spatial predicates are supported.

Table 9:

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.

1.2.4.1.2.2 Supported Spatial Data Types

The following spatial data types are supported in data models and query language for storing and accessing geospatial data.

Table 10:

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.

1.2.4.1.3 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.

1.2.4.1.4 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 25]

1.2.4.1.4.1 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.

1.2.4.1.5 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.

1.2.4.1.6 Join Properties

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

The tool lets 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 [page 28].
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. The system determines the cardinality when executing the join.
Language Column	The value of this property specifies the language column that modeler must use for executing text joins. For more information, see Text Joins [page 23].
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 [page 24].
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 [page 27].

1.2.4.1.7 Supported Join Types

When creating a join between two data sources (tables or column views), you specify the join type. The table shows the supported join types in SAP Web IDE for SAP HANA graphical calculation view editor.

Table 11:

Join Type	Description
Inner	This join type returns all rows when there is at least one match in both the data sources.
Left Outer	This join type returns all rows from the left data source, and the matched rows from the right data source.
Right Outer	This join type returns all rows from the right data source, and the matched rows from the left data source.
Text Join	This join type is used to obtain language-specific data from the text tables using a language column.
Full Outer	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.
Referential	This join type is similar to inner join type, but assumes referential intergrity is maintained for the join tables.

1.2.4.2 Create Unions

Use union nodes in 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 to 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 calculation view in the view editor.
2. From the editor's tools palette, 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. On the *Mapping* pane, select the columns you want to add to the output of the union 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. Assign constant value.

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

   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 *Constant Value* field, enter a constant value.
   e. Choose *OK*.

6. Create a constant output column.

   To create a new output column and assign a constant value to it,

   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 32]
Empty Union Behavior [page 33]
Prune Data in Union Nodes [page 30]

1.2.4.2.1 Prune Data in Union Nodes

Pruning data in union nodes help optimize the query execution. You create a pruning configuration table, which specifies the filter conditions to limit the result set, and prune data using this table.

Context

For pruning data in union nodes, create a pruning configuration table that the tool must refer to, and define this table in the view properties.

The tool cannot prune the data in union nodes if queries executed on the union node are unfolded, and if it does not perform any aggregation. In such cases, switch off the unfolding behavior with the hint, `NO_CALC_VIEW_UNFOLDING`. Unfolding is the normal query execution behavior in which the query execution is passed to the SQL engine or the optimizer after the calculation engine instantiates the query.

> **Note**
>
> Unfolding is not possible for complex calculation views.

Procedure

1. Open the required calculation view in which you want to prune data in union nodes.
2. Select the *Semantics* node.
3. Choose the *View Properties* tab.
4. In the *Pruning Config Table* field, choose value help.
5. In the *Find Data Sources* dialog, enter and search for the pruning configuration table.

   > **Note**
   >
   > You can use catalog tables or views as pruning configuration tables.

6. Select the table from the list.
7. Choose *Ok*.

Related Information

Pruning Configuration Table [page 31]
Example: Pruning Data in Union Nodes [page 32]

1.2.4.2.1.1 Pruning Configuration Table

Pruning configuration table helps prune data in union nodes. The table holds the filter conditions that limit the result set when you execute a query on the union node.

The tool refers to the pruning configuration table while executing queries on the union node. The pruning configuration table must have the following table structure:

Table 12:

SCHEMA	CALC_SCENARIO	INPUT	COLUMN	OPTION	LOW_VALUE	HIGH_VALUE

> **Note**
>
> All columns must be of type VARCHAR or NVARCHAR only.

The description for each of the column value is as follows:

- SCHEMA: Name of the schema that contains the calculation view
- CALC_SCENARIO: Name of the calculation view (fully qualified name).
- INPUT: Name of the data source in the union node (the data source or the view node ID in the repository model).
- COLUMN: Target column name
- OPTION: Operator. The following operations are allowed. (=, <, >, <=, >=, BETWEEN)
- LOW_VALUE and HIGH_VALUE: Filter conditions

> **Note**
>
> If you have already defined filters on columns outside of the pruning configuration table, for example, consider:
>
> Filter where “U1”. “Column1” = ‘5’
>
> Pruning Table: U1 , Column1, =, ’5‘
>
> In the above case, U1 is not pruned, because pruning table shows that all records in U1 has value “5” in Column1.
>
> Similarily, consider:
>
> Filter : where “U1”. “Column1” = ‘5’
>
> Pruning Table: U1 , Column1, =, ’6‘
>
> Here, U1 is pruned because pruning table shows that all records in U1 has value “6” in Column1. But, you cannot obtain any result even if the union operation is executed.

If you are creating multiple filter conditions using the same column, then the filter conditions are combined using the logical `OR` operator. Similarly, if you are using different columns to provide the filter conditions, then the filter conditions are combined using the logical `AND` operator.

Related Information

Example: Pruning Data in Union Nodes [page 32]

1.2.4.2.1.2 Example: Pruning Data in Union Nodes

Pruning data in union nodes of calculation views help optimize query execution. The below is an example of a pruning configuration table, and a query that you can possibly execute.

Table 13:

SCHEMA	CALC_SCENARIO	INPUT	COLUMN	OPTION	LOW	HIGH
a	b	u	C1	BETWEEN	2000	2005
a	b	u	C1	=	2008	
a	b	u	C2	<	1998	
a	b	u	C2	>	2005	

The above example is an equivalent of ('2000' <= C1 <= '2005' OR C1 = '2008') AND (C2 < '1998' OR C2 > '2005')

> **Note**
>
> The SQL queries must have numerical constants enclosed within single quotes. For example, the below query cannot be pruned:
>
> SELECT * from p1.p2/CV1 WHERE YEAR = 2008;
>
> You can prune the above query only if the numerical constants are enclosed within single quotes as shown below:
>
> SELECT * from p1.p2/CV1 WHERE YEAR = '2008';

1.2.4.2.2 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
3000	P2	ACTUAL
4000	P1	PLANNED
4000	P2	PLANNED

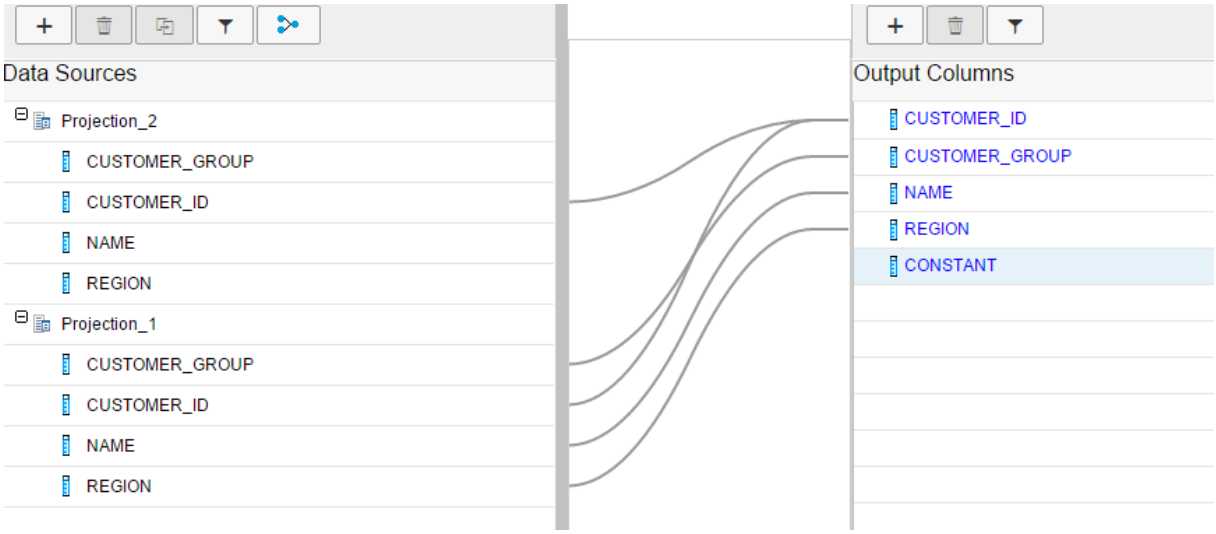
Now, you can identify the data source and its underlying data.

1.2.4.2.3 Empty Union Behavior

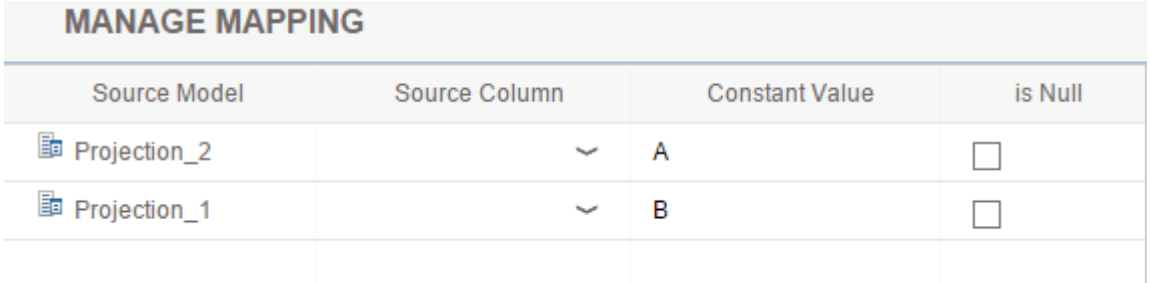
The *Empty Union Behavior* property determines whether queries on union nodes, ones with constant output columns, will return values when no other column from the data source is queried.

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 its value 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 a calculation view with this union node, and 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, as follows:

- If the *Empty Union Behavior* property is set to *No Row*, no data from *Projection _2* appears in the output data. Only data from *Projection_1* appears in the output data.
- If the *Empty Union Behavior* property is set to *Row with Constant*, 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.

1.2.4.3 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 14:

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

Sort Direction	Description
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.

1.2.4.4 Create Graph Nodes

SAP HANA Graph lets you create graph nodes in calculation views for various calculation scenarios. A graph node helps execute one of the available actions on a graph workspace and provides the output as a table.

Prerequisites

- You have at least one vertex table, one edge table, and one graph workspace, in the same HDB module as the calculation view.

Context

The SAP HANA Graph provides basic information for creating graphs to depict data relationships visually. Graphs are a powerful abstraction that can be used to model different kinds of networks and linked data coming from many areas, such as logistics and transportation, utility networks, knowledge representation, text processing, and so on.

Procedure

1. Open the required calculation view in the view editor.
2. From the editor's tools palette, drag a graph node to the editor.

   > **Note**
   >
   > You can add graph nodes in calculation views as leaf nodes only.

3. Add data sources.
   a. Select the graph node.
   b. Choose  .
   c. In the *Find Data Sources* dialog, enter the name of a graph workspace and select it from the list.

      > **Note**
      >
      > You can add only a graph workspace as the data source in graph nodes. A HDB module can have multiple graph workspaces, vertex tables or edge tables.

   d. Choose *Ok*.
4. Select an action.

   Select a supported action (a graph operation or algorithm) that you want to execute on the graph workspace.

   a. In the *Actions* tab, choose  .
   b. Select the required action.
5. Define the parameters necessary to execute the action.

   Based on the action you have selected, define values for all parameters necessary to execute the action. You can also create and use input parameters to define some of the values.

   For pattern matching, you can use the graphical editor to visually compose the pattern. The pattern comprises of vertices connected by edges. Use the + icon in the menu bar to add a vertex to the editor and drag the cursor from one vertex to the other to create an edge. After composing your pattern, define conditions on the vertex. In the *Mapping* tab, define the output columns.

   For pattern matching with cypher editor, write the required pattern matching query using the supported subset of the Cypher query language on the given graph workspace. After writing the query, choose *Generate Colum* to create the output columns.

