

HA300

SAP HANA 2.0 SPS02 - Modeling

PARTICIPANT HANDBOOK INSTRUCTOR-LED TRAINING

Course Version: 14
Course Duration: 5 Day(s)
Material Number: 50145521

SAP Copyrights and Trademarks

© 2018 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. Please see <http://global12.sap.com/corporate-en/legal/copyright/index.epx> for additional trademark information and notices.

Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors.

National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP SE or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP SE or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

In particular, SAP SE or its affiliated companies have no obligation to pursue any course of business outlined in this document or any related presentation, or to develop or release any functionality mentioned therein. This document, or any related presentation, and SAP SE's or its affiliated companies' strategy and possible future developments, products, and/or platform directions and functionality are all subject to change and may be changed by SAP SE or its affiliated companies at any time for any reason without notice. The information in this document is not a commitment, promise, or legal obligation to deliver any material, code, or functionality. All forward-looking statements are subject to various risks and uncertainties that could cause actual results to differ materially from expectations. Readers are cautioned not to place undue reliance on these forward-looking statements, which speak only as of their dates, and they should not be relied upon in making purchasing decisions.

Typographic Conventions

American English is the standard used in this handbook.

The following typographic conventions are also used.

This information is displayed in the instructor's presentation



Demonstration



Procedure



Warning or Caution



Hint



Related or Additional Information



Facilitated Discussion



User interface control

Example text

Window title

Example text

Contents

vii Course Overview

1 Unit 1: Information Views

2	Lesson: Introducing Information Views
18	Lesson: Connecting Tables
32	Lesson: Creating Dimension Calculation Views
41	Lesson: Using Measures in Calculation Views

69 Unit 2: Modeling Functions

71	Lesson: Creating Restricted and Calculated Columns
82	Lesson: Filtering Data
89	Lesson: Using Variables and Input Parameters
104	Lesson: Using Hierarchies
114	Lesson: Implementing Currency Conversion

126 Unit 3: SAP HANA Studio Modeling

127	Lesson: Creating Information Models in SAP HANA Studio
-----	--------------------------------------------------------

133 Unit 4: SAP Supplied Virtual Data Models

134	Lesson: SAP HANA Live
152	Lesson: Embedded Analytics and CDS

163 Unit 5: Using SQL in Models

164	Lesson: Introducing SAP HANA SQL
184	Lesson: Defining the Persistence Layer using CDS
187	Lesson: Working with SQLScript
195	Lesson: Query a Modeled Hierarchy Using SQLScript
198	Lesson: Creating and Using Functions
204	Lesson: Creating and Using Procedures

215 Unit 6: Optimization of Models

216	Lesson: Implementing Good Modeling Practices
233	Lesson: Using Tools to Maximize Optimization

242 Unit 7: Management and Administration of Models

244	Lesson: Working with Modeling Content in a Project
262	Lesson: Creating and Managing Projects
273	Lesson: Working with GIT within the SAP Web IDE
294	Lesson: Using SAP Enterprise Architecture Designer
299	Lesson: Migrating Modeling Content

309	Unit 8:	Security in SAP HANA Modeling
310		Lesson: Understanding Roles and Privileges
322		Lesson: Defining Analytic Privileges
335		Lesson: Defining Roles
346	Unit 9:	Advanced Data Processing
347		Lesson: Introducing Advanced Data Modeling
351		Lesson: Optional: Implementing Text Processing
363		Lesson: Optional : Working with Geospatial Data
371		Lesson: Optional: Developing Predictive Models
376		Lesson: Optional: Working with SAP HANA Graph

Course Overview

TARGET AUDIENCE

This course is intended for the following audiences:

- Application Consultant
- Data Consultant/Manager
- Database Administrator

UNIT 1

Information Views

Lesson 1

Introducing Information Views

2

Lesson 2

Connecting Tables

18

Lesson 3

Creating Dimension Calculation Views

32

Lesson 4

Using Measures in Calculation Views

41

UNIT OBJECTIVES

- Explain information views
- Describe the main types of information views
- Connect tables
- Determine the type of join to use when connecting tables
- Create dimension calculation views
- Define calculated attributes
- Create time-based dimension views
- Use base table aliases
- Define label columns and hide attributes in a dimension calculation view
- Use measures in calculation views
- Explain the benefits of each type of node in calculation views
- Create and combine nodes in calculation views
- Use the features of calculation views to enhance the flexibility of this type of view

Unit 1

Lesson 1

Introducing Information Views

LESSON OVERVIEW

This lesson introduces the general concept of information views, and then describes the different types of views and their benefits.

Business Example

As part of an SAP HANA implementation, you want to use the modeling capabilities of SAP HANA to build flexible information models and easily report on your data.

Before getting into more detail, you want to understand what information views are. This is also an opportunity to review the basic concepts used in reporting; such as dimensions, measures, attributes, hierarchies, and so on.



LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Explain information views
- Describe the main types of information views

Key Vocabulary of Modeling

Before introducing modeling in SAP HANA, you must become familiar with some key concepts that are frequently used when reporting on financial or operational data, for example, the following:

- Measure
- Attribute
- Dimension
- Star schema
- Hierarchy
- Semantics

Measure and Attribute

When you report on data, you have to distinguish between the following important concepts:

- Measure
- Attribute



Table 1: Measure Versus Attribute

	Measure	Attribute
Definition	A numeric value, such as a price, quantity, volume, on which you can process arithmetic or statistics operations, such as sum, average, top N values, and calculations.	An element that is used to describe a measure.
Examples	<ul style="list-style-type: none"> • Number of products sold • Unit Price • Total Price 	<ul style="list-style-type: none"> • Product ID • Product Name • Customer ID • Customer Name • Sales Organization • Sales Org. Country • Sales Org. Region • Currency

Attributes are used to filter or aggregate the measures, in order to answer questions such as the following:

- What are the total sales originating from Sales Org. located in the EMEA region?
- What is the sales revenue generated by the product Cars ?



Note:

One key objective of modeling in SAP HANA is to create a relevant association between attributes and measures to fulfill a particular reporting requirement.

Dimension

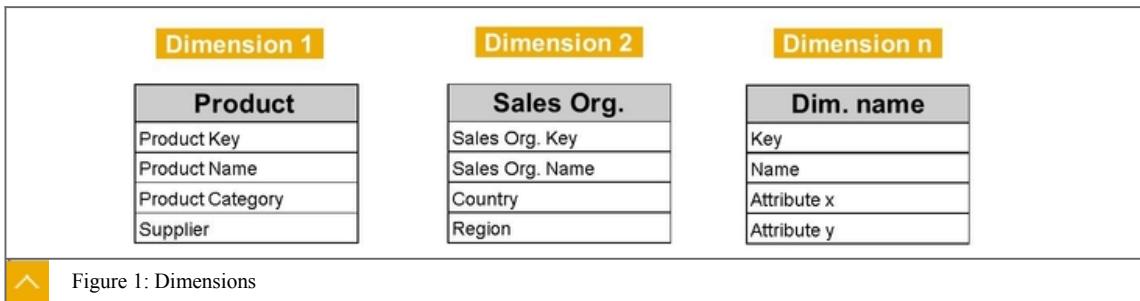
In a number of cases, analyzing the measures is easier if you group attributes together by dimension.

In the examples, the sales organization would be treated as a dimension, with the following associated attributes:

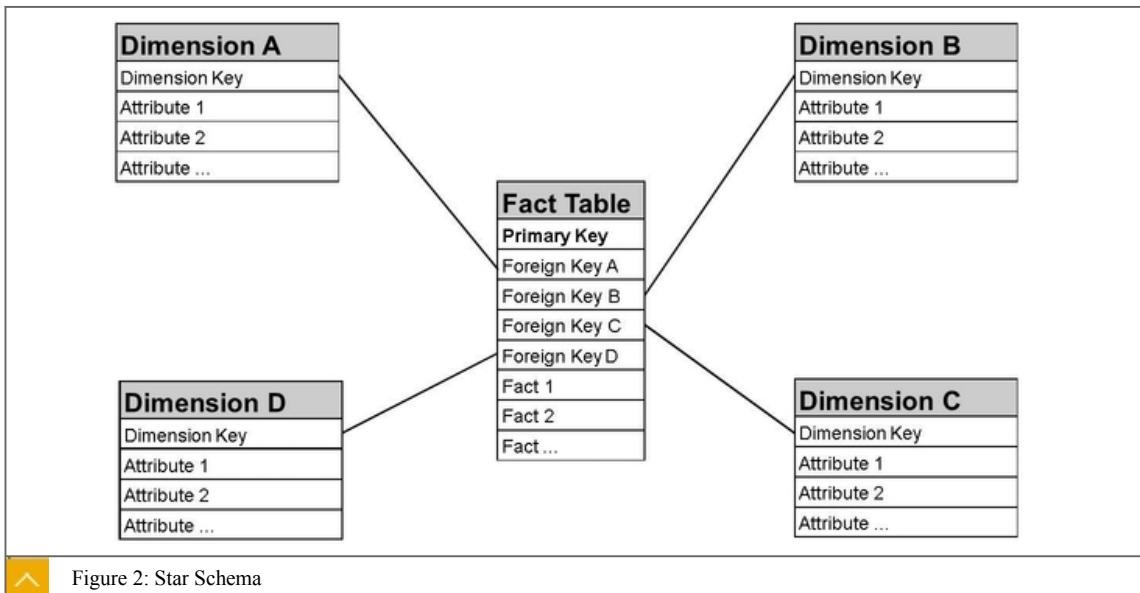
- Country
- Region

Similarly, a Product ID dimension could be associated with several attributes, such as product name, product category, or supplier.

Unit 1: Information Views



Star Schema



A Star schema consists of one fact table that references one or several dimension tables.

The fact table contains facts, or measures, as well as the keys used to identify – for each dimension – which dimension member corresponds to each fact.

Each dimension contains attributes, such as the name of the dimension member, description, and other attributes that can be used to filter the dimension members, build a hierarchy, and perform specific calculations.



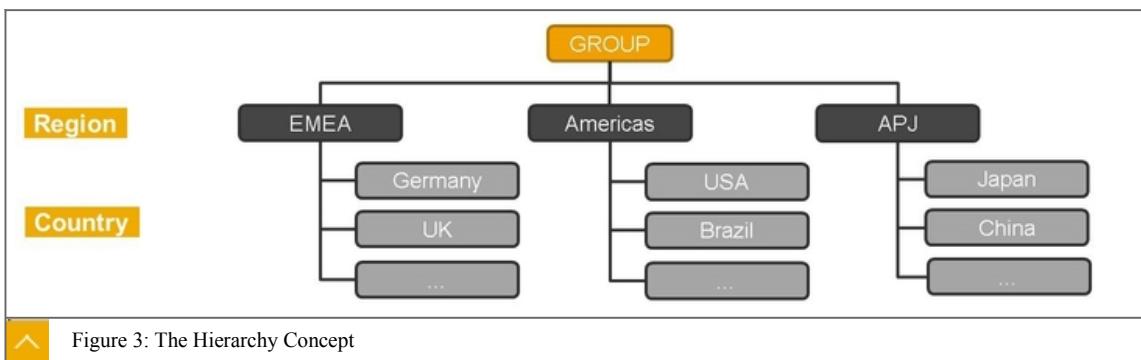
Note:

The term **fact table** is used in a generic way. Later in the course, you will learn that calculation views in SAP HANA allow you to create this fact table.

Hierarchy

A hierarchy is a structured representation of an organization, a list of products, the time dimension, and so on, by levels.

It is used to navigate the entire set of members with more ease (the location of the company, the products, or the days, weeks, months, or years) when analyzing the data.



In the figure, The Hierarchy Concept, you see a geographical representation of the structure of a company by regions and countries. The level of detail (granularity) of the hierarchy will depend on how the company wants to analyze its data, and its design too. For instance, the geographical hierarchy can be based on where the subsidiaries of a company are located.

In the figure, the top node of the hierarchy, GROUP, represents the entire company.



Note:

Similarly, the list of products that a company sells could be organized into a hierarchy, by classifying the products by product area or product type.

Semantics

The term **semantics** is sometimes used to describe what a piece of data means, or relates to. A piece of numeric data that you report can be of different types. Here are a few examples:



- A monetary value

For example, the total amount of sales orders.

In this case, you might need a dimension to specific the currency (for example USD, EUR, or GBP), if it is not implicit.

- A number of items

For example, a number of sales orders, or a number of calls to support services.

- A weight, volume, distance, or a compound of these measures

For example, the payload-distance in freight transportation.

You often need to specify the unit in which the data is expressed.

- A percentage

For example, a discount rate, or a tax rate.

Unit 1: Information Views**Note:**

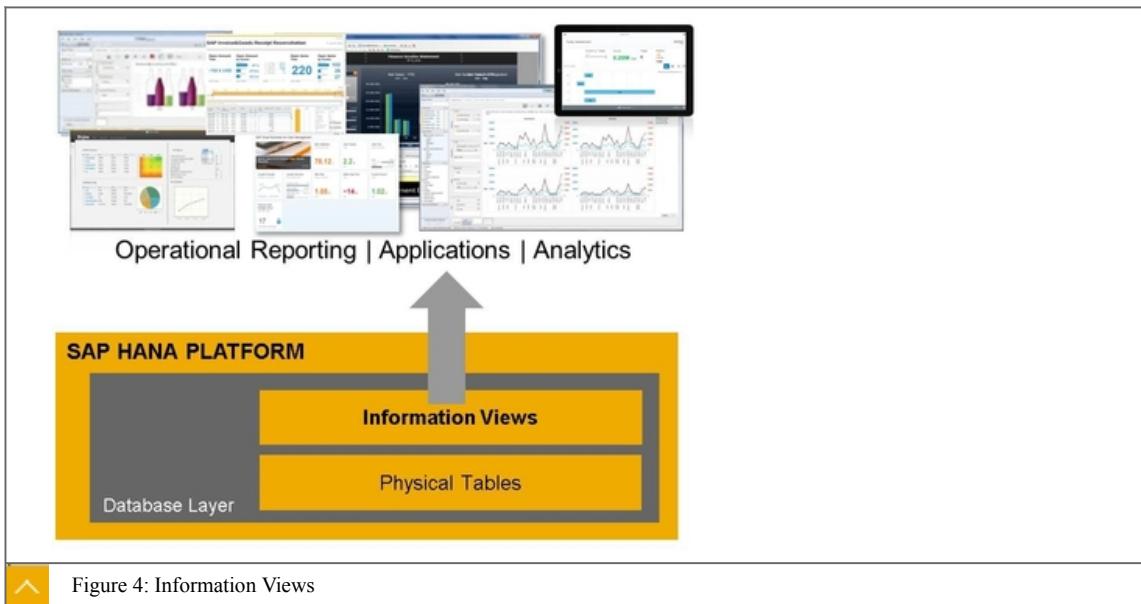
Even if the semantics are not always expressed explicitly in the data repository itself, it is very important to know and understand the semantics of any measure to avoid misinterpretation of the data or irrelevant calculation.

For example, before summing up the amount of sales orders, you must make sure that they are all expressed in the same currency.

As another example, you have to be careful with aggregations when using columns that contain ratios.

Information Views in SAP HANA

Information views are used in SAP HANA to create a Virtual Data Model, based on the data that resides in the SAP HANA database schemas.



The purpose of information views is to organize the data from the individual transactional tables, and to perform a variety of data calculations, in order to get a relevant and meaningful set of measures and dimensions or attributes to answer a specific reporting need. You can make the data more meaningful than it is in the source tables by customizing the column names, assigning label columns to key columns, and calculating additional attributes.

Benefits of Information Views

The main benefits of using information views in SAP HANA are as follows:



- All calculations are performed on the fly within the database engines

No aggregates need to be pre-calculated, and the front-end reporting tools delegate most of the data processing workload (filtering, aggregation, calculations) to the SAP HANA in-memory engines in order to achieve very high performance.

- Reusability

Each information view can be used (referenced) by other information views.

- Flexibility

Information views provide a number of features that make them very flexible. For example, defining hierarchies, filtering data, generating prompts for variables and input parameters, and performing currency conversion.

- **Adaptability**

An SAP HANA information view can adapt its behavior to the list of columns that are selected or projected on top of it. For example, the granularity of a Rank node can be determined dynamically depending on whether you query results by country or by country and customer.

- **Easy to transport**

SAP HANA provides powerful tools to transport information models between different SAP HANA databases; for instance, to install SAP-delivered information models, or transport your own information models between your development, quality assurance, and production landscapes.



Note:

These benefits are illustrated in the following lessons of this unit, and in the following units.

Design-Time Versus Runtime Information Views

In SAP HANA, information views, like a large variety of development objects, can exist both as a design-time and runtime object.

- **Design-Time View**

The design-time view is the object that you create and modify with a graphical or text-based editor.

- **Runtime View**

The runtime view is a database catalog object, more specifically a column view, that is used when you preview the data within SAP HANA, execute a SQL query, or execute a query on top of the information view with an external tool, such as BI tools, Microsoft Excel, and so on.



Caution:

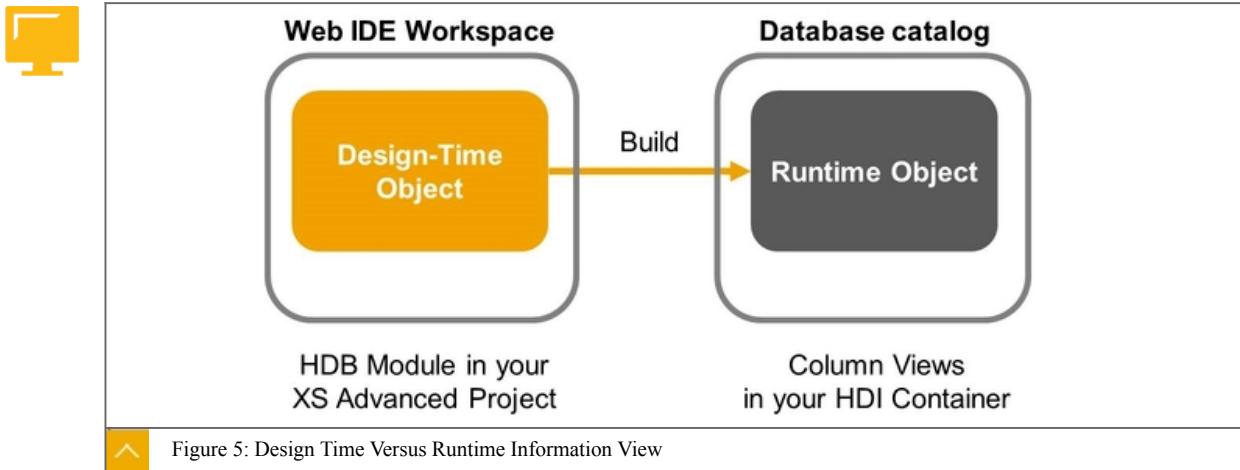
A view cannot expose its data before a runtime version of this view is successfully created in the SAP HANA database catalog. Similarly, any change to a design-time view is only visible in a query after the runtime view has been successfully updated.

Building Information Views

In many cases, building a single design-time object creates several related runtime objects in the database schema. For example, in addition to the main column view itself, which contains the measures, you can find a column view with the list of all measures, and column views materializing the hierarchies.

Apart from column views, building an information view also creates the necessary metadata that makes this view consumable by external tools.

Unit 1: Information Views



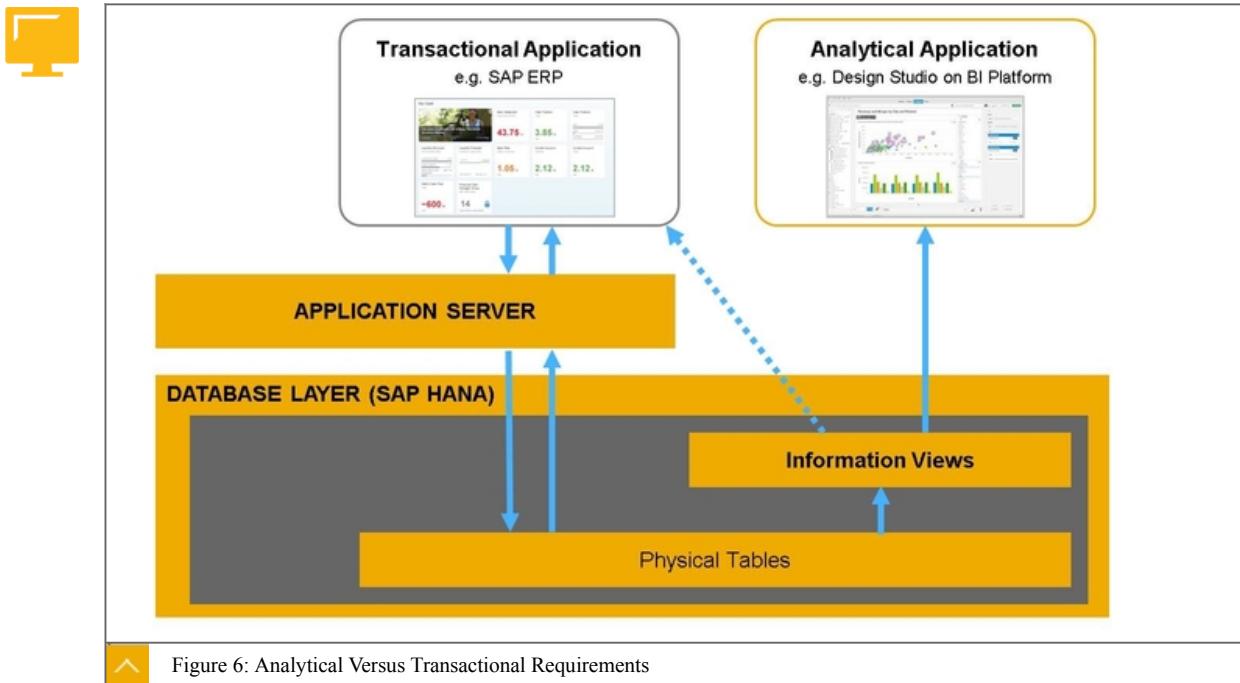
When you create or modify an information view with the SAP Web IDE for SAP HANA, the design-time object is located in a project within your workspace. This workspace is linked to your SAP Web IDE user and is not visible to other users in the same SAP HANA system.

When you build a file with the SAP Web IDE, SAP HANA creates the corresponding runtime objects in a so-called container, which is an abstraction layer for a database schema. This container is specific to the HDB module of your project.

**Note:**

You will learn more about the information model's lifecycle and SAP HANA Deployment Infrastructure (HDI) later on, in a dedicated unit, Management and Administration of Models .

Analytical Versus Transactional Requirements



In transactional applications, such as SAP ERP, the underlying data (stored in physical tables) is generally handled by the application server. This layer is necessary to handle the business

process, however complex it can be, while still ensuring data consistency at any time when updating multiple tables and checking the authorizations of the user.

On the other hand, information views are used only to retrieve data, without making any changes to the source transactional data.

For this reason, as shown in the figure, Analytical Versus Transactional Requirements, the analytical process can bypass the application server, which makes it even faster. The reporting tools directly query the SAP HANA information views, where data is calculated on-the-fly.

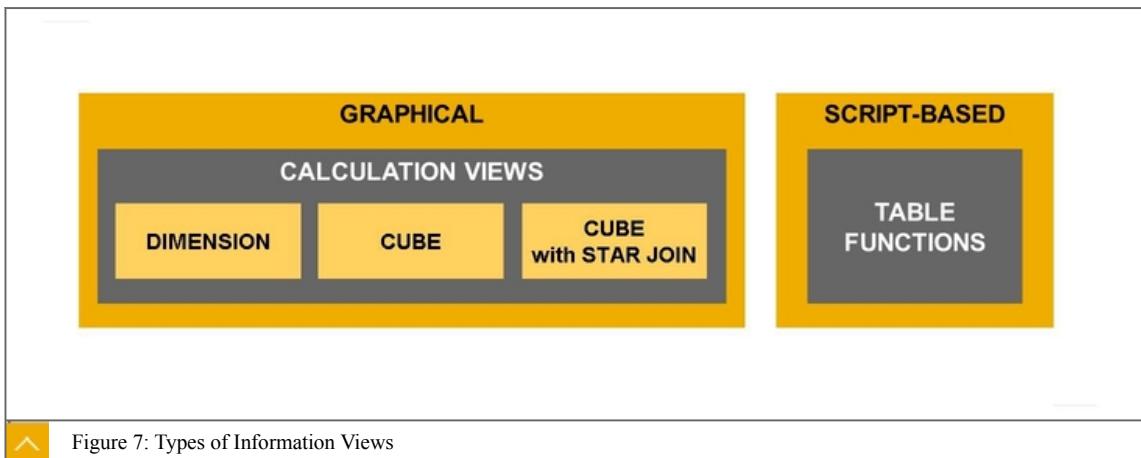


Note:

In some cases, the transactional application can also leverage the real-time capabilities of SAP HANA information views by displaying inside the transactional application (for example in the SAP Fiori launchpad) real-time analytical data that is relevant for the end user.

Information View Types

There are several types of information views.



The figure, Types of Information Views, presents the key modeling artefacts that you will be studying during this course. In addition to graphical calculation views, table functions, which are script-based objects, can be used to address modeling requirements that cannot be fulfilled with graphical information views.

Graphical Calculation Views

Before creating Calculation Views, you must understand the purpose of each type of graphical information view.

SAP HANA supports the following main types of graphical calculation views:

Unit 1: Information Views



Calculation View Type	Properties	Default Upper Node
DEFAULT	No multidimensional support. Never exposed to any client tool.	Projection
DIMENSION *	No multidimensional support.	Projection
CUBE	Designed for data analysis with multidimensional reporting	Aggregation
CUBE With Star Join	Similar to a CUBE Calculation view, but the upper node is a Star Join where you join all the attributes (calculation views of type DIMENSION *)	Star Join

* Only **DIMENSION** Calculation Views can be joined to the fact table in the **Star Join** node of a **CUBE With Star Join** Calculation View.



Figure 8: Graphical Calculation View Types

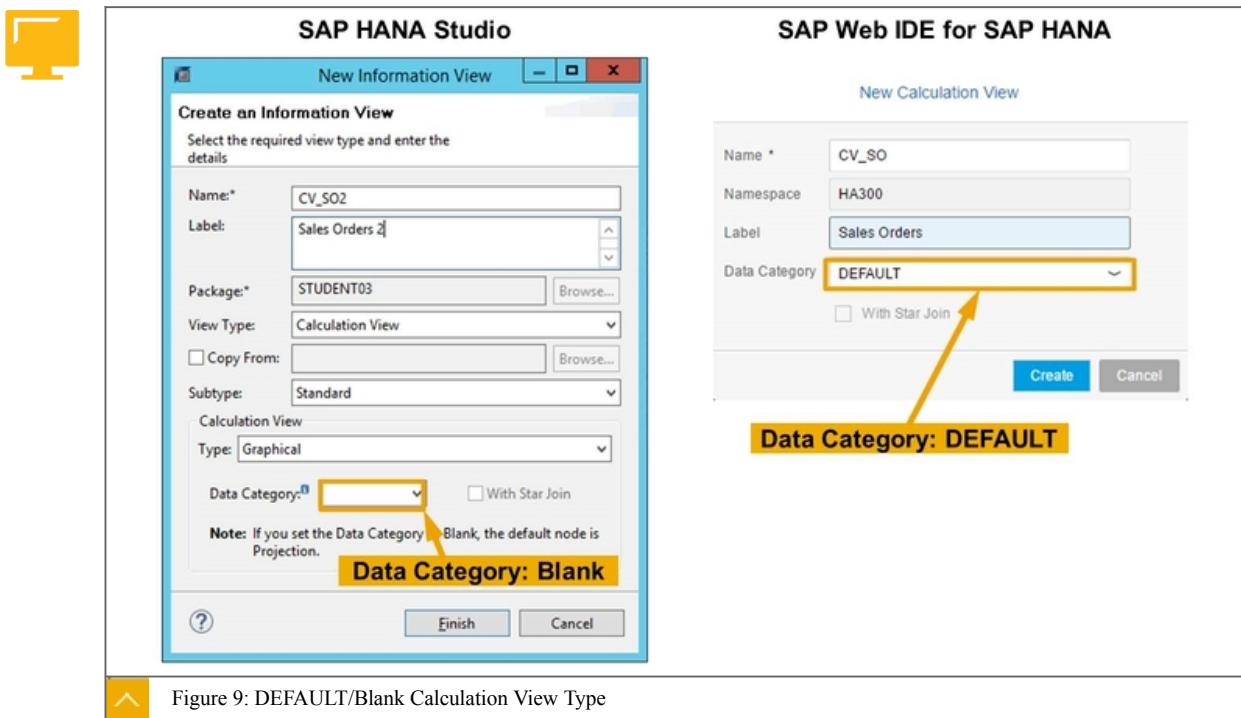
DEFAULT Calculation View Type

A graphical calculation view defined with the **DEFAULT** data category is never exposed to client tools. It can contain both attributes and measures. The main use case for this type of view is when you create views that will be reused in other views but do not need to be accessed by the end user.



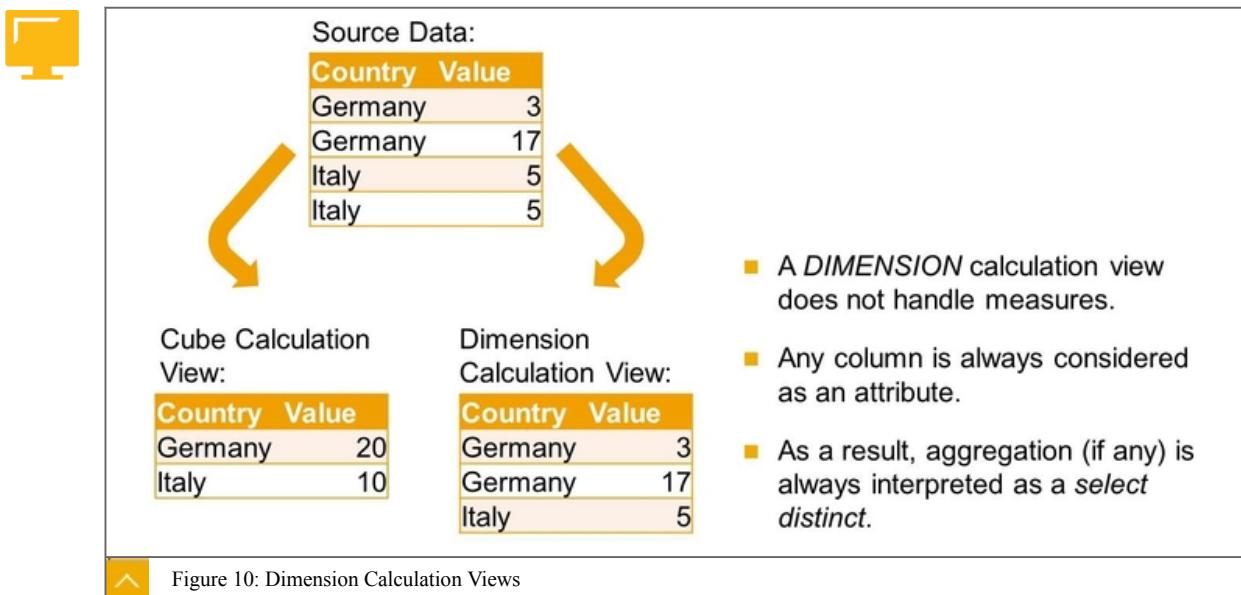
Note:

When creating a Calculation View in SAP HANA Studio, this calculation view type is not labelled **DEFAULT** but appears as “blank” in the corresponding dropdown list.



For example, in the SAP HANA Live Virtual Data Model, which is described later in this course, a number of views use this DEFAULT category type (in SAP HANA Studio, the data category is “blank”) and, as a consequence, are not exposed to the end user. On the contrary, the views that must be exposed to the end-users are assigned the category CUBE.

DIMENSION Calculation Views



Note:

Dimension calculation views do not allow measures; any numeric columns or values will always be treated as attributes.

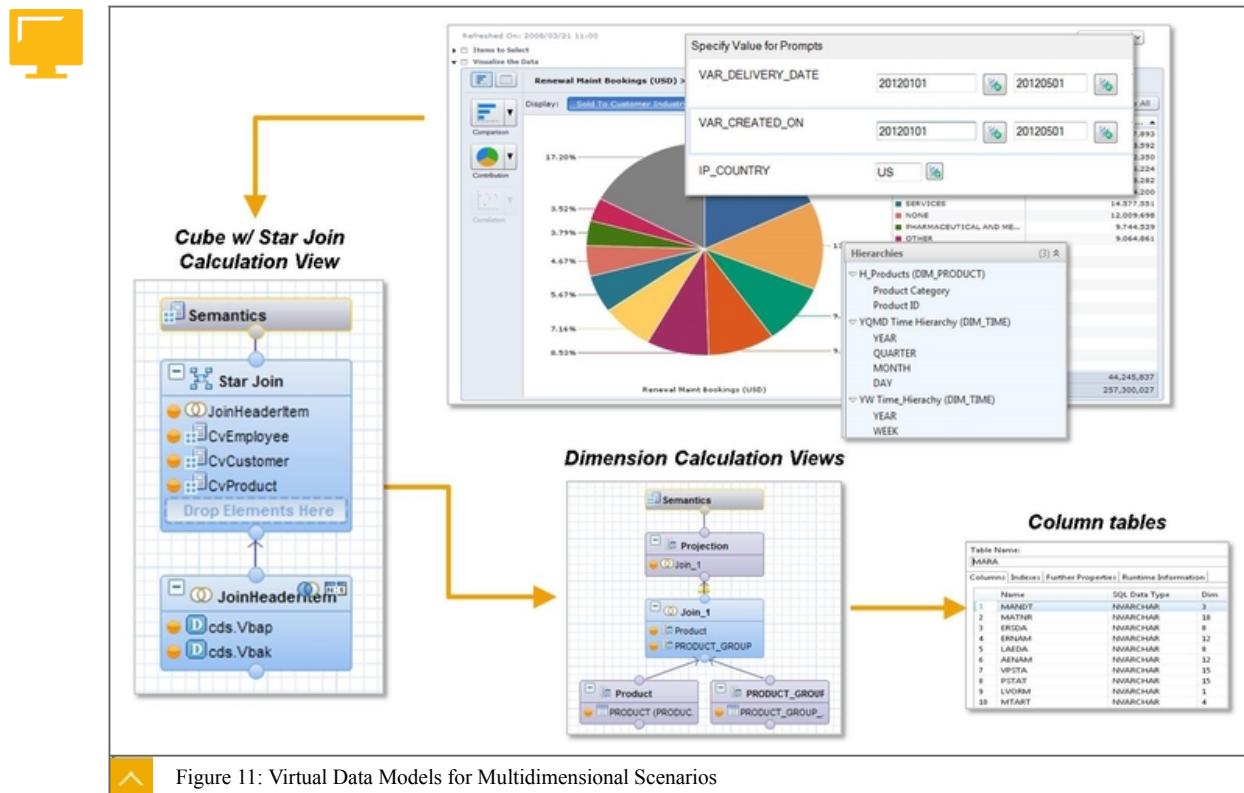
Unit 1: Information Views

CUBE Calculation Views

When you analyze data including measures, you use a calculation view of the type CUBE or CUBE with Star Join .

These types of views provide a number of features to filter data, control the aggregation of data, perform complex data calculations on measures, combine data from several data sets, retrieve sub-sets of data based on measure ranking (for example, the top countries for revenue or margin).