6. View output columns.

   The output columns of a graph node varies depending on the action you select.

   a. Select the *Columns* tab to view output columns of the graph node
7. Create input parameters, if required.

   a. Select the *Parameters* tab and create a new input parameter.You can create input parameters to parameterize the execution of graph node and provide values at run time to execute the graph node.

      You can use this input parameter while defining the parameters necessary to execute a selected action.
8. Add the graph node as input to other nodes.

   You can add the output of graph nodes as inputs (to data source) to multiple view nodes at the same time. If you are using Shortest Path action in your graph node, you can optionally add the output edge table from the graph node as an input to other view nodes. Select the graph node and choose  to add the edge table as an input to a view node.

Related Information

SAP HANA Graph Model [page 38]
Graph Workspaces [page 39]
Supported Graph Actions in Calculation Views [page 39]

1.2.4.4.1 SAP HANA Graph Model

SAP HANA Graph is an integral part of SAP HANA core functionality. It expands the SAP HANA platform with native support for graph processing and allows executing typical graph operations on the data stored in an SAP HANA system.

The SAP HANA Graph model provides basic information for creating graphs to depict data relationships visually. It allows typical graph operations to be run on the data stored in an SAP HANA system.

Graphs are powerful abstraction that can be used to model different kinds of networks and linked data coming from many areas, such as logistics and transportation, utility networks, knowledge representation, text processing, and so on.

In SAP HANA, a graph is as a set of vertices and edges. Each edge connects two vertices; one vertex is denoted as the source and the other as the target. Edges are always directed and there can be two or more edges connecting the same two vertices. Vertices and edges can have an arbitrary number of attributes. Both vertices and edge attributes consists of a name, which is associated with a data type and a value. Edge attributes consist of the same information.

For more information about the SAP HANA Graph model, see SAP HANA Graph Reference Guide.

1.2.4.4.2 Graph Workspaces

A graph workspace is an object in the database that defines a graph that consists of tables and columns.

A graph workspace includes the following:

- Vertex table
- Edge table
- Key column in the vertex table
- Key column in the edge table
- Source in the edge table
- Target in the edge table

> **Note**
>
> You can use CDS tables or SQL syntax to create vertex tables and edge tables. For more information on how to create vertex tables and edge tables, see the SAP HANA Graph Reference guide.

Example Artifact Code

The following code shows a simple example of a graph workspace definition for XS advanced HDI:

Code Syntax

`/src/my_graph_workspace.hdbgraphworkspace`

```
GRAPH WORKSPACE MY_GRAPH_WORKSPACE
EDGE TABLE THE_EDGE_TABLE
  SOURCE COLUMN source TARGET COLUMN target KEY COLUMN edge_id
VERTEX TABLE THE_VERTEX_TABLE
  KEY COLUMN vertex_id
```

Plug-in Configuration

In the configuration file for the HDI container (`.hdiconfig`), the plugin configuration should look like the following example:

Code Syntax

`.hdiconfig`

```
"hdbgraphworkspace" : {
   "plugin_name" : "com.sap.hana.di.graphworkspace",
   "plugin_version": " "
}
```

1.2.4.4.3 Supported Graph Actions in Calculation Views

A graph node allows executing one of the available actions on the given graph workspace and providing results as table output.

The below table lists the supported graph actions.

Table 15:

Action	Description
Neighborhood	Retrieves the neighboring vertices within the given radius (depth) from the given start vertices.
Shortest Path	Provides the information for the shortest path from the starting vertex to all available vertices
Strongly Connected Components	Calculates strongly connected components (SCC) in the given graph workspace.
Pattern Matching	Identifies all the subgraphs (within a graph) that match the given pattern.

Related Information

Neighborhood [page 40]
Shortest Path [page 41]
Strongly Connected Components [page 42]
Pattern Matching [page 42]

1.2.4.4.3.1 Neighborhood

The Neighborhood graph action retrieves the neighboring vertices within the given radius (depth) from the start vertices.

This action lets you specify multiple start vertices, choose a traversal direction, set filters on vertices and edges, and set minimum and maximum depth (radius) of the neighborhood.

This action has the following additional parameters:

Table 16:

Parameter	Value
`startVertices`	Set of start vertex keys.
`direction`	Traversal direction, can use one of the following values: `any`, `incoming` and `outgoing` (default: `outgoing`).
`minDepth`	Minimum depth (radius) of the neighborhood. 0 means the start vertices are included in the result.
`maxDepth`	Maximum depth (radius) of the neighborhood.

Parameter	Value
`vertexFilter`	Vertex filter expression analogous to SQL's where-clause on the vertex table (default: `empty`).
`edgeFilter`	Edge filter expression analogous to SQL's where-clause on the edge table (default: `empty`).
`depthColumn`	Depth column name (default: DEPTH).

The output includes the following:

Table 17:

Parameter	Value
vertex table	Contains the set of explored vertices with vertex key and optionally the depth level of the neighborhood.

1.2.4.4.3.2 Shortest Path

This action provides information for the shortest path from the starting vertex to all available vertices.

Shortest Path action returns the shortest paths from the provided start vertex to all reachable vertices in the graph also known as single-source shortest path (SSSP). The resulting shortest paths form a tree structure with the start vertex at the root. All other vertices carry the shortest distance (smallest weight) information. The non-negative edge weights are read from the column provided in the edge table.

This action has the following additional parameters:

Table 18:

Parameter	Value
`startVertex`	Start vertex key
`inputWeightColumn`	A column in the edge table that contains edge weights. If omitted, edge weights are set to 1.
`outputWeightColumn`	Output weight (shortest distance) column name (default: WEIGHT)

The output includes the following:

Table 19:

Parameter	Value
vertex table	Contains vertex keys and corresponding smallest weights (shortest distances)
edge table	Optional edge table with shortest path(s)

1.2.4.4.3.3 Strongly Connected Components

The Strongly Connected Components action calculates strongly connected components (SCC) in the given graph workspace.

Strongly connected components partition the given graph workspace into largest subgraphs possible, where every vertex is reachable from every other vertex. Knowing strongly connected components is helpful for cycle detection, because, by definition, the cycles can only exist within strongly connected components and they cannot cross strongly connected component boundaries.

The only output of this action is a table containing the vertex key column and a `COMPONENT` column containing strongly connected component indices. All vertices having the same component index belong to the same strongly connected component.

This action has the following additional parameters:

Table 20:

Parameter	Value
`componentColumn`	Component index column name (default: COMPONENT)

The output includes the following:

Table 21:

Parameter	Value
vertex table	contains vertex key column and component index column

1.2.4.4.3.4 Pattern Matching

Pattern matching is a kind of graph query, which involves finding all the subgraphs (within a graph) that match the given pattern.

The graphical calculation view editor in the SAP Web IDE for SAP HANA tool provides two options to model calculation views with graph nodes that help execute graph pattern queries. The first option is to use the graphical pattern matching editor. The second option is to describe the pattern in Cypher query language (Cypher is a registered trademark of Neo Technology, Inc.).

Pattern Matching with Graphical Pattern Editor

The graphical pattern editor lets you visually compose pattern matching queries, execute them, and visualize the results.

The graphical pattern contains a set of vertex variables, a set of edge variables, a set of filter conditions, a projection list, an order-by list, a limit and an offset.

The result of the graphical pattern matching query is a projection of subgraphs within a given graph workspace that are isomorphic to the given pattern. The columns of the result table correspond to the projection list of the given pattern. Every row in the result table corresponds to a matching isomorphic subgraph.

Pattern Matching with Cypher Query Language

Pattern matching action with Cypher query language lets you execute pattern matching queries written in the supported subset of the Cypher query language on the given graph workspace.

Table 22:

expression	A query string written in the supported subset of the Cypher query language containing one MATCH clause and one RETURN clause.

The output includes the following:

Table 23:

result table	Every row corresponds to a matching subgraph. Every row contains attributes of vertices and edges specified in the RETURN clause.

1.2.4.5 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)
   ```

   .

1.2.5 Supported Data Categories for Calculation Views

Modeler supports three types of data categories to classify calculation views. The table below explains each of these data category types.

Table 24:

Data Category	Description
Cube	Calculation views with data category Cube are visible to the reporting tools and support data analysis with multidimensional reporting. For graphical calculation views with data category as *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.
Dimension	Calculation views with a data category Dimension are not visible to the reporting tools and do not support data analysis. However, you can use these calculation views as data sources in other calculation views 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 can 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. For graphical calculation views with data category *Dimension*, the tool provides projection as the default view node.

Data Category	Description
Default	Calculation views with data category Default 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 calculation views, which have data category as Cube, for any multidimensional reporting purposes. For graphical calculation views with data category *Default*, modeler provides projection as the default view node.

Related Information

Create Graphical Calculation Views [page 6]

1.3 Preview Calculation View Output

After modeling calculation views based on your requirements, deploy them and preview its output within the same tool.

Context

The tool displays the output data in tabular format. You can also export and download the output data to `.csv` files.

> **Note**
>
> You can preview a deployed calculation view not only to preview the output data, but also to preview the SQL query that the tool generates for the deployed calculation view. In addition, for an activated view, you can also preview the output of its intermediate nodes.

Procedure

1. Start the SAP Web IDE for SAP HANA tool in a browser.
2. In the *Project Explorer* view, select the required calculation view to preview its output.
3. In the context menu, choose *Data Preview*.
4. Provide input parameter or variable value.

   If you have defined any input parameters in the calculation view, provide the required input parameter values.

a. Select the required operator.
b. Provide values for the *From* and *To* fields based on the selected operator.
c. Choose  .
d. Select *Open Content*.

   The tool displays the output data in tabular format.

> **Note**
>
> The tool does not provide a value help list for the input parameter, if the input parameter in the calculation view is mapped to an the input parameter in its underlying calculation view, or if the input parameter in the calculation view is of type, *Static List*.

5. Apply filters.

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

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

   a. If you want to view the SQL query that the tool generates for the deployed calculation view, choose  .
8. Preview output of intermediate nodes.

   If you have activated the calculation view, you can also preview the output of any of its intermediate nodes. This helps know the output data, which is then passed on to its higher node levels.
9. Open the calculation view in the view editor.
10. Select the required intermediate node.
11. In the context menu, choose *Data Preview*.

2 Working With Attributes and Measures

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

Columns in calculation views are either a measure or attribute. Measures are columns where you define an aggregation. If calculation views are used in SQL statements, 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 because they don't need to be aggregated.

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

Related Information

Create Counters [page 47]
Create Calculated Columns [page 48]
Assign Semantics [page 51]
Create Input Parameters [page 57]
Assign Variables [page 55]
Using Currency and Unit of Measure Conversions [page 72]
Enable or Disable Attributes for Drilldown in Reporting Tools [page 81]
Create Restricted Columns [page 50]
Handle Null Values in Columns [page 81]
Convert Attribute Values to Required Formats [page 82]
Group Related Measures [page 83]
Using Hierarchies for Reporting [page 63]

2.1 Create Counters

Use counters to count the number of distinct values in one or more attribute columns. Counters are columns that display 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, the counter displays the count of distinct combinations of both columns.

Procedure

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

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

3. In the *Calculated Column/Counters* tab, choose  .
4. Select *Counter*.
5. In the *General* section, provide a name and description for the new counter.
6. If you want to hide the counter for data preview, select *Hidden*.
7. In the *Counter* section, choose  .
8. In the dropdown list, choose an attribute column.

   > **Note**
   >
   > Transparent Filter Flag
   >
   > You have to set the transparent filter flag on attribute columns to *True* to get correct counter results in the following scenarios:
   > - Ccalculation views are stacked on top of other dependent calculation views, and you have defined count-distinct measures in the dependent views.
   > - Queries on main calculation views let you filter column that you do not want to project.
   >
   > For the above scenarios, you have to set the transparent filter flag to *True* for the filtered, non-projected columns. The filter must be set for these columns in all nodes of the upper calculation view and in the default node of the lower dependent calculation view. This helps correct unexpected counter numbers.

2.2 Create Calculated Columns

Create new output columns and calculate their values at run time, 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 calculation view in the view editor.
2. Select the view node in which you want to create the calculated column.
3. In the *Calculated Columns* tab, choose  .
4. In the *General* section, enter a name and description for the new calculated column.
5. In the *Data Type* dropdown list, select the data type for the calculated column.
6. Enter values for length and scale based on the selected data type.
7. Define calculated columns semantics.

   In the *Semantics* section, provide semantics information for the new calculated column. You can create calculated attributes or calculated measures using attributes or measures respectively.

   a. In the *Column Type* dropdown list, select a value.
   b. If you want to create a calculated measure and enable client-side aggregation for the calculated measure, select *Enable client side aggregation* and, in the *Aggregation Type* dropdown list, select an aggregation type that the client must perform on calculated measures..

      > **Note**
      >
      > After selecting the *Enable client side aggregation* checkbox, in the *Aggregation Type* dropdown list, select an aggregation type that client needs to perform on the calculated measures.

   c. Drilldown in the calculated attributes in the reporting tools.

      By default, the tool lets 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.
8. If you want to hide the calculated column in reporting tools, select *Hidden*.
9. Provide an expression.

   You can create an expression using the SQL language or the column engine language.

   a. In the *Expression* section, choose *Expression Editor*.
   b. In the dropdown list, select the language you want to use to create the expression.
   c. In the expression editor, enter a valid expression.

      The tool computes this expression at run time 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 `'NOTEBOOK'` 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 drag and drop the expression elements, operators, and functions to the editor and create expressions.

   d. Choose *Validate Syntax* to identify any inconsistencies in the expression.
   e. Choose *Back*.

2.3 Create Restricted Columns

Create restricted columns as an additional measure 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

For restricted columns, modeler uses the aggregation type of the base column. You can define restrictions on measures using any of the below approaches:

- Deinfe restrictions on measures using values from other attribute columns.
- Define restriction on measures using expressions based on SQL language or column engine language.

Procedure

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

   > **Note**
   >
   > You can create restricted columns in star join nodes or aggregation nodes only.

3. Choose the *Restricted Columns* tab.
4. Choose +
5. In the *General* section, provide a name and label for the new restricted column.
6. In the *Base Measure* dropdown list, choose a base measure value on which you want to apply restrictions.
7. If you want to hide the restricted column in reporting tools, select the *Hidden* checkbox.
8. Define restrictions using column values.

   You can define restrictions on the base measure using values from other attribute columns.

   a. In the *Restrictions* section, select *Columns*.
   b. Choose + .
   c. In the *Column* value help, selecte an attribute column.
   d. In the *Operator* dropdown list, select a required operation to define the condition.
   e. In the *Value* field, select the value type.

      The value type is either a fixed value from the attribute column or an input parameter. Provide or select the required value and choose *OK*.
   f. If you want to apply restrictions based only on the defined conditions, choose *Include*.

> **Note**
>
> You can define multiple conditions using the same attribute columns or different attribute columns. For example, the expression, ("CUSTOMER_ID" = '10' OR "CUSTOMER_ID" = '2010') AND ("CUSTOMER_NAME" = '' ") has three conditions.

9. Apply restrictions using expressions.

   You can create an expression using the SQL language or the column engine language.

   > **Note**
   >
   > Only a limited list of SQL functions are supported for expressions in SQL language.

   a. In the *RESTRICTIONS* section, select *Expression*.
   b. In the *Language* dropdown list, select an expression language.
   c. In the *Expression Editor*, enter a valid expression.

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

2.4 Assign Semantics

Assigning semantics to measures or attributes in calculation views helps define output structure of views.

Context

Client tools use semantic types to represent data in appropriate format. Assigning semantics to attributes or measures at design time helps provide meaning to attributes and measures.

Procedure

1. Open the required calculation view in the view editor.
2. Select the *Semantics* node.
3. Choose the *Columns* tab.
4. Select a measure or attribute.
5. Choose  .
6. In the *Semantic Type* dropdown list, select the required semantic type.

7. Choose *OK*.

Related Information

Extract and Copy Semantics From Underlying Data Sources [page 52]
Propagate Columns to Semantics [page 53]
Supported Semantic Types for Measures [page 54]
Supported Semantic Types for Attributes [page 54]

2.4.1 Extract and Copy Semantics From Underlying Data Sources

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

Context

For example, let's say you are modeling a complex calculation view with multiple underlying data sources, and these data source have their own semantic definitions for their columns. In this case, you can extract and copy the semantic definitions of columns from their underlying data sources to define the semantics of the calculation view. Extracting and copying the semantic definitions this way helps you save the effort of manually defining the semantics of the calculation view.

Procedure

1. Open the required calculation view in view editor.
2. Select the *Semantics* node.
3. In the *Columns* tab, choose  .

   In the *Extract Semantics* dialog box, the tool displays the output columns from the underlying data sources.
4. Select the columns.

   By default, the column properties, *Label*, *Label Column*, *Aggregation Type* and *Semantic Type*, is enabled. This means that the values of these column properties are extracted from the underlying views to the semantic definition of the calculation view. If you want to avoid copying the definition, you can clear the *Aggregation Type* checkbox.

> **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.

5. Select the hierarchies.

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

   a. In the *Extract Semantics* dialog box, select the *Hierarchies* tab.

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

   b. Select the hierarchies defined in the underlying views.
   c. If the calculation view already has hierarchies with same name, in the *New Name* field, provide a different name.
6. If you want to override the existing semantic definition of the calculation view with the extracted semantics, select *Overwrite semantics already defined*.
7. Choose *Ok*.

2.4.2 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 columns from underlying view nodes as output columns in all view nodes and up to the semantic node.

Context

The tool lets you propagate columns from an underlying view node to all nodes in the joined path up to the semantics node. This means you don't have to define the output columns of each node, if the same columns are already defined in its underlying node and you need these output columns in the above nodes up to the semantic node. Propagating columns are useful in complex calculation views that many levels of view nodes.

Procedure

1. Open the required calculation view in the 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 tool propagates the columns you select to all view nodes and up to the semantics node. If some columns are already available in any of the view nodes in the propagated path, the tool does not propagate columns to those nodes.

2.4.3 Supported Semantic Types for Measures

The following semantic types are supported for measures.

- Amount with Currency Code
- Quantity with Unit of Measures

2.4.4 Supported Semantic Types for Attributes

The following sematic types are supported 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

For associating attributes with Quantity with Unit of Measures or Amount with Currency Code, you need to specify the unit column and currency column respectively.

2.5 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 25:

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.

Related Information

Supported Variable Types [page 57]

2.5.1 Supported Variable Types

Graphical calculation views support the following variable types for assigning variables to attributes.

Table 26:

Type	Description
Single value	Filter and viewoutput data, based on a single attribute value. For example, to view the sales of a product, where the month is equal to January.
Interval	Filter and view a specific set of output data. For example, to view the expenditure of a company from March to April.
Range	Filter view output 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.

2.6 Create Input Parameters

Input parameters help you parameterize calculation views and execute them based on the values that users provide to the input parameters at query run time. The engine the values that users have provided for the input parameters as the `PLACEHOLDER` clause of the SQL statement.

Context

You create an input parameter at design time (while creating calculation views), and provide values to the engine and then run calculation views accordingly. For example, if you want the calculation view to provide data for a specific region, `REGION` is a possible input parameter. In that case, create the input parameter `REGION` and use it in a filter expression. At run time, the data is filtered based on the value you provided to the input parameter, `REGION`.

Procedure

1. Open the required calculation view in the view editor.
2. Select a view node in which you want to create the input parameter.

> **Note**
>
> Input parameters are not specific to the view nodes and are defined at the view level.