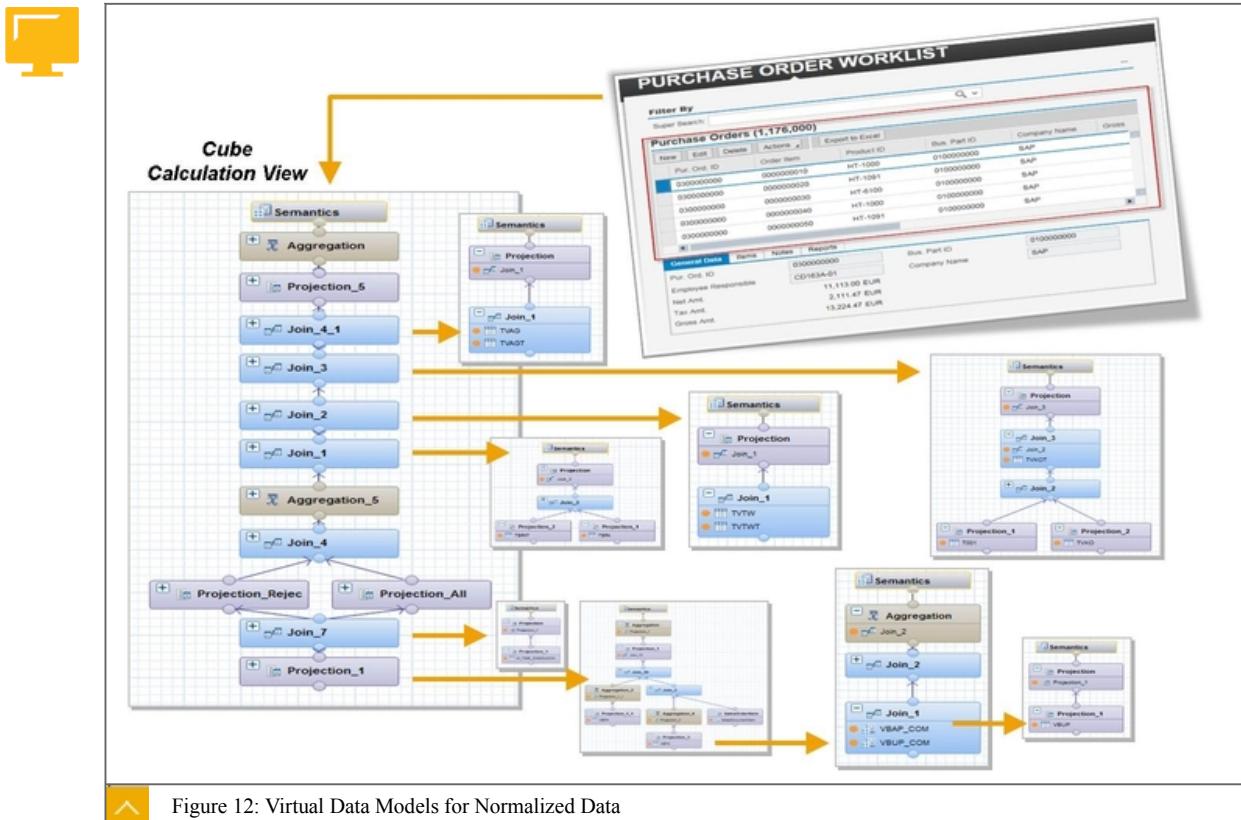
Virtual Data Models for Multidimensional Scenarios



In this scenario, a Cube with Star Join calculation view enables a multidimensional reporting that leverages the source data (from the Vbap and Bvak tables) and dimension calculation views created for the main dimensions, such as Product , Employee , or Customer .

Multidimensional tools support hierarchies for navigation, filtering and aggregation, as well as prompts (variables and input parameters) for efficient pre-filtering of data.

Virtual Data Models for Normalized Data



In this second scenario, SAP HANA calculation views typically feed data to business applications, such as SAP HANA XS build applications or enterprise analytical applications. They provide the means to model sophisticated views based on normalized data structures.

To model the purchase order worklist, a cube calculation view combines a number of different source views, with joins, projections, aggregations, and so on. Each of the different source views provides a custom set of measures or attributes.

Supported Data Source Types in Graphical Calculation Views

The following is a list of the main data source types that are supported in SAP HANA calculation views. It makes a distinction based on whether the data source is located in the same database as the calculation view that consumes it, or in a different database within the same multi-database container (MDC) system.



Table 2: Supported Data Source Types in SAP HANA Calculation Views

Data Source Type	Located in the Same Database	Located in Another Database in the Same Multi-Database Container System
Row Table	Yes (only in table functions)	No
Column Table	Yes	Yes
SQL Views	Yes	Yes (from version 1.0 SPS12)
Graphical Calculation Views	Yes	Yes

Unit 1: Information Views

Data Source Type	Located in the Same Database	Located in Another Database in the Same Multi-Database Container System
CDS Views without parameters	Yes	Yes (from version 1.0 SPS12)
CDS Views with parameters	Yes	No
Table Functions	Yes	No
Virtual Tables	Yes	Yes
Graph workspace	Yes (in the same development container)	No

Row or Column Table

To identify whether an existing table in SAP HANA is a column table or a row table, you have the following options:



- From the Systems view catalog
 - Check the table icon
 - Open the table definition (right-click the table and choose Open Definition)
- Within a node in an information view that consumes the table
 - Check the table icon
 - Check the Properties view
- From the SQL Console
 - Query the system table M_TABLES

Example:

```
SELECT "SCHEMA_NAME", "TABLE_NAME", "TABLE_TYPE" FROM "M_TABLES"
WHERE "SCHEMA_NAME" = 'TRAINING'
AND "TABLE_NAME" = 'SALES_DATA'
```

Table Functions

Table functions can be used when the graphical view types (default, dimension, and cube) do not fulfill the reporting requirements.

In particular, they are useful when you need to apply a complex logic that is not supported by graphical information views. For example, when you want to use syntax such as conditions (IF... THEN... ELSE) or loops (FOR or WHILE).



Note:

You will learn more about table functions later in the course, in a dedicated unit Scripting in Models .

Calculation Views and the SAP HANA Engines

The index server of SAP HANA provides several engines to process queries against calculation views.

Main SAP HANA Engines for Executing Calculation Views



- Join Engine
- OLAP Engine
- Calculation Engine

The distribution of the overall view execution between these engines is complex, and out of the scope of this course.

However, you can keep in mind that optimization processes exist in order to delegate the execution of a calculation view to the most relevant engine.

Overview of Calculation View Optimization

When executing a query against a calculation view, SAP HANA uses optimization processes in order to achieve the best possible performance. The optimization processes are enhanced over time, with each release of SAP HANA.

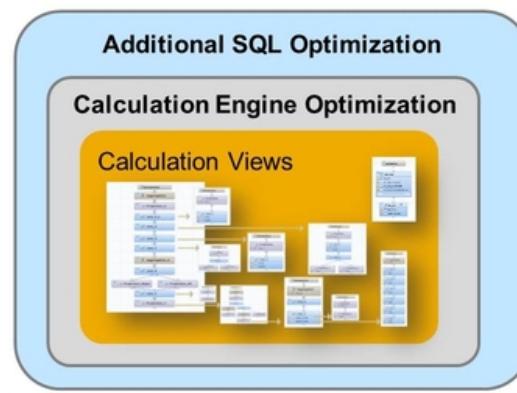


Figure 13: Calculation View Optimization

In SAP HANA SPS10 and above, when you query a calculation view, the optimization is made in two main steps:

- The initial calculation engine optimization generates a single SQL statement across a stacked model, which is then passed to the SQL optimizer.
- The SQL optimizer adds additional optimizations and delegates operations to the best database execution operator.

For example, it delegates the execution of the star join to the online analytical processing (OLAP) engine whenever it is possible.

Querying Calculation Views Data

When building an information view, the SAP Web IDE for SAP HANA allows you to preview the data of the view or to execute a custom query on top of the view. It is important to understand the difference between these approaches.

Unit 1: Information Views

The screenshot illustrates the SAP Web IDE interface for information views. It shows two main panes: 'Standard Data Preview in Web IDE' and 'Custom SQL Query'.

Standard Data Preview in Web IDE: This pane displays a table with 190 rows. The columns are: #, COUNTRY, PRODUCT_ID, BP_COMPANY_N..., GROSS_AMOUNT, and RANK. The data includes entries for African Gold And Diamar, PicoBit, and others.

Custom SQL Query: This pane shows a SQL query being typed into a text area:

```
SELECT TOP 1000
    "COUNTRY",
    "PRODUCT_ID",
    "BP_COMPANY_NAME",
```

A callout box labeled 'Display and copy the default query' points to the text area.

Execution Process:

- 1 SQL
- 2 Modify Query in SQL Console
- 3 Execute
- 4 Check Results

Custom Result: This pane shows the results of the executed SQL query, matching the data preview.

#	COUNTRY	PRODUCT_ID	GROSS_AMOUNT	RANK
AR	HT-1037	25676444.48	1	
AR	HT-1106	2441955.68	2	
AR	HT-1137	1869144.72	3	
AR	HT-1070	303284.64	4	
AR	HT-1107	30644.90	5	
AT	HT-1037	25676444.48	1	
AT	HT-1021	4013066.8	2	



Figure 14: Standard Preview or Custom Query

- Standard Data Preview

With the standard data preview, you select all the columns that are included in the semantics of the information view (provided that they are not hidden).

You can move the columns (using drag and drop), apply a temporary filter on one or several attribute columns, and order the result set by one (and only one) column.

- Custom SQL Query

An alternative to the standard data preview is to execute a custom SQL query. As shown in the figure, Standard Preview or Custom Query, after copying the SQL statement corresponding to the data preview into a SQL console, you can modify it and, for example, change the selected columns or the GROUP BY clause, and order the result set by one or more columns.

It is particularly useful to perform a thorough test of calculation views with a complex scenario (stacked calculation views, counter measures, join between several aggregation nodes with different GROUP BY columns).



Note:

Calculation views behave differently depending on which columns are selected, or whether you explicitly define a group by or not. You must ensure that a view does not give the wrong results if it is not correctly queried upon, or document the view so that it is correctly consumed by end users with reporting applications.

Standard Data Preview Features

Even though the Data Preview Editor is not a reporting tool, it still offers analysis functionality that can be useful during modeling or troubleshooting. It is comprised of the following tabs, each offering specific capabilities:

Table 3: Data Preview Tabs

Tab	Displays	Use Case
Raw Data	All data	Basic display of contents
Analysis	Selected attributes and measures in tables or graphs	Profiling and analysis

A setting in the SAP Web IDE for SAP HANA defines whether a filter must be set before fetching view or table data. To check or change this setting, choose Tools → Preferences → Data Preview .

If the option is selected, the default data preview is not executed immediately when you choose Data Preview .

Deferred Default Query Execution

Deferring the default query execution can be useful in the following situations:

- When the user wants to apply additional criteria to the data preview.
For example, to set an additional filter on one or several columns (measures or attributes).
- When the user wants to execute a custom query derived from the standard data preview query and does not need to execute the standard data preview.



Note:

Even with the option **Require that at least one filter be set before fetching view or table data** selected, it is still possible to execute the data preview without any filter. A dialog box is displayed to confirm that you want to fetch the data anyway.

Suppose you are testing a very complex view, and – just temporarily – you want to reduce the result set to one specific order ID. Executing the default query in this case is not useful, and might cause you to lose time. Instead you can perform the following steps:

- In the SAP Web IDE preferences, select the **Require that at least one filter be set before fetching view or table data** option.
- Define a filter on the ORDERID column.
- Click Refresh Data .



LESSON SUMMARY

You should now be able to:

- Explain information views
- Describe the main types of information views

Unit 1

Lesson 2

Connecting Tables

LESSON OVERVIEW

This lesson describes the different join types that are available in SAP HANA and explains how to connect tables.

In addition to the classical Inner, Left or Right Outer Joins, and Full Outer Join, SAP HANA offers other types of joins that can provide better execution speed or address some specific business scenarios. These SAP HANA specific joins include the following:

- Referential join
- Text join
- Temporal join
- Star join
- Spatial join

Business Example

SAP HANA system hosts an SAP ERP system, in which the data model is highly normalized, which means that to avoid as much as possible data redundancy, the data is distributed in a huge number of tables. For example, the table containing sales orders does not contain customer addresses or cities. The information about customer is located in a set of different tables. The same applies to the product details, and so on.

To retrieve your sales order data based on the name and location of the customer, or the product category, you need to join several tables.

You want to know more about the different types of join and how they behave in SAP HANA Information Models.



LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Connect tables
- Determine the type of join to use when connecting tables

Connecting Tables

One important activity when you create information models is to express the relationships between the different data sources used by a model. Most often, this is done by using joins, a classical artefact in any relational database management system (RDBMS).

To illustrate the behavior of the different types of joins in SAP HANA, consider the following tables:

- Sales Order

- Customer
- State

The objective is to join these tables in order to retrieve the sales order amounts (facts) with the customer information, including the states in which the customers reside.



Sales Order				Customer			
	ORDER_ID	C_ID	AMOUNT		C_ID	CNAME	STATE
1	1	1	100	1	1	WERNER	MI
2	2	1	100	2	2	MARK	MI
3	3	2	100	3	3	TOM	TX
4	4	4	100	4	4	BOB	TX
5	8	77	100				

We want to connect the **Sales Order table** to the **Customer table** linked to the **State table**.

State		
	STATE	SNAME
1	MI	MICHIGAN
2	AL	ALABAMA

Figure 15: Sample Data for Business Example

To begin with, you can make the following observations:

- Sales Order 8 does not have a customer master record.
- Customer TOM does not have any orders.
- State TX does not have a description.
- No customer resides in Alabama.

Unit 1: Information Views

Join Type Summary



Join type	Use when you want to report on	Be aware that
INNER	Facts with matching dimensions only	* Facts without a dimension will be excluded * Dimensions without a fact will be excluded * Join is always executed
LEFT OUTER	All posted facts whether there is a matching dimension or not	* Dimensions without a fact will be excluded * Best performance since join is ommissible
RIGHT OUTER	All dimension whether there are matching facts or not	* Facts without a dimension will be excluded * Right outer join is rarely used
FULL OUTER	All posted facts and all dimensions	* Combines the effects of LEFT and RIGHT OUTER join
REFERENTIAL	Facts with matching dimensions only where referential integrity is ensured	* The join is only executed if columns from the right table are queried
TEXT	A multi-language table	* Requires a language column (SPRAS or equivalent) * Acts as a LEFT OUTER JOIN
TEMPORAL	A key date within a validity period	* Needs a temporal condition, from and to date.
SPATIAL	Geospatial data	* Executes specific computations on spatial data

 Figure 16: Join Type Summary

There are different types of joins. The inner and outer joins are similar to other database management systems. Others are specific to SAP HANA.

You can review the typical use cases for the different join types.



Note:

This table builds on a scenario similar to the one just described (Business Example), assuming that a fact table (considered as the left table) is joined to a dimension table (considered as the right table).

(Sample Data for
left table) is

Each join type will now be presented in detail.

Inner Join



Select * FROM "Customer" AS c -- table alias with explicit AS Inner Join "State" s -- table alias (simple form) On c.STATE = s.STATE					STATE	SNAME
C_ID	CNAME	STATE	AGE	STATE	C_ID	CNAME
1	WERNER	MI	10	MI	1	WERNER
2	MARK	MI	11	MI	2	MARK
3	TOM	TX	12	TX	3	TOM
4	BOB	TX	13	TX	4	BOB
C_ID	CNAME	AGE	STATE	C_ID	CNAME	STATE

 Customer (3 & 4) is not returned due to no corresponding entry (TX) in the state table.

 Figure 17: Inner Join in a Dimension Calculation View

The inner join is the most basic of the join types. It returns rows when there is at least one match on both sides of the join.

Inner Join in a Cube Calculation View



```

=Select
    o.ORDER_ID, o.C_ID, sub1.CUSTOMER, sub1.AGE,
    sub1.STATE, sub1.STATETXT, o.AMOUNT
  FROM "SalesOrder" o
  Inner Join
    (Select
        c.C_ID, c.CNAME As CUSTOMER,
        c.AGE, c.STATE, SNAME as STATETXT
      FROM "Customer" c
      Inner Join "State" s
      On c.STATE = s.STATE) as sub1
    On o.C_ID = sub1.C_ID
  ORDER BY o.ORDER_ID

```

ORDER_ID	C_ID	CUSTOMER	AGE	STATE	STATETXT	AMOUNT
1	1	WERNER	10	MI	MICHIGAN	100
2	2	WERNER	10	MI	MICHIGAN	100
3	3	MARK	11	MI	MICHIGAN	100

AMOUNT	ORDER_ID	C_ID
100	4	4
100	8	77
100	1	1
100	2	1
100	3	2
		TOM
		BOB

STATE	SNAME
AL	ALABAMA
MI	MICHIGAN

C_ID	CNAME	AGE	STATE
1	WERNER	10	MI
2	MARK	11	MI
3	TOM	12	TX
4	BOB	13	TX

Inner Joins lose facts with fragmented dimensions. Order (4 & 8) lost due to no corresponding customer or state record.

Figure 18: Inner Join in a Cube Calculation View

The figure, Inner Join in a Cube Calculation View, shows the behavior of inner joins in a cube calculation view.

With the sample scenario data, some facts are not retrieved because customer information is missing.

Left Outer Join

```

=Select
    c.C_ID, c.CNAME As CUSTOMER,
    c.AGE, c.STATE, SNAME as STATETXT
  FROM "Customer" c
  Left Outer Join "State" s
  On c.STATE = s.STATE

```

C_ID	CUSTOMER	AGE	STATE	STATETXT
1	WERNER	10	MI	MICHIGAN
2	MARK	11	MI	MICHIGAN
3	TOM	12	TX	?
4	BOB	13	TX	?

STATE	SNAME
AL	ALABAMA
MI	MICHIGAN

C_ID	CNAME	AGE	STATE
1	WERNER	10	MI
2	MARK	11	MI
3	TOM	12	TX
4	BOB	13	TX

No matches for TX in the right table.

Figure 19: Left Outer Join in a Dimension Calculation View

A left outer join returns all rows from the left table, even if there are no matches in the right table.

© Copyright. All rights reserved.

21

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

Unit 1: Information Views

Left Outer Joins and Design Time Filters



CUSTOMER

C_ID	CNAME	STATE	AGE
1	WERNER	MI	10
2	MARK	MI	11
3	TOM	TX	12
4	BOB	TX	13

Design time filter applied (AGE < 13) of left/central table

STATE

STATE	SNAME
MI	MICHIGAN
AL	ALABAMA

Design time filter applied to (STATE = MI) on right table

RESULT

C_ID	CNAME	STATE
1	WERNER	MICHIGAN
2	MARK	MICHIGAN
3	TOM	NULL

Filters are applied to both tables and then afterwards the join is executed.
Due to the left outer join TOM will be included in the result set even though he resides in TX.

Figure 20: Left Outer Joins and Design Time Filters

The figure, Left Outer Joins and Design Time Filters, shows the behavior of left outer joins with design time filters.

Left Outer Join in a Cube Calculation View



```

Select
    o.ORDER_ID, o.C_ID, sub1.CUSTOMER, sub1.AGE,
    sub1.STATE, sub1.STATETXT, o.AMOUNT
  FROM "SalesOrder" o
  Left Outer Join
  (
    Select
      c.C_ID, c.CNAME As CUSTOMER,
      c.AGE, c.STATE, SNAME as STATETXT
    FROM "Customer" c
    Left Outer Join "State" s
    On c.STATE = s.STATE) as sub1
  On o.C_ID = sub1.C_ID
 ORDER BY o.ORDER_ID

```

ORDER_ID	C_ID	CUSTOMER	AGE	STATE	STATETXT	AMOUNT
1	1	WERNER	10	MI	MICHIGAN	100
2	2	WERNER	10	MI	MICHIGAN	100
3	3	MARK	11	MI	MICHIGAN	100
4	4	BOB	13	TX	?	100
5	8	?	?	?	?	100

Customer TOM is not returned due to no corresponding sale item record in sales order table.

AMOUNT	ORDER_ID	C_ID	C_ID	CNAME	AGE	STATE	STATE	SNAME
100	1	1	1	WERNER	10	MI	MI	MICHIGAN
100	2	1	2	MARK	11	MI	MI	MICHIGAN
100	3	2	3	TOM	12	TX	TX	ALABAMA
100	4	4	4	BOB	13	TX	TX	ALABAMA
100	8	77						

Figure 21: Left Outer Joins in a Cube Calculation View

The figure, Left Outer Joins in a Cube Calculation View, shows the use of left outer joins in a cube calculation view.

Compared with the Inner join, all the sales order data (including those with no corresponding customer information) is retrieved, but still an analysis of sales by customer or state will return irrelevant data.

Right Outer Join



- Right Outer Join returns all the rows from the right table, even if there are no matches in the left table.

```
>Select
    c.C_ID, c.CNAME As CUSTOMER,
    c.AGE, c.STATE, SNAME as STATETXT
FROM "Customer" c
Right Outer Join "State" s
On c.STATE = s.STATE
```

C_ID	CUSTOMER	AGE	STATE	STATETXT
1	1 WERNER	10	MI	MICHIGAN
2	2 MARK	11	MI	MICHIGAN
3	?	?	?	?
				ALABAMA

		STATE	SNAME
1	WERNER	10	MI
2	MARK	11	MI
3	TOM	12	TX
4	BOB	13	TX
C_ID	CNAME	AGE	STATE

- Alabama is included in the result set, though there is no match in the left table.

Figure 22: Right Outer Join in a Dimension Calculation View

A right outer join returns all the rows from the right table, even if there are no matches in the left table.

Right Outer Join in a Cube Calculation View



```
Select
    o.ORDER_ID, o.C_ID, sub1.CUSTOMER, sub1.AGE,
    sub1.STATE, sub1.STATETXT, o.AMOUNT
FROM "SalesOrder" o
Right Outer Join
(
    Select
        c.C_ID, c.CNAME As CUSTOMER,
        c.AGE, c.STATE, SNAME as STATETXT
    FROM "Customer" c
    Right Outer Join "State" s
    On c.STATE = s.STATE) as sub1
On o.C_ID = sub1.C_ID
ORDER BY o.ORDER_ID
```

ORDER_ID	C_ID	CUSTOMER	AGE	STATE	STATETXT	AMOUNT
1	?	?	?	?	?	ALABAMA ?
2	1	1 WERNER	10	MI	MICHIGAN	100
3	2	1 WERNER	10	MI	MICHIGAN	100
4	3	2 MARK	11	MI	MICHIGAN	100

- Right Outer Join results in NULL measure.

AMOUNT	ORDER_ID	C_ID								STATE	SNAME
100		4	4								
100		8	77								
?	*	?	?						?	AL	ALABAMA
100		1	1								
100		2	1	1	WERNER	10	MI	MI	MICHIGAN		
100		3	2	2	MARK	11	MI	MI	MICHIGAN		
				3	TOM	12	TX				
				4	BOB	13	TX				
C_ID	CNAME	AGE	STATE								

Figure 23: Right Outer Join in a CUBE Calculation View

Unit 1: Information Views

Full Outer Join



				STATE	SNAME
C_ID	CNAME	AGE	STATE	AL	ALABAMA
1	WERNER	10	MI	MI	MICHIGAN
2	MARK	11	MI		
3	TOM	12	TX		
4	BOB	13	TX		

Figure 24: Full Outer Join

A full outer join combines the behaviors of the left and right outer joins.

The result set is composed of the following rows:

- Rows from both tables that match on joined columns
- Rows from the left table with no match in the right table
- Rows from the right table with no match in the left table



Caution:

A full outer join is supported by calculation views only, in the standard Join and Star Join nodes.

However, in a Star Join node, a full outer join can be defined only on **one** dimension calculation view, and this view must appear last in the star join node.

Referential Join

A referential join is semantically an inner join that assumes that referential integrity is given, meaning that the left table always has a matching entry in the right table.

It is an optimized or faster inner join where the right table is generally not checked if no field from the right table is requested.



Note:

From SPS11 onwards, the referential join is supported in any type of join in calculation views. Before this version, you could only use it in a Star Join node.



Relies on Referential Integrity	Each entry in the left table MUST have a corresponding entry in the right table
Optimized for performance	Join is only performed if at least one field from the right table is requested.
Like an Inner Join when join is executed	When field from both tables are requested an inner Join is performed.

Figure 25: Referential Join

Referential Joins in a Calculation View

In calculation views, referential joins are executed in the following way:

- If at least one field is selected from the right table, it will behave as an inner join.
- If no field from the right table is selected, the execution of the referential join depends on the cardinality:
 - If the cardinality is 1..1 or n..1, the join is not executed.

This corresponds to the most common situation, and in particular when you join a (left) fact table or view with a (right) view or table that contains only dimensions.

This is where the optimization occurs.

- In the rare case where you use a 1..n cardinality, the join is executed as an inner join.
Indeed, this is a requirement to get the correct number of rows in the output, which depends on the number of matching rows in the right table or view.

From a performance perspective, the left outer join is almost equally as fast as the referential join, while the inner join is slower due to the join always being executed.

- Referential joins must be used with caution because they assume that referential integrity is ensured at any time.



Note:

If you consider a join between a fact table and its related attributes, keep in mind that facts without corresponding attributes violate referential integrity, while attributes without facts (for example, a customer without any order) do not.

Using referential joins in a context where referential integrity is not ensured might lead to different results depending on whether or not you select columns from the right table.

Unit 1: Information Views

Referential Joins in a Cube Calculation View

 **SELECT C_ID, CNAME, STATE, AGE, SNAME**

C_ID	CNAME	STATE	AGE	SNAME
1	WERNER	MI	10	MICHIGAN
2	MARK	MI	11	MICHIGAN

*** Customer 77, TOM and BOB are not returned since C_ID is a Joined key field resulting in an Inner Join. BOB has no corresponding description for TX and TOM has no corresponding facts.

SELECT C_ID, CNAME, AGE

C_ID	CNAME	AGE
1	WERNER	10
2	MARK	11
4	BOB	13

*** TOM is not returned because there are no corresponding facts in the sales table.

SELECT SUM(AMOUNT)

AMOUNT
500

*** The Amount includes all facts including Customer 77 and BOBs order even though master records do not exist: when only non-key fields are selected from the left table, all joins to other tables will be omitted.

 Figure 26: Referential Joins in a Cube Calculation View

The figure, Referential Joins in a Cube Calculation View, shows an example of referential joins in a cube calculation view.

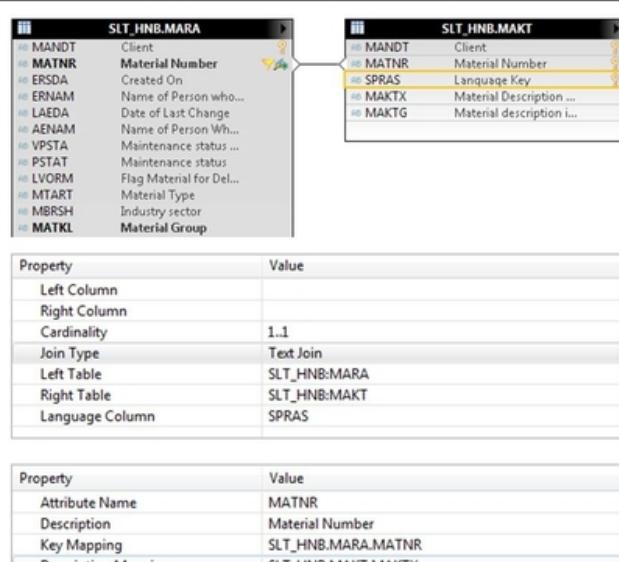
Text Join

 **SLT_HNB:MARA**

MANDT	Client
MATNR	Material Number
ERSDA	Created On
ERNAME	Name of Person who...
LAEDA	Date of Last Change
AENAM	Name of Person Wh...
VPSTA	Maintenance status ...
PSTAT	Maintenance status
LVORM	Flag Material for Del...
MTART	Material Type
MBRSH	Industry sector
MATKL	Material Group

SLT_HNB:MAKT

MANDT	Client
MATNR	Material Number
SPRAS	Language Key
MAKTX	Material Description ...
MAKTG	Material description i...



Property	Value
Left Column	
Right Column	
Cardinality	1..1
Join Type	Text Join
Left Table	SLT_HNB:MARA
Right Table	SLT_HNB:MAKT
Language Column	SPRAS

Property	Value
Attribute Name	MATNR
Description	Material Number
Key Mapping	SLT_HNB:MARA.MATNR
Description Mapping	SLT_HNB:MAKT.MAKTX

 Figure 27: Text Joins

A text join enables SAP HANA to handle the translation of attribute labels in a way that corresponds to the way translation texts are stored in the master data or SAP systems, such as SAP ERP.

Text Join Example



- Text join is used when a translation for a dimension is available
- Designed for ERP tables (and typically SPRAS column)
- User language is used as a filter at runtime to find the right translation for that attribute

PRODUCT table

SPRAS	ID	DESC
E	1	Car
D	1	Auto
E	2	Motorbike
D	2	Motorrad

Text join applied on **ORDER.PR_ID = PRODUCT.ID**
using **SPRAS** as the language column

ORDER table

ORDER_ID	PR_ID	VALUE
00215753	1	1000
00237469	2	2000

Figure 28: Text Join Example

The figure, Text Join Example, is a simplified example of a text join. Depending on the session language of the end user, the DESC column displays the product description in English or in German.

Temporal Join

It is possible to add a temporal condition to a join in order to find matching records from two tables based on a date. The records are matched only if a date column of one table is within a time interval defined by two columns of the other table.

This is useful to manage time-dependent attributes.

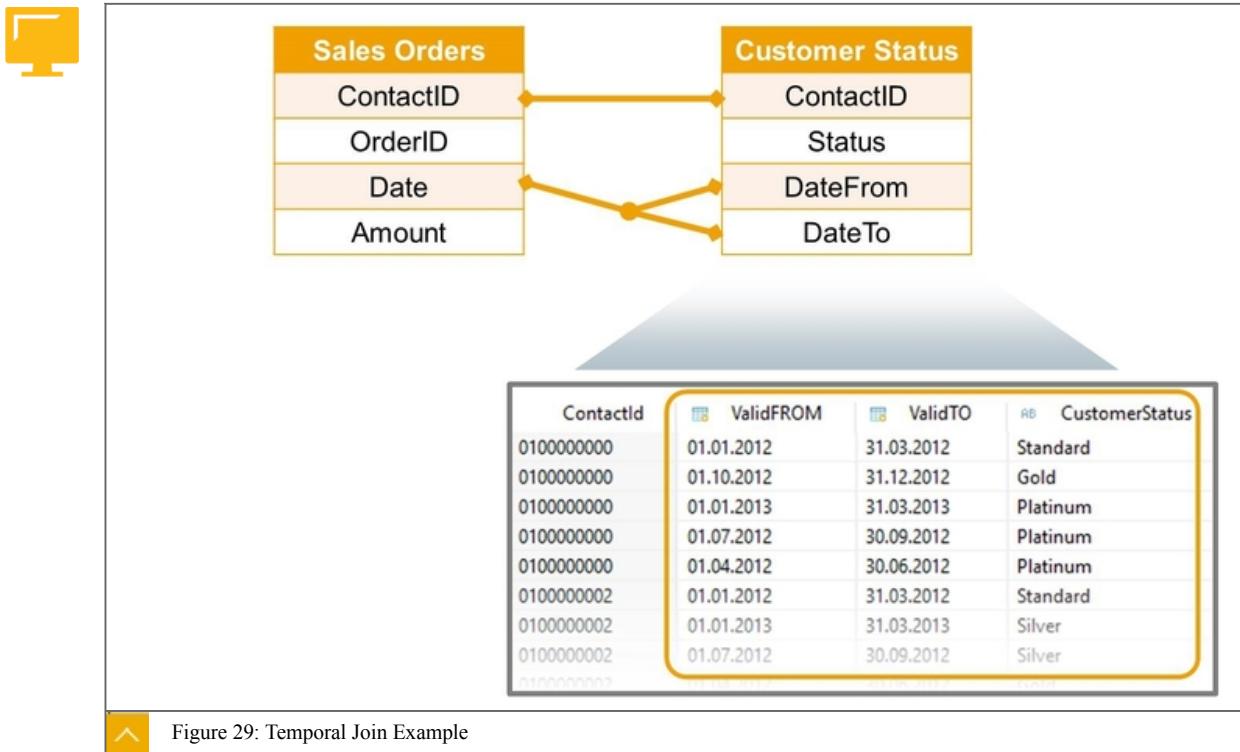


Caution:

Temporal joins are only supported in the star join of calculation views of the type Cube with Star Join . The join type must be defined as Inner .

Unit 1: Information Views

Temporal Join Example



In this example, the status of the customers can change over time, and this information is captured in a dedicated table (Customer Status). If you need to analyze the sales orders and include the status of each customer when they issued the order, you create an inner join on the ContactID column and add a temporal condition as follows:

- Temporal column: Date (Sales Orders)
- From Column: DateFrom (Customer Status)
- To Column: DateTo (Customer Status)
- Temporal Condition: Include Both

Note:

- Temporal conditions can be defined on columns of the following data types:
 - timestamp
 - date
 - integer
- Only columns already mapped to the output of the Star Join node can be defined as **Temporal Column** in the temporal properties of the join.

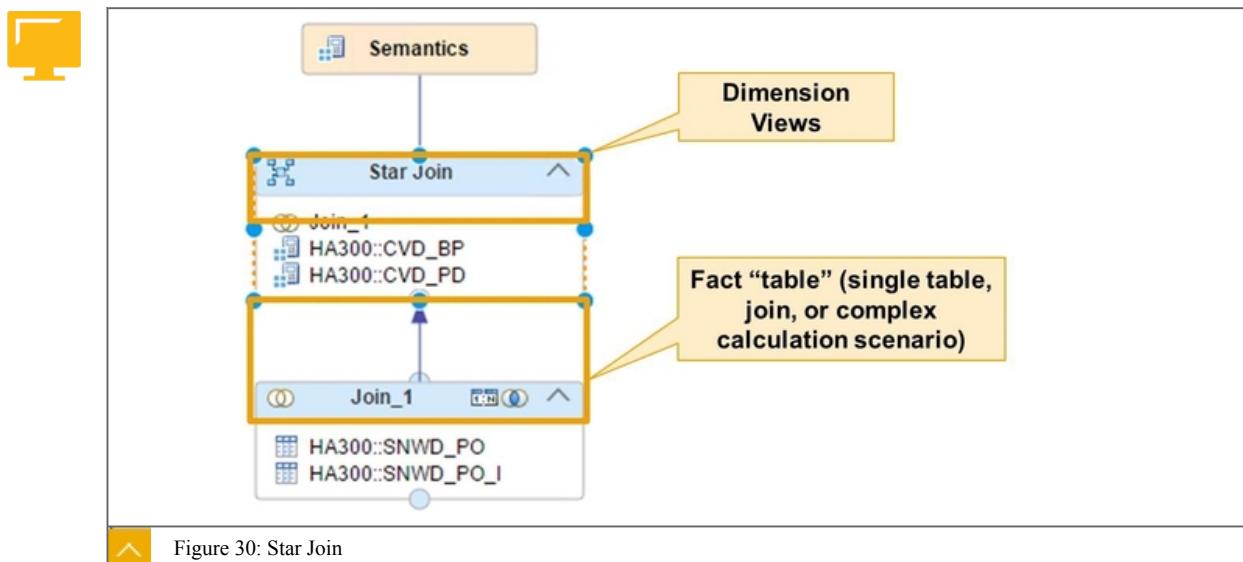
Star Join

The star join in calculation views of the type **Cube with Star Join** is a **node type**, rather than a join type.

It is used to structure data in a star schema. The fact table (data source) of a star join can be any type of input node. However, only calculation views of the data category Dimension are allowed as input nodes for dimensions.

The type of joins between the fact and dimension tables within the star schema can be defined in the star join node. The available joins are as follows:

- Referential join
- Inner join
- Left outer join
- Right outer join
- Full outer join (new join type in SAP HANA SPS11, with some specific restrictions when used in a star join)
- Text join



Spatial Join

A spatial join enables the modeler to use the native spatial capabilities of SAP HANA in graphical modeling tools.



Note:
Spatial data modeling is covered in a dedicated lesson.

Join Cardinality

The cardinality of a join defines how the data from two tables joined together are related, in terms of matching rows.

For example, if you join the Sales Order table (left table) with the Customer table (right table), you can define an n..1 cardinality. This cardinality means that several sales orders can be related to the same customer, but the opposite is not possible (you cannot have a sales orders that relates to several customers).

Unit 1: Information Views



Caution:

We recommend that you specify the cardinality only when you are sure of the content of the tables. If not, just leave the cardinality blank.

Validating a Join

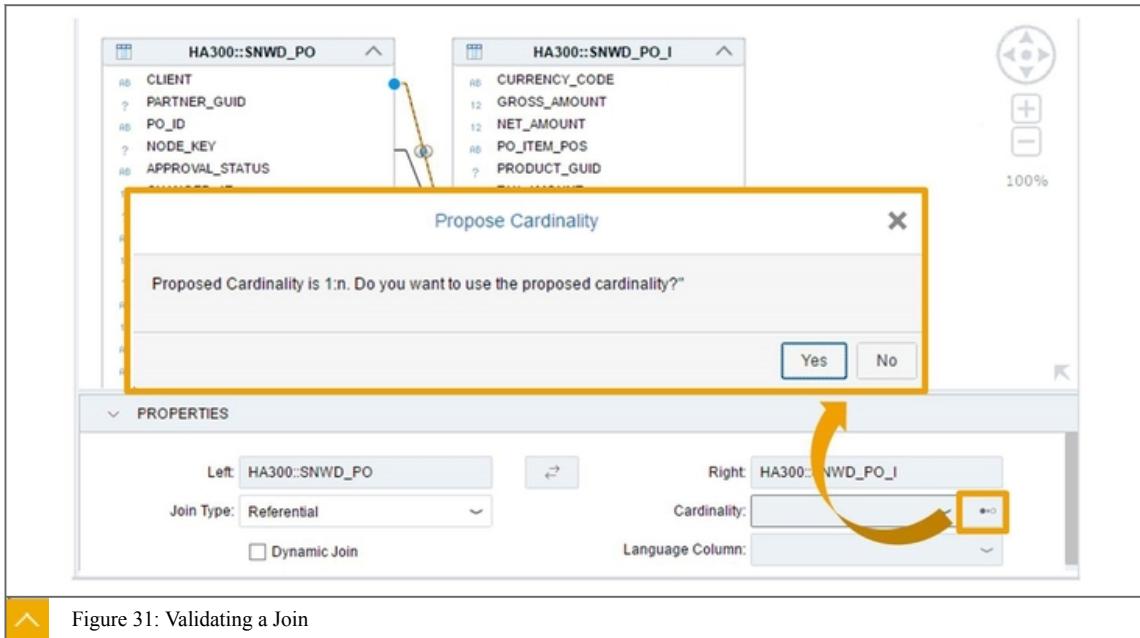


Figure 31: Validating a Join

A feature in the SAP Web IDE for SAP HANA suggests the recommended cardinality, based on an analysis of the tables that are joined together.



Caution:

This analysis of joined tables is performed at the moment you define the join. If the content of the table evolves after that, the cardinality you have defined might become incorrect.

For example, you are validating the join between the Sales Order and Customer tables, but your data contains only one sales order per customer. In this case, the join validation might suggest a 1..1 cardinality, which does not correspond to the expected scenario in real life.

Combining Several Data Sets: Union Versus Join

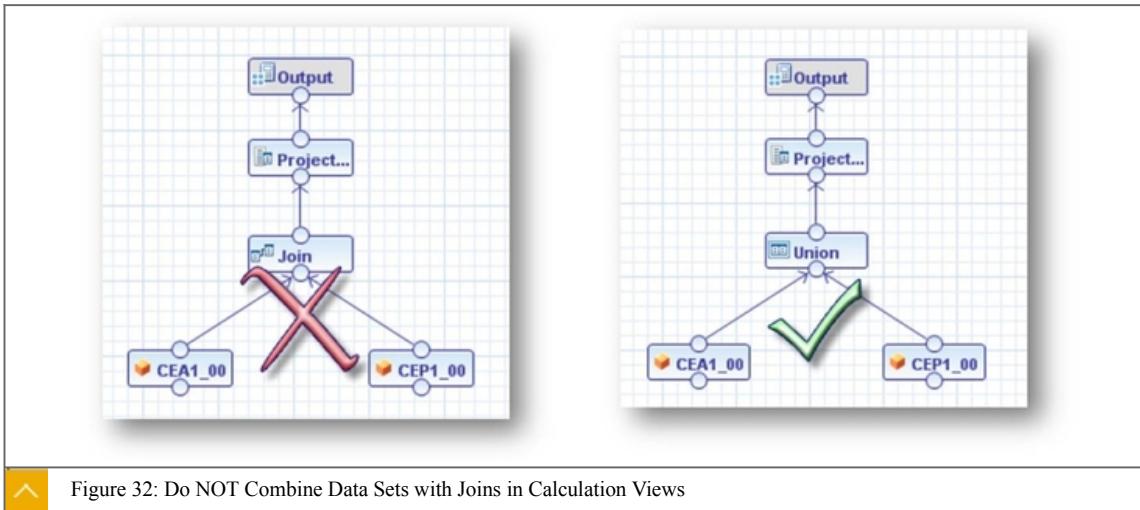
When you need to combine measures originating from more than one table or calculation view, you might want to create a join between these tables. However, under most circumstances, you must avoid using a join to address this requirement, because it is very costly from a computing standpoint.

It is more beneficial to use a union node, which provides much better performance.



Note:

Technically, a union is not a join type.



LESSON SUMMARY

You should now be able to:

- Connect tables
- Determine the type of join to use when connecting tables

Unit 1

Lesson 3

Creating Dimension Calculation Views

LESSON OVERVIEW

This lesson covers the creation of Dimension Calculation Views, which are used to organize the master data available in the underlying database tables. These views are, in turn, joined to fact table in Cube Calculation Views to analyze the facts in a way that suits your reporting needs.

Dimension Calculation Views can be shared by several Cube calculation views, which makes the maintenance of the virtual data model much easier.

Business Example

You want to create a view on your customer information (for example, ID, name, country, city, address, category, and so on) based on different tables.

In SAP HANA, you generally retrieve this information with Calculation View of type DIMENSION, which enables you to structure the master data and adapt it to your reporting requirements, for example:

- Join several tables to define a dimension that contains all the attributes you need
- Select only some of the columns from the source tables, as some data could be unnecessary
- Modify the name or labels of the columns
- Semantically associate the name or code and the description of objects (for example, product code and product description)
- Define calculated attributes
- Define hierarchies based on the master data



LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Create dimension calculation views
- Define calculated attributes
- Create time-based dimension views
- Use base table aliases
- Define label columns and hide attributes in a dimension calculation view

Dimension Calculation Views

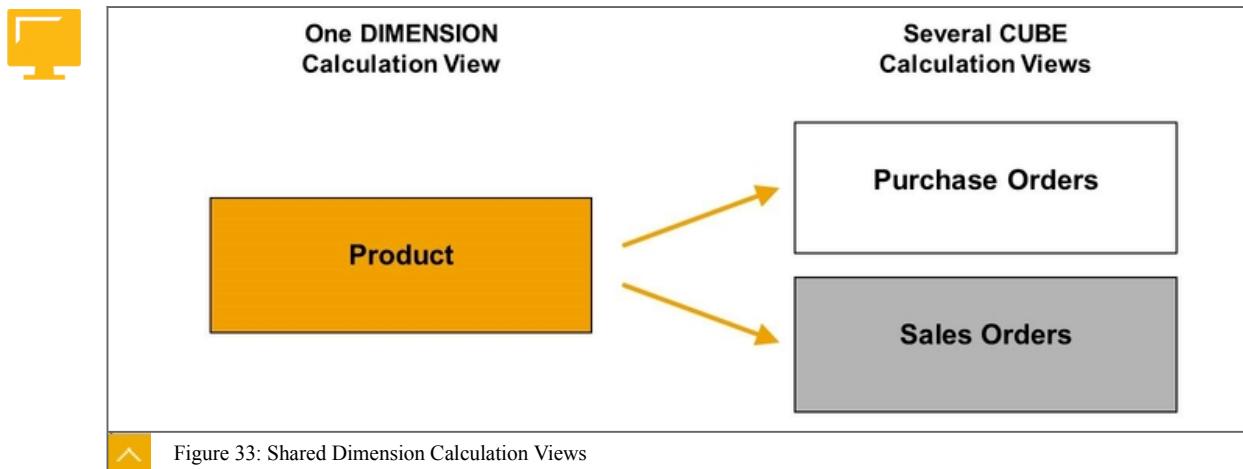
Dimension calculation views are used to give context. This context is provided by text tables which give meaning to data.

For example, if a fact table or cube calculation view in a car sales database only contains a numeric ID for each dealer, you can use a dimension calculation view to provide information about each dealer. Using this method, you could then display the names and addresses of the car dealers, and give context to the data.

Dimension calculation views are used to select a subset of master data columns and rows from tables. They are not meant to handle measures or aggregates.

A dimension calculation view does not need to be based on a single table. On the contrary, you can use them to join master data tables to each other. For example, to join Products to Product Categories .

Shared Dimension Calculation Views



Like many other types of information views, dimension calculation views are reusable objects and can be shared between several cube calculation views.

For example, the product attribute view can be used both in a purchase order cube calculation view and in a sales order cube calculation view.

Supported Characters for Views Names and View Objects

SAP HANA supports all the Unicode characters in object names; that is, views, columns, input parameters, and so on.

However, a number of characters are forbidden in the object names. These characters are \ / : * ? " < > | . ; ' \$ % , ! # + and the **space** character.

Supported Types of Nodes

Dimension calculation views support all types of nodes (Projection, Join, Union, Aggregation, and Ranking). This allows a very flexible design of dimension calculation views. The most commonly used nodes are Join nodes (to join master data tables) and Projection nodes (to filter data, select specific columns from the master data tables, and create calculated attribute columns).

Only the Star Join node is not supported in this type of view, because it is specific to calculation views of the type cube with star join.

Calculated Attributes

It is possible to create additional calculated columns in a dimension calculation view.

Unit 1: Information Views

For example, you have two columns containing the first and last name of the customer, but you would like to have all this information (first and last name) in a single column. You can do this by creating a calculated column based on string manipulations.



Figure 34: Calculated Columns

- The calculation can be an arithmetic or just a character string manipulation.
- Calculated columns also support non-measure attributes as part of the calculation.
- It is possible to nest calculated columns, so that one calculated column in turn is based on other calculated columns.

Time-Based Dimension Calculation Views

Apart from the Standard type for calculation views, the Time type is used to create dimension calculation views that apply exclusively to time data.

The tables containing measures generally have date or date-time columns to clearly define when an event occurred, or what period of the year a measure relates to, such as a month or a quarter. The purpose of these time-based dimension views is to ease the manipulation of measures across time.

In order to report in a simple and flexible way, you can use time-based dimension views to build a time hierarchy that corresponds to your needs, and convert a timestamp into simpler attributes, such as year, month, or day.

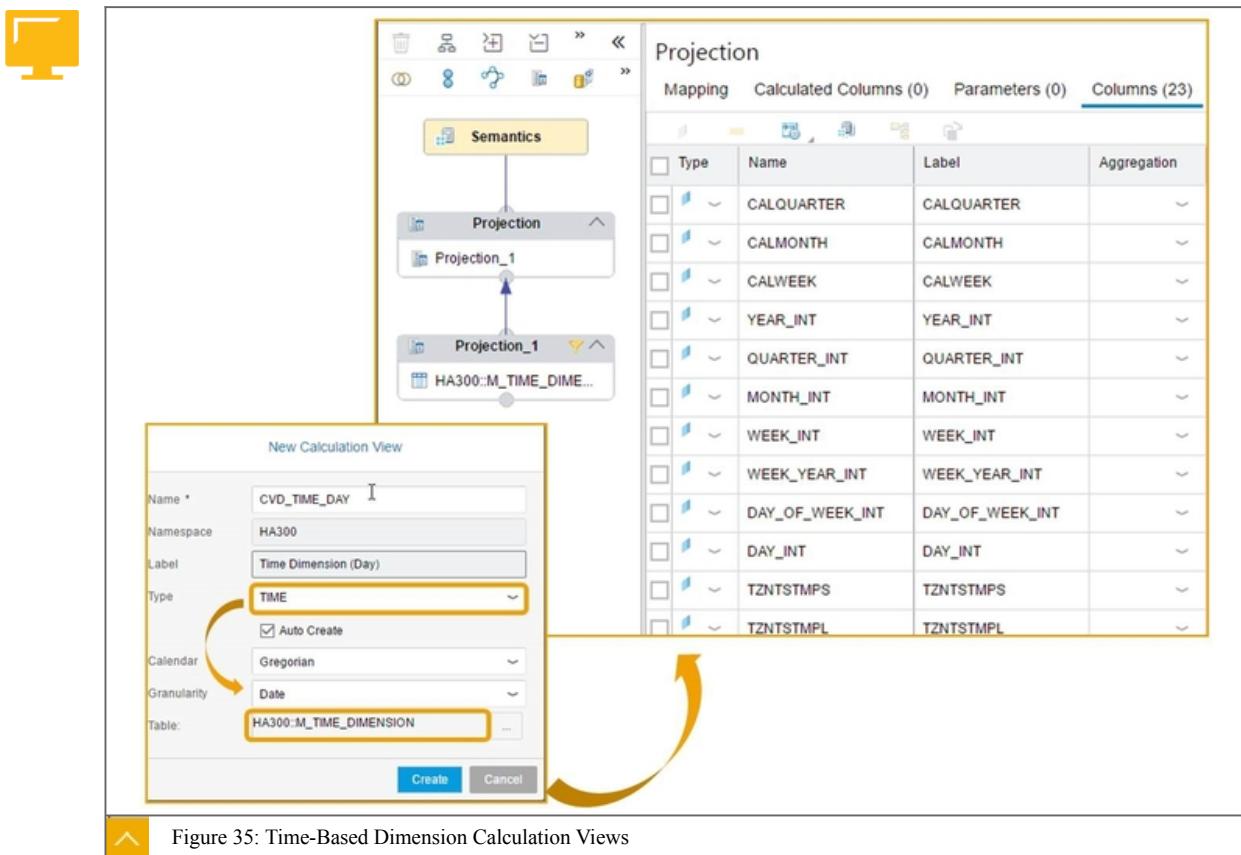


Figure 35: Time-Based Dimension Calculation Views

Note:

Even if it does not appear explicitly in the user interface, a calculation view of the subtype Time is automatically created as a dimension calculation view. You can check that in the semantics of the view (in the View Properties tab).

Different Calendar Types in Time Calculation View

The following are different calendar types available in time calculation views:

- Gregorian

The Gregorian calendar is made up of years, months, and days. You can adjust the level of granularity, down to hours, minutes, or seconds.

- Fiscal

The fiscal calendar is organized into fiscal years and fiscal periods. Several fiscal year variants can be defined depending on your reporting needs.

Note:

The fiscal calendar is especially useful to display data in your information models according to the fiscal calendar tables available in your SAP ERP system.

Unit 1: Information Views



To Create a Time Dimension Calculation View
The time data has been generated in your SAP HANA system.



Note:

This is generally done by a system administrator, and it populates some **M_TIME_DIMENSION_*tables** (Gregorian calendar) or the **M_FISCAL_CALENDAR** table (fiscal calendar), located in the **_SYS_BI** schema.

1. Create a new calculation view of subtype **Time**.
2. Choose a calendar type (Gregorian or fiscal).
3. Define the granularity level (Gregorian) or the variant (fiscal).
4. To have the relevant table from the **_SYS_BI** schema automatically included in the calculation view scenario, select the **Auto Create** checkbox and choose the table as suggested by the search field, for example, **M_TIME_DIMENSION**



Note:

Because the technical tables for the time dimension are located in the **_SYS_BI** schema, synonyms for these tables must exist in your project.

5. Choose **Finish**.
6. Add the columns that fit your reporting needs to the output.
Make sure that you include a relevant column that will be used as a key to join the time calculation view to the tables or views that contain your measures.
7. If necessary, organize time data into level hierarchies.
8. Save and activate your calculation view.

Base Table Alias



- When adding multiple instances of the same table to a view, an alias will be proposed.
- It is possible to modify the alias name in the data source properties (*Mapping* tab).



Figure 36: Adding Multiple Base Tables Using Aliases

In some cases, you need to use the same base table more than once in the same calculation view. In this case, you can define a table alias for any additional instance of the same source table.



Note:

The SAP Web IDE for SAP HANA automatically suggests an alias, but you can choose your own.

Unit 1: Information Views

Hidden Columns



- To hide a column, select the *Hidden* checkbox of the column in the *Semantics* node.
- The column will not be exposed to reporting tools but can be used within the calculation view itself, for example in calculated columns.

Semantics

View Properties Columns (9) Hierarchies (0) Parameters (0)

Type	Name	Hidden	Data Type	Semantics	Conversion Function
LANGUAGE	LANGUAGE	<input type="checkbox"/>	NVARCHAR(1)	<input type="checkbox"/>	<input type="checkbox"/>
FIRST_NAME	FIRST_NAME	<input checked="" type="checkbox"/>	NVARCHAR(40)	<input type="checkbox"/>	<input type="checkbox"/>
LAST_NAME	LAST_NAME	<input checked="" type="checkbox"/>	NVARCHAR(40)	<input type="checkbox"/>	<input type="checkbox"/>
FULL_NAME	FULL_NAME	<input type="checkbox"/>	VARCHAR(100)	<input type="checkbox"/>	<input type="checkbox"/>

Figure 37: Hidden Columns

You can choose to hide the attributes that are not required for client consumption. This typically occurs in the following scenarios:

- When an attribute column is calculated using a complex computation of source columns or intermediary calculated columns, you can hide any column that you do not want to expose to the end user.

In the figure, Hidden Columns, for example, the FULL_NAME column is a concatenation of the first and last name (calculated column), and the source columns FIRST_NAME and LAST_NAME have been hidden.

- When a column that contains a description has been assigned to the corresponding code, name or ID column as a Label Column, you can hide this column. This action is particularly useful in scenarios where the reporting tools are able to interpret this code, name or ID to description assignment.

For example, SAP BusinessObjects Analysis and SAP Design Studio are able to expose Label Columns. The code, name, or ID, and the label, are generally referred to as Key and Text, respectively.



Note:

Only a column that is added to the output of the top node of a calculation view can be defined as a Label Column.

Label Columns and Hidden Attributes



- Label columns can refer to hidden attributes
- You define label columns and hidden attributes in the *Semantics*

Semantics					
Type	Name	Label Column	Hidden	Data Type	Semantics
	BP_COMPANY_NAME	BP_COMPANY_NAME	<input checked="" type="checkbox"/>	NVARCHAR(80)	
	BP_ID	BP_COMPANY_NAME	<input type="checkbox"/>	NVARCHAR(10)	
	CLIENT		<input type="checkbox"/>	NVARCHAR(3)	
	PARTNER_GUID		<input type="checkbox"/>	VARBINARY(16)	

When reporting on an information model containing a label column, with a tool that supports Label Columns:

- Label is displayed as "Text".
- The base column is displayed as "Key"



Figure 38: Label Columns and Hidden Attributes

General Properties of Views

For each view, you can define a number of Properties in the **View Properties** tab of the **Semantics** node. Depending on the type of view, the list of available properties may vary.

The following table gives a list of these properties and a description. You will find more information on some of them in this course later on, and you can also consult the SAP HANA documentation, in particular the **SAP HANA Modeling Guide**.



Table 4: General Properties of Views

Property	Description
Data Category	For calculation views, determines whether the view supports multi-dimensional reporting.
Type	Standard or Time
Default Client	Defines how to filter data by SAP CLIENT (aka MANDT).
Run With	Defines how to apply security when executing a script-based calculation view.
Apply Privileges	Specifies whether analytic privileges must apply when executing a view (SAP Web IDE modeling supports SQL analytic privileges only).
Enable History	The value of this property determines whether your calculation view supports time travel queries.
History Input Parameter	Specifies which input parameter must be used to specify the timestamp in time travel queries.
Deprecate	Identifies views that are not recommended for reuse, though still supported in SAP HANA Modeler.

Unit 1: Information Views

Property	Description
Cache	Defines whether the data retrieved by the view should be cached.
Cache Invalidation Period	If the view data are cached, determines whether the cache must be deleted on a daily or hourly basis, or after each transaction that modifies any of the underlying tables.
Execute in	Determines whether the model must be executed by the SQL engine or column engine.
Pruning Configuration Table	Identifies which table contains the settings to prune Union nodes.
Default Member	Defines the default member to be used for all the hierarchies of this calculation view.
Propagate instantiation to SQL Views	If the calculation view is referenced by a SQL view or a CDS view, determines whether the calculation view must be instantiated (for example, by pruning columns that are not selected by the above queries) or executed exactly as it is defined.
Analyticview Compatibility Mode	If this setting is activated, the join engine ignores joins with cardinality n..m defined in the star join node when no column at all is queried from one of the two joined tables.
Execution Hints	This property is used to specify how the SAP HANA engines must handle the calculation view at runtime.
Enable Hierarchies for SQL access	In a calculation view of type cube with star join, determines whether the hierarchies defined in shared dimension views can be queried via SQL.



LESSON SUMMARY

You should now be able to:

- Create dimension calculation views
- Define calculated attributes
- Create time-based dimension views
- Use base table aliases
- Define label columns and hide attributes in a dimension calculation view

Unit 1

Lesson 4

Using Measures in Calculation Views

LESSON OVERVIEW

This lesson will give you an overview of how to handle measures in graphical Calculation Views, before you deep dive into specific modeling functions, such as Calculated and Restricted Measures, Variables and Input Parameters in the next unit.

You will learn the different types of nodes you can use in Calculation Views and what possibilities each of them provides.

Business Example

You have created a dimension calculation view and understood the main concepts of node, data sources, output columns, semantics.

You are now ready to model Calculation Views that include measures. After creating a simple cube with Star Join Calculation View, you will explore additional features of cube calculation views.



LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Use measures in calculation views
- Explain the benefits of each type of node in calculation views
- Create and combine nodes in calculation views
- Use the features of calculation views to enhance the flexibility of this type of view

Measures in Calculation Views

Compared with attributes, measures bring an additional level of complexity to calculation views. In particular, you must define the relevant behavior of views when measures are aggregated based on a number of attributes.



Note:

Attributes are regular columns and, generally, they do not need to be aggregated. Aggregating data in a dimension calculation view would just be like a `Select Distinct` statement.

In SAP HANA, two types of graphical modeling objects handle measures:

- Calculation views of type Cube
- Calculation views of type Cube with Star Join

In a calculation view, the **measures can originate from several data sources**, that is, several tables, information views, table functions, and so on.

Unit 1: Information Views

For example, you can join two different tables and extract measures from both of them.

Standard Preview or Custom Query

When you design an information view, you have to keep in mind that the output data depends heavily on the query that the client tool executes on top of the view. Common criteria impacting the result set include the following:

- Selected attributes (and measures)
- Aggregate functions applied by the client query
- Ordering defined on one or several columns

The screenshot illustrates the process of generating a custom SQL query and executing it to produce a result set. It includes three main sections:

- Standard Data Preview in Web IDE:** A table showing rows (190) with columns: COUNTRY, PRODUCT_ID, BP_COMPANY_N..., GROSS_AMOUNT, and RANK. The data includes entries for African Gold And Diamar, PicoBit, and others.
- Custom SQL Query:** A section where a user can type an SQL query. A yellow box highlights the default query: `SELECT TOP 1000 [COUNTRY], [PRODUCT_ID], [BP_COMPANY_NAME]`. Step 1 is labeled "Display and copy the default query".
- SQL Console:** A section where the user can modify the query. Step 2 is labeled "Modify Query in SQL Console". The modified query is shown as:


```
SELECT
    "COUNTRY", "PRODUCT_ID",
    SUM("GROSS_AMOUNT") AS "GROSS_AMOUNT",
    SUM("RANK") AS "RANK"
  FROM
    "HA300_03_HDI_CONTAINER_1"."HA300::CVC_SO_RANK"
  GROUP BY "COUNTRY", "PRODUCT_ID"
  ORDER BY 1,4;
```
- Execute:** A button labeled "Execute" (Step 3) which runs the modified query.
- Custom Result:** The final result set produced by the executed query, showing data for AR and AT countries with their respective product IDs, gross amounts, and ranks.
- Check Results:** A step (Step 4) indicating the results have been checked.

To check the behavior of an information view that has a complex calculation scenario, we recommend that you create custom SQL queries on top of the view and test several scenarios. This is especially useful when some of the measures, or all of them, cannot be aggregated with the common Sum function, for example when working with ratios, averages, counter columns and rank nodes.

This approach enables you to enhance the design of the view, if possible, or at least to understand how end users must query the view to get relevant results.

Caution:

Always keep in mind that calculation views in SAP HANA have a special behavior because they are instantiated at runtime, which means that their execution depends on the query that is run on top of them by the front-end tool (SQL console, reporting tool, and so on).

Overview of the Possible Node Types

A calculation view can use as many nodes as you need. Each node has its own capabilities and behavior.



Icon	Node Type	Use Case
	Projection	To filter data or obtain a subset of required columns from a data source
	Aggregation	To summarize measures by grouping them together by attribute columns values
	Join	To query data from two or more data sources
	Union	To combine the data from two data sources
	Star Join	To join attributes to the very last node of a CUBE With Star Join Calculation view
	Rank	To order the data for a set of partition columns and select only the top 3/4/.../n elements

Figure 40: Types of Nodes in Graphical Calculation Views

In addition to these nodes, SAP HANA 2.0 SPS01 introduces to nodes, Minus and Intersects , that allow you to perform set operations. These two nodes will be discussed later on.

Semantic Node

Every calculation view has a semantic node. You do not add this, it is already present and will always be the top node regardless of the data category of the calculation view. In this node, you assign semantic to each column in order to define its behavior and meaning. This is important information used by consuming clients so that they are able to handle the columns appropriately.

The most important setting for each column is its column type. You can choose between **attribute or measure** .

In the semantic node, you can also optionally assign a **semantic type** to each column. A semantic type describes the specific meaning of each column and this can be helpful to any client that consumes the calculation view so it can then represent the columns in the appropriate format. For example, if you define a column as a date semantic type, the front-end application can then format the value with separators rather than a simple string. The key point is that it is the responsibility of the front-end application or consuming client to make use of this additional information provided by the semantic type.

Semantic Attributes

Semantic types for **attributes** can be defined as:

- Amount with Currency Code

Unit 1: Information Views

- 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

Semantic Measures

Semantic types for **measures** can be defined as:

- Amount with currency code
- Quantity with unit of measure

Apart from the semantic type there are other important settings that can be defined for each column such as the following:

- Assigning a description column to another column — for example assign the product id column to a product description column so a user sees a value that is more meaningful
- Hiding a column — perhaps a column is only used in a calculation, or is an internal value that should not be shown, for example — hide the unhelpful product id when we have assigned a description column that should be shown in its place
- Assigning a variable — allow a user to select a filter value at run time for the attribute

One of the most frequently maintained values in the semantic node is the name and label of the column. It is possible to define an alternative name and label to any column so that it will make more sense to a user than what was originally proposed from the data source. For example, who wants to see the word MATNR in a business report column heading when we really should be seeing the words Material Number ?

Projection Node

A Projection node is typically used in the following scenarios:

- To filter measures based on attributes values.
- To extract only some columns from a data source.

Using a Projection node is also a prerequisite to compute a calculated measure before an aggregation. Indeed, calculated columns in the Aggregation nodes of calculation views are always executed after the aggregation.

Join Node

The Join node is used to join two (and only two) sources.

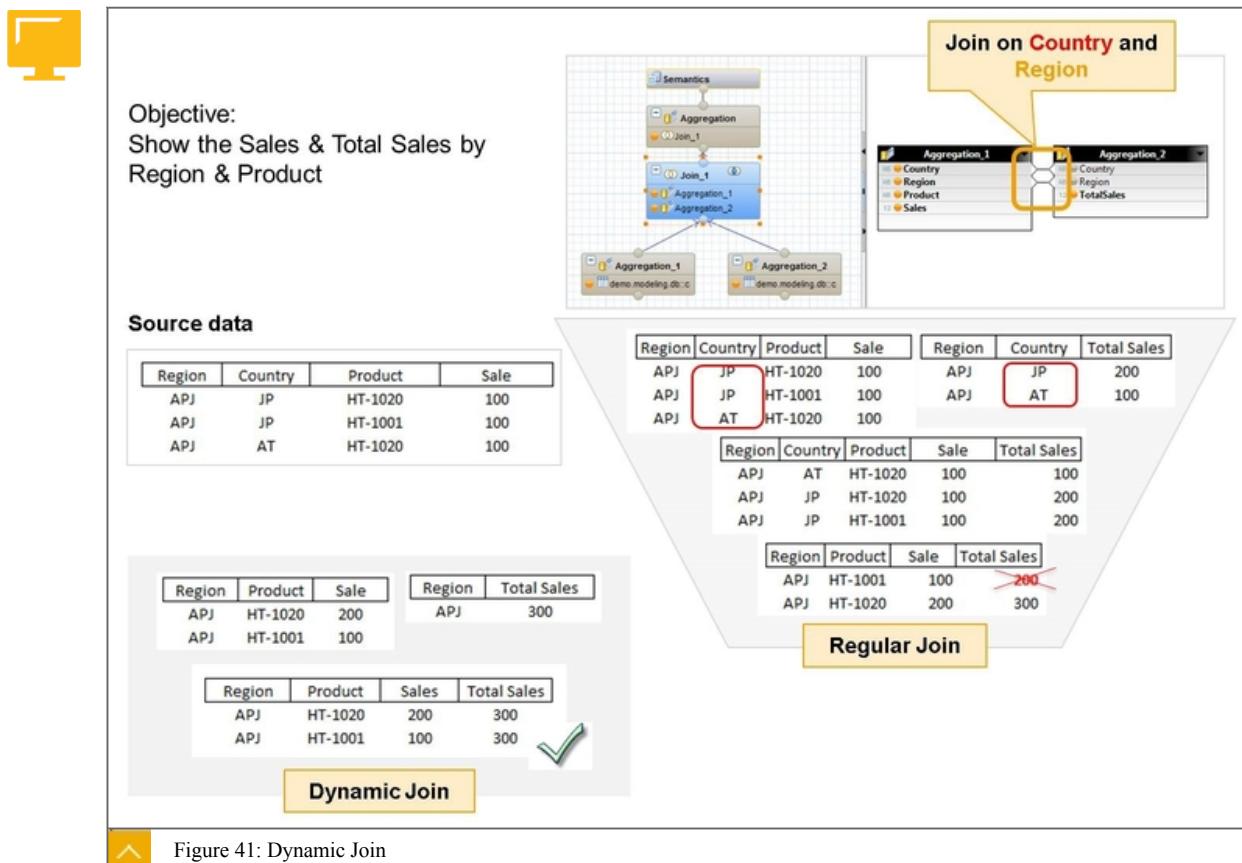
If you need to join more than two sources, you can stack several joins in the scenario of your calculation view.

For each Join node, you must define which columns of the two joined sources must participate in the join, as well as the join type, as discussed in the lesson Connecting Tables. You can also specify the cardinality (with the help of the Propose Cardinality feature), but only if you are sure that the cardinality you provide corresponds to how the data is actually organized.

Spatial Join Type

The Join node type supports spatial joins, which means that you can now join tables that contain spatial data types in graphical calculation views. In some cases, graphical spatial join modeling avoids creating script-based artifacts, such as table functions, to perform spatial operations.

Enhancing Model Flexibility with Dynamic Join



In some scenarios, you want to allow data analysis at different levels of granularity with the same calculation view.

This is generally possible in an Aggregation node when measures support the aggregation at different levels, that is, when it is possible to report measures grouped by one set of columns or another. For example, calculating the total sales by country and product in one case, and by region and product in another case.

The figure, Dynamic Join, shows an example of a more complex scenario, where you want to present two different measures side by side, either by Country, or Region:

- The sales by product
- The total sales (all products)

Unit 1: Information Views

In this case, assuming that you model your calculation view with a regular join on Country and Region , you will get correct results if you analyze the data by country, but the results will be inconsistent if you analyze the data by region.

Note:

On further analysis of the example, you can see that the details of total sales by region and product are inconsistent for products that are not sold in all the countries of the region (HT-1001)... or (to be specific) in all the countries of the region that report sales.

Benefits of a Dynamic Join

With a dynamic join, only the join columns requested in the query are brought into context and play a part in the join execution. As a consequence, the same calculation view can be used for both purposes, that is, to analyze data by country or by region.

Note:

A dynamic join can be defined only with multi-column joins.

If we consider the behavior of the join from an aggregation perspective:

- In a regular (static) join, the aggregation is executed after the join.
- In a dynamic join, when a joined column is not requested by the client query, an aggregation is triggered to remove this column, and then the join is executed based only on the requested columns.

Caution:

With a dynamic join, if none of the joined columns are requested by the client query, you get a query runtime error.

Star Join Node

If your source data for a graphical calculation view is of a star schema type, you can create a calculation view of type Cube with Star Join .

A Star Join node enables you to join the fact data with the descriptive data. The input allowed to the star join include the lower node and all the relevant calculation views of type dimension.

This way, you are logically creating a star schema where the join is created from the central entity to the other entities.

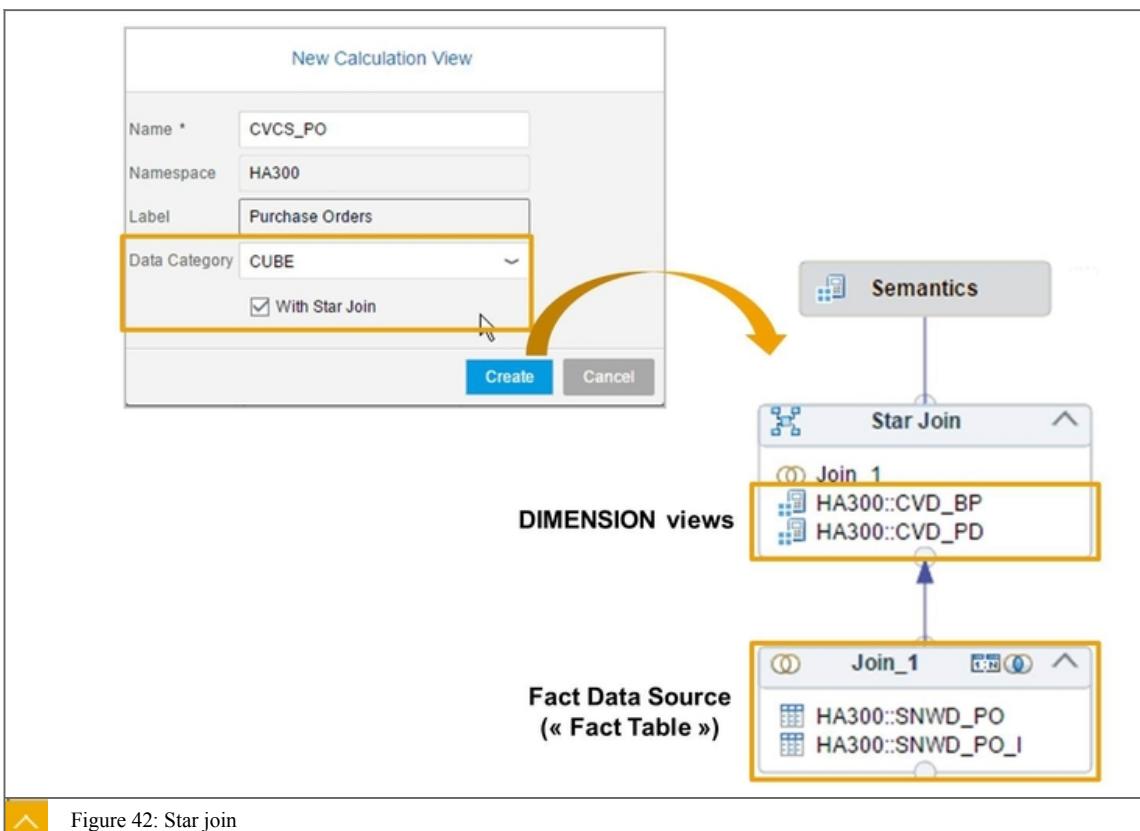


Figure 42: Star join

During deployment, the star join is always deployed with an Aggregation node on top of it. The star join is deployed first with a series of joins, and then the aggregation logic is deployed. This allows the view to aggregate the measures dynamically, based on the attribute columns you include in the result set.

The star join in calculation views supports the Referential join type, which can speed up the execution of the calculation view when you do not query any column from one or several of the dimension calculation views assigned to the Star Join node, because the join will not be executed (at least for the cardinalities 1..1 and n..1).



Caution:

As always, using a referential join requires that the referential integrity of the joined tables is ensured.

As already said, in addition to the Facts Data Source (“Fact Table”), the only other possible sources for a Star Join node are Calculation Views of type DIMENSION. But DIMENSION views can also be used in classical CUBE Calculation Views (without star join) if relevant.

Unit 1: Information Views

Union Node



- If you want to combine multiple result sets with identical structures into one result set, you can use a union node
- A mapping of the sources to the target is required and will allow you to adapt structural differences
- This can be done via a drag and drop interface.