3. Choose the *Parameters* tab.
4. Choose + .
5. Select *Input Parameter*.
6. Maintain input parameter details
   a. In the *General* section, provide the input parameter name and description.
   a. If you want to configure the input parameter to mandatorily accept a value at run time, select the *Is Mandatory* checkbox.
   b. If you want to configure the input parameter to accept multiple values at run time, 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 run time.
7. Define input parameter type.
   a. In the *Parameter Type* dropdown list, select an input parameter type.

      Table 27:

Input Parameter Type	Description	Next Steps
Column	At run time, the tool provides a value help with attribute data. You can choose a value from the attribute data as an input parameter value.	a. 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 calculation view that contains the required attribute. b. In the *Reference Column* dropdown list, select an attribute.
Direct	Specify the data type and length and scale of the input parameter value that you want to use at run time. 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 a semantic type as Currency.	a. In the *Data Type* dropdown list, select the data type. b. Provide the *Length* and *Scale* for the data type you choose. c. Optionally, in the *Semantic Type* dropdown list, specify the semantic type for your input parameter.

Input Parameter Type	Description	Next Steps
Static List	At run time, the tool 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 + to provide the list values.
Derived From Table	At run time, the tool uses the value from the table's return column as the input parameter value. This means you don't need to provide any values to the input parameter at run time. 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 selected, in the *Return Column* dropdown list, select a column value. c. In the *Filters* section, define filter conditions to filter the values of the return column. d. If you want to provide a different value to the parameter at run time (overriding the default value) and do not want the tool to automatically use the value returned by the table as the input parameter, select *Input Enabled* checkbox. If this checkbox is enabled, at run time the tool displays the value returned by the table as the default value, but you can override this value based on your requirements.
Derived From Procedure/Scalar Function	At run time, the tool uses the value returned from the procedure or scalar as the input parameter value.	a. In the *Procedure/Scalar Function* field, provide the name of procedure or scalar function. b. If you want to provide a different value to the parameter at runtime (override the default value) and don't want the tool to automatically use the value returned by the procedure, select *Input Enabled* checkbox. If this checkbox is enabled, then at run time the tool displays the value returned by the procedure or scalar function as the default value, but you can override this value based on your requirements.

> **Note**
>
> You cannot configure input parameters of types, *Derived from table* or *Derived from Procedure/Scalar Function* to mandatorily accept a value or to accept multiple values at run time

8. Provide default values.

Provide a default value that the tool must consider as the input parameter value if no value is provided to the input parameter at run time.

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

> **Note**
>
> The tool cannot accept a combination of expressions and constants as default values for input parameters.

Table 28:

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 + . 2. In *Type* dropdown list, select *Constant*. 3. In *Value* field, provide a constant value.
Expression	You can create an expression in SQL language or the column engine language. 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, 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* to identify any inconsistencies in the expression. 6. Choose *Back.*. For example, you can evaluate the expression `date(Now())`, and use the result as the default input parameter value at run time.

> **Note**
>
> Providing multiple default constant values. If you have configured the input parameter to accept multiple values at the run time 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 run the calculation view.

Related Information

Map Input Parameters [page 61]
Input Parameters [page 62]

2.6.1 Map Input Parameters

If you are modeling calculation views on top of other calculation views that have input parameters defined for them, you can map the input parameters of underlying data sources to the input parameters of calculation views that you are modeling.

Context

When you map parameters of the current view to the parameters of the underlying data sources, the filters are moved down to the underlying data sources at run time. This reduces the amount of data transferred across them.

> **Note**
>
> Input parameters are available for mapping only when used in the dependent data sources.

Procedure

1. Open the required calculation view in the view editor.
2. Select the *Semantics* node.
3. Choose the *Parameters* tab.
4. Choose  .
5. In the *Select Type* dropdown list, select the required value.

Value	Description
Data Sources	If you are using other calculation views as data sources in a calculation view, and if you want to map input parameters of the underlying data sources to the input parameters of the calculation view.
Procedures/Scalar functions for input parameters	If you are using input parameters of type procedure/scalar functions, and if you want to map the input parameters defined in the procedures or scalar functions to the input parameters of the calculation view.
Views for value help for variables/input parameters	If you are using input parameters that refer to external views for value help references, and if you want to map input parameters of external views to the input parameters or variables of the calculation view.
Views for value help for attributes	If you are filtering the attributes in the underlying data sources using other calculation views or tables as value help, and if you want to map the input parameters in these value help views to the input parameters of the calculation view.

6. Manage mappings for the source and target input parameters by dragging a value from the source to a value in the target.

7. If you want to auto-map, based on the source and target input parameter names,
   a. On the toolbar, choose  .

   > **Note**
   >
   > If you are using auto-map, to map all unmapped parameters at the source, the system creates input parameters with same name at the target.

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

Related Information

Create Input Parameters [page 57]
Input Parameters [page 62]

2.6.2 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 29:

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.

2.7 Using Hierarchies for Reporting

The modeler in SAP Web IDE for SAP HANA supports creating hierarchies using graphical modeling tools. Hierarchies help 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-related hierarchy comprises of levels such as Fiscal Year, Fiscal Quarter, Fiscal Month, and so on. You can graphically create the following two types of hierarchies:

- Level Hierarchies
- Parent-Child Hierarchies

Related Information

Create Level Hierarchies [page 64]
Create Parent-Child Hierarchies [page 66]
Root Node Visibility [page 71]
Orphan Nodes [page 71]

2.7.1 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. Start the SAP Web IDE for SAP HANA tool in a web browser.
2. Open the required calculation view in the view editor.
3. Select the *Semantics* node.
4. Choose the *Hierarchies* tab.
5. Choose the icon  .
6. Choose the *Level Hierarchy* menu option
7. Provide hierarchy details

   In the *General* section, provide additional details for the hierarchy.

   a. Provide a name and label to the new hierarchy.
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* tab, choose  to create a level.
   b. In the *Column* value help list, select the required 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.

      > **Note**
      >
      > MDX client tools use attribute values to sort 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.

    For supporting different business scenarios, the tool allows you to define certain additional properties for the hierarchy. In the *Properties* section, define the required values.

    a. Select the *Aggregate All Nodes* checkbox 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 the *Aggregate All Nodes* checkbox, 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, do not select the checkbox. This helps the engine to execute the hierarchy more efficiently.

    b. If you want the level hierarchy to support multiple parents for its elements, select the *Multiple Parent* checkbox.
    c. If you want to cache the hierarchy data after executing a query on the hierarchy, select *Cache*.
    d. If you wanto convert the empty string values in the fact table that do not have corresponding values in the master table to NULL, select *Convert empty string values to NULL*.

       The empty values are replaced with NULL in reporting tools.
    e. 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.
    f. In the *Root Node Visibility* dropdown list, select a value.

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

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

    > **Note**
    >
    > If you select *Stepparent* option to handle orphan nodes, in the *Stepparent* text field, 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

Node Style [page 66]
Query Shared Hierarchies [page 70]
Orphan Nodes [page 71]

Root Node Visibility [page 71]

2.7.1.1 Node Style

Node style value deinfed for a level hierarchy helps modeler identify the format of the node ID in reporting tools. For example, whether the node ID must comprise of the level name or node name or both in the reporting tools.

Table 30:

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

2.7.2 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. Start the SAP Web IDE for SAP HANA tool in a web browser.
2. Open the required calculation view in the view editor.
3. Select the *Semantics* node.
4. Choose the *Hierarchies* tab.
5. Choose the icon  .
6. Choose the *Parent Child Hierarchy* menu option
7. Provide hierarchy details

   In the *General* section, provide additional details for the hierarchy.

   a. Provide a name and label to the new hierarchy.
8. Create parent-child elements

   a. In the *Nodes* tab, choose  .
   b. In the *Child* value help list, select the required column value as the child attribute.
   c. In the *Parent* value help list, select the required column value as a parent attribute of 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* text field, 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.

      You can provide the *Root Node* value either as a fixed value or as an input parameter. In the *Value Help* dialog, select (input parameter) or provide (fixed value) the required value and choose *OK*.

      > **Note**
      >
      > If you have selected root node visibility value as *Add Root node*, it is mandatory to provide the root node value.
9. Define parent-child hierarchy properties.

   For supporting different business scenarios, the tool allows you to define certain additional properties for the hierarchy. In the *Properties* section, define the required values.

   a. Select the *Aggregate All Nodes* checkbox 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 the *Aggregate All Nodes* checkbox, 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, do not select the checkbox. This helps the engine to execute the hierarchy more efficiently.

   b. If you want the parent-child hierarchy to support multiple parents for its elements, select the *Multiple Parent* checkbox.

c. If you want to cache the hierarchy data after executing a query on the hierarchy, select *Cache*.
d. If you wanto convert the empty string values in the fact table that do not have corresponding values in the master table to NULL, select *Convert empty string values to NULL*.

   The empty values are replaced with NULL in reporting tools.
e. 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.
f. In the *Root Node Visibility* dropdown list, select a value.

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

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

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

h. Handling cycles in hierarchy

   A parent-child hierarchy is said to contain cycles if the parent-child relationships in the hierarchy have a circular reference. You can use any of the following options to handle the behavior of such hierarchies at load time.

   Table 31:

Options	Description
Break up at load time	The nodes are traversed until a cycle is encountered. The cycles are broken-up at load time.
Traverse completely, then breakup	The nodes in the parent-child hierarchy are traversed once completely and then the cycles broken up.
Error	Displays error when a cycle is encountered.

10. If you want to add additional attributes to execute the hierarchy, then

    a. In *Additional Attributes* section, choose + .
    b. In the *Attributes* value help list, select the required 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 + .
    b. In the *Order By* value help list, select the required column value that modeler must use to order hierarchy members.
    c. In *Sort Direction* dropdown list, select the required value that modeler must use to sort and display the hierarchy members.

> **Note**
>
> MDX client tools use attribute values to sort hierarchy members.

12. Enable hierarchy for time dependency

    In some business cases, the elements in an hierarchy are changing elements (time dependent elements). In such cases, you can enable the parent-child hierarchy as a time dependent hierarchy. This helps to create hierarchies that are relevant for specific time periods and to 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 the required column value.
    c. In the *Valid To Column* dropdown list, select the required column value.

       The tool uses the *Valid From Column* and *Valid To Column* values as the required validity time for 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

Query Shared Hierarchies [page 70]
Orphan Nodes [page 71]
Root Node Visibility [page 71]

2.7.3 Query Shared Hierarchies

Enable SQL access to shared hierarchies and query them using SQL statements at runtime. This is necessary to obtain correct aggregation results for hierarchy nodes.

Context

Calculation views in star join nodes are referred to as shared dimensions. The tool includes all attributes and hierarchies of a shared dimension to the output of the star join calculation view. You can enable SQL access to shared hierarchies and query them at runtime.

> **Note**
>
> Not all reporting tools support SQL access to shared hierarchies. For example, this feature does not work with BI clients such as MDX or Design Studio.

Procedure

1. Open the required star join calculation view.
2. Select the *Semantics* node.
3. If you want enable SQL access to only a selected list of shared hierarchies, then:
   a. Choose the *Hierarchies* tab.
   b. In the *Shared* section, select a hierarchy to which you want to enable SQL access.
   c. Expand the *SQL Access* section
   d. Select the *Enable SQL access* checkbox.
4. If you want to enable SQL access to all shared hierarchies of the current version of the calculation view, then:
   a. Select the *View Properties* tab.
   b. In the *General* section, select the *Enable Hierarchies for SQL access* checkbox.

Results

After you enable SQL access to shared hierarchies, the tool generates a *Node column* and a *Hierarchy Expression Parameter* for the shared hierarchy with certain default names. You can use the node column to filter and perform SQL `GROUP BY` operation and use the hierarchy expression parameter to filter the hierarchy nodes (for example, if you want query only the children nodes of a parent-child hierarchy).

For example, the below query shows uses the node column to filter and perform SQL `GROUP BY` operation:

Sample Code

```
select "HierarchyNodeColumn",
sum("Revenue") as "Revenue"
FROM "_SYS_BIC"."mini/CvSalesCubeHier" group by "HierarchyNodeColumn";
```

2.7.4 Orphan Nodes

The tools provides different options to handle orphan nodes in hierarchies. For example, you can specify whether the orphan nodes must be treated as error or root nodes or so on.

Table 32:

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.

2.7.5 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 33:

Root Node Visibility	Description
Add Root Node If Defined	This is applicable only for parent-child hierarchies. The tool adds a root node only if you have defined a root node value.
Add Root node	The tool 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 requires a root node for reporting purposes. Modeler creates a root node with the technical name ALL.
Do Not Add Root Node	The modeler does not add any additional root node to the hierarchy.

2.8 Using Currency and Unit of Measure Conversions

If measures in calculation views represent currency or unit values, associate them with currency codes or unit of measures accordingly in the calculation view.

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.

Modeler performs currency conversions based on the source currency value, target currency value, exchange rate, and date of conversion. It also 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 run time.

Related Information

Associate Measures with Currency [page 72]
Associate Measures with Unit of Measure [page 77]
Reuse a Currency Conversion or Unit Conversion Definition [page 80]

2.8.1 Associate Measures with Currency

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

Prerequisites

The HDB module that contains the calculation view in which you are performing currency conversions either has all tables necessary for currency conversions or it has the synonyms referring to the standard SAP currency tables TCURC, TCURF, TCURN, TCURR, TCURT, TCURV, TCURW and TCURX that are available in other schemas.

> **Note**
>
> If you are not using synonyms, the structure of the tables (table definition) in the HDB module for currency conversions must be similar to the standard SAP currency tables.

Context

Associating measures with currency codes is also necessary for currency conversions. For example, let's say you want to generate a sales report for a region in a particular currency code, but you have the sales data in the database table with a different currency code. In this case, create a calculation view by using the table column containing the sales data in the different currency as a measure, and associate the measure with your desired currency to perform the currency conversion. Activate the calculation view to generate the reports you need.

Procedure

1. Open the required calculation view in the view editor.
2. Select the *Semantics* node.

   > **Note**
   >
   > You can also assign assocaite measures in any of the *Aggregation* nodes with currency code values and perform currency conversion.

3. In the *Columns* tab, select a measure to associate it with currency code.
4. In the menu bar, choose  .
5. Choose the *Assign Semantics* menu option.
6. In the *Semantic Type* dropdown list, select *Amount with Currency Code*.
7. Copy currency code and currency conversion definition.

   You can copy the currency code values and currency conversion definition from a selected measure to another measure.

   a. In the *Copy From* dropdown list, select the required measure.

   The tool populates the *Assign Semantic* dialog with the currency code values and currency conversion definition associated with the selected measure.
8. Select a display currency code value.

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

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

      Table 34:

Value	Description
Fixed	Associate the measure with a currency code available in the currency table. This table either is a synonym to the SAP currency table, TCURC, or has a structure similar to the standard SAP currency table, TCURC.
Column	Associate the measure with an attribute column available in the calculation view.

   b. In the value help list, select the required value based on the selected display currency type.

> **Note**
>
> The tool populates the value help list only if you have defined all necessary conversion tables. Define the conversion tables in the *Conversion Table* section. If the value help list is not populated, you can manually enter the required display currency code.

c. Choose *OK*.

9. Enable decimal shift.

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

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

10. Enable conversion.

    a. If you want to convert the measure value to another currency, select the *Conversion* checkbox.

11. Enable rounding.

    If you have enabled conversion, and you want to round the result value after currency conversion to the number of digits of the target currency, do the following:

    a. Select the *Rounding* checkbox.

> **Note**
>
> Use rounding cautiously because rounding errors could accumulate after subsequent aggregations.

12. Enable 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 caused by 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 *Shift back*.

13. Specify the conversion tables to refer to for currency conversion.

    a. In the *Rates* field, select a table from the value help that provides the exchange rate information.

       This table either is a synonym to the SAP currency table, TCURR, or has a structure similar to standard SAP currency table, TCURR.

    b. In the *Configuration* field, select a table from the value help that provides the exchange rate types.

       This table either is a synonym to the SAP currency table, TCURV, or has a structure similar to standard SAP currency table, TCURV.

    c. In the *Prefactors* field, select a table from the value help that provides the conversion factors information.

       This table either is a synonym to the SAP currency table, TCURF, or has a structure similar to standard SAP currency table, TCURF.

    d. In the *Notations* field, select a table from the value help that provides the quotations information.

This table either is a synonym to the SAP currency table, TCURN, or has a structure similar to standard SAP currency table, TCURN.

e. In the *Precisions* field, select a table from the value help that provides information on decimal places in currencies.

   This table either is a synonym to the SAP currency table, TCURX, or has a structure similar to standard SAP currency table, TCURX.

14. Provide additional details for currency conversions.

    a. In the *Client* dropdown list, select the required value that modeler must use for currency conversion rates.

    Table 35:

Value	Description
Fixed	Set a fixed client value or 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.
Input Parameter	Set an input parameter to provide the client value to the tool at run time. Select the required input parameter from the value help.

    b. Specify the source currency.

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

    Table 36:

Value	Description
Fixed	Select the source currency value from the currency table. 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.

    c. Specify the target currency.

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

    Table 37:

Value	Description
Fixed	Select the target currency value from the currency table. 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 the tool at run time. Select the required input parameter from the value help.

    d. Specify the exchange rate type.

    In the *Exchange Type* dropdown list, select the required value.

Table 38:

Value	Description
Fixed	Select the exchange rate value from the currency table. Provide the required value in the value help.
Column	Attribute calculated 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 the tool at run time. Select the required input parameter from the value help.

e. Specify the date for currency conversion.

   In the *Conversion Date* dropdown list, select the required value.

   Table 39:

Value	Description
Fixed	Select the fixed conversion date from the calendar.
Column	Attribute value 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 the tool at run time. Select the required input parameter from the value help.

f. Specify the exchange rate

   If you want to specify the exchange rate using a column value from the calculation view, in the value help for *Exchange Rate*, select a column value.

15. 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.
16. Generate a result currency column.
    a. If you want the tool to generate a column to store the result currency conversion values, select the *Generate Result Currency Column* checkbox.

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

17. Manage error handling.

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

    Table 40:

Value	Description
Fail	The tool displays error for conversion failures at data preview.
Set to NULL	The tool sets the values for corresponding records to NULL at data preview.

Value	Description
Ignore	The tool displays unconverted values for the corresponding records at data preview.

18. Choose *OK*.

2.8.2 Associate Measures with Unit of Measure

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

Prerequisites

The HDB module that contains the calculation view in which you are performing unit conversions either has all tables necessary for unit conversions or has synonyms referring to the standard SAP unit tables T006, T006D, and T006A available in other schemas.

> **Note**
>
> If you are not using synonyms, the structure of the tables (table definition) in the HDB module for unit conversions must be similar to the standard SAP unit tables.

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 calculation view in the view editor.
2. Select the *Semantics* node.

   > **Note**
   >
   > You can also assign assocaite measures in any of the *Aggregation* nodes with unit code values and perform unit conversion.

3. In the *Columns* tab, select a measure to associate it with unit code.
4. In the menu bar, choose  .

5. Choose the *Assign Semantics* menu option.
6. In the *Semantic Type* dropdown list, select *Quantity with Unit Of Measure*.
7. Copy unit code and unit conversion definition.

   You can copy the unit code values and unit conversion definition from a selected measure to another measure.

   a. In the *Copy From* dropdown list, select the required measure.

   The tool populates the *Assign Semantic* dialog with the unit code values and unit conversion definition associated with the selected measure.
8. Select a display unit value.

   The tool displays the measure values with this unit in reporting tools.

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

      Table 41:

Value	Description
Fixed	Associate the measure with a unit code available in the unit table. This table has a structure similar to any of the standard SAP unit tables, T006, T006A or T006D.
Column	Associate the measure with an attribute column available in the calculation view.

   b. In the value help list, select the required value based on the selected display unit type.

      > **Note**
      >
      > The tool populates the value help list only if you have defined all necessary conversion tables. Define the conversion tables in the *Conversion Table* section. If the value help list is not populated, you can manually enter the required display unit code.

   c. Choose *OK*.
9. Enable conversion.

   a. If you want to convert the unit measure value to another unit, select the *Enable Conversion* checkbox.
10. Specify the unit tables to refer for unit conversion.

    a. In the *Rates* field, select a table from the value help that provides the unit conversion information.

       This table either is a synonym to the SAP unit table, T006, or has a structure similar to standard SAP unit table, T006.

    b. In the *Dimension* field, select a table from the value help that provides information on dimensions.

       This table either is a synonym to the SAP unit table, T006D, or has a structure similar to standard SAP unit table, T006D.
11. Provide additional details for unit conversions.

    a. In the *Client* dropdown list, select the required value that the tool must use for unit conversion factors.

       Table 42:

Value	Description
Fixed	Set a fixed client value or select a session client for unit conversion factors. Provide the required value in the value help.

Value	Description
Column	Attribute column available in the calculation view to provide the client value. Select the required value from the value help.
Input Parameter	Input parameter to provide the client value to the tool at run time. Select the required input parameter from the value help.

b. Specify the source unit.

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

Table 43:

Value	Description
Fixed	Select the source unit value from the unit tables. 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.

c. Specify the target unit.

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

Table 44:

Value	Description
Fixed	Select the target unit value from the unit tables. 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.

12. 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 isn't available in reporting tools. You can only consume them using other calculation views to perform calculations.

13. Error handling.

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

Table 45:

Value	Description
Fail	The tool displays error for conversion failures at data preview.
Set to NULL	The tool sets the values for corresponding records to NULL at data preview.

Value	Description
Ignore	The tool displays unconverted value for the corresponding records at data preview.

14. Choose *OK*.

2.8.3 Reuse a Currency Conversion or Unit Conversion Definition

Reuse the currency conversion or unit conversion definition of a selected measure in multiple other measures, at a time.

Context

Procedure

1. Open the required calculation view in the view editor.
2. Select the *Semantics* node.
3. In the *Columns* tab, select a measure for which you have already enabled and defined currency conversion or unit conversion.
4. In the menu bar, choose  .
5. Choose *Copy Conversion Properties*.
6. In the *Copy From* dropdown list, select the required measure to reuse its conversion definition.
7. Choose *Next*.
8. Select all target measures in which you want to reuse the definition.
9. For any of the selected target measure, if you have already defined currency conversion or unit conversion, and if you want to override the existing conversion definition, select *Override Existing Conversion Properties*.
10. Choose *Finish*.

2.9 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.

2.10 Handle Null Values in Columns

Define default values for columns (both attributes and measures) in the event that no value is provided during an INSERT operation. The system uses these default values in the reporting tools to replace any null values in columns.

Context

Columns, both attributes and measures can contain undefined values or null values. You can handle such cases by defining default values that replaces the null values in reporting tools. For example, you can replace the column values that appear with null value representation, **?** (question mark) with default value **Null** or with any other user defined value.

Procedure

1. Open the calculation view in the view editor.
2. Select the *Semantics* node.
3. Choose the *Columns* tab.
4. Select a measure or attribute.
5. Select the *Enable Null Handling* checkbox

> **Note**
>
> Configure default column properties.
>
> If you do not see *Enable Null Handling* checkbox, choose the  icon in the menu bar and ensure you have selected the *Enable Null Handling* column property. Use the horizontal scroll bar to view all the available column properties.

6. In the *Default Value* text field, provide a default value.

> **Note**
>
> If you have enabled null handling for columns, and if you have not provide any default value, the tool considers the integer `0` as the default value for columns. For columns of data type `VARCHAR` and `NVARCHAR`, if you have not defined a default value after enabling null handling, the tool considers empty string as the default value.

2.11 Convert Attribute Values to Required Formats

Assign conversion functions to attribute columns. These functions help maintain conversion from any internal to external format and from any external to internal format.

Context

For each attribute column, or input parameters or variables, you can assign two scalar functions. These functions help maintain conversion of values to required formats. For example, the ABAP table stores data in YYYYMMDD format. You can use a scalar function that converts the internal value, 20160503 to 2016.05.03 and use the new formatted value for reporting purposes.

You can also preserve the order of value, for example, see the below table:

Table 46:

Stored Data Type Format	Stored Value	Formatted Value	Preserving Order
ABAP Date	20160503	05.03.2016	No
ABAP Date	20160503	2016.05.03	Yes

The below are example scenarios where you can use scalar functions to convert and format values

- If you want to display attribute values in the reporting tools in a specific format, but these values are stored internally in a different format.
- If you want to provide values to filters, variablesor parameters in a specific format, but internally these values are stored in different formats.

Procedure

1. Open the required calculation view in the view editor.
2. Select the *Semantics* node.
3. Choose the *Columns* tab.
4. Select an attribute.
5. In the *Semantics* pane, choose 
6. Assign conversion functions.
   a. If you want to use a scalar function to format any internally stored value, in the *Internal to External* field provide the required scalar function.

      Use the *Find Data Sources* dialog to search for the required scalar function.
   b. If you want to format an external value, for example, variables or input parameters, in the *External to Internal* field provide the required scalar function.

      Variables and parameters of type column implicitly inherit the conversion properties.
   c. If you want to preserve the order, for example, 20150305 as 2015.03.05, select the *Preserve Order* checkbox.
   d. Choose *OK*.
7. Assign conversion functions to parameters and variables.

   You can also explicitly assign conversion functions to input parameters and variables. For example, for input parameters of type derived, static list, direct and so on.
   a. Select the *Parameters* tab.