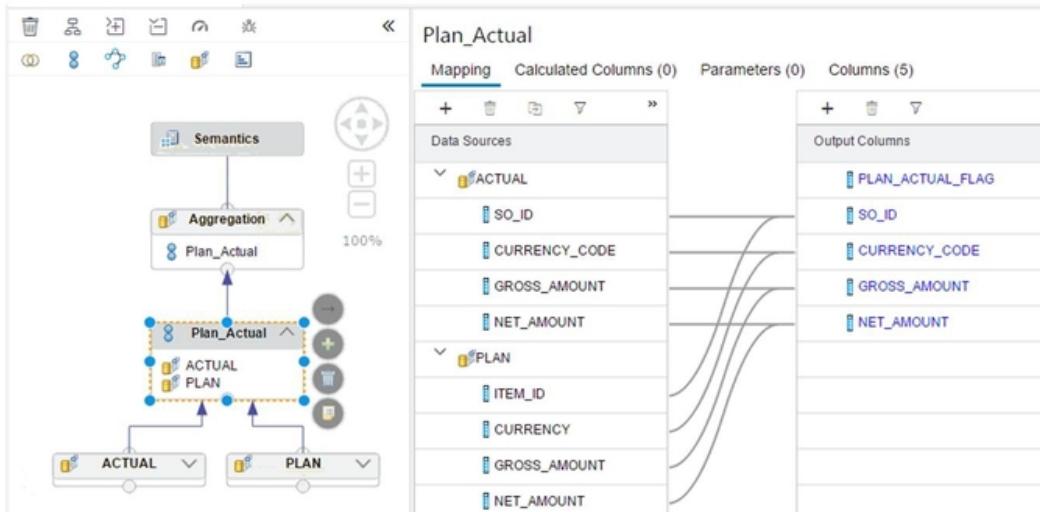


Figure 43: Union Node

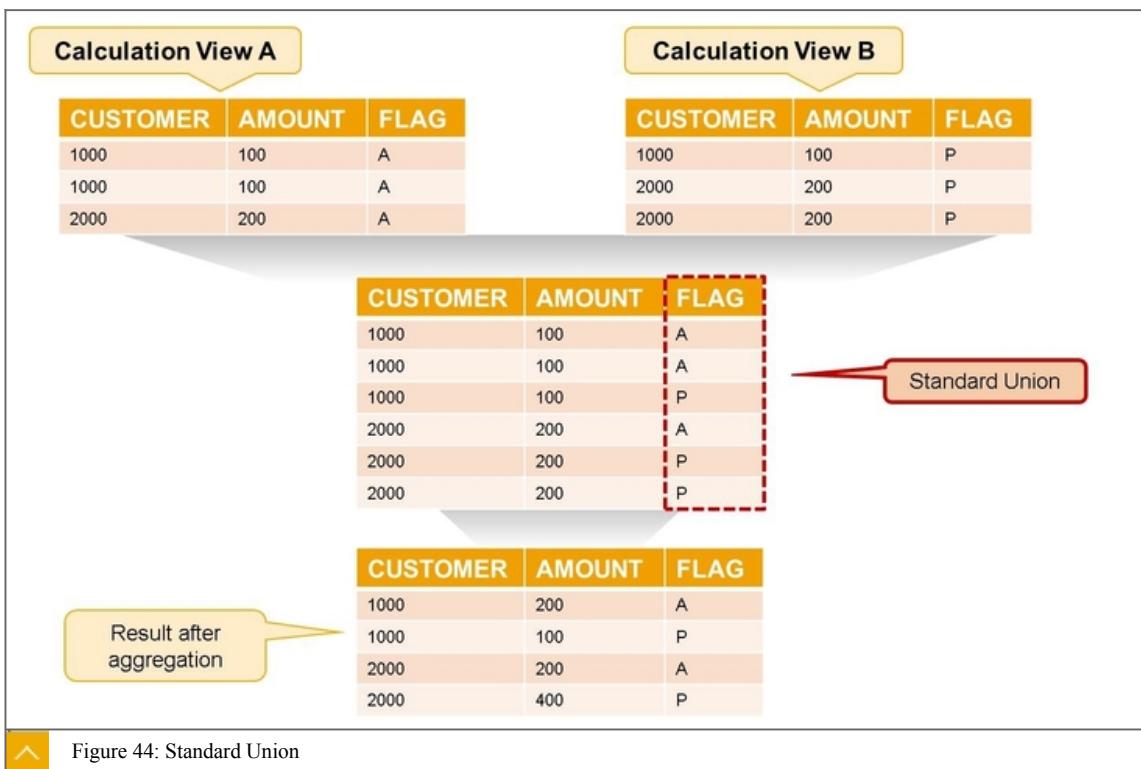
A Union node is used to combine two or more data sets to a common set of columns.

Depending on how different the column names are from the data source, you can map the columns automatically by name (columns with identical names will be mapped automatically) or define the mapping manually.

Standard Union

Depending on the requirement, you can use one of the following approaches:

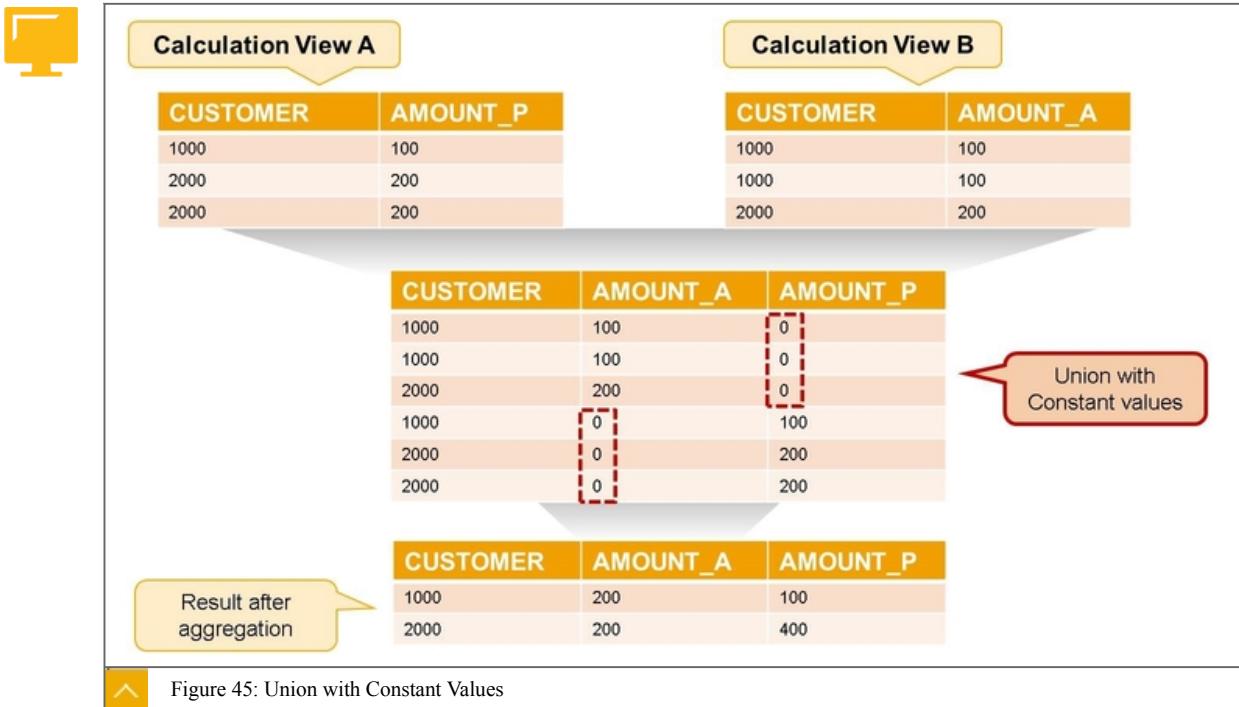
- A standard union
- A union with constant values



A standard union is where, for each target column in the union, there is always one source column that is mapped. In the example, you see how both sources provide a mapped column to the union. The source column does not have to have the same name as the target column. For example, a source column `month` could be mapped to a target column `period`. Also, a standard union does not have to provide a mapping for each source column. In other words, there could be source columns that are left behind and play no part in the union. This scenario is useful when you want to combine measures from multiple sources into one column. Because the data sources can provide an attribute that describes the type of measure (for example, Plan or Actual) it means that we do not lose the meaning of each row.

Unit 1: Information Views

Union with Constant Values



A union with constant values is where you provide a fixed value that is used to fill a target column where the data source cannot provide a mapped column. For example, you have a data source that provides the actual amount and you also have a second data source that provides the planned amount. You decide to avoid combining these measures into one column as they would lose meaning, as there is no attribute to describe the meaning. In this case you would map the measure to a separate target column and provide a constant value '0' to the target column for the missing measure on each side.

The choice between a standard union and a union with constant values depends on the data you are using and the way you want the end users to report on data.

If it is more beneficial to present different measures in different columns, you can use a union with constant values so that you have a way to provide a value (probably zero) to column where there is no source mapping. On the contrary, if it is easier to present measures in a single column and differentiate them with an attribute (such as an 'amount type'), use the standard union.

Mapping Columns Based on their Names

It is possible to map columns in a union node based on their names. This helps to save time when the data structures have some similarities in terms of column names.

The **Auto Map by Name** feature can be used in two different ways, depending on whether you make a selection in the **Data Source** area before clicking the icon.

Table 5: Two Options to Map Columns by Name Automatically

Scenario	Result
You have not selected any data source (default behavior)	All the columns of all data sources are added to the output. Columns with matching names are mapped together.

Scenario	Result
You have selected one or several data source(s)	Only the columns of the selected data source(s) are added to the output. The columns from the other data source(s) are added to the output only if they have a matching column in the selected data source(s). They are mapped to their matching columns.

**Note:**

Selecting a data source before triggering the **Auto Map by Name** is useful when one or several other data sources have a lot of columns that you do not want to include in the output.

Unmapped Columns in a Union Node

There could be instances when a union needs to be performed between data sources that do not provide the same data structure.

- If column names are different but contain the same type of information (for example, CURRENCY versus CURRENCY_CODE, you must define the mapping manually. This can be done with drag and drop.)
- If a column exists in only one data source but you need it in the target, you map it to the target. Then, for the other source, you must decide whether you allow the data in this column to be null or you can define a **constant** value that will be assigned to all the rows from this data source.

As an example, you have a data source that provides the customer status attribute and you have a second data source where the customer status attribute cannot be provided; there is no column for this. You know this second data source contains only customers with the status Active. So you simply fill the target column with a constant value for this data source to Active for every row.

Even if you have a source column available for mapping you can choose to use a constant value instead. This creates the effect of overriding the source value.

Manage Mapping of Unmapped Columns in a Union Node

Although we described how constants are often used when one of the data sources cannot, or should not, provide a value to the union target column, we can also use constants when of the data sources can provide a value. To do this, we first create an empty column in the union target. We then define a constant value for each data source. For example, I would like to create a union between table A that contains part-time employees' data and another table B that contains full-time employees' data. The employment status is very important for my analysis but neither table contains a column that indicates employment status. So I simply add a new column to the union target called 'Employment Status' and define a constant value for one data source as Full Time and the other data source as Part Time. Basically, I have manually tagged each source with a fixed label that now appears in each row and I can use this new column for filtering, aggregation, and so on.

To set the constant value, right-click the target column and choose **Manage Mappings**.

Unit 1: Information Views



Figure 46: Unmapped Columns in a Union Node

Union Node Pruning

In some circumstances, the query that is executed on top of a Union node can ignore completely one or several of the data sources.

Think about a union node that combines two data sources: Cold (data up to 2016) and Hot (data for 2017 and later).

If a select statement queries only the data for 2017, the Cold data source can be ignored from the start (this is called pruning), which will provide a better performance than if the query execution scans the entire Cold data source looking for records for 2017 (and eventually not finding a single one).



Note:
You will learn how to define Union node pruning in the unit Optimization of Models .

Empty Union Behavior

We know that the data sources to a union can provide different columns. So what happens when a query requests columns that are present in one source but not another source? Do you want to be made aware that one source was not able to provide a row? Or is this not important?



Figure 47: Empty Union Behavior

To illustrate this feature, imagine you have a data source A that contains the columns Product and Product Group and another data source B that contains only the column Product. If a query requests only the column Product Group how does data source B respond when it doesn't have this column?

The answer depends on the setting in the property Empty Union Behavior which is set for each data source. The default behavior, as you might expect, is to provide No Row for data source B.

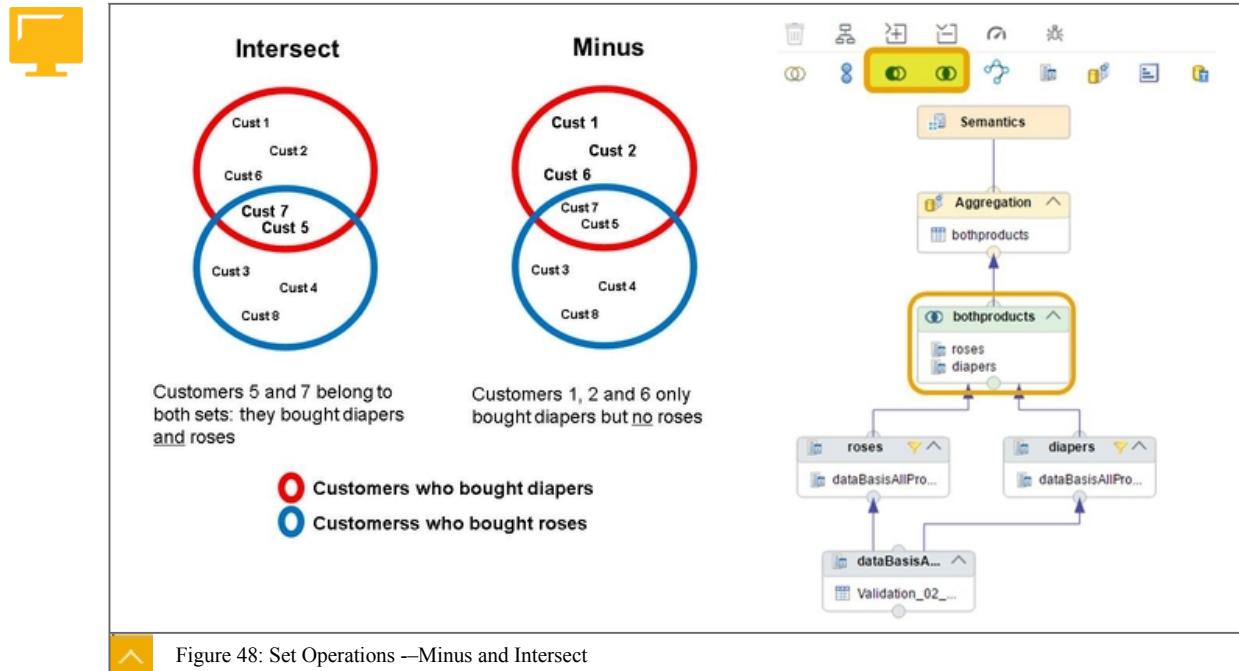
But there are times when you might prefer to know that no rows could be returned from data source B. In that case you should proceed as follows:

1. Add a constant column to the union output.
2. Provide a constant value for each data source with a suitable value that help you identify each source.
3. Change the property Empty Union Behavior to Row with Constants .

To test this, you simply consume the calculation view that contains the union, with a query that does not request any column from one of the data sources. In our case above, the query should request only the column Product Group . The results will then contain multiple rows from data source A, and also a single row with the constant value that you defined for the data source B and nulls will fill the empty columns that it could not provide.

Unit 1: Information Views

Set Operations Nodes – Intersect and Minus



SAP HANA 2.0 SPS01 introduced a new type of node, dedicated to set operations: **Intersect** and **Minus**.

The purpose of these nodes is to easily filter members that belong to two data sets (Intersect), or members that belong to one data set and not another (Minus).



Note:

As per the figure, Set Operations — Minus and Intersect, you see that **Intersect** is symmetrical (it gives the same results regardless of the order of the two data sources) while **Minus** is not.



Set operation is based on fields requested at runtime

Input nodes

	Amount	Country	Name	Product
1	22	US	Hemingway	roses
2	21	G	Kafka	roses

	Amount	Country	Name	Product
1	20	US	Hemingway	diapers
2	40	US	Kafka	diapers

bothproducts

Intersect node

Mapping Calculated Columns (0)

Data Sources

- roses
 - Country
 - Name
 - Amount
 - Product
- diapers
 - Country
 - Name
 - Amount
 - Product

Output Columns

- Country
- Name

```
SELECT
  "Country",
  "Name"
FROM "Container::setExample"
GROUP BY
  "Country",
  "Name"
  
```

Country	Name
US	Hemingway

Kafka has bought diapers and roses in different countries and country is in query:
No match for Kafka

```
SELECT
  "Name"
FROM "Container::setExample"
GROUP BY
  "Name"
```

Name
Hemingway
Kafka

Country is not in the query and therefore not compared at runtime:
There is also a match for Kafka

Figure 49: Minus and Intersect – Detailed behavior

Filtering relies on the list of attributes that are queried at runtime. In other words, a column that is provided by both source nodes (for example, Country , but is not queried at runtime, is ignored.

Aggregation Node

The purpose of an Aggregation node is to apply aggregate functions to measures based on one or several attributes.



Note:

This function acts similarly to a GROUP BY clause in an SQL query, where you must specify the aggregate function for each measure. Here, the GROUP BY clause is not materialized as such, but takes each and every attribute selected for the output.

A graphical calculation view can support SUM, MIN, MAX, COUNT.

From version 1.0 SPS11, SAP HANA allows you to apply the following additional aggregate functions to the calculation views:

- Average
- Variance
- Standard deviation

In an aggregation node, a calculated column is always computed AFTER the aggregate functions. If you need to calculate a column BEFORE aggregating the data from this column, you have to define the calculated column in another node, for example a projection node, executed BEFORE the aggregation node in the calculation tree.

Unit 1: Information Views



Caution:

These aggregate functions should not be used in stacked scenarios.

Attribute Versus Measure

When adding a column to the output of an Aggregation node, you define whether the column is used for grouping (Add to Output) or if it must be combined with an aggregate function (Add as Aggregated Column). In the second case, you must specify the aggregate function to apply (Sum is applied by default).

Controlling the Behavior of the Aggregation Node

When you work with Aggregation nodes, the list of columns requested by the client query can influence the way the aggregation is executed, especially in complex scenarios.

The following features can help you control the aggregation of measures, in order to build more flexible models:

- Keep Flag
- Transparent Filter

Keep Flag

Let's consider a scenario where a data source contains the details of sales orders, by order ID. The only measures available are quantity and price.



Note:

Each order ID relates to one store, one customer, and only one product (in order to simplify the example).

You want to calculate the quantity and total sales for the product

Mouse and the month of

February.



Objective:
Show Mouse sales in February!

Order	Month	Product	Store	Customer	Quantity	Price	Results
1	2	Ipod	TigerDirect	John	20	20	
2	2	Mouse	TigerDirect	Susan	2	5	10
3	2	Mouse	TigerDirect	John	2	5	10
4	2	Mouse	Amazon	John	2	5	10
5	2	Headset	Amazon	Susan	3	5	
6	2	Headset	Ebay	John	2	5	
7	2	Ipad	Amazon	Susan	3	250	
8	2	Ipad	Ebay	Susan	3	250	
9	2	Mouse	Amazon	Peter	2	5	10
10	3	Ipad	Amazon	John	3	250	40



40

Output

- Columns
 - ORDER_ID: ORDERS.ORDER_ID
 - PRODUCT: ORDERS.PRODUCT
 - STORE: ORDERS.STORE
 - CUSTOMER: ORDERS.CUSTOMER
 - QUANTITY: ORDERS.QUANTITY
 - UNIT_PRICE: ORDERS.UNIT_PRICE

Properties

General	Value	ORDER_ID
Name	Mapping	"TRAINING".ORD...
Length	Scale	
Data Type	Filter	INTEGER
Keep Flag	True	
Transparent...	False	

Regular SQL

```
SQL Result
SELECT "Month", "Product",sum("Quantity"), sum("Price")
FROM "DEMO"."demo.modeling.db::cds.StoreOrders"
WHERE ("Month" = '2' AND "Product" = 'Mouse')
group by "Month", "Product"
```

Month	Product	SUM(Quantity)	SUM(Price)	Sales
1	2	Mouse	8	20 \$160

Solution:
Within the Calculation Model,
Set **Keep Flag = True** on **Order ID**

Figure 50: Keep Flag

Here, the issue is that the columns you request mandate a level of aggregation (Month , Product) that is generic. Hence, the total sales is calculated by multiplying the sum of quantities by the sum of unit prices.

Setting the **Keep Flag** property to true for the **Order ID** column forces the calculation to be triggered at the relevant level of granularity, even if this level is not the grouping level defined by the client query.

From SAP HANA 2.0 SPS02 onwards, the **Keep Flag** option can also be defined on the shared columns in the Star Join node of a CUBE with Star Join Calculation View (shared columns are the ones defined in the underlying DIMENSION calculation views)..

Transparent Filter

In some scenarios, using a filter (where clause) in a client query forces a column to be used in the **Group By** columns set.

In this scenario, for example, calculating the number of stores that sold mice to John or Susan triggers an intermediate calculation of the storecount sum by product and by customer, which makes the total by product irrelevant.

Unit 1: Information Views

**Objective:**

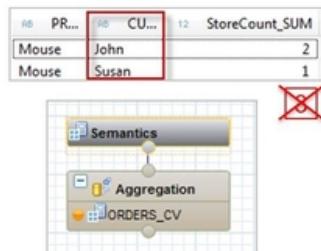
Count the amount of stores that sold a Mouse to John or Susan

```
select product, sum(quantity), sum(storecount) from model
where Customer in (john, susan) and product = mouse group by product
```

Order	Month	Product	Store	Customer	Quantity	Price
1	2	Ipod	TigerDirect	John	20	20
2	2	Mouse	TigerDirect	Susan	2	5
3	2	Mouse	TigerDirect	John	2	5
4	2	Mouse	Amazon	John	2	5
5	2	Headset	Amazon	Susan	3	5
6	2	Headset	Ebay	John	2	5
7	2	Ipad	Amazon	Susan	3	250
8	2	Ipad	Ebay	Susan	3	250
9	3	Ipad	Amazon	John	3	250
10	2	Mouse	Amazon	Peter	3	250

Results		
Product	Quantity	StoreCount
Mouse	6	2

Stacked Calculation Model



Solution:
Set the **Transparent Filter** property for the column **Customer** to **True**, on all models and nodes that reference the Customer

Figure 51: Transparent Filter

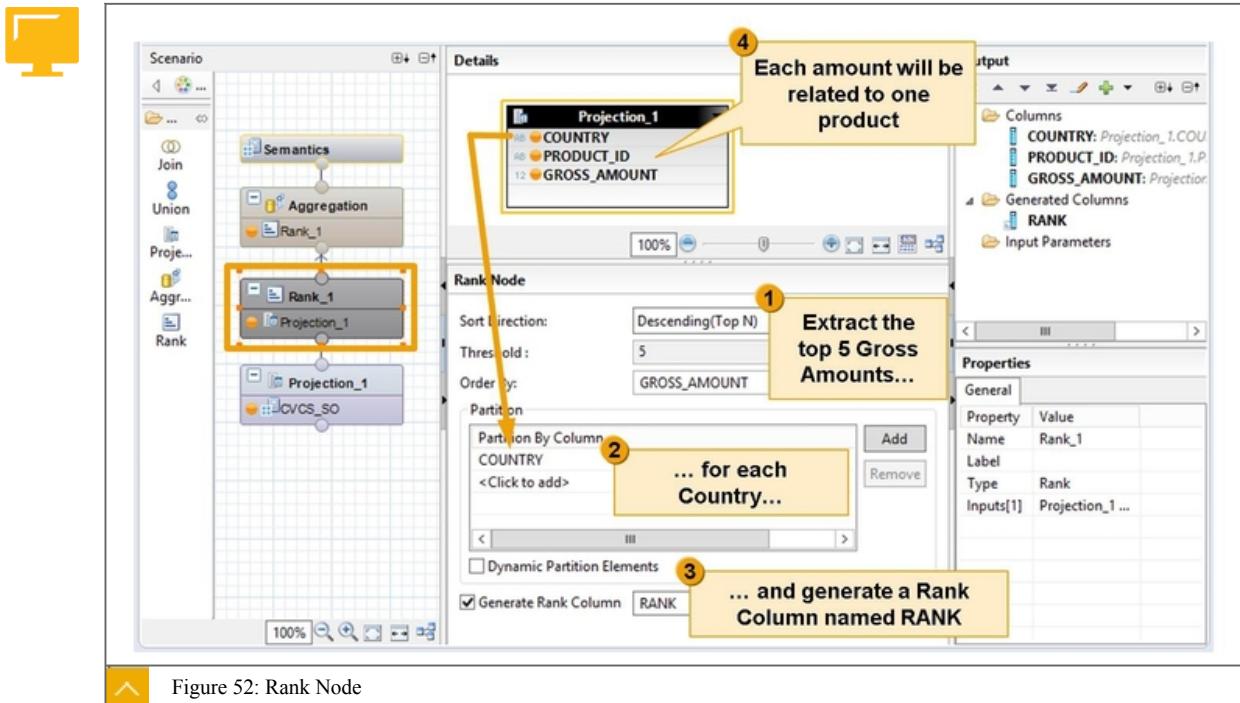
Setting the **Transparent Filter** property to true for all models and nodes that reference the Customer column, will stop the Customer column from being unnecessarily used in the By clause.

Group

This property is required in the following situations:

- When using stacked views where the lower views have distinct count measures
- When queries executed on the upper calculation view contain filters on columns that are not projected

Rank Node



The purpose of the Rank node is to enable the selection, within a data set, of the top or bottom 1, 2, ... n values for a defined measure, and to output these measures together with the corresponding attributes and, if needed, other measures.

For example, with a Rank node, you can easily build a subset of data that gives you the five products that generated the biggest revenues (considering the measure Gross Amount) for each country.

Main Settings of a Rank Node

The main settings of a Rank node are as follows:

Table 6: Main Settings of a Rank Node

Setting	Purpose
Sort Direction	Descending (Top N) or Ascending (Bottom N).
Threshold	Number of rows returned for each partition (N).
ORDER BY	The column that is used to order the data and extract the top or bottom N values.
Partition	One or several columns by which you want to provide the top or bottom N values. The result will give you N values for each partition “node”.
Dynamic Partition Elements	Define whether the partition can be adjusted automatically based on the columns that are selected by an upper node, an upper view, or query you run on top of the view.
Generate Rank Column	Whether you want to output a column with rank position (with values 1 to N).

Unit 1: Information Views

The top or bottom N values that are returned by the rank node depend totally on the attributes that you use in the node, project with an upper node, or select with the query you run on top of the view.

Considering the same example (for each region, select the top five products in terms of gross revenue), let's describe how the Rank node will behave if you modify the list of attributes in the output:

- If you remove the `PRODUCT_ID` attribute from the output:

The Rank node no longer behaves as a rank. It just sums up gross amount by country, and each row in the output has the rank 1.

- You add an attribute `BP_COMPANY_NAME` to the output:

Each amount is now related to a product and business partner pair. That is, for each country, you get the 5 biggest revenues generated by product and business partner in each country.

Assigning a Type to the Rank Column

Depending on how you want to use the ranking information, you can decide to assign to the rank column the type `Attribute` or `Measure`.

Rank Column: Measure or Attribute?



- Attribute

Assigning the type `Attribute` is a simple approach.

It is probably a bit less error-prone, because you are not tempted to perform an irrelevant aggregation of ranking positions. Removing the rank column from the top query will just apply the defined aggregate function — very often `SUM` — to the ranked values of each partition.

- Measure

With the type `Measure`, the rank column provides more flexibility.

For example, if you set the default aggregate function to `MAX`, you can retrieve summarized data, such as the total sales generated by the five biggest orders in each country, while keeping the information about how many orders are actually totalled in each country. Indeed, there might be countries that have received less than five orders over the considered period, and this information could be of interest when analyzing the data.

Specific Features to Enhance Flexibility of Calculation View Design

Because calculation views offer a lot of flexibility, in particular the possibility to have a large number of nodes in a view, SAP HANA Studio provides advanced functionality to manage nodes and semantics.



Feature	Purpose
Switching Node Types	Change a node type with another one without losing the link to the data source/lower and upper nodes
Replacing a Data Source	Replace a node source by another one without losing the output columns (throughout all the upper nodes), calculated columns, etc.
Extract Semantics	Apply to the Semantics of the Calculation view the semantics from an underlying node or data source.
Propagate to Semantics	Map columns to the output across all upper nodes, up to the top node of a calculation view.
Previewing the Output of Intermediate Nodes	To troubleshoot problems in view design, preview the data of an intermediate node rather than the whole Calculation View.
Map Input Parameters between Nodes	Map Input Parameters of a view to input parameters defined in underlying (source) views of other nodes.



Figure 53: Useful Features for Calculation View Definition

Switching Between Certain Types of Node

It is possible to convert a Projection node into an Aggregation node and the other way around without losing the reference to the data sources of the node.

Additionally, in the particular case of the upper node of a calculation view (that is, the last node of the calculation logic), you can switch between Projection, Aggregation, and Star Join nodes.

Replacing the Data Sources of a Node

Depending of the type of node and view configuration, you can replace the data source of a node (for example, a table, an information view, or another node) by another one.

This functionality is useful, for example, when you want to have the columns still propagated to all the upper nodes. You can also keep the existing calculated columns defined in this node working despite the change of data source.

When you do the change, a column mapping dialog box appears. It helps you assign the columns of the new data source to the output columns previously defined in the node.

Extracting Semantics

It is possible to extract the semantics from the underlying data sources of the nodes (especially when these data sources are information views with rich semantics) and to propagate them to the semantics of the calculation view.

Propagate to Semantics

This feature can be used in views with multiple stacked nodes. The purpose is to map one or several columns, already in the output of a node, to the output of all upper nodes and up to the Semantics, or in other words, the top node.

This is an alternative approach, faster than adding successively the same column(s) to the output of several stacked nodes.

To do this, select one or several columns in the output of a node. Then right-click the selection and choose Propagate to Semantics .

Unit 1: Information Views

Previewing the Output of Intermediate Nodes

To understand and fine-tune the different steps of a calculation view, each node can be previewed independently from the calculation view itself.

To achieve this preview, the view must have already been successfully built.

Map Input Parameters between Nodes

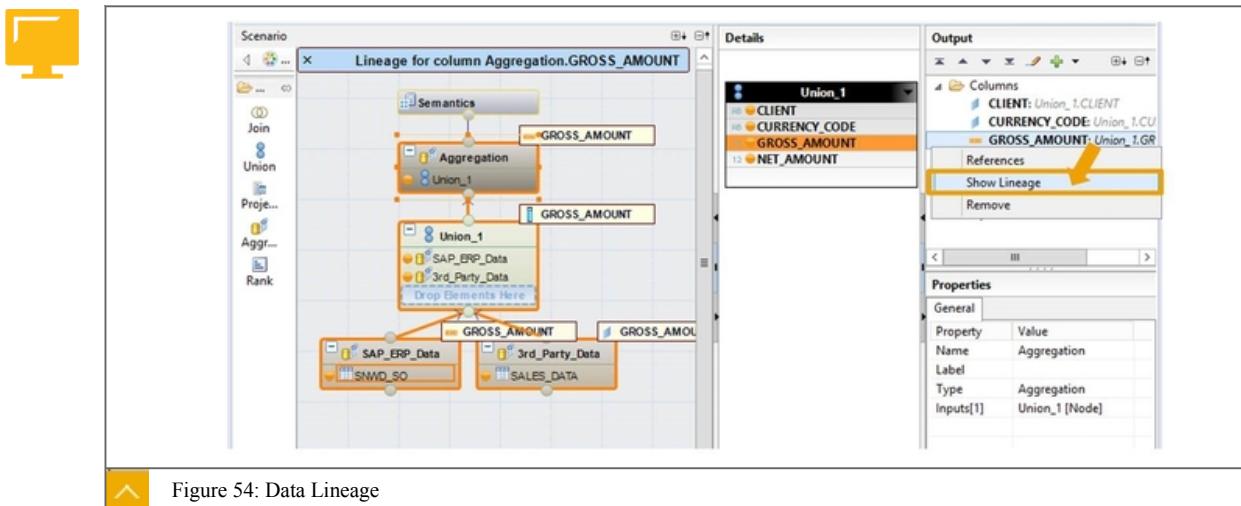
You can map input parameters (and also variables) defined in a calculation view to the input parameters/variables defined in the source views. You can do this to pass the parameter value itself, but also to pass the relevant list of values to the input parameter dialog box at runtime (when the view is executed).

Note:
In the Manage Mappings view, you can even copy and map an input parameter from a source view.
That is, you do not need to create a new input parameter in the calculation view (and define all its settings) before mapping it. This task can be done automatically in just one operation.

Copy and Paste Part of a Scenario

It is possible to copy and paste a node within the scenario pane of the same calculation view. If this node uses other nodes as its data sources, all the nodes below are also copied, together with the connections between them.

Data Lineage



The data lineage functionality enables you to track the origin of a column along the calculation scenario, and down to the first node where it appears. This node can be a bottom-level node of the view where the column is selected from a source, such as another information view, a column table, and so on. It can also be an intermediate node where the column is calculated.

Data lineage is a great feature for locating the origin of columns that might have been renamed during the flow.



LESSON SUMMARY

You should now be able to:

- Use measures in calculation views
- Explain the benefits of each type of node in calculation views
- Create and combine nodes in calculation views
- Use the features of calculation views to enhance the flexibility of this type of view

Unit 1

Learning Assessment

- When can you use the referential join?

Choose the correct answer.

- A** When you have to report on facts with matching dimensions
- B** When you have to report on all posted facts
- C** When referential integrity is assured
- D** When you have to report on all dimensions to confirm if there are matching facts or not.

- A text join, which acts as a right outer join, is used to join a text table to a master data table.

Determine whether this statement is true or false.

- True
- False

- Identify the join that checks the From and To column on the left table.

Choose the correct answer.

- A** Left Outer Join
- B** Right Outer Join
- C** Temporal Join
- D** Full Outer Join
- E** Text Join

4. What type of node is recommended to combine two data sources that have a similar set of dimensions?

Choose the correct answer.

- A** Join
- B** Union
- C** Star join

5. The columns labels you define in a dimension calculation view are used internally and are not exposed to client tools.

Determine whether this statement is true or false.

- True
- False

6. When do you use the hidden attribute in a dimension calculation view?

Choose the correct answer.

- A** When you want to hide an attribute that is not required for client consumption.
- B** When you want to hide the attribute in your cube calculation view.
- C** When you need to include the attribute in your calculation view to define a join.

7. Identify the features and benefits of a calculation view.

Choose the correct answers.

- A** It can be defined as a graphical view or a scripted view.
- B** It is not possible to join several fact tables in a calculation view.
- C** It supports you to include more advanced calculations.
- D** It can contain calculated attributes.

8. What is the function of the Aggregation node in a graphical calculation view?

Choose the correct answer.

- A** Map the sources to the target.
- B** Control the aggregation.
- C** Combine the result sets.

Unit 1

Learning Assessment - Answers

- When can you use the referential join?

Choose the correct answer.

- A** When you have to report on facts with matching dimensions
- B** When you have to report on all posted facts
- C** When referential integrity is assured
- D** When you have to report on all dimensions to confirm if there are matching facts or not.

- A text join, which acts as a right outer join, is used to join a text table to a master data table.

Determine whether this statement is true or false.

- True
- False

- Identify the join that checks the From and To column on the left table.

Choose the correct answer.

- A** Left Outer Join
- B** Right Outer Join
- C** Temporal Join
- D** Full Outer Join
- E** Text Join

Correct. The temporal join is a special join that uses date information from the left table in order to join the relevant data from the right table.

4. What type of node is recommended to combine two data sources that have a similar set of dimensions?

Choose the correct answer.

- A** Join
 B Union
 C Star join

Correct. To combine two data sources with similar data structures, a union node will perform better than a join node.

5. The columns labels you define in a dimension calculation view are used internally and are not exposed to client tools.

Determine whether this statement is true or false.

- True
 False

6. When do you use the hidden attribute in a dimension calculation view?

Choose the correct answer.

- A** When you want to hide an attribute that is not required for client consumption.
 B When you want to hide the attribute in your cube calculation view.
 C When you need to include the attribute in your calculation view to define a join.

7. Identify the features and benefits of a calculation view.

Choose the correct answers.

- A** It can be defined as a graphical view or a scripted view.
 B It is not possible to join several fact tables in a calculation view.
 C It supports you to include more advanced calculations.
 D It can contain calculated attributes.

Unit 1: Learning Assessment - Answers

8. What is the function of the Aggregation node in a graphical calculation view?

Choose the correct answer.

- A** Map the sources to the target.
- B** Control the aggregation.
- C** Combine the result sets.