   b. Select the required input parameter or variable.
   c. In the *Conversion Function* section, assign the required function.
   d. If you want to preserve the order, for example, 20150305 as 2015.03.05, select the *Preserve Order* checkbox.

2.12 Group Related Measures

Create folders in calculation views to logically group related measures in a calculation view. For example, you can group planned measures and related measures in separate folders.

Context

A single calculation view contain multitple measures. While modeling your calculation view, create a *Display Folder* to group related measures within them. You can also create multiple folders within a single display folder.

Procedure

1. Open the required calculation view in the view editor.
2. Select the *Semantics* node.
3. Choose the *Columns* tab.
4. In the *Semantics* pane, choose .

   The *Display Folder* dialog displays the list of measures in the calculation view and a folder structure.
5. Create a new display folder.

   a. In the *Display Folder* dialog, choose .

      You can create more than one display folder. If you want to create a folder within another display folder, select the required folder and choose .

   b. Double-click the folder to edit the folder name.
6. Group related measures

   a. From the *Measure* pane, drag and drop the required measure to it respective folder.
7. Choose *OK*.

3 Working With Calculation View Properties

While modeling calculation views, you can define certain properties for the calculation views so that the tool can refer to those values and run the view accordingly.

For example, you can define a property, which controls how the tool must access the data from the database or identify how to execute the calculation view and so on. This section describes the commonly used 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 85]
Invalidate Cached Content [page 88]
Deprecate Calculation Views [page 89]
Quick Reference: Calculation View Properties [page 90]
Enable Calculation Views for Time Travel Queries [page 87]

3.1 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 86]

3.1.1 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 47:

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.

3.2 Enable Calculation Views for Time Travel Queries

Time travel queries are queries against the historical states of the database. You can execute a time travel query on a calculation view to query data for a specified time in past.

Context

If you have enabled time travel for calculation 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 calculation views as a timestamp query:

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

SAP HANA supports creating history tables. These tables help associate time-related information to your data.

When you use history tables as data sources in calculation views, you also need to specify a parameter that you want to use to provide the timestamp to the time-travel queries at run time.

Procedure

1. Open the required calculation view in the view editor.
2. Select the *Semantics* node.
3. In the *View Properties* tab, select the *Enable History* checkbox.
4. 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.

3.3 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.

Related Information

Enable Support for Cache Invalidation [page 89]

3.3.1 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.

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.

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

Procedure

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

3.4 Deprecate Calculation Views

Deprecated calculation views signifies that although a calculation view is supported for modeling activities within the tool, it is not recommended to use it in other calculation views or in analytic privileges.

Context

As a data modeler, because of various reasons, you can deprecate calculation views, which you do not recommend for use in other calculation views or analytic privileges. This helps other users or data modelers using this calculation view know that the view is deprecated or not recommended for use.

Procedure

1. Open the required calculation view in the view editor.
2. Select the *Semantics* node.
3. In the *View Properties* tab, select the *Deprecate* checkbox.

Results

The tool displays a warning in the menu bar of the view editor for those calculation views or analytic privileges that are either deprecated or modeled with deprecated calculation views.

3.5 Quick Reference: Calculation View Properties

While modeling a calculation view, you can define values for calculation view properties. These values determine the behavior of a calculation view at runtime.

When you are modeling calculation views, you can define the following properties in the *View Properties* tab of the *Semantics* node.

Table 48:

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 Calculation Views [page 44] .
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 93].
Default Member	The value of this property helps the tool identify the default member for all hierarchies in the calculation views.
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 88].
Default Client	The value of this property determines whether the tool must filter the calculation view data based on a fixed client value, a session client value, or as cross client (does not filter data) at run time. For more information, see Filter Data for Specific Clients [page 85]

Properties	Description
Propagate Instantiation to SQL Views	The value of this property helps the tool identify whether it has to propagate the instantiation handled by the calculation engine to the CDS or SQL views built on top of this calculation view. If the value is set to True, the tool propagates the instantiation to the CDS or SQL views. This means that, attributes that a query (on a SQL view built on top of this view) does not request are pruned and not considered at run time. For information on calculation engine instantiation process, see SAP Note 1764658.
Always Aggregate Result	This value of this property determines whether the engine must always enforce an aggregation in the semantics node of the calculation view irrespective of whether you have defined aggregations in the default node.
Deprecate	The value of this property determines whether a user does not recommend using the calculation view in other modeler objects. If the value is set to True, it indicates that although the calculation view is supported for modeling activities, it is not recommended for use. For more information, Deprecate Calculation Views [page 89].
Translate	The value of this property determines whether the tool must write the translation texts strings into repository text tables. If the value is set to True, the tool writes the text strings in calculation views into the repository text tables.
Run With	The value of this property helps modeler identify the authorization it has to use while selecting the data from the database and for executing the calculation view or procedure. If the property is set to *Definer's Rights*, the tool uses the authorizations of the user who defines the view or procedure. Similarly, if the property is set to *Invoker's Rights*, then modeler uses the authorizations of the current user to access data from the database.
Cache	The value of this property determines whether you have enabled support for cache invalidation. For more information, see Enable Support for Cache Invalidation [page 89]
Enable History	The value of this property determines whether you can execute time travel queries on the selected calculation view. If the property is set to True, you can execute time travel queries on the calculation view. For more information, see Enable Calculation Views for Time Travel Queries [page 87].
History Input Parameter	Input parameter that specifies the timestamp in time-travel queries.
Execute In	The value of this property impacts the output data. It determines whether the tool must execute the calculation view in SQL engine or column engine. For more information, see 1857202.
Pruning Config Table	The value of this property determines the pruning configuration table that the tool must use to prune data in union nodes. For more information, see Prune Data in Union Nodes [page 30].
Count Star Column	The value of this property is set to *row.count* in calculation views, which were created by migrating analytic views having the row.count column. The row.count column was used internally to store the result of `SELECT COUNT(*)` queries. You can also select a column from the calculation view as *Count Star Column*. In this case, the column you select is used to store the result of `SELECT COUNT(<column_name>)`.

Properties	Description
Analyticview Compatibility Mode	The value of this property helps the join engine identify whether it has to ignore joins with N:M cardinality, when executing the join. If the value of this property is set to True, the join engine prunes N:M cardinality joins if the left table or the right table in the star join node does not request for any field, and if no filters are defined on the join.
Enable Hierarchies for SQL access	This view property is applicable only for calculation views with star join. The value of this property helps the tool identify whether you have enabled SQL access to shared hierarchies. For more information, see Query Shared Hierarchies [page 70]

4 Defining Data Access Privileges

Use the analytic privilege editor in the SAP Web IDE for SAP HANA tool to create analytic privileges.

You create analytic privileges to 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 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 calculation 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.

Related Information

Create Static SQL Analytic Privileges [page 93]
Create Dynamic SQL Analytic Privileges [page 96]
Create Analytic Privileges Using SQL Expressions [page 97]
Static SQL Analytic Privileges [page 99]
Dynamic SQL Analytic Privileges [page 99]

4.1 Create Static SQL Analytic Privileges

For creating static SQL analytic privileges, you use attribute columns from views to define fixed restrictions on data access. These restrictions are defined in the analytic privilege editor at design time.

Prerequisites

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

2. If you want to use a SQL analytic privilege to apply data access restrictions on CDS views, you have defined the referenced CDS views using the `WITH STRUCTURED PRIVILEGE CHECK` clause.

Procedure

1. Start the SAP Web IDE for SAP HANA in a browser.
2. If you want to create a new project for the calculation view, do the following:
   a. In the SAP Web IDE for SAP HANA, choose *File* > *New* > *Project from Template*.
   b. Choose the project template type.

      Currently, there is only one type of project template available, namely: *Multi-Target Application Project*. Select *Multi-Target Application Project* and choose *Next*.
   c. Type a name for the new MTA project (for example, **myApp** and choose *Next* to confirm.
   d. Specify details of the new MTA project and choose *Next* to confirm.
   e. Create the new MTA project; choose *Finish*.
3. Select the HDB module in which you want to create the analytic privilege.
4. Browse to the *src* folder.
5. In the context menu, choose *New* > *Analytic Privilege*.
6. Enter details for the new analytic privilege.
   a. In the *Name* field, enter the name of the analytic privilege.
7. Choose *Create*.

   The tool opens the analytic privilege editor where you can define the analytic privilege.
8. Select models to define the scope of the analytic privilege.

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

   a. Choose  .
   b. In the *Find Data Sources* dialog, search and select the models for which you want apply the analytic privilege restrictions.

      > **Note**
      >
      > You can only add calculation views and CDS views to the secured models list.

   c. Choose *OK*.
9. Define validity.

   In the *Privilege Validity* section, specify the time period for which the analytic privilege is valid. You can specify more than one time period for which the analytic privilege is valid.

   a. Choose  .

b. In the *Operator* dropdown list, select the required operator.
c. Based on the selected operator, specify the time period (From and To) for the which the analytic privilege is valid.

10. Define the attribute restrictions.

    The tool 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 define more than one attribute restriction.

    a. In the *Assign Attribute Restrictions* section, choose + .
    b. In the *Restriction Type* dropdown list, choose + Restriction .

       You can add more than one restriction for a single attribute.
    c. Selected the required operator to define the condition.
    d. In the *Value* field specify the required value.

11. Define attribute restrictions using hierarchy node column.

    You can define hierarchy analytic privileges only if you have enabled SQL access to shared hierarchies.

    For example, if you have enabled SQL access to shared hierarchies and if `SalesRepHierarchyNode` is the node column that modeler generates for a parent-child hierarchy, then `"SalesRepHierarchyNode" = "MAJESTIX"` is a possible filter expression in analytic privileges.

    a. In the *Hierarchy Privilege* section, choose + .
    b. In the *Hierarchy* dropdown list, select a hierarchy.
    c. In the *Value* field, provide a node column value.

    > **Note**
    >
    > You can create hierarchical analytic privileges only for the following conditions:
    > - All models in the *Secured Models* are star join calculation views with shared dimensions.
    > - You have enabled SQL access to the shared hierarchies in star join calculation views.

12. Build an HDB module.

    The build process uses the design-time database artifacts to generate the corresponding actual objects in the database catalog.

    a. From the module context menu, choose *Build*.

13. Assign privileges to a user.

Related Information

Static SQL Analytic Privileges [page 99]

4.2 Create Dynamic SQL Analytic Privileges

Dynamic SQL analytic privileges determine the filter condition string to restrict data access at run time. You use the analytic privilege editor to define the dynamic SQL analytic privilege.

Prerequisites

1. If you want to use a SQL analytic privilege to apply data access restrictions on calculation views, set the *Apply Privileges* property for the calculation view to *SQL 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 *SQL Analytic Privileges*.
2. If you want to use a SQL analytic privilege to apply data access restrictions on CDS views, you have defined the referenced CDS views using the `WITH STRUCTURED PRIVILEGE CHECK` clause.

Procedure

1. Start the SAP Web IDE for SAP HANA in a browser.
2. If you want to create a new project for the calculation view, do the following:
   a. In the SAP Web IDE for SAP HANA, choose *File* > *New* > *Project from Template*.
   b. Choose the project template type.

      Currently, there is only one type of project template available, namely: *Multi-Target Application Project*. Select *Multi-Target Application Project* and choose *Next*.
   c. Type a name for the new MTA project (for example, **myApp** and choose *Next* to confirm.
   d. Specify details of the new MTA project and choose *Next* to confirm.
   e. Create the new MTA project; choose *Finish*.
3. Select the HDB module in which you want to create the analytic privilege.
4. Browse to the *src* folder.
5. In the context menu, choose *New* > *Analytic Privilege*.
6. Enter details for the new analytic privilege.
   a. In the *Name* field, enter the name of the analytic privilege.
7. Choose *Create*.

   The tool opens the analytic privilege editor where you can define the analytic privilege. By default, it opens the editor to define static SQL analytic privileges.
8. Switch analytic privilege editor view.
   a. In the analytic privilege editor, choose  .
   b. Choose the *Dynamic* menu option.

c. In the *Procedure* value help list, search for the required procedure.

   The procedure helps define the filter conditions in the dynamic SQL analytic privilege.

d. Choose *Ok*.

9. Build an HDB module.

   The build process uses the design-time database artifacts to generate the corresponding actual objects in the database catalog.

   a. From the module context menu, choose *Build*.

10. Assign privileges to a user.

Related Information

Dynamic SQL Analytic Privileges [page 99]

4.3 Create Analytic Privileges Using SQL Expressions

The analytic privilege editor provides you the flexibility to create SQL based analytic privileges using the familiar SQL environment. You can create both static and dynamic SQL analytic privileges by writing relevant SQL expressions.

Prerequisites

1. If you want to use a SQL analytic privilege to apply data access restrictions on calculation views, set the *Apply Privileges* property for the calculation view to *SQL 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 *SQL Analytic Privileges*.
2. If you want to use a SQL analytic privilege to apply data access restrictions on CDS views, you have defined the referenced CDS views using the `WITH STRUCTURED PRIVILEGE CHECK` clause.

Procedure

1. Start the SAP Web IDE for SAP HANA in a browser.
2. If you want to create a new project for the calculation view, do the following:
   a. In the SAP Web IDE for SAP HANA, choose *File* > *New* > *Project from Template*.
   b. Choose the project template type.

Currently, there is only one type of project template available, namely: *Multi-Target Application Project*. Select *Multi-Target Application Project* and choose *Next*.

c. Type a name for the new MTA project (for example, **myApp** and choose *Next* to confirm.

d. Specify details of the new MTA project and choose *Next* to confirm.

e. Create the new MTA project; choose *Finish*.

3. Select the HDB module in which you want to create the analytic privilege.
4. Browse to the *src* folder.
5. In the context menu, choose *New* > *Analytic Privilege*.
6. Enter details for the new analytic privilege.

   a. In the *Name* field, enter the name of the analytic privilege.

7. Choose *Create*.

   The tool opens the analytic privilege editor where you can define the analytic privilege. By default, it opens the editor to define static SQL analytic privileges.

8. Switch analytic privilege editor view.

   a. In the analytic privilege editor, choose .

   b. Choose the *SQL Expression* menu option.

   c. For creating static SQL analytic privilege, in the SQL editor define the attribute restrictions and its validity.

   For example:

   Sample Code

   ```
   (("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