UNIT 2

Modeling Functions

Lesson 1

Creating Restricted and Calculated Columns

71

Lesson 2

Filtering Data

82

Lesson 3

Using Variables and Input Parameters

89

Lesson 4

Using Hierarchies

104

Lesson 5

Implementing Currency Conversion

114

UNIT OBJECTIVES

- Create restricted and calculated columns
- Filter data
- Create client dependent views
- Restrict data when modeling using domain fix values
- Use variables and input parameters
- Create variables and use them to filter data
- Create input parameters
- Use hierarchies
- Create parent-child hierarchies
- Work with time-based hierarchies
- Explain the general principles of currency conversion

Unit 2: Modeling Functions

- Apply currency conversion in Calculation Views

Unit 2

Lesson 1

Creating Restricted and Calculated Columns

LESSON OVERVIEW

This lesson explains how to use different modeling capabilities on columns. In particular, you will learn how to create calculated columns (attributes or measures) and restricted columns (for attribute only).

Business Example

You want to ensure that you minimize the processing in the client tool, pushing down expressions to SAP HANA using calculated and restricted columns.



LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Create restricted and calculated columns

Restricted Columns



What is a restricted column?

- As the name implies it is a column that does not give the complete picture of a column, it is restricted to a **subset** of the original column.
- The benefit of a restricted column is that it expands the modeling options in a view, giving the modeler the possibilities of creating objects that can be easily reported on or reused.

Country	Month	Amount	US Amount
DE	2010-01	12.345,00	0
DE	2010-02	15.678,00	0
DE	2010-03	25.814,00	0
DE	2010-04	21.586,00	0
DE	2010-05	21.861,00	0
DE	2010-06	11.258,00	0
DE	2010-07	12.387,00	0
DE	2010-08	13.589,00	0
DE	2010-09	12.345,00	0
DE	2010-10	15.678,00	0
DE	2010-11	25.814,00	0
DE	2010-12	21.586,00	0
US	2010-01	21.861,00	21.861,00
US	2010-02	11.258,00	11.258,00
US	2010-03	12.387,00	12.387,00
US	2010-04	13.589,00	13.589,00
US	2010-05	12.345,00	12.345,00
US	2010-06	15.678,00	15.678,00
US	2010-07	25.814,00	25.814,00
US	2010-08	21.586,00	21.586,00
US	2010-09	21.861,00	21.861,00

Restricted column based on **Amount**, where **Country = US**

Figure 55: The Benefits of Restricted Columns

Unit 2: Modeling Functions

The restricted column is one of a set of columns available in SAP HANA models, which includes the following:

- Columns
- Calculated columns
- Restricted columns

The restricted column is restricted based on one or more attributes. These columns can be anything in the base table or view that the modeler defines to help reporting or further modeling.



Note:

The restriction cannot be based on a column defined as a **Measure** in the semantics.

Example Without Using a Restricted Column



- You have a transactional table with cost data items, with each cost type split on a different line.
- If you want to find out the shipping cost you could create an analytic view with Cost Type as an attribute, and Amount as a measure.
- You could then restrict your report by reporting on Cost Type, setting the column filtered on Cost Type = "Shipping Cost" in the reporting tool.

Cost Type	Amount
Purchasing Price	€ 1 200
Shipping Cost	€ 80
VAT	€ 346
Processing Cost	€ 150
Margin	€ 300



Figure 56: Example Without Using a Restricted Column

With this data, you could restrict your report by filtering by **Cost Type = Shipping Cost**. If your reporting tool is SAP Business Objects, you could create a data provider with a query restriction where **Cost Type** is filtered to only display **Shipping Cost**.

Creating Restricted Columns



- A restricted column can be created in an aggregation node of a calculation view.
- You assign a measure to the restricted column, and also which columns define the restriction. For example, we can create restricted column based on Gross Amount restricted to certain product categories.

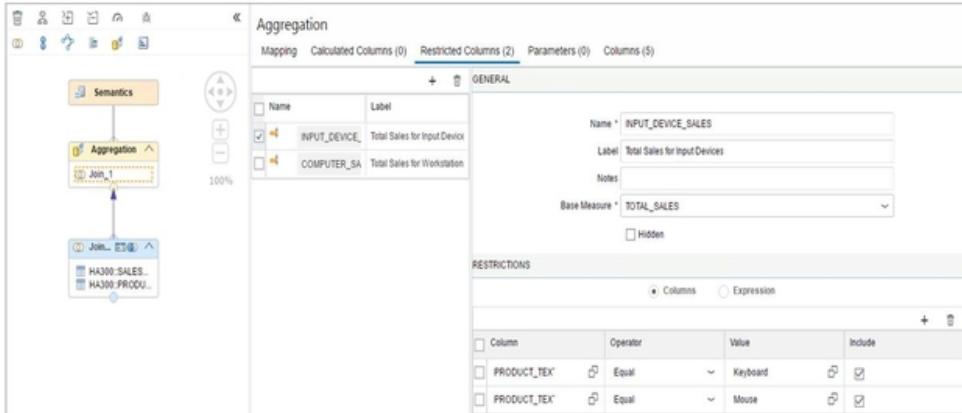


Figure 57: Creating Restricted Columns

Setting the Restriction

Continuing with the example of using SAP Business Objects for reporting, when you have access to this restricted column, you can report on both the total gross amount and the gross amount for flat screens, in the same data provider or query.



- The restrictions for a restricted column do not have to be limited to one single Column: you can filter based on multiple columns depending on your reporting requirements.
- Multiple operators can be used for the filter restrictions, such as Equal, Greater Than, Less Than and others.
- You have the option to hide the restricted column, for example to reuse it in a calculated column and thereby make it unavailable for reporting.

The screenshot shows the 'RESTRICTIONS' tab of the Aggregation dialog. The 'Column' dropdown is set to 'PRODUCT_TEX'. The 'Operator' dropdown shows 'Equal' selected. The 'Value' dropdown shows 'Keyboard'. The 'Include' checkbox is checked. Below this, other operators like 'Between', 'Greater', 'GreaterEqual', 'Less', 'LessEqual', and 'LessThan' are listed.

Figure 58: Setting the Restriction

Unit 2: Modeling Functions

If there are different lines in the restriction, all the lines defined on the same column are combined with the logical operator **OR**, and then, all the sets of restrictions for different columns are combined with the logical operator **AND**.



Note:
This is the case regardless of the order in which you define the lines.

As of SAP HANA 1.0 SPS09, in calculation views, it is possible to create restricted columns with the **Between** operator, and using input parameters.

Displaying and Editing Restriction Expressions



- Several restrictions on the same column are combined with an **OR** operator.
- Restrictions on different columns are combined with an **AND** operator.
- To visualize the expression, select the **Columns** tab.
- You can also edit the expression to enrich it.
In this case, the column-based definition can no longer be edited.



Figure 59: Displaying and Editing Restriction Expressions

You can visualize the expression corresponding to your restrictions. This can be useful for example when you want to check the precedence of logical operators.

It is also possible to modify this expression, especially in complex scenarios when the features offered in the **Column** tab do not fulfill your requirements.

As of SPS11 onwards, the expression can be written using SQL, in addition to the Column Engine syntax. However, for expressions in SQL, SAP HANA modeling supports only a limited list of SQL functions.



Note:

The general SQL support within expressions is not specific to restricted measure expressions. SQL can also be used in the following expressions:

- Calculated columns (since SAP HANA SPS10)
- Filters (new in SPS11)
- Default values for variables and input parameters (new in SPS11)

You will learn about these different types of modeling functions later.

Calculated Columns



- In a data model sometimes not all measures available in the base data will give your users sufficient information for reporting if you just provide the base measures in your views.
- In order to enhance the data, SAP HANA provides Calculated Columns where the modeler is able to include calculations already within the information model in order to help reporting or further modeling.

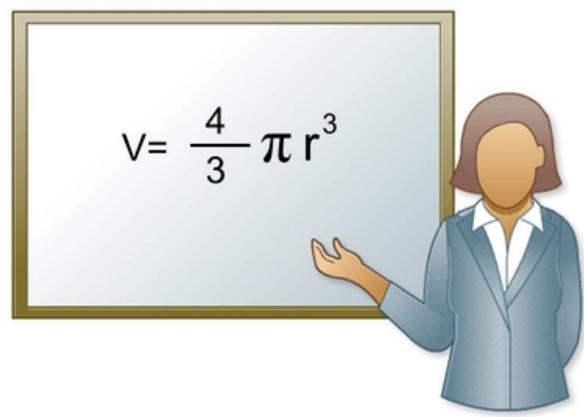


Figure 60: When to Use Calculated Columns

A calculated column does not have to be a complex formula, it can also be a simple calculation.

Unit 2: Modeling Functions

When to Use Calculated Columns



- When you include calculations in your views using calculated columns you take advantage of the speed of SAP HANA letting the database engine perform calculations instead of doing these calculations in your end client reporting tool.
- Having ready defined calculations in information models can also help simplifying reporting by unifying calculations.
- **HINT:** Only create calculated columns where there is a specific reporting need, as data transfer will take place between the OLAP engine and Calculation engine slowing down execution.

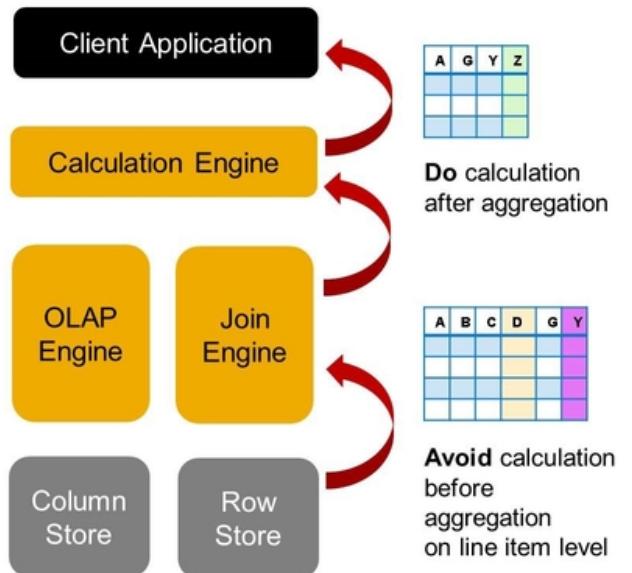


Figure 61: When to Use Calculated Columns

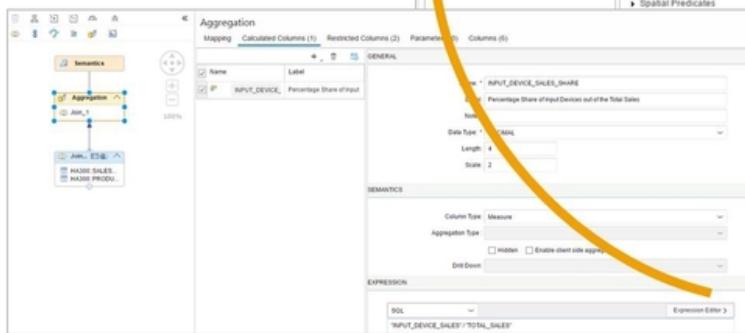
The Calculated Columns Wizard

When creating a calculated column, the first step is to choose the column type, measure, or attribute.



- A calculated column is defined within an information model and can use the string functions, mathematical functions etc. available in the editor.

- You can define a calculated column as measure or attribute.



- Double-click or drag-and-drop Elements, Operators and Functions to build the expression
- When you type, use Ctrl+Space to autocomplete the name of functions, columns, input parameters.

Figure 62: Calculated Columns Wizard

Consider Granularity When Creating Calculated Columns



Product	Units	Price	Total Sales (Units * Price)
Keyboards	100	€ 40	€ 4 000
LCD Screens	50	€ 300	€ 15 000
Network Switches	75	€ 90	€ 6 750
VOIP Telephones	200	€ 65	€ 13 000
Servers	30	€ 2 000	€ 60 000
SUM:		455	€ 1 135 225



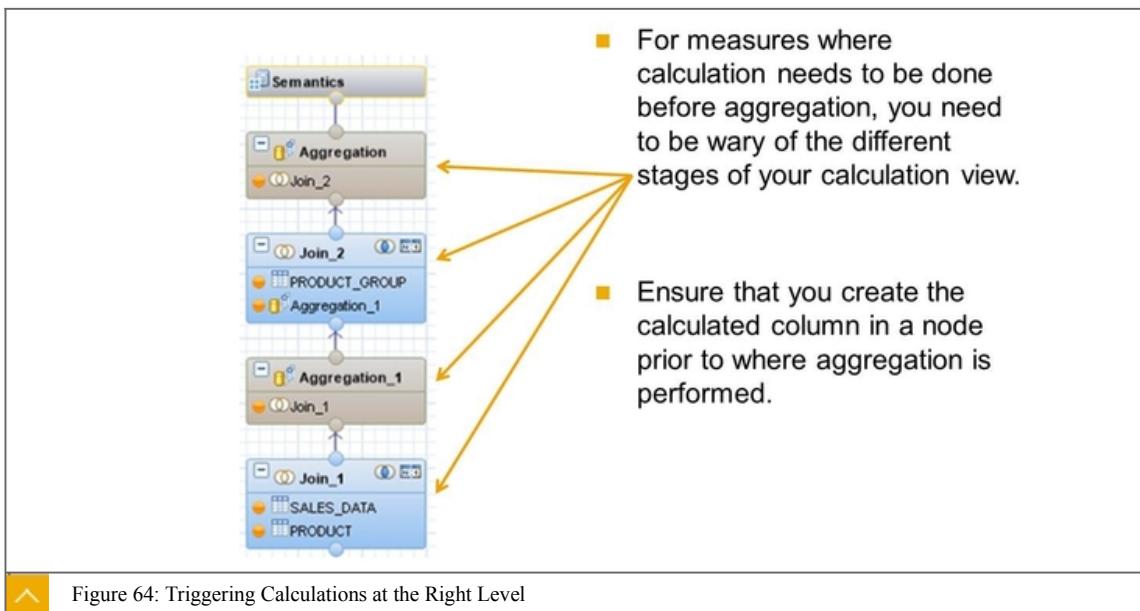
- For certain measure it is not possible to perform the calculations when the measures are already aggregated. The aggregated granularity of, for example, Price does not mean anything.

Figure 63: Consider Granularity When Creating Calculated Columns

In the figure, Consider Granularity when Creating Calculated Columns, in the sum line highlighted in red, the units have been aggregated as well as the price. Multiplying these to aggregates will not give a meaningful result.

Unit 2: Modeling Functions

Triggering Calculations at the Right Level



By analyzing your reporting requirements, you can arrive at a decision at which stage the calculation should be performed.



Caution:

Try to minimize calculations before aggregation; for example, when calculations include multiplication or division.

If the calculations are just additions or subtractions, it is not required. It will also slow down the execution of the view.

Calculation Before Aggregation



Product	Units	Price	Total Sales (Units * Price)
Keyboards	100	€ 40	€ 4 000
LCD Screens	50	€ 300	€ 15 000
Network Switches	75	€ 90	€ 6 750
VOIP Telephones	200	€ 65	€ 13 000
Servers	30	€ 2 000	€ 60 000
		SUM:	€ 98 750

■ This way you can be sure that you end up with a correct sum as the calculations are performed on the correct granular level.

Figure 65: Calculation Before Aggregation

When data is calculated at the correct level of granularity, the total sales measure is correctly calculated; that is, 98,750.

Calculated Columns and Persistence

Creating calculated columns in an SAP HANA information view does not mean the calculation is persisted, it is simply projected through the generated column view.

The calculated column is available only during the runtime of the model and can be displayed in a report, or consumed by another view.

By contrast, if you need to keep the result of a calculation over time, you can apply one of the following methods:

Storing the Value of a Calculated Column

- Create an ad-hoc SQL artifact that updates and inserts the calculated column into a table.

You can either use a write-enabled procedure or a table function.

- Use the transformation and calculation features provided by an ETL tool, such as SAP Landscape Transformation (SLT) or SAP Data Services.

You can pre-calculate columns during the data provisioning phase.

Calculated Columns in Analytic Views

Calculating measures before aggregation in analytic views works in a way that is different from calculation views.

Indeed, as you have learned in the lesson, Understanding Deprecated Graphical Views , the scenario of analytic views is made up of only three nodes: Data Foundation, Star Join, and Semantics, and calculated measures are defined in the Star Join node, not in the Data Foundation.



Figure 66: Calculate Before Aggregation Option in Analytic Views

Unit 2: Modeling Functions

Because there is only one place to define calculated measures, analytic views provide an additional option in the Calculated Column wizard.

Expression Language and Validation

As of SAP HANA SPS10 onwards, you can specify which of the two following languages the expression of a calculated column uses.

Calculated Columns: Explicit Language Definition



- SQL

The expression only uses plain SQL.

- Column Engine

The expression uses any valid SQLScript expression.

Validating an Expression Against a Specific Language: Key Benefits



- It helps to validate the syntax of calculated columns according to the specified language.

For example, an expression with an IF condition (supported only with SQLScript) will not validate if it is defined as SQL.

- In turn, it helps the modeler to optimize the calculation execution.

Indeed, plain SQL expressions enable a better optimization process, compared to SQLScript. So, by validating the expression against SQL, you can make sure it will be fully optimized.



Note:

SAP HANA SPS11 has introduced the same distinction for filters, restricted columns, and default value expressions for variables and input parameters.

Client Side Aggregation

If you want to create a calculated measure and enable client side aggregation for the calculated measure, select the Enable client side aggregation checkbox.

This allows you to propose the aggregation that client needs to perform on calculated measures.



Figure 67: Client Side Aggregation

Consider an example where you have created the measure MAXIMUM_GROSS_AMOUNT, giving you the maximum value of the gross amount. By selecting the Enable client side aggregation checkbox, you propose to the reporting client to also apply a maximum aggregation on the client side as defined in the Client Aggregation drop-down list.



LESSON SUMMARY

You should now be able to:

- Create restricted and calculated columns

Unit 2

Lesson 2

Filtering Data

LESSON OVERVIEW

Most of the time, users do not need the whole data perimeter stored in the source system. In these instances, filters can be used to display only relevant data. Moreover, by decreasing data volume, performance can be improved.

Business Example

You notice that a large amount of data stored in base tables are never used for reporting, because users decide to apply WHERE clause filters in reporting tools.

In order to speed up query execution you decide to introduce filters into information models.

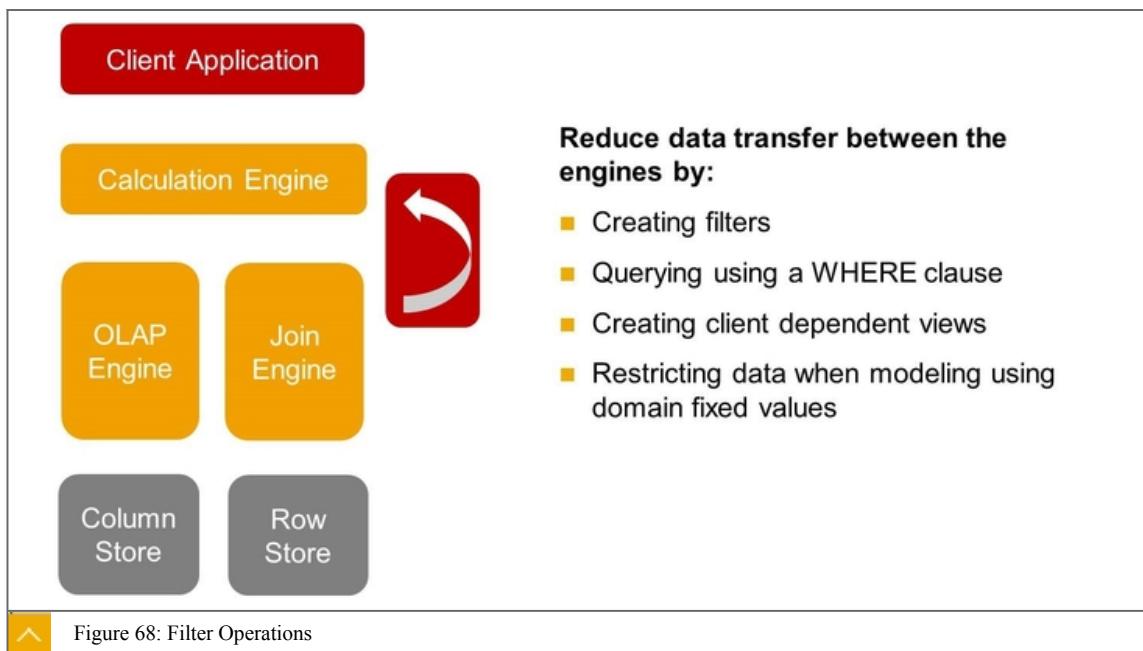


LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Filter data
- Create client dependent views
- Restrict data when modeling using domain fix values

Using Filter Operations



A primary goal during modeling is to minimize data transfers between engines. This statement holds true both internally, that is, in the database between engines, but also between SAP HANA and the end user client application. For example, an end user will never

need to display a million rows of data. Such a large amount of information just cannot be consumed in a meaningful way.

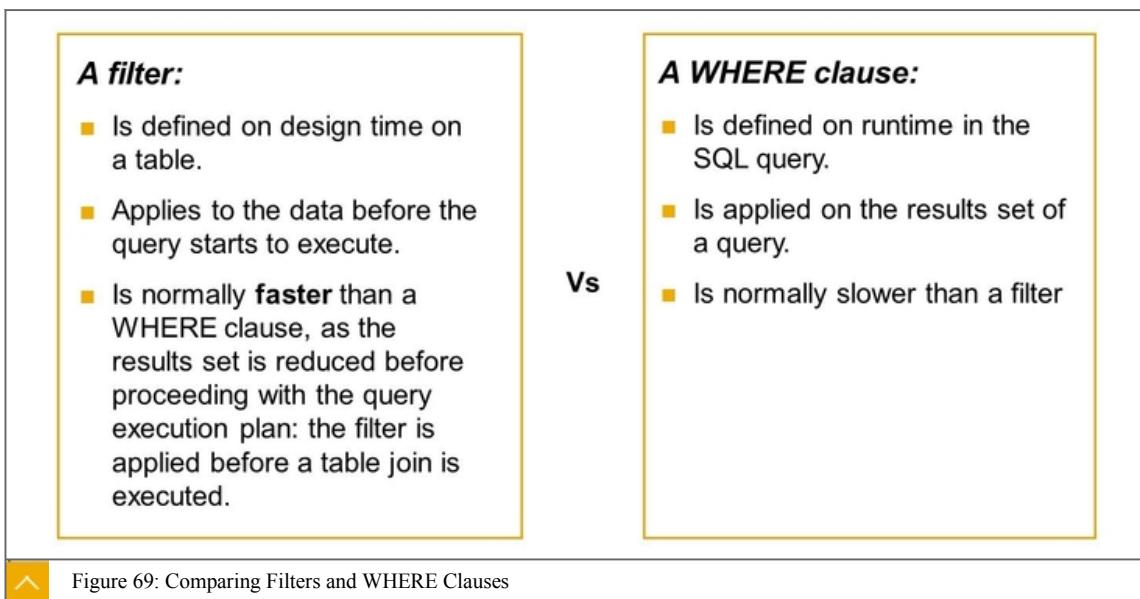
Whenever possible, data should be aggregated and filtered to a manageable size before it leaves the data layer.

When deciding which records should be reported upon, a best practice is to think at a set level, not a record level.

A set of data can be aggregated by a region, a date, or some other group, to minimize the amount of data passed between views.

Comparing Filters and WHERE Clauses

Both WHERE clauses and filters are used to reduce the result set of data. However, from a semantic perspective, these two operations are not the same.



Client-Dependent Views

The CLIENT (sometimes also called SAP Client to avoid any confusion with the concept of Customer) is a general concept in SAP Systems such as SAP Business Suite and SAP Business Warehouse. The main purpose is to isolate different types of data (for example, development versus test data) based on a specific column (generally named **CLIENT** or **MANDT**, which stands for **Mandant** (a German word for **Client**)). The values in this column are three-digit numbers, such as 001, 200, 800.

Almost all tables in the SAP Business Suite data model are client-dependent. Only a small number of tables do not have the client as the first primary key field.

SAP HANA handles the SAP Client column in order to enable client filtering when you work on SAP Applications such as the SAP ERP.

In the SAP Web IDE for SAP HANA, you have to define explicitly in the properties of a source table which column holds the SAP Client information.

Unit 2: Modeling Functions



Note:

In SAP HANA Studio, the client columns handling is different. In particular, a column named CLIENT or MANDT is automatically considered as an SAP Client column, and filtering settings can be defined for each Calculation View without the need to specify explicitly which column holds the SAP Client info in each data source.

Whenever you create SAP HANA models on SAP data sources that include a Client column, we recommend that you always define joins on this column. Otherwise, your view might return inconsistent results in case there is data for several clients. For example, you might join data for clients 200 and 800 (cartesian product) in a join, which in general would not make sense.

You should also pay attention to aggregations when the SAP Client is not part of the requested columns, because this might result in an irrelevant aggregation.

Defining Client-Specific Filtering in Calculation Views

Client-specific filtering makes sense only when some (or all) of the source tables have an SAP Client column.



1 Define the Client Column in the properties of the table

The screenshot shows the SAP HANA Studio interface. A callout points to the 'Properties' dialog for a table node named 'HA300::CITIES'. The 'Client Column' property is set to 'MANDT'.

2 Set the Default Client property to Session Client

The screenshot shows the 'Semantics' tab of a calculation view node. The 'Default Client' dropdown is set to 'Session Client'.

3 In the Security (e.g. SAP HANA Studio), assign a Session Client to the user.

The screenshot shows the SAP HANA Studio security configuration dialog. The 'Session Client' dropdown is set to '800'.

When the user runs a query based on this model, the user's Session Client (in this example: 800) is used to filter the source data.

By default, a Calculation View defined in the SAP Web IDE returns all the values, regardless of the SAP Client value. To implement client-based filtering, you proceed as follows:

- Define, for all relevant data source tables, the SAP client column.

This is done in the Mapping tab of the node where the source table is referenced, with the Client Column property.

- Define how you want to retrieve data.

This is done in the **Semantics** node of the Calculation View. You choose one of the three following options for the **Default Client** setting:

- **Cross-Client** : All the data is retrieved regardless of the client number
- **Fixed Client** : You specify an SAP Client number (for example, 200) and the data sources are automatically filtered to include only the rows with this client number
- **Session Client** : The source tables are filtered dynamically based on a client value that is specified for each user in the **USERS** table of SAP HANA.

The **Session Client** is the default setting applied to a new view.

Let's consider a simple table that contains data for two different clients, 200 and 800. This information is stored in a column named **MANDT**. You create and build a calculation view based on this table, and then query the data with your user. The following table shows what data is retrieved depending on both the default client setting for the Calculation View and whether or not the **MANDT** column has been defined as the (SAP) Client Column in the view properties.

Table 7: Default Client Setting Effect

Default Client Setting	Effect when NO Client Column is specified	Effect when MANDT is defined as the Client Column
Cross-Client	All rows	All rows
Fixed Client: 200	All rows	Only rows for which MANDT=200
Session Client (use case 1: the user has no default client assigned)	All rows	No rows at all
Session Client (use case 2: the user has default client 800)	All rows	Only rows for which MANDT=800

Important Note Regarding the Technical User in the SAP Web IDE

In the SAP Web IDE for SAP HANA, you connect with a regular XS Advanced user, but most of the tasks you perform with defining a Calculation View and previewing its data are executed on your behalf by a technical user (you will learn about this technical user in the Unit about Security).

Because the technical user has NO default client assigned, if you specify a **Column Client** property for a Calculation View and set the **Default Client** to **Session Client**, the data preview in the Developer perspective will not retrieve any data.

There are two alternative approaches to by-pass the technical user:

- You can query the data from an external tool with your regular user (for example, STUDENT##)
- You can display the column view data in the **Database Explorer** perspective from a connection to the classic database (for example, in the training environment, H00 DB). Indeed, a query from the classic database connection does not involve the technical user.

Unit 2: Modeling Functions

This can be done either by opening the column view or by executing an SQL query on top of it.

Session-Dependent Functions

To dynamically select table data in calculation views, you can use the following functions. These functions return client or locale information based for the connected user.

Session-Dependent Functions



- SESSION_CONTEXT('CLIENT')

Returns the client's value based on the current user profile.

- SESSION_CONTEXT('LOCALE')

Returns the session's language in POSIX format (set by 'locale' parameter of JDBC/ODBC/ODBO connection, for example, en_en, de_de, de_at).

- SESSION_CONTEXT('LOCALE_SAP')

Returns the session's language following the SAP internal format (like the SPRAS column in the text tables of Master Data).

Example

```
SELECT field1, field2 FROM my_table WHERE mandt =
SESSION_CONTEXT('CLIENT') AND spras = SESSION_CONTEXT('LOCALE_SAP')
```

It is also possible to refer to the system variable `$$client$$` in a filter expression of a Calculation View. This variable picks up the client value stored in the user ID.

Filtering Using Domain Fixed Values

Domains and domain fixed values are data concepts that originate in the SAP ABAP data dictionary (DDIC). They can be exploited in SAP HANA modeling to synchronize the filter values between SAP source applications and SAP HANA calculation views.

What Are Domain Fixed Values?

Before you can define domain fixed values, you first must define a **domain**. A domain is a highly reusable, primitive data entity such as a delivery status code or an employment status or person's gender. A domain can also have fixed values assigned to it and these values represent the **only possible values that are valid** in that domain. For example, employment status could have the fixed domain values FULL-TIME, PART-TIME, RETIRED. Although it is optional to supply a list of fixed domain values, doing this is a very popular approach to removing the need for each ABAP program to validate the field value (perhaps from a user's input) each time it is used. Imagine developers having to write the same lookup routine to check the values are valid each time the domain is used in all their programs? Essentially, using domain fixed values means that you have 'pre-programmed' the domain with its allowed values and you can be sure that only valid values will then be processed. An error is raised if an invalid value is used. Developers can simply use the domain knowing that only valid entries will be allowed and they don't have to worry about validating the entries in their program code.

Another benefit of using fixed values with domains is that whenever a domain value changes, for example, a new employment status is required, we simply add the new value to the domain list. Immediately, all references to this domain are then validated against the new value. We don't have to adjust every table and program that uses the domain. Therefore we have centralized the valid list maintenance and this ensures consistency.

Domain fixed values are stored in ABAP tables. The tables are DD07 for the values and DD07T for the language descriptions of the values.



Note:

Use transaction SE16 in the source ABAP system to access the domain tables and browse their contents.

Modeling in SAP HANA Using Domain Fixed Values

Often the calculation views we create in SAP HANA are based on data sources from SAP ABAP sources. If we copy the fixed domain value tables to SAP HANA then we can use these to filter the data to ‘allowed’ values. Therefore we synchronize the filters between SAP ABAP sources and SAP HANA.

Ideally, to ensure complete synchronization you should replicate the domain fixed value tables to SAP HANA. For this you could use SAP SLT, but it really doesn’t matter how you get the tables to SAP HANA.

If you are using SAP SLT then the domain fixed value tables are replicated to the SAP HANA schema that aligns to the name of the SLT configuration.

What you should do is to create a dimension calculation view for each domain you will use. In each view include only the tables DD07 and DD07T. Apply a filter based on the domain name you require. Use a text join to ensure that you include the language description for the domain values (if you need them to be part of the overall output). The text join is based on the language field DDLANGUAGE. In the output, include the field DOMVALUE_L as key field (the domain values) and also the field DDTEXT (the language description of the domain values).

Finally, consume the dimension calculation view in a join node of your main calculation view (probably with an inner join) to restrict your data source to the values that are available in the domain fixed values. Then you have full synchronisation between the source ABAP system and the SAP HANA calculation views without having to manually define and maintain the filter values.

Unit 2: Modeling Functions

Restricting Domain Fix Values Using Text Join



- In order to restrict data, you define a text join and a cardinality 1..1.
- Language Column is defined as DDLANGUAGE column.

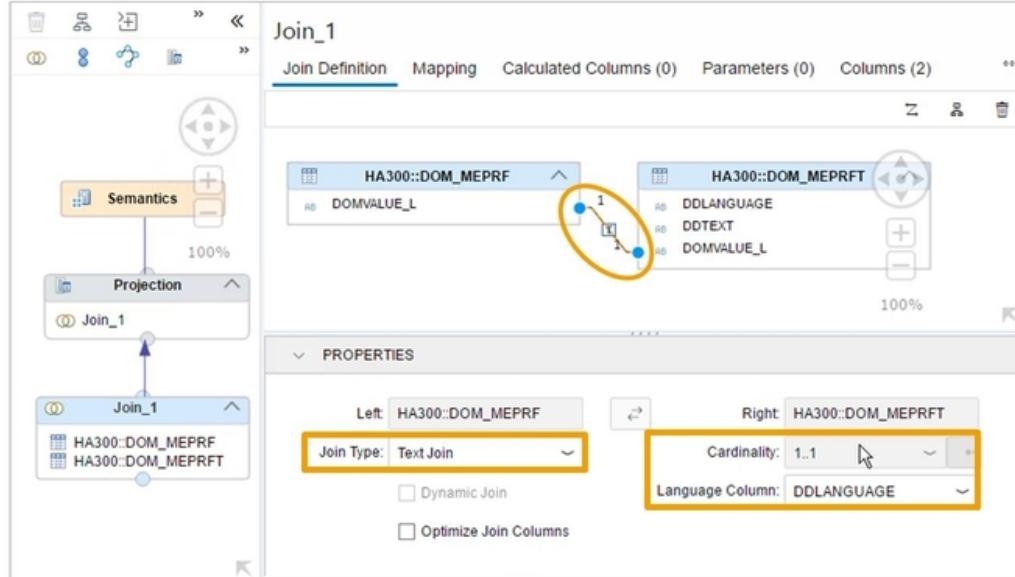


Figure 71: Restricting Domain Fix Values Using Text Join

In the diagram you will notice that the view is not using the domain tables DD07 and DD07T. That is because SAP also supplies SQLScript to extract domain values and texts from the large DD07 and DD07T tables and move the values to individual domain-specific tables. The domain-specific tables use the naming conventions DOM_<Domain Name> for the domain values and DOM_<Domain Name>T for the language-dependent texts. Examples include DOM_MEPRF and DOM_MEPRFT, and DOM_BFART and DOM_BFARTT. The SQLscript is available in the attachment to SAP Note 739432. This approach can be helpful in isolating the domain values to individual tables so that modelers do not need to then know the name of each domain to use in their own filters. They simply refer to the domain-specific table which contains only the values they need.

You do not have to follow this approach; you can use DD07 and DD07T tables directly.



LESSON SUMMARY

You should now be able to:

- Filter data
- Create client dependent views
- Restrict data when modeling using domain fix values

Unit 2

Lesson 3

Using Variables and Input Parameters

LESSON OVERVIEW

This lesson explains how you can customize your information views by creating variables and input parameters.

These artifacts will result in dedicated dialog boxes that will appear when the view is executed, requesting the end user to pass parameters to the query to modify the result set or trigger parameter-based calculations.

These artifacts provide the flexibility to use the same information view with parameters entered at runtime, instead of hard-coding such parameters in different views during the design. It also helps restrict the set of data that is transferred between SAP HANA and the reporting tools.

You can also compute calculated columns based on a parameter entered by the user at runtime instead of hard-coding it in the information view itself.

Business Example

You want to build a view to report sales by product, but only for the country (or region) the user specifies when executing the view.

Additionally, you would like to create in your information models calculations based on user-defined parameters.



LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Use variables and input parameters
- Create variables and use them to filter data
- Create input parameters

Variables and Input Parameters

Variables and input parameters enable the modeler to create more dynamic information models. The end user consuming the data will be prompted to provide attribute values or parameters, either to reduce the result set of a view, or to trigger calculations based on the provided parameters.

Difference Between Variables and Input Parameters

Variables and Input parameters in SAP HANA can be described as types of input boxes.

It is a way of asking the reporting user for input on how to provide, present, or restrict the data that will be displayed.

Unit 2: Modeling Functions



Note:

For course participants familiar with SAP BusinessObjects terminology, the variable or input parameter is often referred to as a **prompt** in Business Object tools.

In SAP Business Warehouse, they are referred to as **variables**.

Difference Between Variables and Input Parameters

**Variables**

- Variables are used to filter the content of an information model. They do not impact the execution workflow of the information model and are applied to a query to filter out some values.
- Variables can be used to filter attributes. As an example, a variable can be used to filter a result to a specific country or a specific customer.
- When filtering using variables, a WHERE clause is added to the SQL query.

Input parameters

- Calculations performed by a model can use input parameters as input values
- Input parameters can be defined as compulsory for the evaluation of an information model.
- In SQL, input parameter values are passed via the PLACEHOLDER reserved word.



Figure 72: Variables Compared to Input Parameters

Variables have a **single** purpose: to pass dynamic values for filtering. Variables are excellent at filtering and there are many options that make implementing variables easy and flexible.

Variables are passed using a WHERE clause which is easily understood by most applications that call calculation views.

Input parameters are **multi-purpose** and can be used in a variety of ways, for example as placeholders in expressions (calculated column, restricted column, filter) and also to pass values to table functions. Input parameters are passed using an SQLScript **PLACEHOLDER** keyword. Not all applications are able to pass values to the PLACEHOLDER. Once you have defined an input parameter, you must figure out how to use it in an expression; otherwise it is ignored. In contrast, once you define a variable, it is used immediately to filter its assigned attribute. You do not have to figure out how to implement it.

Impact of Variables and Input Parameters

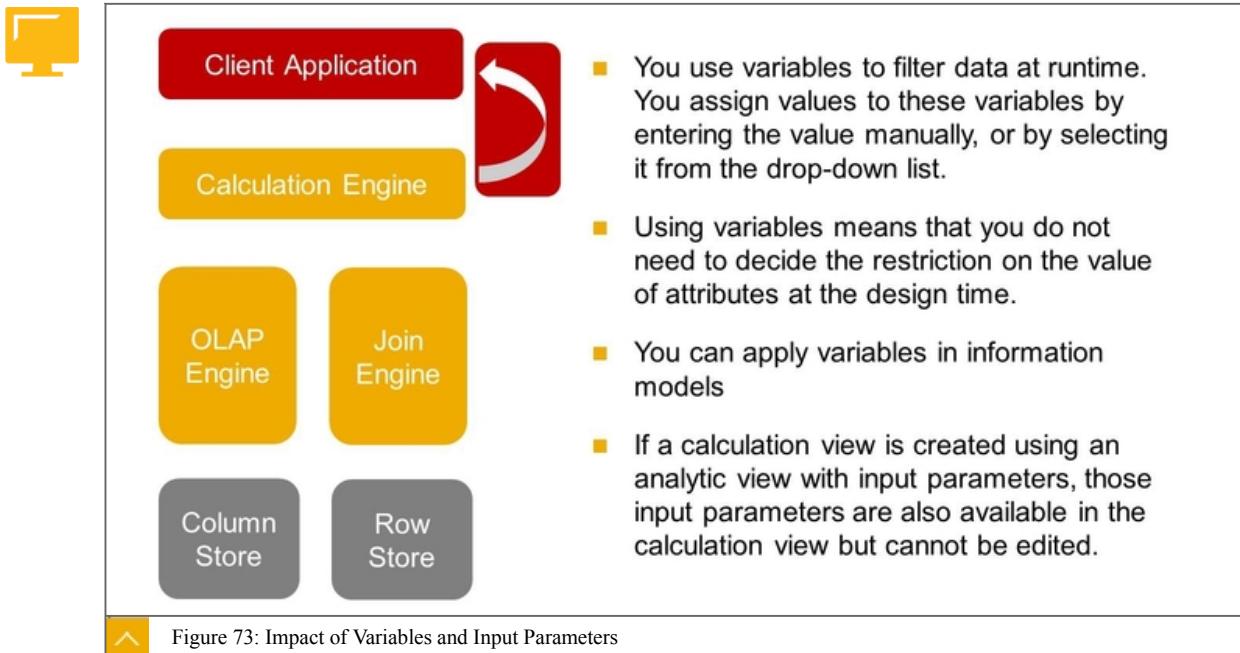


Figure 73: Impact of Variables and Input Parameters

Variables

The following types of Variables are supported:

Type	Description
Single Value	Use this to filter data on a single value of the column
Interval	Use this where you want the user to specify an interval between two values of the column.
Range	Use this when you want the end user to use operators such as "Greater Than" or "Less Than".

Range Variable Example:

GENERAL

Name: * VAR_CUSTOMER
Label: Customer Selector
Notes:
Selection Type: Range (This field is circled in yellow)
 Is Mandatory
 Multiple Entries
View/Table Value Help: * HA300-CV_CAMPAIGN_ANALYSIS(Current View)
Reference Column: * CUSTOMER
Hierarchy:

Figure 74: Variables Types

Unit 2: Modeling Functions

Creating Variables



GENERAL

Name:	VAR_CUSTOMER
Label:	Customer Selector
Notes:	
Selection Type:	Single Value
<input checked="" type="checkbox"/> Is Mandatory <input checked="" type="checkbox"/> Multiple Entries	
View/Table Value Help:	HA300:CV_CAMPAIGN_ANALYSIS(Current)
Reference Column:	CUSTOMER
Hierarchy:	

DEFAULT VALUE

Type	Operator	From Value	To Value
Constant	Equal		

APPLY FILTER

Attribute
CUSTOMER



Figure 75: Creating Variables (I)

A variable definition includes:

- **View/Table for Value Help and Attribute:** These settings define which view/table and which attribute from this view/table is used as a reference to provide a list of values at runtime
- **Selection Type:** Whether selections should be based on intervals, ranges or single values.
- **Multiple Entries:** Whether multiple occurrences of the selection type are allowed.
- You can also define whether specifying the variable at runtime is **Mandatory** and/or if it should have a **Default Value**.
- You define which **attribute(s)** of the current view the variable should be applied to.

The behavior of variables at runtime depends on whether an entry is required for the variable or not, as follows:

- If a variable is defined as mandatory, the user needs to provide the values, ranges, or intervals at runtime.
- For non-mandatory variables, if nothing is specified at runtime, all the data for the corresponding attributes is returned by the view without filtering.

Creating Variables (2)

Semantics

View Properties Columns (5) Hierarchies (0) **Parameters (2)**

Name	Label
[a] INP_EMAIL_DATE	The date when the e-mail was sent
<input checked="" type="checkbox"/> VAR_CUSTOMER	Customer Selector

GENERAL

Name: * VAR_CUSTOMER
Label: Customer Selector
Notes:
Selection Type: Single Value
 Is Mandatory
 Multiple Entries
ViewTable Value Help: * HA300_CV_CAMPAIGN_ANALYSIS(Current View)
Reference Column: **CUSTOMER**
Hierarchy:

- In the semantics of a view, you can see, and also define, which variable is assigned to which attribute.
- Note that one variable can be assigned to multiple attributes.

Figure 76: Creating Variables (II)

Creating Variables (3)

Value Help

Specify a name to find a content of "CUSTOMER" attribute

6 item(s) matched

- High Tech Park
- Electronics Delivery
- Computer Services
- Becker Berlin
- Omega Soft-Hardware Markt
- Lampen-Markt GmbH

OK Cancel

Figure 77: Creating Variables (III)

More than one value can be chosen for a variable when you select the checkbox.

Multiple Entries

Unit 2: Modeling Functions



Note:

In the data preview, From and To are displayed in the Variable Values dialog even when a variable has not been defined as range .

Value Help for Variables

When a dialog box appears to the user they must make a selection. But rather than an empty field appearing and the user having to guess valid values or figure out the format for a value (for example, is the country code UK or GB?) we can have a dialog box populated with the run-time values available. To do this, you make a selection in the setting View/Table for value help . The default entry is the calculation view where the variable is being created. This means that you present **all possible values** from the column that is assigned to the Attribute setting in the variable definition.

While this might seem like a great idea, remember that the list may be huge and would be difficult for the user to navigate. Imagine presenting a list of every employee in a very large organization? If the list should be restricted to offer limited values (such as employees in your line of business or country) then you should reference an external calculation view or table that exposes a restricted list. It is also good practice, from a performance perspective, to refer to an external calculation view or table. This is because it means you are burdening the main calculation view with the task of providing value help for unfiltered columns.

When creating a variable on an attribute that is associated with one or several hierarchies, you can specify one of the hierarchies in the variable definition. With this option, the user can navigate the hierarchy, rather than a flat list, to select the values in the value help. This makes navigation much easier when there are a lot of values; imagine being able to first select your country, then your department, before the list of employees appears? You can use either parent-child or level hierarchies.

There are some basic rules when implementing value help based on hierarchies:

- If you refer to a parent-child hierarchy, the variable attribute column must be defined as the **parent** attribute and not the child.
- If you refer to a level hierarchy, the variable attribute column must be defined at the **leaf** level; that is, the bottom level.

Input Parameters



- You might not want a variable to just restrict the data of a view.
- You might want to take input from the user and process it, returning dynamic data based on the user selection.
- Input Parameters make this possible.



Figure 78: Input Parameter Use Cases

You can use input parameters to define internal parameterization of the view. Input parameters used in the views enable you to obtain a desired functionality when the view is executed.

Input Parameter Types



The following types of Input Parameters are supported:

Type	Use Cases
Direct: Currency	For currency conversion, when you want the end user to specify a source or target currency.
Direct: Date	To retrieve data based on a date entered by the end user (or chosen in a calendar type input box).
Direct: Unit of Measure	To retrieve data based on a unit of measure chosen by the end user
Static List	To provide the end user with a predefined list of values in which he/she chooses one or several items.
Column	To provide the end user with a list of values from a column of the information model
Derived From Table	When you want the end user to have a set list of values from a table (not necessarily included in the view)
Derived From Procedure	When you want the parameter value to be passed to the information model based on the scalar output of a stored procedure
Direct (without semantic type)	When none of the above applies and/or when you want the user to enter a parameter without choosing it from a predefined list.



Figure 79: Input Parameter Types

The figure, Input Parameter Types, shows the different types of Input Parameters that can be defined.

The Direct parameter type can be combined with a semantic type such as Date , Currency , or Unit of measure . This means that the value help will be based on these types of values. For example, if you specify Date then a pop-up calendar will appear for the user prompt. If you specify Currency then a list of valid currencies will be presented in the value help. This allows us to provide flexible input for the user but also allows us to control the type of values that are allowed.

Currency and Unit of Measure Semantic Types

For the Currency and Unit of measure semantic types, the list of proposed values will be created based on the corresponding reference tables in SAP HANA. This setup requires that the default schema assigned to the view contains the reference tables.

Input parameters support multiple values, which means that, at runtime, the end user has the possibility to provide several values to the parameter. Some examples of use cases include the following:

- Applying filters of the types List of values and Not in List
- Expression of calculated columns and expression of filters in projection nodes, provided that the expression requires a multi-value input.

Unit 2: Modeling Functions

When you define an input parameter of the type **Derived from Procedure/Scalar Function**, it is possible to map parameters to the input of the scalar function or procedure.

**Note:**

Input parameters of the types **Derived from table** and **Derived from Procedure/Scalar Function** do not generate a prompt for the end user (they pass the parameter values directly), except if you select the **Input Enabled** option. In this case, the values returned by the table, procedure, or scalar function, can be modified by the end user.

Creating Input Parameters

Unlike a variable, an input parameter can also be used in a conditional expression. For example, we can use an input parameter to determine which measure should be displayed in a particular column.

To illustrate this, we create a calculated column **AMOUNT** that can be filled with either the gross amount or the net amount, depending on the value that the user chooses when querying the view. In our example we have chosen to use an input parameter of the type **Static List**. This means that we pre-define the allowed value that can be chosen by the user in a list. This is fine for short lists, but when the list becomes large it becomes cumbersome to manage, as you would have to edit the calculation view and re-build it each time. Of course you could choose the type **Direct** which would mean the user could input anything. But that would mean, apart from the user not having any guidance, the user could also mis-type the value, or enter the value in the wrong format (perhaps adding leading zeros when they were not required). A good solution would be to define the input parameter with the type **Column** and then in the **View / Table Value Help** enter the name of a table or view where the allowed entries are presented. This also means that this list can be used by multiple input parameters and encourages central maintenance of the consistent, allowed values list.

Input Parameters



The screenshot shows the SAP Fiori Launchpad with a configuration interface for creating an input parameter. The 'GENERAL' section includes fields for Name (GROSS_OR_NET), Label (Gross or Net Amount), Notes, and checkboxes for Is Mandatory and Multiple Entries. The 'Parameter Type' is set to 'Static List'. The 'PARAMETER TYPE - STATIC LIST' section shows 'Data Type' as 'VARCHAR', 'Length' as 5, and 'Scale' as 0. It lists two values: 'GROSS' (Label: GROSS Amount) and 'NET' (Label: Net Amount), with 'NET' being selected. The 'DEFAULT VALUE(S)' section shows a 'Type' of 'Constant' with a value of 'GROSS'. The 'CONVERSION FUNCTION' section is empty.

Figure 80: Creating Input Parameters

- If we want the end user to decide whether Gross or Net amount should be shown in a view, the first step is to create an input parameter that will be used in a calculation.
- The Input Parameter can be of any suitable type, for example a Static List type.
- In this example, the user will be able to choose either "Gross Amount" or "Net Amount".
- Default value GROSS will be assigned to the input parameter if the user does not specify anything.

An input parameter used within a formula does not necessarily have to be of the type **Static List**. For example, it can also be a **Direct numeric value** used in multiplication or any other calculation type.

Unit 2: Modeling Functions

Calling an Input Parameter in a Calculation



The second step is to use the Input Parameter in a Calculated column.

This is done by calling it within single quotes and double dollar signs.

In this example, the input parameter is used in the condition of an *IF* expression

if(''\$GROSS_OR_NET\$\$='GROSS','GROSS_AMOUNT','NET_AMOUNT')



Figure 81: Calling an Input Parameter in a Calculation

In the example in the figure, Calling an Input Parameter in a Calculation, if the user selects GROSS , the calculated column (of type Measure) will display the GROSS_AMOUNT measure in the AMOUNT column. Any other selection will result in NET_AMOUNT being displayed.

Input Parameter Using Dates



GENERAL

Name:	INP_CONV_DATE
Label:	INP_CONV_DATE
Notes:	
<input checked="" type="checkbox"/> Is Mandatory <input type="checkbox"/> Multiple Entries	
Parameter Type:	Static List

PARAMETER TYPE - STATIC LIST

Data Type:	DATE						
Length:							
Scale:							
List of Values <table border="1"> <thead> <tr> <th>Name</th> <th>Label</th> </tr> </thead> <tbody> <tr> <td>20121231</td> <td>31/12/2012</td> </tr> <tr> <td>20131231</td> <td>31/12/2013</td> </tr> </tbody> </table>		Name	Label	20121231	31/12/2012	20131231	31/12/2013
Name	Label						
20121231	31/12/2012						
20131231	31/12/2013						

DEFAULT VALUE(S)

Type	Value
Constant	



Figure 82: Input Parameter Using Dates

- An Input Parameter type of type "Direct" with a semantic type "Date" can be useful when you want to create calculations based on a date specified by the reporting user.

- You can create a date range by creating a pair of input parameters (for example, "Date From" and "Date To")

- Note that the Data Type must be set to "DATE".

Using a Calendar Dialog for Date Input Parameters



Variable/Input Parameter	SQL Data Type	Operator	From	To	Multiple Entries	Value Type	Selection Type
(A) *INP_CONV_DATE	DATE	#			<input type="checkbox"/>	StatList	Single

Value Help

July 2017

26	25	26	27	28	29	30	1
27	2	3	4	5	6	7	8
28	9	10	11	12	13	14	15
29	16	17	18	19	20	21	22
30	23	24	25	26	27	28	29
31	30	31	1	2	3	4	5



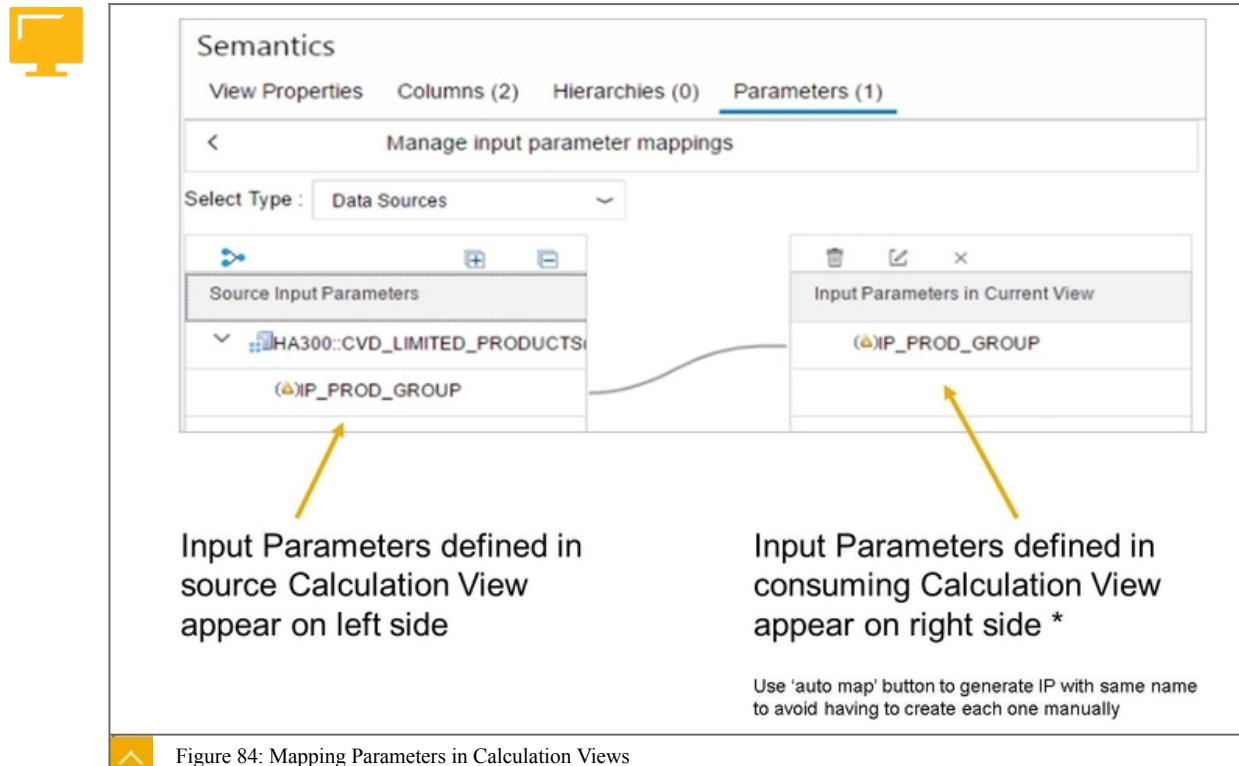
Figure 83: Using a Calendar Dialog for Date Input Parameters

In the figure, Using a Calendar Dialog for Date Input Parameters, the user is asked for a single value. Dates can also be selected as ranges.

Unit 2: Modeling Functions

Pushing Down Input Parameters and Variables to Lower Level Calculation Views

In many cases, calculation views use other calculation views as data sources. This is not necessarily confined to two levels; we can go on and layer the calculation views to create a **stacked model**. When you execute a calculation view in which variables or input parameters are defined, it is possible to pass their values (entered by the end user at runtime) to the lower level calculation views. In fact, the input parameters and variables at the lower levels will usually be ignored unless you define input parameters and variables at the top level and map them to the input parameters and variables in the lower levels. This is called **parameter mapping** and is an important feature of SAP HANA calculation view modeling.



Input Parameter/Variables Mapping

To enable this, you must use the **Input Parameter/Variables Mapping** feature that you can find in the Semantic node of the calculation view.

When you open this pane you must first decide the type of mapping you want to work with.

Parameter Mapping Types

There are four types of parameter mapping and you choose the type from the **Manage Mapping** properties:



Parameter mapping types

Value	Description
Data Sources	Map input parameters of the underlying data source to input parameters of the calculation view
Procedures/Scalar Functions For input parameters	If you are using input parameters of type procedure/scalar function, and if you want to map the input parameters defined in the procedure or scalar function to the input parameters of the calculation view
Views for value help for variables/input parameters	If you are using input parameters or variables, which refer to external views for value help references and if you want to map input parameters or variables of external views with the input parameters or variables of the calculation view.
Views for value help for attributes	If you are creating a calculation view, and for the attributes in the underlying data sources of this calculation view, if you have defined a value help view or a table that provides values to filter the attribute at runtime.

Figure 85: Parameter Mapping Types

Once you make your selection, you will then see on the left side the input parameters and variables that are defined in the calculation views from the lower layers in the stack, which are related to the mapping type you selected. On the right side, you will see the input parameters and variables that are defined in the current calculation view (the one you are editing). You simply drag a line between the left to right side to map them. There is also an auto-map feature which means that if the names are the same, the mapping is done with one click. In the SAP Web IDE for SAP HANA, the auto-map feature can also generate the input parameters with the same name and map them, so that you don't have to create the input parameters manually.



Note:

This feature is also available in SAP HANA Studio but there is an explicit menu option **Copy and Map 1:1**.

Pushing filters down to the source views using parameter mapping is a very common scenario. Choose the type **Data Sources** from the dropdown list in the **Manage Mapping** dialog to enable this.



Note:

Mapping parameters of the current view to the parameters of the underlying data sources moves the filters down to the underlying data sources during runtime, which reduces the amount of data transferred across them. This is a great way to improve performance.

Another common scenario is when you want to push parameters down from the main calculation view to a calculated column in a lower view to support a calculation. Again, this would be the type **Data Sources**.

Unit 2: Modeling Functions

Mapping for Value Help Views



- Variables and Input Parameter can be mapped to variables and input parameters from external views
 - Allows filtering and customizing value help lists from external views
 - Supported with Analytic and Calculation Views (Graphical and Script)
- A Manage Mapping dialog box can be opened from:
 - The variable/input parameter creation dialog box
 - The "Semantics" node

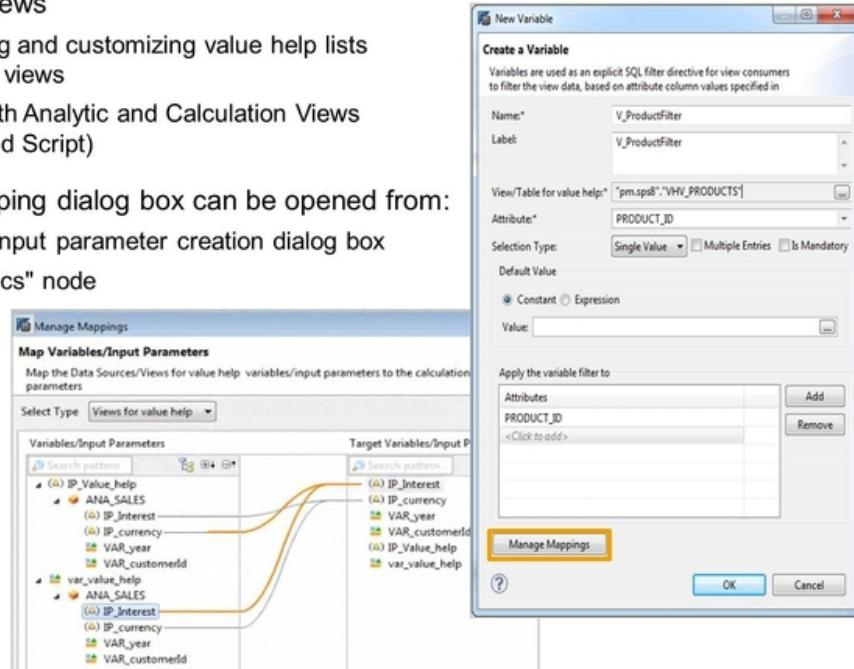


Figure 86: Mapping for Value Help Views

Another important use case for mapping input parameters and variables is to enable dynamic **value help views**.

Cascading Prompts Architecture

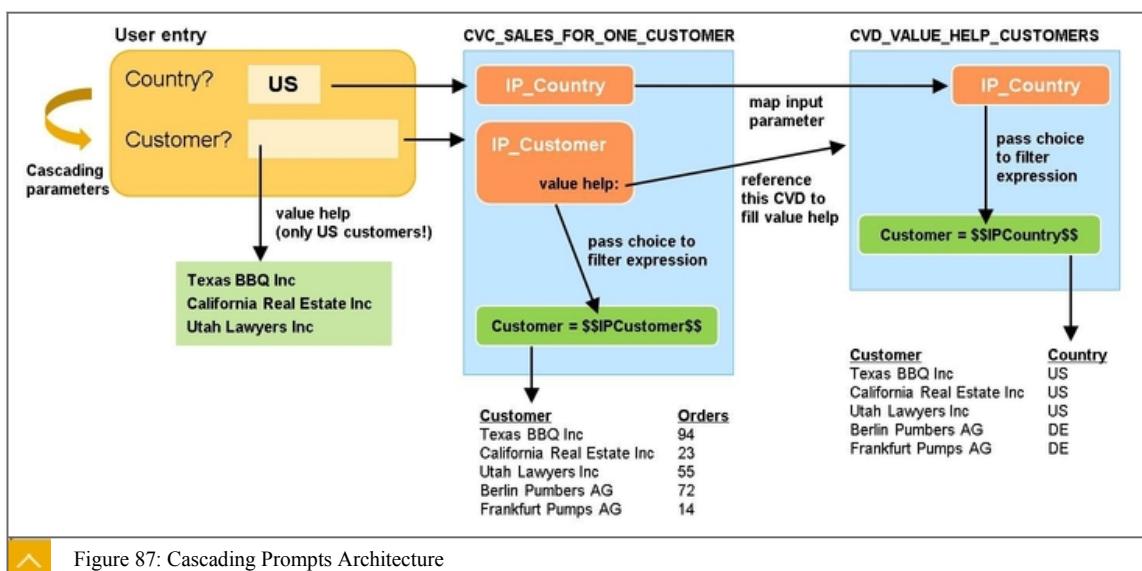


Figure 87: Cascading Prompts Architecture

When you define input parameters and variables, the default data source that generates the **value help** list is taken from the calculation view itself. So essentially you are getting an unrestricted list of all possible values to choose from. But you can also redirect the value help to use a list from another table or view. The main reason we do this is to expose a restricted

value help list. This is also good practice for performance because the value help is not competing with the main calculation view for data. For example, you could create a calculation view on a table that contains all possible cities. Here, your calculation view would include a fixed filter expression that restricts the cities to a specific country. This means that the value help list presents to the user only cities of a specific country. But what if you wanted to change the country? Well, you could go back to the calculation view and change the fixed filter expression, but this would be inefficient.

What we should do is replace the fixed value in the filter expression with an input parameter based on the country. Then we should map this parameter to an input parameter we define in the main calculation view for the country. This means that when a user is prompted for a country in the main view, the value chosen is passed through the mapping to the value help calculation view, so that the cities are filtered by the country that was chosen. The list of cities is then presented as the value help for the **Cities** column. This is also known as **cascading prompts**. Cascading is not restricted to two levels; you can also cascade prompts across multiple levels. For example you could prompt for **Continent** which then restricts the list of countries which in turn restricts the list of **Cities**, and so on.



Note:

A hierarchy could also be considered in this scenario.

To implement value help parameter mapping, you must select the option **Views for value help** for variables/input parameters from the dropdown list in the **Manage Mapping** dialog.



LESSON SUMMARY

You should now be able to:

- Use variables and input parameters
- Create variables and use them to filter data
- Create input parameters

Unit 2

Lesson 4

Using Hierarchies

LESSON OVERVIEW

This lesson explains how to implement hierarchies in information models using the SAP Web IDE for SAP HANA.

Business Example

Hierarchies are usually used in business intelligence reporting to display characteristics across aggregated nodes.

For example, you may have business requirement to display customers in a geographical organization with country, state, and city in a hierarchy.



LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Use hierarchies
- Create parent-child hierarchies
- Work with time-based hierarchies

Hierarchy



The type of hierarchy you want to create will depend on the structure of the source data

- A level hierarchy requires each node in a separate column
- A parent child hierarchy requires separate columns for parents and children.

Parent Child Hierarchy source data:

Parent	Child
DACH	Germany
Germany	Bayern
Germany	Hamburg
EMEA	Ireland
Ireland	Kerry

Level Hierarchy source data:

Region	Country	State
DACH	Germany	Bayern
DACH	Germany	Hamburg
EMEA	Ireland	Kerry



Figure 88: Choosing Hierarchy Type



Note:

Parent-child hierarchy columns usually contain IDs or key fields instead of plain text.

Hierarchy Comparison



Parent-Child Hierarchy					Level Hierarchy				
Parent	Child	X	Y	Z	Region	Country	State	X	Y
1	2				DACH	Germany	Bayern		
1	3				DACH	Germany	Hamburg		
2	4				EMEA	Ireland	Kerry		

- Distinct fields define the parent-child relation
- Parent and child fields usually have the same data type
- Recursive data structure defines the hierarchy

- Heterogeneous fields (possibly with different data types) are combined in a hierarchy

Figure 89: Hierarchy Comparison

Level Hierarchies

A level hierarchy is rigid in nature, and the root and child nodes can only be accessed in a defined order.

To implement level hierarchies, use the following procedure:

Unit 2: Modeling Functions



- Select the source table(s) for the view:

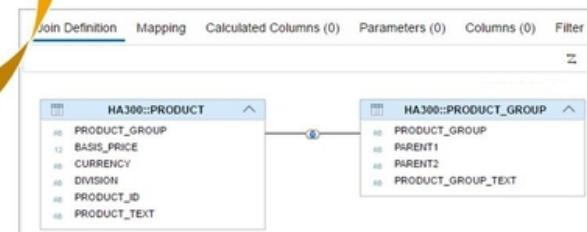
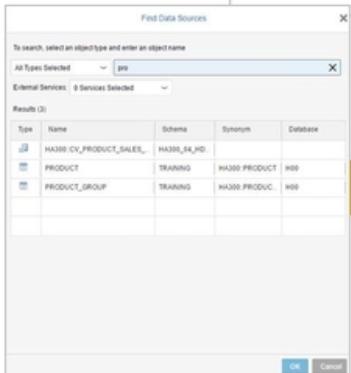
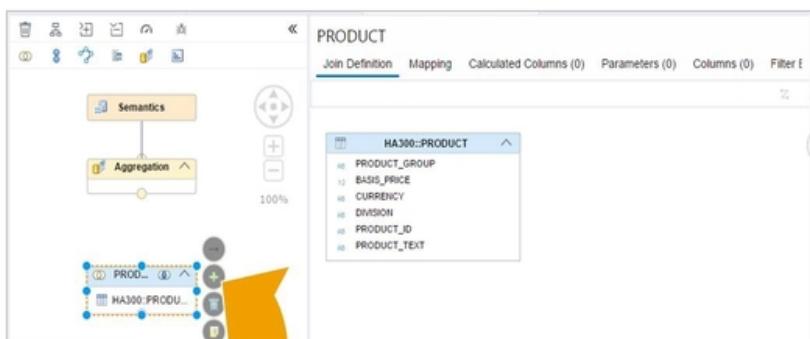


Figure 90: Implement Level Hierarchies (I)

Level Hierarchies (2)



- Select the columns that should be part of the view, including any columns required for the hierarchy:

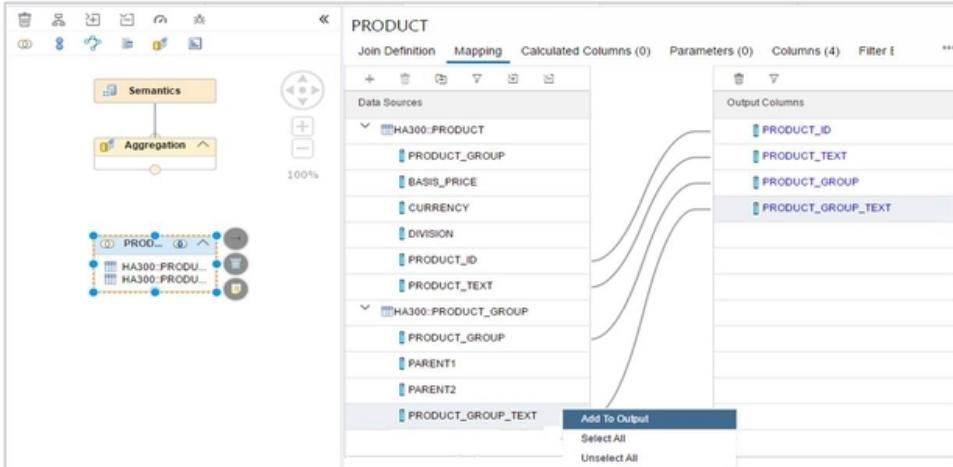


Figure 91: Implement Level Hierarchies (II)

Level Hierarchies (3)



- In the Semantics node, select the Hierarchies tab and click the '+' button in the Hierarchy pane:

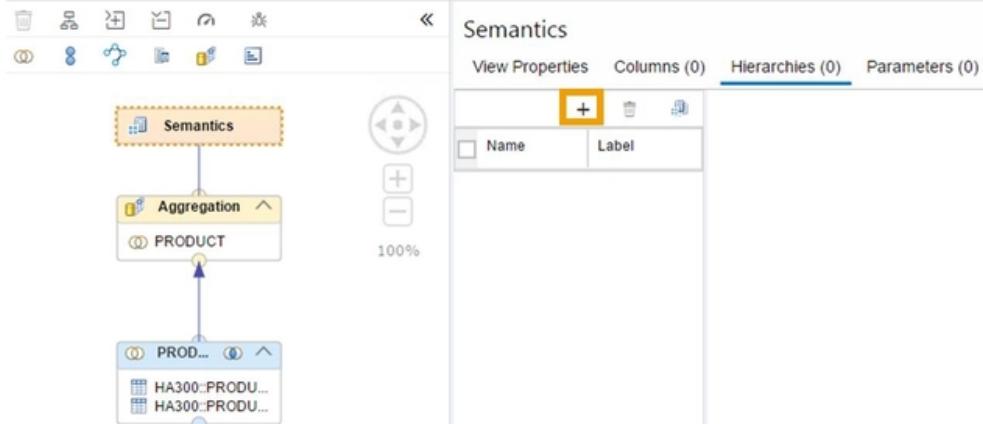
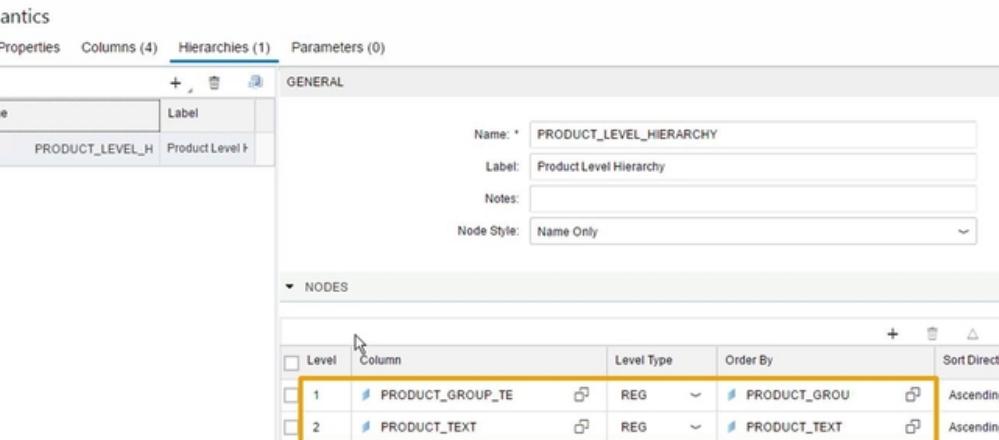


Figure 92: Implement Level Hierarchies (III)

Level Hierarchies (4)



- Add the columns to the hierarchy in the correct level order from top to bottom, with the lowest granularity at the lowest level of the hierarchy.
- Additionally, you can define an ascending or descending sort direction per level.



Figure 93: Implement Level Hierarchies (IV)



Note:

When you preview a calculation view containing hierarchies using the SAP Web IDE for SAP HANA, you will not be able to see the hierarchy in the same way that it is displayed when you use a reporting tool supporting hierarchies.

Unit 2: Modeling Functions

Node Styles

Node styles are used to define the output format of a node ID.

Using a fiscal hierarchy example, the following table demonstrates the different node styles:



Table 8: Node Styles

Level Style	Output	Example
Level Name	Level and node name	MONTH.JAN
Name Only	Node name only	JAN
Name Path	Node name and its ancestors	FISCAL_2015.QUARTER_1.JAN

Level Types

A level type specifies the semantics for the level attributes. For example, the level type TIMEMONTHS indicates that the attributes are months such as January, February, or March.

The level type REGULAR indicates that the level does not require any special formatting.

Hierarchy Member Order

Using the Order By dropdown list, an attribute can be selected for ordering the hierarchy members in the order specified in the Sort Direction column.

Orphan Nodes

An orphan node in a hierarchy is a member that has no parent member.