   If you have enabled SQL access for calculation views (of type dimensions used in a star join calculation view), the tool generates a node column. You can maintain a filter expression using this node column when creating analytic privileges. For example, if `SalesRepHierarchyNode` is the node column that the tool generates for a parent-child hierarchy, then `"SalesRepHierarchyNode" = "MAJESTIX"` is a possible filter expression.

   d. For creating dynamic SQL analytic privilege, in the SQL editor specify the procedure within the `CONDITION PROVIDER` clause.

   For example:

   Sample Code

   ```
   CONDITION PROVIDER schema_name.procedure_name
   ```

9. Build an HDB module.

   The build process uses the design-time database artifacts to generate the corresponding actual objects in the database catalog.

   a. From the module context menu, choose *Build*.

10. Assign privileges to a user.

4.4 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, <=, >=, <, >.

The user creating the analytic privileges must have corresponding privileges on the database objects (tables/views) involved in the defining the restrictions.

4.5 Dynamic SQL Analytic Privileges

In dynamic analytic privileges, you use a database procedure to dynamically obtain the filter condition string at run time. 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.

> **Note**
>
> For creating dynamic SQL analytic privileges using SQL expressions, the tool supports using only simple filter conditions and does not support subqueries for defining dynamic SQL analytic privileges.

5 Additional Functionality for Calculation Views

After modeling calculation views or during design time itself you can perform certain additional functions to understand the performance of the view at runtime and to efficiently model calculation views.

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

Related Information

Trace View Objects with Data Lineage [page 100]
Trace Dependent Objects to Analyze Impacts [page 101]
Open Calculation Views in Performance Analysis Mode [page 104]
Maintain Comments for Calculation View Objects [page 108]
Replacing Nodes and Data Sources [page 110]
Using Functions in Expressions [page 112]
Manage Calculation Views with Missing Objects [page 124]
Generate Calculation View Documentation [page 125]

5.1 Trace View Objects with Data Lineage

The graphical modeling tool in SAP Web IDE for SAP HANA supports data lineage to graphically visualize the source of objects used in modeling a calculation view. With data lineage, you can essentially identify from where the calculation view gets its data from.

Context

Use data lineage to graphically visualize the flow of an object within a calculation view. For example, for a selected calculation view, you can graphically visualize the source for its attributes, measures, underlying data sources (tables and views) and so on. Data lineage is useful, especially to analyze impacts, to trace errors and to debug errors.

The tool supports two types of data lineage. In the first scenario, you can view the flow of an object from its source and up to the semantics node within the calculation view. In the second scenario, you can view the source of all data sources (tables and views) used for modeling a calculation view.

Procedure

1. Trace columns in a calculation view.
   a. Start the SAP Web IDE for SAP HANA tool.
   b. In the *Project Explorer* view, select the required calculation views.
   c. In the context menu, choose *Open With* > *Calculation View Editor*.
   d. Select the *Semantics* node.
   e. Choose the *Columns* tab.
   f. Select a column you want to trace.
   g. Choose  .

   In the same view editor, the tool highlights the selected column, its source object and the flow of the selected column from its source to the *Semantics* node.

   > **Note**
   >
   > For calculated columns, the tool displays the source of all columns used in the valid calculated column expression, and for restricted columns, the tool displays the source of the base measure used in defining the restricted column.

2. Trace source of all data sources within the calculation view.
   a. Start the SAP Web IDE tool for SAP HANA.
   b. In the *Project Explorer* view, select the required calculation views.
   c. In the context menu, choose *Data Lineage*.

      In a new tab, the tool opens a new editor. You use this editor to trace the source of all data sources used for modeling the selected calculation view. Use the collapse and expand buttons for each such data source to view its next level details.

5.2 Trace Dependent Objects to Analyze Impacts

Modifying a calculation view can impact other calculation views that are modeled on top of the view. It is necessary to identify all such dependent objects before making any changes to the view, which otherwise may lead to run time errors.

Context

You perform impact analysis at the view level. The tools helps to identify all dependent objects of a target calculation view, one level at a time. This means that, for each of the dependent object, you can further drilldown and identify the next level of dependent objects and until the leaf object.

Procedure

1. Start the SAP Web IDE for SAP HANA tool.
2. In the *Project Explorer* view pane, select the required calculation view.
3. In the context menu, choose *Impact Analysis*.

   In a new tab, the tool opens a editor for impact analysis. You use this editor to identify all dependent objects of the target calculation view. Use the collapse or expand button for each data source to view its next level of dependent object.
4. In the new editor, expand the *Properties* section (at the bottom of the editor) to view additional details such as the object type, object name, and so on for a selected object.

5.3 Assign Value Help for Attributes

If you are using attribute data to provide values to variables and input parameters at run time, 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 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 required calculation view or table that you want to use for providing values.
7. Select an attribute.

   The tool 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 run time, the tool provides a value help that has values from the selected attribute. You can use these values for input parameters and variables.

5.4 Performance Analysis

The SAP Web IDE for SAP HANA tool supports opening a calculation view in performance analysis mode. The objective of the performance analysis mode is to provide such information to users that helps them understand the performance of the calculation view when it is executed.

When you open a calculation view in performance analysis mode, you obtain information on the catalog tables modeled in the view. For example, information on table partitions, number of rows in the tables, and so on.

The information that the tool displays in performance analysis mode depends on the view node that you select and the data sources within this view node. In addition, opening a calculation view in performance analysis mode also helps to:

Identify number of rows in a table

Identify those data sources that have number of rows above a certain threshold value. If you want to configure the threshold value,

1. In the menu bar, choose  .
2. Choose *Modeler*.
3. In the *Threshold Value* textbox field, provide the required threshold value.
4. Choose *Save*.

> **Note**
>
> The tool displays a warning icon across tables in the view editor, if the tables have rows more than the threshold value.

Identify Table Partitions and Table Types

If you have modeled a calculation view with partitioned tables, identify the partitioned tables and its partition type (Hash, Range, Round Robin).

Related Information

Open Calculation Views in Performance Analysis Mode [page 104]
Debug Calculation Views [page 106]

5.4.1 Open Calculation Views in Performance Analysis Mode

When you open a calculation view in performance analysis mode, you obtain information on joins, join tables, table partitions, table types and other such information that to better understand the performance of calculation views when it is executed.

Context

For example, the number of rows in a data source and table partitions impact the performance of your queries. The performance analysis mode provides information on such details at design time. Based on this infomration you can model more efficient calculation views and improve its performance when it is executed.

Procedure

1. Start the SAP Web IDE for SAP HANA tool.
2. Open the required calculation view in the view editor.
3. Select a view node that contains catalog tables.

   Note

   You cannot analyze the performance of the *Semantics* node.

4. In the menu bar, choose the icon to switch to the performance analysis mode.

   For the selected view node, the modeler displays the following information in the *Performance Analysis* tab.
   - Join Details (If the selected view node is a join node.)
   - Data Source Details

   Note

   When you are in performance analysis mode, you can switch to normal mode by choosing the same icon in the menu bar. If you want to always open a calculation view in performance analysis mode as a default, you have configure the preferences.

   1. In the menu bar, choose .
   2. Choose *Modeler*.
   3. Select *Always open Calculation Views in performance analysis mode*.
   4. Choose *Save*.

Related Information

Performance Analysis [page 103]
Join Details [page 105]
Data Source Details [page 105]

5.4.1.1 Join Details

Open a calculation view in performance analysis mode and select a join view node that has catalog tables as data sources.

If you have defined a join for the catalog tables, then the *JOIN DETAILS* section in *Performance Analysis* tab provides the following information:

- Catalog tables participating in the join. This includes the left table and right table.
- The cardinality and join type that you have selected for each join.
- Information on whether you have maintained the referential integrity for the join table.
- If the cardinality that modeler proposes is different from the cardinality that you select or if you have not maintained referential integrity, modeler displays a warning.

> **Note**
>
> Only users with SELECT privileges on the catalog tables participating in the join can view join validation status.

5.4.1.2 Data Source Details

Open a calculation view in performance analysis mode and select a view node that has catalog tables as data sources.

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).
- 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 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 identify whether a system is using a scale-out architecture.

5.4.2 Debug Calculation Views

For debugging calculation views, the graphical modeling tool in SAP Web IDE for SAP HANA provides a debugger editor. Open the calculation view in the debugger editor (in debug mode) by executing a debug query that the tool proposes or by executing your own debug query.

Context

The debugging operation helps analyze the run time behavior of a calculation view. Based on the analysis, you can make necessary changes to the view at design time and improve its performance when it is executed. The tool supports several debugging operations within the debugger editor. For example, write a SQL query for debugging a calculation view and identify those attributes or data sources in the calculation view that the engine consumes for executing the query, and also those objects that the engine does not consume.

Procedure

1. Open the required calculation view in the view editor.
2. In the view editor menu bar, choose .
3. Select the *Semantics* node.
4. Select the *Debug Query* tab.

   The SQL editor in the *Debug Query* tab by default proposes a query. You can use this query to debug the calculation view or use your own query in the SQL editor to debug the calculation view. At any point in time, you can reset to the default query. Choose to reset the query.

   > **Note**
   >
   > The tool proposes a query in the SQL editor after analyzing the existing version of your calculation view. If you have made changes to the existing version, then ensure to build the calculation view before debugging.

5. Start debugging.

   Once you have identified the query to debug the calculation view, begin the debugging process.

   a. In the SQL editor menu bar, choose to start the debugging process.

Results

This operation opens the calculation view in the debugger editor in read-only mode. Use the debugger editor to analyze the performance of your calculation view at run time.

> **Note**
>
> You can modify the calculation view only in the standard calculation view editor outside of the debug session.

Related Information

Using the Debugger Editor [page 107]
Predefined Validation Rules [page 108]

5.4.2.1 Using the Debugger Editor

The debugger editor opens the calculation view in debug mode and helps in analyzing the runtime performance of calculation views.

The data in the debugger editor largely depends on the query you execute to debug the calculation view. The debugger editor provides the following information when you debug a selected calculation view.

Helps identify pruned and unpruned data sources in calculation views.

The debugger editor provides information on all pruned and unpruned data sources and columns. Pruned data sources refer to those underlying calculation views that the engine does not require for executing the debug query on the selected calculation view. Pruned data sources are greyed out in the debugger editor.

Similarly, pruned columns are those columns in a view nodes that the engine does not require or consume to execute the debug query. Select the required view node and choose the *Columns* tab. The pruned columns are greyed out.

Allows drilldown on underlying data sources for detailed analysis

You can drilldown any underlying data source (calculation view) and analyze its performance. For performing the drilldown operation, in the debugger editor, select an underlying data source (calculation view) and choose . This operation opens the selected underlying data source in a new debugger editor. You can perform similar debugging operations on the underlying data source.

Provides simple intermediate data preview

Select a view node in the debugger editor and choose the *Debug Query* tab. By default, the SQL editor in the *Debug Query* tab displays the sub-query relevant to the selected view node. Choose to preview the output of the selected view node.

Displays results of executing the performance validation rules on the calculation view.