Level hierarchies offer four different ways to handle orphan nodes:

- Root Nodes
Any orphan node will be defined as a hierarchy root node.
- Error
When encountering an orphan node, the view will throw an error.
- Ignore
Orphan nodes will be ignored.
- Stepparent
Orphan nodes are assigned to a step parent you specify.

Parent-Child Hierarchies

When creating a parent-child hierarchy, the first step is to define the nodes that make up the hierarchy.



The screenshot shows the SAP Semantics interface. In the top navigation bar, 'Hierarchies (1)' is selected. The main area displays a table with one row, 'EMPLOYEE_H', labeled 'Employee Parent-Child'. On the right, there are fields for 'Name' (set to 'EMPLOYEE_HIERARCHY') and 'Label' (set to 'Employee Parent-Child Hierarchy'). Below this is a 'NODES' section containing a table with columns 'Child' (containing 'DNUMBER') and 'Parent' (containing 'PARENT_DN'). At the bottom, there are sections for 'PROPERTIES', 'ADDITIONAL ATTRIBUTES', 'ORDER BY', and 'TIME DEPENDENCY'.

Figure 94: Defining the Nodes of a Parent-Child Hierarchy

The Child column contains the attribute used as the child within the hierarchy, whereas the Parent column contains the attribute used for its parent.

You can define multiple parent-child pairs to support the compound node IDs. For example:

- CostCenter → ParentCostCenter
- ControllingArea → ParentControllingArea

The preceding list of parent-child pairs constitutes a compound parent-child definition to uniquely identify cost centers.



Caution:

Multiple parents and compound parent-child definitions are not supported by MDX.

Advanced Properties of a Parent-Child Hierarchy

Additional attributes can also be added to the hierarchy, making it easier to report on.



The screenshot shows the 'PROPERTIES' section of the SAP Semantics interface. It includes several checkboxes and dropdowns:

- Aggregate All Nodes
- Cache
- Convert empty string values to NULL
- Multiple Parents
- Default Member: [empty input field]
- Root Node Visibility: Add Root node If Defined
- Orphan Nodes: Root Nodes
- Cycles: Break up at load time

Figure 95: Advanced Properties of a Parent-Child Hierarchy

Unit 2: Modeling Functions

Aggregate All Nodes

The **Aggregate All Nodes** property defines whether the values of intermediate nodes of the hierarchy should be aggregated to the total value of the hierarchy's root node. If you are sure that there is no data posted on aggregate nodes, you should set the option to **False**. The engine then executes the hierarchy faster.



Note:

The value of the **Aggregate All Nodes** property is interpreted only by the SAP HANA MDX engine.

Default Member

The **Default Member** value helps identify the default member of the hierarchy. If you do not provide any value, all members of the hierarchy are default members.

Orphan Nodes

In a parent-child hierarchy, you might encounter orphan nodes without a parent. The **Orphan Nodes** property defines how these should be handled as follows:

- Root Nodes

Any orphan node will be defined as a hierarchy root node.

- Error

When encountering an orphan node, the view will throw an error.

- Ignore

Orphan nodes will be ignored.

- Stepparent

Using the **Stepparent** option, orphan nodes will fall under the stepparent node ID as defined on the **Node** tab.



Caution:

If you choose to assign an orphan node to a stepparent, the following rules apply:

- The stepparent node must be already defined in the hierarchy at the ROOT level.
- The stepparent ID must be entered according to the node style defined in the hierarchy.

Root Node Visibility

The **Root Node Visibility** property is used to define if an additional root node needs to be added to the hierarchy.

Cycles

Cycles are typically not desirable in a hierarchy.

In such cases, the `Cycles` property is used to define how these should be broken when encountered, or whether an error should be thrown.

Time Dependent Hierarchies

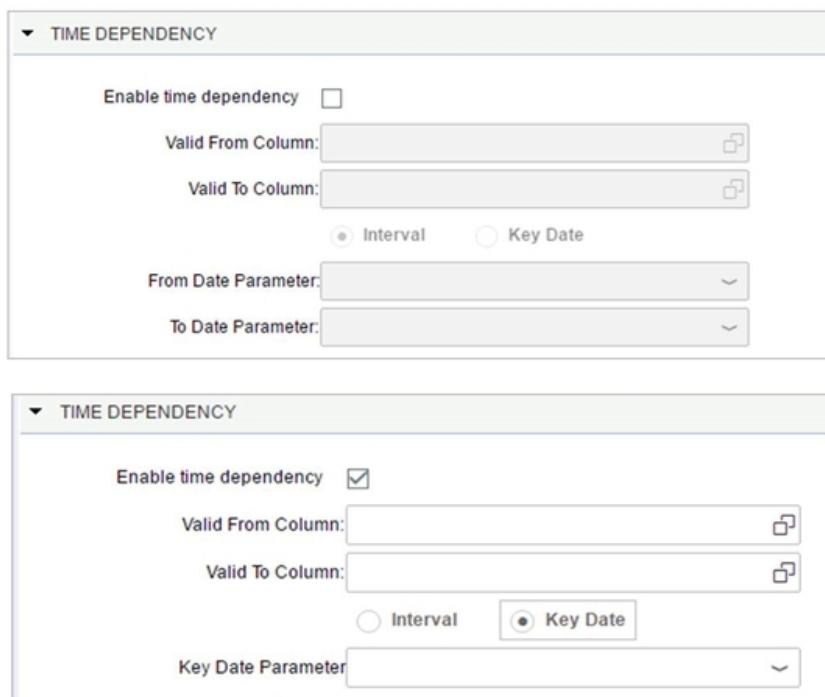
Time dependency is supported for more complex hierarchy data, such as human resources applications with their organizations, or material management systems with BOMs where information is reliant on time.

!

Caution:

Defining a time dependency is only possible in calculation views, and for parent-child hierarchies.

!



!

Figure 96: Defining Time Dependent Hierarchies

Enabling time dependency supports hierarchies based on changing elements valid for specific time periods. This allows displaying different versions of a hierarchy.

Your source data needs to contain definition columns consisting of a `Valid From` and a `Valid To` column.

Determining the Validity Period

When a hierarchy needs to show elements from an interval, you need to define two input parameters; a `From Parameter`, and a `To Parameter`. If the hierarchy needs to show elements valid on a specific date, you need one input parameter defined as the `Key Date`.

Drill Down Enablement

By default, MDX only shows key fields, which are governed by an output field property of the dimension calculation view.

If the `Drill Down Enablement` property is left as default for a non-key field, this field will not show up for reporting in an MDX client such as Microsoft Excel.

Unit 2: Modeling Functions



- By setting Drill Down Enablement to 'Drill Down with flat hierarchy (MDX)', you can make sure that the column can be used for reporting using MDX, even though it is not a key field.

Semantics			
		View Properties	Columns (9)
Type	Key	Name	Drill Down Enablement
<input type="checkbox"/>	<input checked="" type="checkbox"/>	CLIENT	Drill Down
<input type="checkbox"/>	<input checked="" type="checkbox"/>	NODE_KEY	Drill Down
<input checked="" type="checkbox"/>	<input type="checkbox"/>	BP_COMPANY_NAME	Drill Down
<input type="checkbox"/>	<input type="checkbox"/>	BP_ID	Drill Down
<input type="checkbox"/>	<input type="checkbox"/>	FIRST_NAME	Drill Down
<input type="checkbox"/>	<input type="checkbox"/>	LANGUAGE	Drill Down with flat Hierarchy (MDX)

Figure 97: Drill Down Enablement in MDX

When Drill Down Enablement is set to Drill Down with flat hierarchy (MDX) the attribute is enabled for drill down and an additional flat hierarchy is generated.

In the generated flat hierarchy, all the distinct attribute values make up the first and only level of the hierarchy.

Enabling Hierarchies for SQL Access

A hierarchy defined in a dimension calculation view is available (as a shared hierarchy) in any calculation view of the type Cube with Star Join that references the dimension view in its star join. SAP HANA enables you to include a specific column for this hierarchy, which you can query with SQL. This column will display the different nodes of the hierarchy, which will enable filtering or aggregation in SQL queries executed against the Cube with Star Join view.



Note:

This column is only available via SQL. It is not exposed to the graphical information models that consume the view.

For a given Cube with Star Join calculation view, the following two options enable SQL access to the shared hierarchies defined in the underlying dimension views:



- Option 1: Enable all shared hierarchies at once.

You can enable SQL access to all of the shared hierarchies defined in the various dimension calculation views.

In the View Properties tab of the Semantics node, select the Enable Hierarchies for SQL access checkbox.

- Option 2: Select which shared hierarchies you want to enable.

You can specify which of the shared hierarchies you want to enable for SQL access.

In the Hierarchies tab, select a shared hierarchy and in the SQL Access area, select the Enable SQL access checkbox.

You will learn how to consume the hierarchy columns with SQL later on, in the Procedures unit.

Overview of the SAP BI Tools that Support SAP HANA Hierarchies

There are a number of client tools in the SAP and SAP BusinessObjects Portfolio that support hierarchies.

The figure, SAP BI Tools Support for Hierarchies, shows the current status for SAP HANA SPS12 and SAP BusinessObjects Business Intelligence (BI) suite 4.2.



Client Tool	Parent-Child Hierarchies	Level Hierarchies
Analysis, Edition for Office (Excel)	✓	✓
Analysis, Edition for OLAP	✓	✓
Crystal Report for Enterprise	✓	✓
Lumira, Predictive Analysis		✓
Design Studio	✓	✓



Figure 98: SAP BI Tools Support for Hierarchies



LESSON SUMMARY

You should now be able to:

- Use hierarchies
- Create parent-child hierarchies
- Work with time-based hierarchies

Unit 2

Lesson 5

Implementing Currency Conversion

LESSON OVERVIEW

This lesson describes how to set up currency conversion for measures.

Business Example

For a worldwide company, sales are conducted in a lot of different currencies. However, you want to display data and amounts with one single currency to be able to aggregate data. So you need to set up a currency conversion.

You want to know more about the native functionality of SAP HANA to implement currency conversion in your information models.



LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Explain the general principles of currency conversion
- Apply currency conversion in Calculation Views

Currency Conversion in Data Reporting Rationale



- As most front-end tools do not allow defining or switching reporting currency in the UI, and as there might not be such information in master data, we have to convert the possibly many monetary document currencies into just a few.
- SAP HANA has the necessary functions needed to achieve currency conversion during data modeling.



Figure 99: Currency Conversion

When you build information models, the source data is often expressed in a single currency. Typically, for sales data, this would be the transaction currency at the date of sale.

For reporting purposes however, it is very often necessary to convert the currencies. The purposes can include (but are not limited to) the following:

- Corporate Reporting

When using a global reporting currency in corporate reporting, all values should be displayed in the global reporting currency.

- Regional Reporting

For example, the US region might want to see European figures in USD.

- P&L Reporting

You might want to analyze the effects of currency gains and losses.

- Accounting

Data conversion is a strong requirement in multi-currency general ledger transactions.

Currency Conversion



■ As currency exchange rates fluctuate constantly in the global markets, when converting it is necessary not only to define the **source** and **target** currencies when converting, but also to define the **time** when currency conversion should take place.

■ Examples could be:

- Billing Date
- Posting Date
- Financial Year End
- Today's Date



Figure 100: Currency Conversion

Due to the permanent fluctuation of currency exchange rates, you have to define a smart conversion process and, in particular, define which date must be considered to define the conversion rate to apply.

This requirement is even stronger when the source or target currency is volatile.

Native Currency Conversion in SAP HANA Information Models

SAP HANA offers an elaborate conversion mechanism that is based on the following building blocks:

- A set of technical tables to store master data about currencies, exchange rate types, and the exchange rate values.
- The concept of **semantic type**, which allows a flag to measure with the Amount with Currency semantic type.
- An interface to define, for each amount measure, how the conversion should be processed by the information model (which rate and conversion dates should be applied, where to find the source and target currency).

Unit 2: Modeling Functions

Currency Conversion Approaches in SAP HANA

In SAP HANA, data conversion can be implemented in both graphical and script based views.

- Graphical calculation views in SAP HANA provide the easiest way to convert currencies because the conversion modeling can be done using the graphical interface.
- Alternatively, in case of constraints in the master data or because of the complexity of the reporting requirements, you can model the currency conversion within a scripted view.

However, this feature is based on a SQLScript function, `CONVERT_CURRENCY`, which is based on column engine plan operators (or CE functions), which are now deprecated.



Note:

If you have to adapt an existing implementation of this `CONVERT_CURRENCY` function, you can find more information in the SAP HANA SQL and System Views Reference guide, available at <http://help.sap.com/hana>.

Applying Conversion to Lower Aggregation Nodes

From SAP HANA SPS12 onwards, it is possible to define a currency conversion on any aggregation node of a calculation view. This enhancement allows you to maintain a currency conversion at an intermediate level of your calculation view. For example, when you need to combine two different data sets with a Union node, and only one of the data sets needs to be converted.



Note:

Up to SPS11, currency conversion could only be defined at the upmost level of the calculation view, either the upper Aggregation node of a cube calculation view, or the star join node of a cube with star join calculation view.

TCUR Schema

One of the key building blocks to enable data conversion in SAP HANA is a set of tables to define currencies and exchange rate types, and to store the conversion rates.

Table 9: Required Conversion Tables

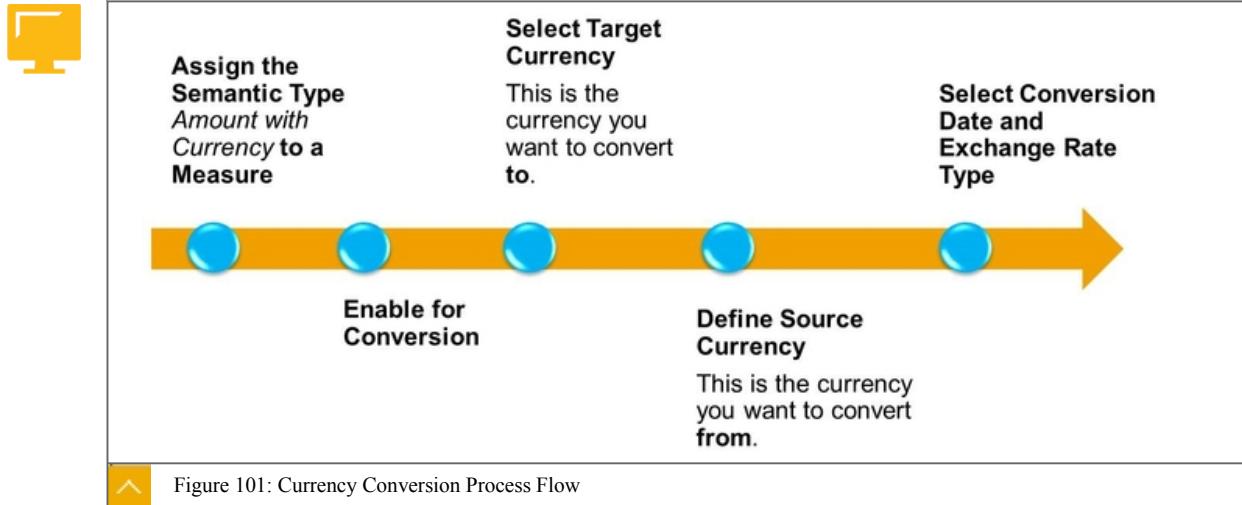
Table Name	Description
TCURC	Currency codes
TCURR	Exchange rates
TCURV	Exchange rate types for currency translation
TCURF	Conversion factors
TCURN	Quotations
TCURX	Decimal places in currencies

These tables exist in most SAP systems (in particular, SAP Business Suite and SAP Business Warehouse).

To enable conversion in SAP HANA, these tables must be available in the SAP HANA database, for example in a dedicated schema for currencies and exchange rates.

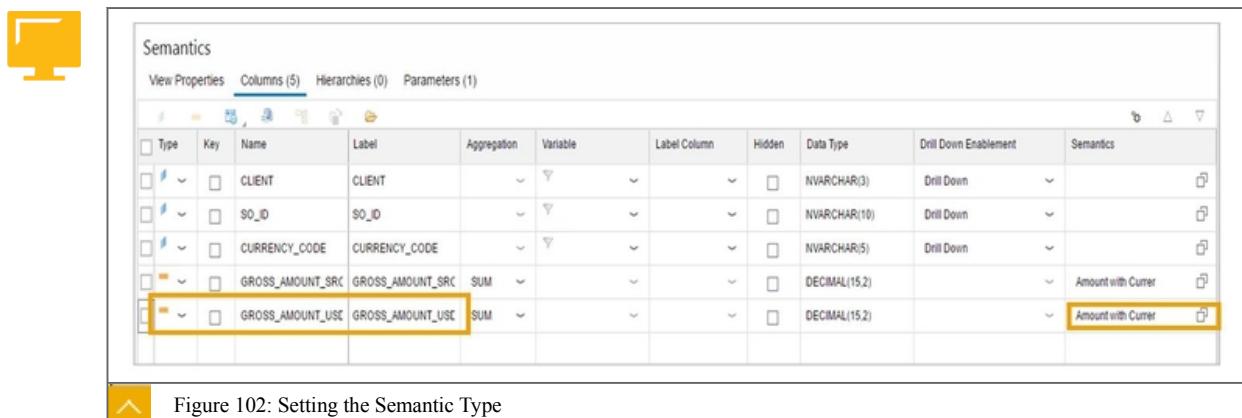
If the models are based on data replicated from another system, the replication process should also include these tables so that they are always synchronized with the ones located on the source system.

Implementing Conversion in Calculation Views



The figure, Currency Conversion Process Flow, depicts the process flow for currency conversion within a calculation view.

Setting the Semantic Type



By default, a measure has no semantic type.

When a measure contains an amount, and if you want to enable conversion, you need to change its semantic type to **Amount with Currency Code**. You can do this in the following ways:

- In the **Output** pane of the Aggregation node, select an aggregated column and, in the **General Properties** pane, define the semantic type. You can also double-click the column. This can also be done on the Star Join node of a cube with star join calculation view.
- Alternatively, when the currency conversion is implemented at the upmost level of the calculation view, select the **Semantics** node, and choose **Assign Semantics**.

Unit 2: Modeling Functions



Note:

The semantic type **Amount with Currency Code** can be used even when conversion is not actually used.

In this case, you only have to set the currency property to identify the currency (or currencies) in which the amounts are expressed.

Defining Currency Conversion Settings



The screenshot shows the 'Assign Semantics For GROSS_AMOUNT_USD' dialog box. It includes fields for 'Copy From', 'Display Currency' (set to Fixed USD), and checkboxes for 'Conversion', 'Decimal Shifts', 'Rounding', and 'Shift Back'. The 'CONVERSION TABLE' section contains fields for Rates (HA300:TCURR), Configuration (HA300:TCURV), Prefactors (HA300:TCURF), Notations (HA300:TCURN), and Precisions (HA300:TCURX). The 'CONVERSION' section contains fields for Client (Fixed 800), Source Currency (Column CURRENCY_CODE), Target Currency (Fixed USD), Exchange Type (Fixed EZB), Conversion Date (Input Parameter INP_CONV_DATE), Exchange Rate (Column), Data Type (DECIMAL Length 15 Scale 2), and Generate Result (checkbox). The 'Upon Failure' dropdown is set to Fail. Buttons for OK and Cancel are at the bottom right.

Figure 103: Currency Conversion Settings

After assigning the semantic type **Amount with Currency Code**, you can enable conversion and define the main parameters used for conversion.

Key Settings for Currency Conversion

Table 10: Key Settings for Currency Conversion

Setting	Description	Options
Schema for Currency Conversion	The schema that contains the TCUR* tables	
Client for Currency Conversion	The client (MANDT) to use to filter the TCUR* tables content	Session Client / Fixed Client Number / Column / Input parameter
Source Currency	The currency in which the amounts to convert are expressed	Fixed / Column
Target Currency	The currency in which the amounts must be converted	Fixed / Column / Input Parameter

Setting	Description	Options
Exchange Type	The type of rate used to convert amounts. Example: Spot rate, average rate...	Fixed / Column / Input Parameter
Conversion Date	The date used to match an amount and the corresponding conversion rate	Fixed / Column / Input Parameter
Exchange Rate	(optional) A column from the source data that contains the exchange rate to be used	
Data Type	The data type of the converted measure	Example: Decimal (15,2)
Generate result currency column	If selected, this option creates a column that indicates for each converted amount the (target) currency in which it is expressed.	
Upon conversion failure	Specifies the behavior if the conversion cannot be executed (for example, if the rate	Fail (a query on the view generates an error), NULL (the column is not populated), Ignore (keeps the source amount without converting it)

It is important to carefully define how the exceptions must be handled when converting data. In addition, to reduce the risk of conversion failure, make sure that the currency conversion tables TCUR* in your SAP HANA system are updated on a regular basis, in particular in a side-by-side scenario where they should always be in sync with the data imported from the remote SAP system.



Note:

The result currency column is never exposed to client tools. It is only available to other SAP HANA views, where it can be used in additional calculations.

Decimal Shift and Rounding

By default, the precision of all values is two digits in SAP ERP tables.

As some currencies require accuracy in value, decimal shift moves the decimal points according to the settings in the TCURX currency table. If you want to round the result value after currency conversion to the number of digits of the target currency, select the **Rounding** checkbox.

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

Reusing Currency Conversion Settings between Columns

From SAP HANA 2.0 SPS02, it is possible to reuse the currency conversion settings for other measures in the same node of a Calculation View. This reduces manual definition and allows

Unit 2: Modeling Functions

for more consistency, especially by avoiding mistakes. The settings can be applied to several measures at the same type.

The currency conversion settings can be reused in two different ways:

Reusing Currency Conversion Settings — Two Options



- Reference

The settings defined in a measure is applied AS IS in the other measures that you select, and cannot be modified in the other measures.

In other words, the settings will always remain consistent and only the source measure for currency conversion setting can be changed, thus impacting the one that reference it.

- Copy

The setting defined in a measure is just copied to the other measures, but they are not bound to each other. The currency conversion settings of the other measures can be freely modified.

Using an Input Parameter for the Currency

The input parameter can also be described as a prompt, in that it asks the user what currency to use.



Figure 104: Creating an Input Parameter

If you want to define the currency at runtime, when the view is executed, you can create an input parameter.

VARCHAR (5) is the way that the currency code is defined in the TCUR* tables, so to be consistent, we recommend that you define the input parameter with the same data type.



LESSON SUMMARY

You should now be able to:

- Explain the general principles of currency conversion
- Apply currency conversion in Calculation Views

Unit 2

Learning Assessment

1. Restricted columns provide a subset of the original column.

Determine whether this statement is true or false.

- True
 False

2. Which of the following benefits characterize the use of a design-time filter?

Choose the correct answers.

- A It reduces the result set of data.
 B The filter is applied on the result set of a query.
 C It is defined on the runtime in the SQL query.
 D The filter is applied on the table before the query is executed.
 E It is applied before a table join is executed.

3. When the Default Client property of an information view is set to Session Client, the data is filtered dynamically based on the CLIENT assigned to the user.

Determine whether this statement is true or false.

- True
 False

4. Identify the object that adds a WHERE clause to the query.

Choose the correct answer.

- A Modeler
 B Attributes
 C Input parameter
 D Variable

5. In a calculation view, for conversion purposes, you can use input parameters as placeholders and formulas like calculated columns.

Determine whether this statement is true or false.

A True

B False

6. What are the features of a parent-child hierarchy?

Choose the correct answers.

A A single dedicated column is used to store the parent of each record of the table.

B Heterogeneous fields are combined in a hierarchy.

C Parent and child field have the same data type.

D Recursive data structure defines the hierarchy.

7. While converting currencies, which parameter is as important as source and target currencies?

Choose the correct answer.

A Money

B Time

C Location

D Season

8. What are the currency conversion functionalities supported by SAP HANA?

Choose the correct answers.

A Conversion based on customer-defined tables

B Enabling of decimal shifts

C Determination of target currency based on attributes or input parameters

D Multiple exchange rate types

Unit 2

Learning Assessment - Answers

1. Restricted columns provide a subset of the original column.

Determine whether this statement is true or false.

True

False

2. Which of the following benefits characterize the use of a design-time filter?

Choose the correct answers.

A It reduces the result set of data.

B The filter is applied on the result set of a query.

C It is defined on the runtime in the SQL query.

D The filter is applied on the table before the query is executed.

E It is applied before a table join is executed.

3. When the Default Client property of an information view is set to Session Client, the data is filtered dynamically based on the CLIENT assigned to the user.

Determine whether this statement is true or false.

True

False

4. Identify the object that adds a WHERE clause to the query.

Choose the correct answer.

A Modeler

B Attributes

C Input parameter

D Variable

5. In a calculation view, for conversion purposes, you can use input parameters as placeholders and formulas like calculated columns.

Determine whether this statement is true or false.

A True

B False

6. What are the features of a parent-child hierarchy?

Choose the correct answers.

A A single dedicated column is used to store the parent of each record of the table.

B Heterogeneous fields are combined in a hierarchy.

C Parent and child field have the same data type.

D Recursive data structure defines the hierarchy.

7. While converting currencies, which parameter is as important as source and target currencies?

Choose the correct answer.

A Money

B Time

C Location

D Season

8. What are the currency conversion functionalities supported by SAP HANA?

Choose the correct answers.

A Conversion based on customer-defined tables

B Enabling of decimal shifts

C Determination of target currency based on attributes or input parameters

D Multiple exchange rate types

UNIT 3

SAP HANA Studio Modeling

Lesson 1

Creating Information Models in SAP HANA Studio

127

UNIT OBJECTIVES

- Create a calculation view using SAP HANA Studio

Unit 3

Lesson 1

Creating Information Models in SAP HANA Studio

LESSON OVERVIEW



LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Create a calculation view using SAP HANA Studio

Core Modeling with SAP HANA Studio

Since SAP HANA 1.0 SPS11, when XSA-based development was introduced along with a brand new deployment infrastructure (HDI), SAP have recommended that data modeling should follow this new approach. This means no more modeling in SAP HANA packages. It also means that modeling should now be done using only the new SAP Web IDE for SAP HANA interface and not in SAP HANA Studio. Modeling content created with SAP HANA Studio is not compatible with SAP Web IDE and the other way around.

So although SAP Web IDE for SAP HANA is the recommended interface for modeling going forward, understanding how content was created with SAP HANA Studio is very worthwhile knowledge.

This is because:

- You cannot access legacy models or objects that were created with SAP HANA Studio or the SAP HANA Web-based Development Workbench using the new SAP Web IDE for SAP HANA. You will still need to use SAP HANA Studio to access those models. SAP HANA 2.0 still supports all content created with SAP HANA Studio and the SAP HANA Web-based Development Workbench, so you may find yourself dipping in and out of these models for some time and will need to use SAP HANA Studio.
- You need to maintain or view deprecated objects, such as attribute views or decision tables. SAP Web IDE does not support these at all.
- You need to use the migration tools that are provided with SAP HANA Studio to migrate information views and other objects in preparation for moving to XSA-based modeling. The migration tools do not exist in SAP Web IDE.
- There are some features of SAP HANA Studio that are not yet available in SAP Web IDE that you want to use. For example, the simple to use, flat file import wizard or the very powerful SQL Plan Visualizer.
- SAP HANA Live and its associated tooling is only available in SAP HANA Studio and not in SAP Web IDE. For example if you wanted to extend an SAP HANA Live view, you can only do this in SAP HANA Studio.

Unit 3: SAP HANA Studio Modeling

Creating a Connection in SAP HANA Studio

The SAP HANA Studio is a multipurpose interface used by a variety of SAP HANA experts. We use the SAP HANA Studio for administration, installation, monitoring and of course, modeling.

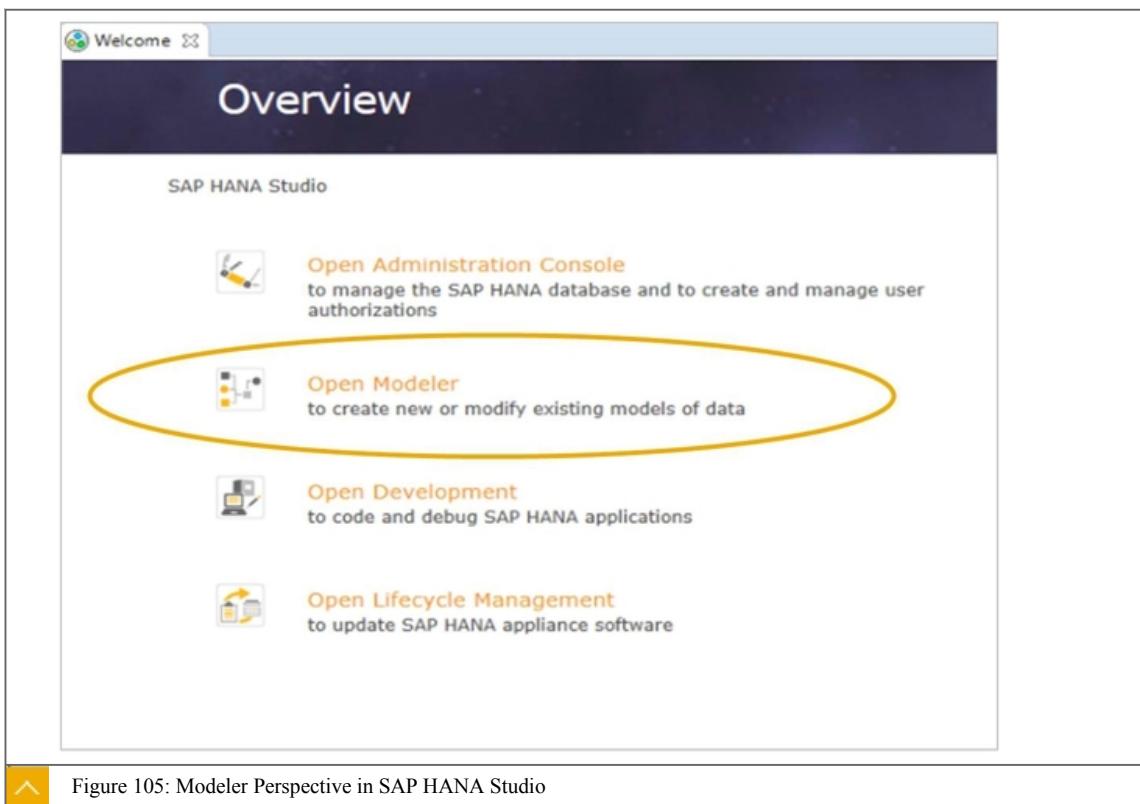


Figure 105: Modeler Perspective in SAP HANA Studio

Modeler Perspective in SAP HANA Studio

The SAP HANA Studio provides many different screen layouts that are built by SAP and organize screen content and tooling for each type of SAP HANA persona. These are called **perspectives**. When you launch SAP HANA Studio, you will be asked to choose a perspective. A modeler would chose the SAP HANA Modeler perspective. It is easy to jump between perspectives once you are inside the interface. For example, you may want to jump to the Administrator perspective in order to check that a service is running, or check memory consumption.

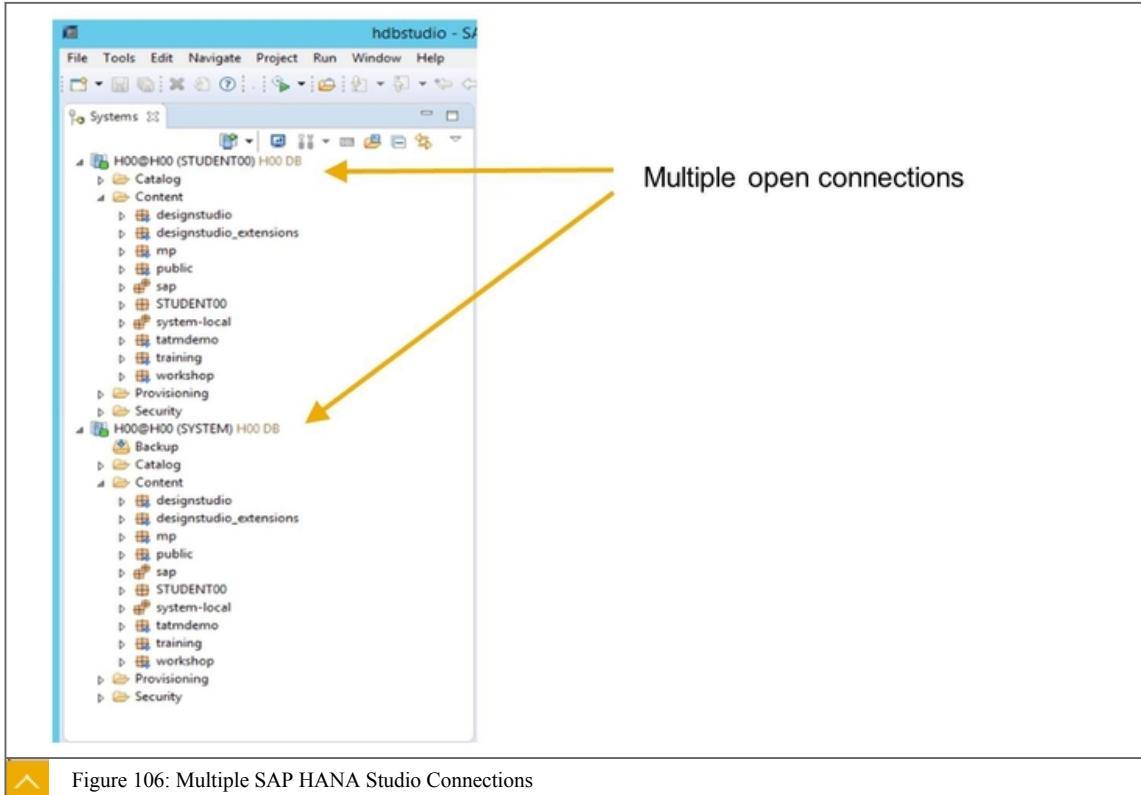


Note:

If you compare the SAP HANA Studio to Web IDE for SAP HANA, one of the key differences you will find is that SAP HANA Studio is full to the brim with features because it is aimed at many types of SAP HANA personnel for all types of tasks ranging from development to administration and monitoring, and so many features are needed to cater for all these roles. SAP Web IDE for SAP HANA is aimed purely at developers. That is why it is called Web Integrated **Development Environment** (IDE). This means that it is a lot simpler, containing only the features needed for developers. “Developers” in this case includes modelers as well as application developers. The administration and monitoring tasks are now managed using another new interface called the SAP HANA cockpit. This is an example of SAP’s approach to simplification – an easy-to-deploy, web-based interface that focuses on one task.

To get started with SAP HANA Studio you need to launch the application and then create at least one connection to the database by providing some properties and also a database user and password.

Multiple SAP HANA Studio Connections



One of the useful features of SAP HANA Studio is the ability to create multiple connections. This means that you can instantly switch between the different SAP HANA systems in your landscape, such as development and QA, without leaving the interface. It's a great way to view content side-by-side in order to make comparisons.



Note:

To open connections to multiple SAP HANA systems using SAP Web IDE you need to open different browser sessions using the URLs that point to the various SAP HANA systems, and then log on to each system. You then switch between the tabs in the browser. You cannot open multiple logon sessions in one browser tab.

Modeling with Studio

In the Modeler perspective of SAP HANA Studio you are able to access two type of content.