The graphical modeling tool in SAP Web IDE for SAP HANA contains predefined validation rules with the objective of validating the performance of a calculation view. When you execute a debug query and open the calculation view in the debugger editor, the tool also automatically executes these predefined validation rules. The tools displays the results of the execution at the bottom of the debugger editor. Hover over the **Warning**

icon for more details. The *Warning* icon does not appear if the execution does not encounter any predefined violations.

5.4.2.2 Predefined Validation Rules

Executing the predefined validation rules helps identify specific design time factors that impact the performance of calculation views.

The tool automatically executes the predefined validation rules when you execute the debug query. For example, calculated columns or aggregated columns in filter expressions impact the performance of calculation views. When you execute the validation rule, Calculation in filter expression, you can identify whether you have modeled the calculation view with calculated columns or aggregated columns in filter expressions. You can make changes to your calculation views at design time accordingly.

Table 49:

Predefined Validation Rule	Objective
Calculation in filter expression rule	Helps identify whether you have modeled the calculation view with calculated columns or aggregated columns in filter expressions.
Calculation in joins rule	Helps identify whether you have modeled the calculation view with calculated columns or aggregated columns in join conditions.
Partition types in join rule	Helps identify whether the tables participating the join are partition tables and if the 1st level partition type of these two tables are different.

5.5 Maintain Comments for Calculation View Objects

When you are modeling a calculation view, you can also maintain comments for the view or for its objects such as parameters, calculated columns, 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 nodes
- Input parameters and variables

- Hierarchies
- Calculated columns and restricted columns in underlying view nodes

> **Note**
>
> The tool does not support translating comments maintained for the calculation view or its objects.

Procedure

1. Start the SAP HANA Web IDE in a browser.
2. Open the required calculation view for which you want to maintain comments.
3. Maintain comments for the calculation view at the view level.
   a. Select the *Semantics* node.
   b. Choose the *View Properties* tab.
   c. In the *Notes* text field, enter a new comment or edit any 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. In the *Notes* column property, choose  .

      > **Note**
      >
      > Configure default column properties.
      >
      > If you do not see *Notes* column property, choose the  icon in the menu bar and ensure you have selected the *Notes* column property. Use the horizontal scroll bar to view all the available column properties.

   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 nodes.
   b. Choose  .
   c. Enter a new comment or edit an existing comment.
6. Maintain comments for input parameters or variables
   a. Select the *Semantics* node.
   b. Choose the *Parameters* tab.
   c. Select an input parameter or variable for which you want to maintain comments.
   d. In the *Notes* text field, enter a new comment or edit any existing comment.
7. Maintain comments for calculated columns or restricted columns.
   a. Select the default view node.
   b. Choose the *Calculated Columns* tab or *Restricted Columns* tab.

c. Select a calculated column or a restricted column for which you want to maintain comments.
d. In the *Notes* text field, enter a new comment or edit any existing comment.

8. Maintain comments for hierarchies
   a. Select the default view node.
   b. Choose the *Hierarchies* tab.
   c. Select the required hierarchy for which you want to maintain comments.
   d. In the *Notes* text field, enter a new comment or edit any existing comment.

5.6 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

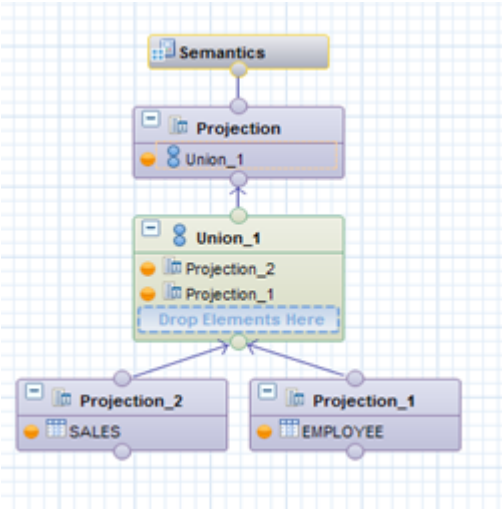
Replace a View Node in Calculation Views [page 110]
Replace a Data Source in Calculation Views [page 111]

5.6.1 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*.

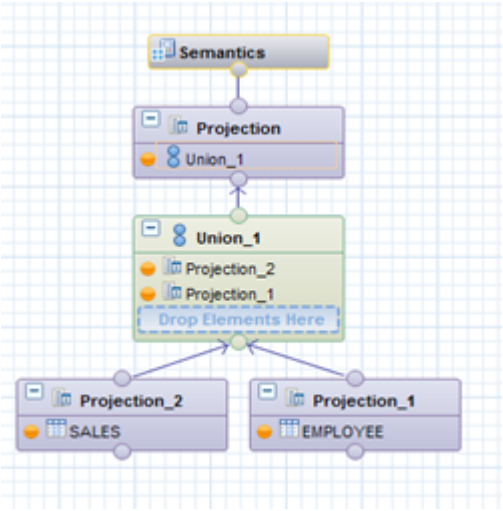
   If you do not select this checkbox, the view node appears as an orphan node in the column view.
7. Choose *Finish*.

5.6.2 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*.

5.7 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 116]
Mathematical Functions [page 117]
Date Functions [page 118]
Miscellaneous Functions [page 122]
Using Functions in Expressions [page 112]
Spatial Functions [page 120]
Spatial Predicates [page 121]

5.7.1 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.

Function	Syntax	Purpose
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 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.

Function	Syntax	Purpose
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 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

Function	Syntax	Purpose
		is interpreted as UTF-8 as it is in the chars() function above.

5.7.2 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 either 8, 12, or 16 byte length, depending on intDigits and fractDigits	fixed(3.2, 8, 2) + fixed(2.3, 8, 3)
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)	convert arg to longdate type, similar to date function above.	longdate(fixed(20000203135026.1234567, 10, 5)) -> longdate('2000-02-03 13:50:26.1234500')

Function	Syntax	Purpose	Example
	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)		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) secondtime(string, string) secondtime(int, int, int)	Converto to secondtime.	

5.7.3 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)	Abs returns arg, if arg is positive or zero, -arg else. Abs is implemented for all numeric types and time.	

Function	Syntax	Purpose	Example
	time abs(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

5.7.4 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.
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,

Function	Syntax	Purpose
		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) quarter(date, month)	Return a string 'yyyy-Qn', yyyy being the year of the quarter and n the 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')

5.7.5 Spatial Functions

The below table lists the supported spatial functions for expressions in the column engine language.

Table 50:

Function	Type	Description
ST_Area	ST_MultiPolygon	Computes the area of the multipolygon.
ST_Area	ST_Polygon	Calculates the area of a polygon.
ST_AsGeoJSON	ST_Geometry	Returns a string representing a geometry in JSON format.
ST_AsText	ST_Geometry	Returns the text representation of an ST_Geometry value.
ST_Buffer	ST_Geometry	Returns the ST_Geometry value that represents all points whose distance from any point of an ST_Geometry value is less than or equal to a specified distance in the given units.
ST_ConvexHull	ST_Geometry	Returns the convex hull of the geometry value.
ST_Difference	ST_Geometry	Returns the geometry value that represents the point set difference of two geometries.
ST_Distance	ST_Geometry	Returns the distance between two geometries in the given unit, ignoring z- and m-coordinates in the calculations.
ST_Envelope	ST_Geometry	Returns the bounding rectangle for the geometry value.
ST_GeomFromText	ST_Geometry	Constructs a geometry from a character string representation.
ST_GeometryType	ST_Geometry	Returns the name of the type of the ST_Geometry value.
ST_Intersection	ST_Geometry	Returns the geometry value that represents the point set intersection of two geometries.
ST_IsEmpty	ST_Geometry	Determines whether the geometry value represents an empty set.
ST_SRID	ST_Geometry	Retrieves or modifies the spatial reference system associated with the geometry value.
ST_SRID(INT)	ST_Geometry	Changes the spatial reference system associated with the geometry without modifying any of the values.
ST_SymDifference	ST_Geometry	Returns the geometry value that represents the point set symmetric difference of two geometries.
ST_Transform	ST_Geometry	Creates a copy of the geometry value transformed into the specified spatial reference system.
ST_Union	ST_Geometry	Returns the geometry value that represents the point set union of two geometries.

The above functions are categorized based on the use case as shown below:

Geometry Construction Functions

- ST_Geometry ST_GeomFromText(String/Clob wkt, Int srid);

Geometry Serialization

- String/Clob ST_AsText(ST_Geometry geometry);
- String/Clob ST_AsGeoJson(ST_Geometry geometry);

Geometry Transformation

- ST_Geometry ST_Transform(ST_Geometry geometry, Int srid);

Geometry Inspection

- String ST_GeometryType(ST_Geometry geometry);
- Int ST_SRID(ST_Geometry geometry);
- Int ST_IsEmpty(ST_Geometry geometry);
- ST_Geometry ST_Envelope(ST_Geometry geometry);

Calculations on a Single Geometry

- Double ST_Area(ST_Geometry geometry);
- ST_Geometry ST_ConvexHull(ST_Geometry geometry);
- ST_Geometry ST_Buffer(ST_Geometry geometry, Double buffer [, String uom]);

Calculations on Two Geometries

- Double ST_Distance(ST_Geometry geometry1,
- ST_Geometry geometry2 [, String uom]);
- ST_Geometry ST_Intersection(ST_Geometry geometry1, ST_Geometry geometry2);
- ST_Geometry ST_Union(ST_Geometry geometry1, ST_Geometry geometry2);
- ST_Geometry ST_Difference(ST_Geometry geometry1, ST_Geometry geometry2);
- ST_Geometry ST_SymDifference(ST_Geometry geometry1, ST_Geometry geometry2);

5.7.6 Spatial Predicates

The below table lists the supported spatial predicates for expressions in the column engine language.

Table 51:

Predicate	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.

Predicate	Type	Description
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 a 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.
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.

5.7.7 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'))

Function	Syntax	Purpose	Example
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	
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	sqladd(if("VAL_B" = 0, int(null), "VAL_C") / "VAL_B", -1)

Function	Syntax	Purpose	Example
		will return NULL if any of its arguments is NULL.	

5.8 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. The modeling tool in SAP Web IDE for SAP HANA uses proxies to help 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.

You can resolve inconsistencies in calculation views by performing any of the following:

- Adjusting mappings of inconsistent objects.
- Deleting inconsistent objects.

5.9 Generate Properties File for Calculation Views

For a calculation view, you can generate a properties file that contains the key-value pairs, such as, name and description values of calculation views objects.

Context

The tool generates the properties file in the same HDB module that contains the calculation view with the extension `.properties`.

For calculation view objects such as columns, input parameters, variables, and so on, you define name and description values. After generating the properties file for the calculation view, activate the HDB module that contains this generated file to store the name and description values in the `BIMC_DESCRIPTION` table.

You can also tranlate the name and description values to multiple langagues and update the `BIMC_DESRIPTION` table. Client tools can read the *BIMC_DESCRIPTION* table and displays values in the reporting tools accordingly.

Procedure

1. Start the SAP Web IDE for SAP HANA tool.
2. In the *Project Explorer* view, select the required calculation views.
3. In the context menu, choose *Generate Properties File*.

5.10 Generate Calculation View Documentation

Generate a single document that captures all details for a selected calculation view.

Context

Some business scenarios requires modeling complex calculation views that includes layers of calculation logic. In such cases, generating a calculation view documentation helps obtain a snapshot of all key details in a single document. The tool generates the calculation view documentation in `.html` format.

For example, the document provides information on columns in the view, the input parameters, variables, hierarchies, and so on.

Procedure

1. Launch the SAP Web IDE for SAP HANA tool in a browser.
2. In the *Project Explorer* view, select the calculation view for which you want to generate the documentation.
3. In the context menu, choose *Generate Documentation*.

Results

In a new browser tab, modeler generates the calculation view documentation. The documentation includes information on all columns, variables, hierarchies, parameters, impact analysis, lineage and other general calculation view details.

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