- Catalog objects — tables, views, synonyms, functions, and so on
- Content objects — calculation views, procedures, decision tables, and so on

Unit 3: SAP HANA Studio Modeling

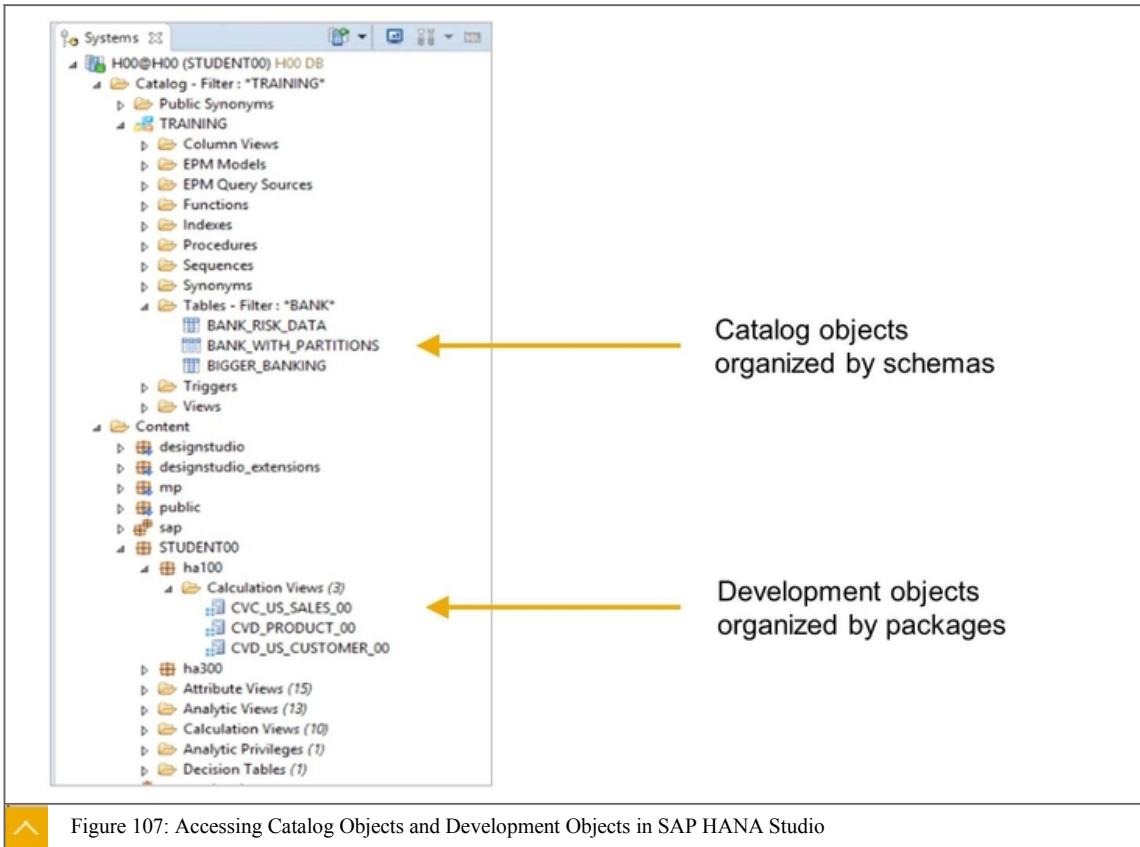


Figure 107: Accessing Catalog Objects and Development Objects in SAP HANA Studio

Development objects (also known as design-time objects) are organized in **packages**. Packages can have sub-packages. The content in each package can be secured so that only developers who have the correct permissions can view or maintain its content. Packages are physically stored in SAP HANA and so the content in each SAP HANA system belongs to that SAP HANA system only.



Note:

Design-time objects built in SAP Web IDE are organized in **folders** and are not physically stored in one SAP HANA system, but in a central shared repository. They can easily be deployed to multiple SAP HANA systems. This is a major improvement over the Studio-based package content which can only deploy to its own SAP HANA system.

Catalog objects (also known as run-time objects) are organized in **schemas**. In order to access content in schemas, developers and modelers need permissions to read, update, and delete. When developers build models or applications, they need to specify the schema in which the database objects reside. This is known as schema-specific development.



Note:

With SAP Web IDE, runtime objects are organized in **containers**. Containers sit above the physical database schema and communicate with physical schemas. Developers and modelers do not access schemas directly; they access only their containers. This is known as schema-less development.

When you open a calculation view in SAP HANA Studio you will see that the screen is very similar to that of SAP Web IDE. Many of the elements are common to both interfaces.

Building Calculation Views in SAP HANA Studio

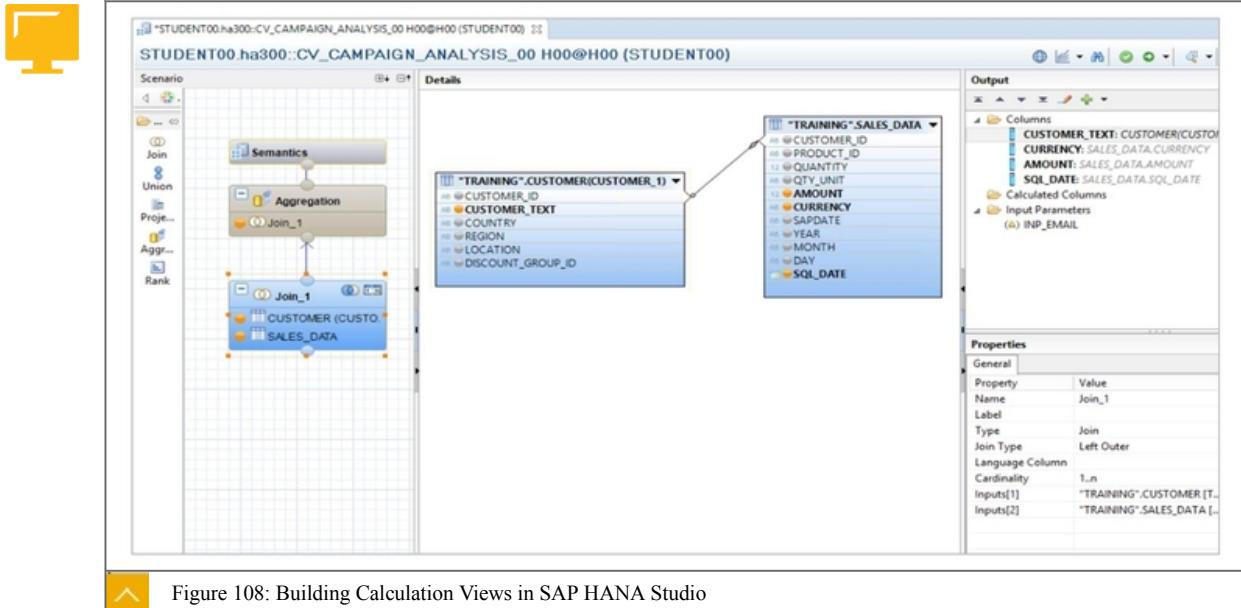


Figure 108: Building Calculation Views in SAP HANA Studio

For someone who is familiar with SAP Web IDE, understanding calculation views in SAP HANA Studio is pretty straightforward.

The following are some key differences between SAP HANA Studio and SAP Web IDE of which you should be aware:

- Unlike SAP Web IDE, SAP HANA Studio does not have a **Mapping** tab. Mapping is done by clicking each column in the **Details** pane and selecting **Add to Output**. The selected columns then appear in the right **Output** pane.
- The folders seen in the **Output** pane of SAP HANA Studio are now tabs in SAP Web IDE, such as **Columns**, **Calculated Columns**, **Input Parameters**, and so on.
- With SAP HANA Studio, the runtime objects are visible in the same structure on the left where you also see the design-time objects. With SAP Web IDE you must switch to **Database Explorer** to access the runtime objects.
- In SAP HANA Studio you **activate** your design-time object to create the runtime object. With SAP Web IDE you **build** your runtime objects.
- A design-time object is a very simple file with a special extension (.hdbculationview) that can easily be copied and pasted anywhere, or even e-mailed to a colleague. In SAP HANA Studio you cannot copy design-time objects with such ease because they are not stored in simple files.

Unit 3: SAP HANA Studio Modeling

Activated Runtime objects in _SYS_BIC

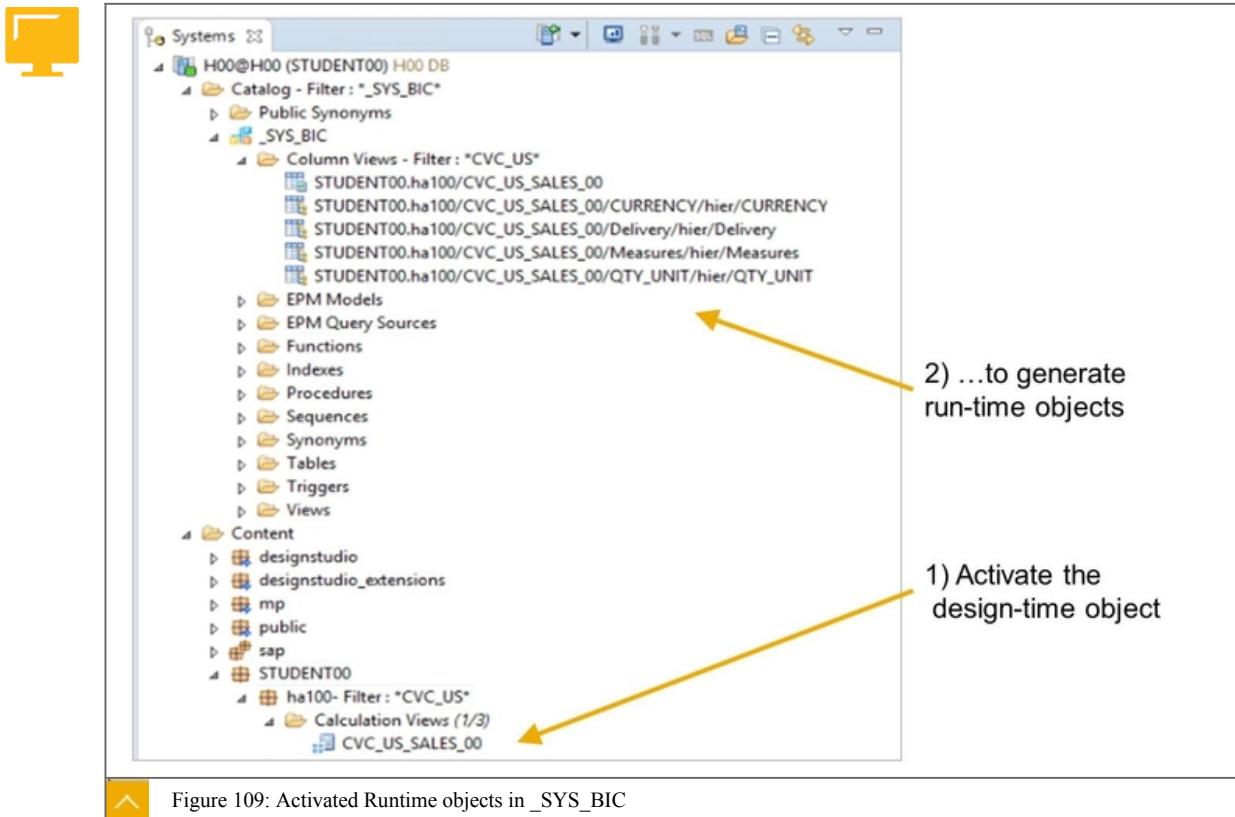


Figure 109: Activated Runtime objects in _SYS_BIC

When you activate a design-time object in SAP HANA Studio, the corresponding runtime object is generated in a fixed schema called **_SYS_BIC**. You will notice that very often there will be multiple objects generated for a single design-time object. SAP HANA automatically generates an optimized runtime object to cope with queries that request the data hierarchically. Even though you did not define a hierarchy in the calculation view, SAP HANA prepares multiple runtime objects that are optimized for hierarchical handling in queries. We do not need to be concerned with these objects because they are automatically selected by the optimizer at runtime.

Note:

Compared to SAP Web IDE, deploying to a single, fixed schema is something of a limitation because it means you can only have one version of the deployed object at any time. With SAP Web IDE, deployed runtime objects can appear in multiple containers. It is possible to deploy different versions of the design-time objects to different containers.

**LESSON SUMMARY**

You should now be able to:

- Create a calculation view using SAP HANA Studio

UNIT 4

SAP Supplied Virtual Data Models

Lesson 1

SAP HANA Live

134

Lesson 2

Embedded Analytics and CDS

152

UNIT OBJECTIVES

- Describe and use SAP HANA Live
- Understand the Virtual Data Model that is based on CDS views

Unit 4

Lesson 1

SAP HANA Live

LESSON OVERVIEW

SAP HANA Live is an optional extension to SAP HANA that provides ready-to-use business views based on SAP Business Suite tables. In this lesson you will discover why implementing SAP HANA Live is essential in the provision of live operational information to business reports and applications.



LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Describe and use SAP HANA Live

SAP HANA Live

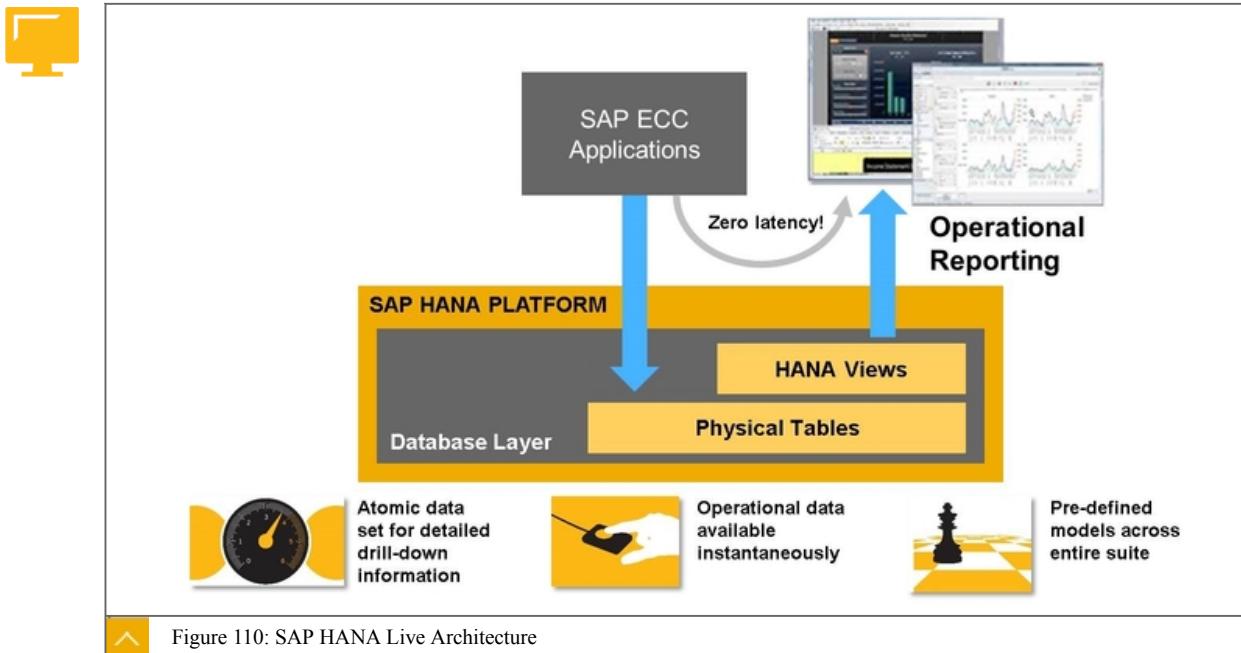
What is SAP HANA Live?

SAP HANA Live is made up of the following features:

- A comprehensive set of SAP supplied calculation views that expose SAP Business Suite tables
- Tools to explore and modify the supplied calculation views
- Sample SAP Fiori applications that consume the supplied calculation views

Due to the normalization of the database schema found in SAP Business Suite, tables can appear complex or fragmented, and are often difficult to consume without a thorough understanding of each schema. As the figure SAP HANA Live architecture illustrates, SAP HANA Live provides ready-to-use business oriented views over tables so that report developers and application builders can easily consume business information in real time without being concerned about the underlying tables and how they must be joined.

SAP HANA Live Architecture



The SAP HANA Live calculation views are created by SAP but can be modified by customers.

Customers can download SAP HANA Live for free and install it in their SAP HANA system.

SAP HANA Live is built **only for use with SAP Business Suite applications**, such as ERP, CRM etc. SAP HANA Live is **not** valid for SAP S/4HANA.

SAP HANA Live calculation views contain all necessary joins, filters, aggregations, and calculations to turn the data from Business Suite tables into meaningful business information ready for consumption.

SAP HANA Live calculation views are technically no different from the calculation views created from custom modeling efforts and SAP have used the very same tools available to customers to create them.

As with any calculation view, SAP HANA Live calculation views can also be consumed by any BI tool, or any custom application development using the standard SAP HANA connectors and query languages supported by SAP HANA.

SAP HANA Live is based on the classic XS/package based repository and not the new XSA / HDI architecture. You maintain the calculation views of SAP HANA Live using SAP HANA Studio. You cannot access SAP HANA Live using SAP Web IDE.

SAP HANA Live

SAP HANA Live has its own release cycle and version numbers, so check the versions available and evaluate which version will work with your Business Suite release.

You can install SAP HANA Live in an SAP HANA system which powers Business Suite, or as a separate side-car deployment of SAP HANA which runs alongside SAP Business Suite.

In a side-car deployment, you must copy the required Business Suite tables to the target SAP HANA system where SAP HANA is installed. How you copy the tables is unimportant, and many tools are available to support the copy process. However, because the central idea of SAP HANA Live is live operational reporting, we recommend you use a replication tool such as SLT to ensure the data from Business Suite is copied in real time to the target SAP HANA.

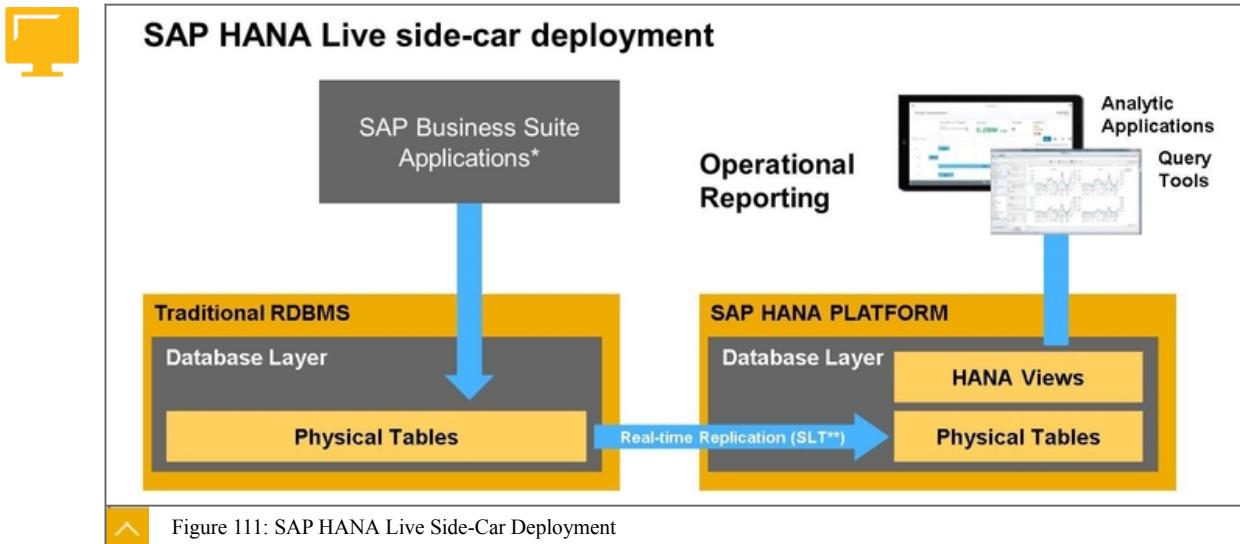
Unit 4: SAP Supplied Virtual Data Models

system. It is not necessary to copy the entire database from SAP Business Suite, you only need to copy the tables that are consumed by the SAP HANA Live calculation views you decide to implement. It is very easy to find out which tables you need to provide by using the supplied SAP HANA Live tools. You simply identify a calculation view you want to use and the tool lists the SAP Business Suite tables that are required.

SAP HANA Live is shipped in packages. You only download and install the packages you need. For example, there is a package for ECC, another package for CRM and so on. All calculation views for each package are installed as one unit in SAP HANA,

As the figure SAP HANA Live Side-Car Deployment illustrates, side-car deployment is an option, and is aimed at customers who do not run their SAP Business Suite on SAP HANA. However, we strongly recommend that you migrate the Business Suite database to SAP HANA so that the side-car deployment is not needed and a fully integrated solution is possible.

SAP HANA Live Side-Car Deployment



The benefits of a side-car deployment of SAP HANA Live include the following:

- Quick start to enable live, operational reporting for customers who do not run their Business Suite on SAP HANA.

The challenges of a side-car deployment of SAP HANA Live include the following:

- Cost and effort to deploy and maintain a separate SAP HANA system to host SAP HANA Live
- Cost and effort to deploy and maintain a data movement technology, such as SLT
- Heavy redundancy of data due to copying
- Risk of incorrect results due to data movement not completing correctly
- Overall complexity of the landscape

**Note:**

Whichever deployment scenario you choose, integrated or side-car, you must define the schema mapping before you import the packages. If you do this, then during the package import, the models will have their schema name automatically swapped from the schema name supplied by SAP to the schema name where your Business Suite tables are found in your system. The schema mapping table identifies the target schema name that is matched to each SAP authoring schema. Failure to take care of the schema mapping will mean SAP HANA Live calculation views retain their original SAP authoring schema, and therefore they will not activate because the schema the SAP HANA Live calculation views refer to will not be present in your system.

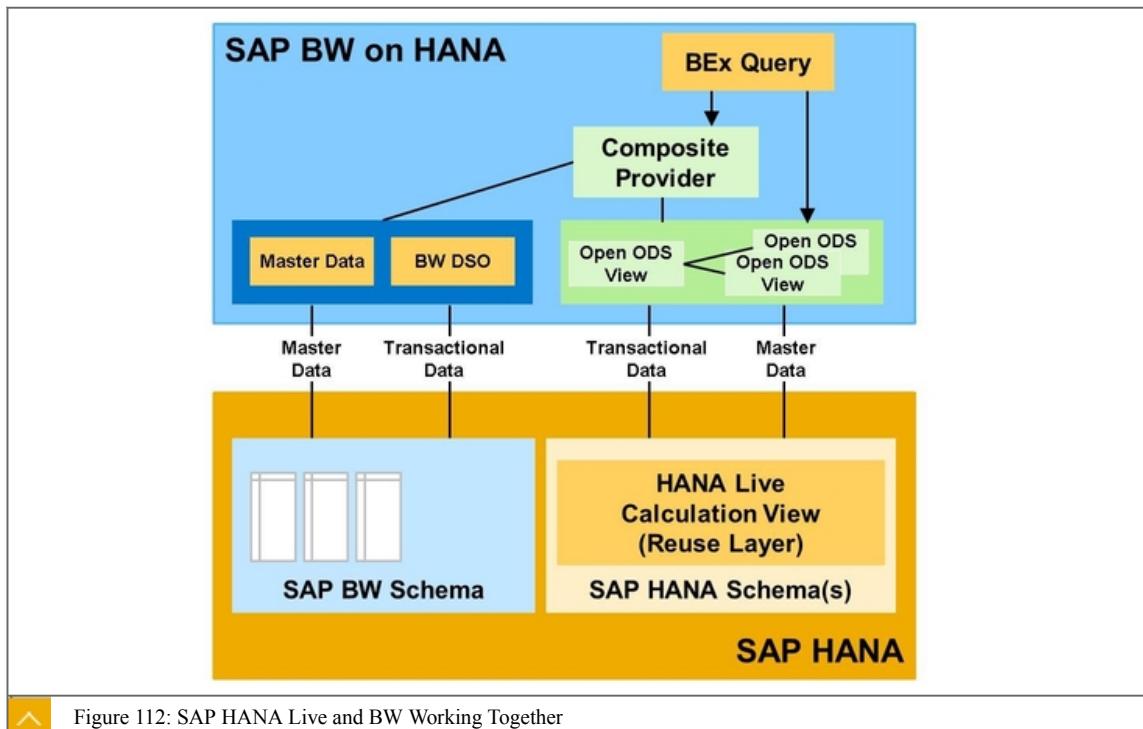
Impact on SAP BW

SAP HANA Live Combined with SAP BW

For many years SAP BW has provided ready to use SAP supplied data extractors and models to cover all key applications of SAP Business Suite. This BW based Business Content continues to provide out-of-the-box reporting on SAP Business Suite tables.

SAP HANA Live can also be regarded as the Business Content for out-of-the box reporting on SAP Business Suite tables. So they might appear at first glance to be doing the same thing.

Does SAP HANA Live make SAP BW and its Business Content obsolete? Do we really need to continue to run SAP BW data warehouse to provide business reporting when we have SAP HANA and SAP HANA Live that can do this and also in real time?



As the figure, SAP HANA Live and BW Working Together shows, SAP HANA Live enhances SAP BW and the other way around. They are both relevant.

Unit 4: SAP Supplied Virtual Data Models

Although there are overlaps with SAP HANA Live, SAP BW is still the recommended solution for scalable, Enterprise Data Warehouse (EDW) scenarios. Remember that SAP BW provides **the full range of data warehousing functions** that are needed to run a large scale data warehouse. The same cannot be said about SAP HANA.** Even with SAP HANA Live installed on SAP HANA, we are still far away from a fully-functional, out of the box enterprise data warehouse. .



Note:

** SAP have recognized that customers may wish to build their own custom SAP HANA based data warehouse instead of, or alongside SAP BW. So there are now a number of components that can be installed into SAP HANA that provide essential data warehousing services, such as scheduling and data request management. But this still leaves the customer with a lot of additional development effort to create the complete solution. To read more about this, look for SAP HANA Data Warehousing Foundation.

However, SAP HANA Live does remove the need to use SAP BW to build live, operational reporting on SAP Business Suite. SAP HANA Live can completely take over that duty, and we can focus SAP BW on providing strategic or historical reporting across a broad landscape of source systems.

SAP HANA Live calculation views are independent of SAP BW and can be consumed separately for live, operational reporting. However, if you combine SAP HANA Live calculation views with SAP BW models, you can develop composite data models that can support real time and also historical data all in one model. The models of SAP BW and SAP HANA Live are easily combined using SAP BW CompositeProviders.

Composite Providers

CompositeProviders are supported only with BW powered by HANA (or BW/4HANA). You cannot build a CompositeProvider using SAP BW powered by non-SAP HANA databases, so there is a strong case for moving SAP BW to run on SAP HANA in order to achieve full integration with SAP HANA Live calculation views and indeed, any customer-developed calculation views.

So, just as a reminder, with almost twenty years of development behind it, SAP BW has many features that are not found in core SAP HANA or SAP HANA Live. The unique or particularly strong features of SAP BW include the following:

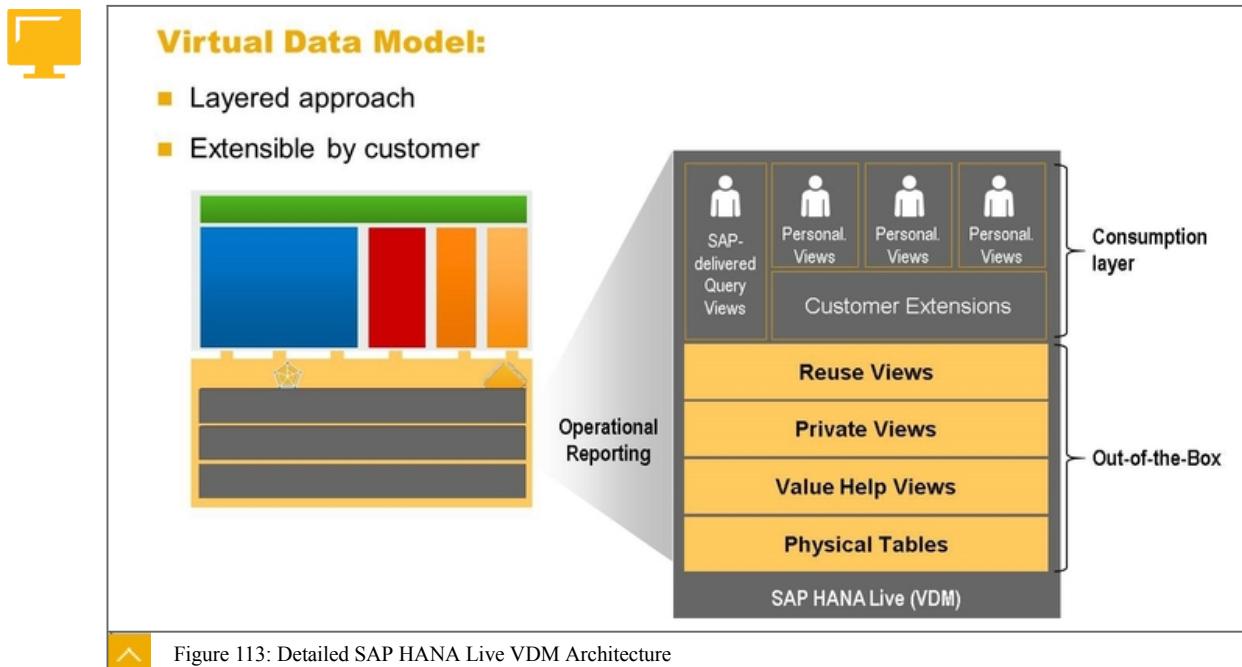
- A highly disciplined, layered, scalable architecture model (LSA / LSA++)
- Advanced automated data loading controls (process chains) and sophisticated error handling and monitoring tools
- SAP provided extractors to take SAP Business Suite data to SAP BW
- A large catalog of applications that run on SAP BW, built by SAP and partners, such as BPC and APO
- Data governance controls (supplied data flow templates)
- Fully integrated Lifecycle Management (NLS)
- Use of ABAP to develop complex business level logic in multiple areas of BW, such as transformations and security

- Sophisticated OLAP security modeling, for example allowing only secure users to access certain levels of a hierarchy, and then only during a defined time period
- Sophisticated hierarchy modeling, for example, graphically develop unbalanced hierarchies with nodes that hold posted and rolled up values
- Advanced OLAP features, for example, nested exception aggregations – show for each country, the count of orders where the average spend per customer within each product category per year was greater than EUR 1000.
- Automated data management (temperatures)
- Out-of-the-box queries and ready to run reports (Business Content), not just models but also data extractors

SAP HANA Live combined with SAP BW powered by HANA provides the best-of-breed suite of analytical services.

The SAP HANA Live Virtual Data Model (VDM)

Understanding the SAP HANA Live Virtual Data Model



As the figure, Detailed SAP HANA Live VDM Architecture, illustrates, the virtual data model (VDM) of SAP HANA Live is a hierarchical structure that is built from calculation views. SAP HANA Live is often described as having “highly stacked views,” which means that calculation views are built in multiple layers, and each layer consumes the output from the lower layers. In this highly modularized approach, calculation views from lower layers are highly reused. As the calculation views appear higher in the stack, more business semantics are added, such as filters and calculated columns to add business meaning to the specific view.

As an example, think of a blocked order list. The lower layers assemble the basic data required, for example, all sales orders, using table joins. Then, we would add filters to the data set to provide business meaning. In this example, we would expect the status codes for blocked orders to be defined in the calculation view filters at the higher levels in the stack. We would not apply the filters lower in the stack because the lower levels could be reused in scenarios where even unblocked order are required.

Unit 4: SAP Supplied Virtual Data Models

The SAP HANA Live VDM is built, maintained, and supported by SAP and follows consistent, standard SAP HANA modeling rules. The VDM has been built with an emphasis on high performance operational reporting.

Key Architectural Feature of SAP HANA Live

A key architectural feature of SAP HANA Live is that the result data set from the top of the calculation view stack is not aggregated. This is because SAP HANA Live needs to support operational line-level reporting. It is possible for the reporting tool to send queries to the SAP HANA Live model that request an aggregated view of data, but there are no in-built pre-aggregation rules defined in the model.

Another key architectural feature is that the stack is built on a relational model and not multi-dimensional. As a result, the only type of calculation view is the **cube without star join**. A multi-dimensional model would use calculation views with the data category cube with star join and we would also need dimension calculation views. There are none of these in SAP HANA Live. Because SAP HANA Live needs to support high-performance operational reporting, we are not trying to create complex OLAP models for deep exploration. With a relational model, all master data attributes are joined with transaction data to create a flattened data set to support flexible reporting. If aggregation is required by the end report then this is possible, but the model does not possess any built-in aggregation rules.

Types of VDM Views

The VDM is made up of layers, which consist of the following types of views:

Query views

Query views are designed for direct consumption by an analytical application or an analytical tool (for example, SAP BI tools). They are always the top layer in the hierarchy of views, and are not designed for reuse in other views. SAP delivers query views to provide a quick start for customers who can use these immediately. However, the SAP-supplied query views only include the attributes and measures chosen by SAP. They do not include all possible attributes and measures from the source tables. They also contain very few variables for user filter selection at run time. It is expected that customers will create their own custom query views, possibly by copying the standard query views and applying their own filters and variables.

Reuse views

Reuse views are the heart of the VDM. They expose the business data in a well-structured, consistent, and comprehensible way, covering all relevant business data in SAP Business Suite systems. They are designed for reuse by other views and must not be consumed directly by analytic tools. You must not make changes to the reuse views because updates to the VDM will overwrite your changes. Only SAP develops and maintains reuse views.

Private views

Private views de-normalize the source database model, which is often far too complex for direct consumption. Private views are built on tables or other private views. You should never consume private views directly because they might not carry clear business semantics. They are intended to be used in reuse views. You should never modify private views as these are strictly developed and maintained by SAP, and the changes you make will be overwritten when you apply updates to the VDM.

Value Help views

Value Help views provide a full list of possible values for popular business entities (for example, material and customers). The Value Help views populate selection lists in business applications or reports to make filter choices easier by having all of the values

visible. Value Help views are not consumed by VDM models, but can be consumed by applications directly.

Text joins are used to support multi-language descriptions of attributes.

Variables are frequently used but only in the top level query views to provide a run-time user selection. Input parameters are sometimes used in lower views, but only where absolutely necessary to add required meaning to data. SAP tried to avoid using input parameters as they are not always easily consumed by some SQL consumers, but if they are used then a default value is always supplied.



Note:

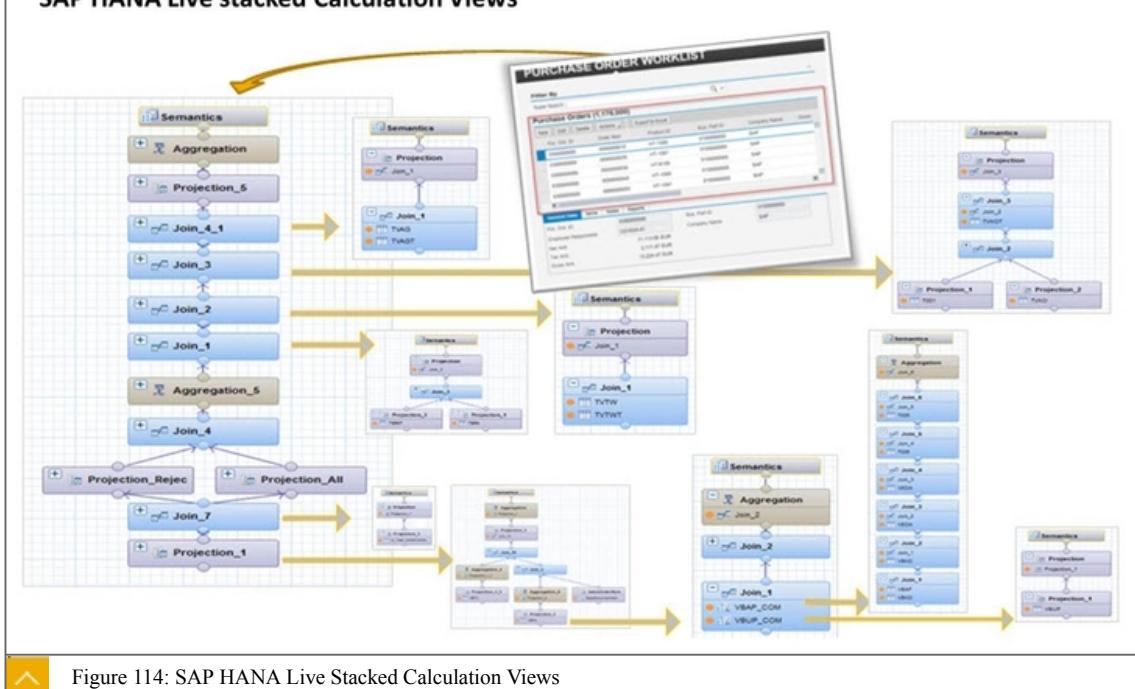
SAP HANA Live is built using the XS classic/SAP HANA repository package environment and **not the XSA/HDI environment**. You must use SAP HANA Studio to view and maintain the models. SAP HANA Live cannot be accessed using SAP Web IDE. SAP have no plans to redevelop SAP HANA Live for XSA/HDI and it will remain fully working in the classic environment of SAP HANA 2.0.

Exploring the Virtual Data Model

You can investigate each VDM in detail with the SAP HANA Studio. When exploring the model with the SAP HANA Studio, pay attention to the icons in the nodes. These identify the type of source (could be another model or a table). Also notice the yellow filter icon on the column in the node as this gives an indication that a node is filtering data that passes through it. Also look for the use of calculated columns and restricted columns. You may also spot the use of variables.



SAP HANA Live stacked Calculation Views



To navigate down the layers of the stacked model from any node, highlight one of the data sources and choose the **Open** right-click menu option. This will open the source object and allow easy step by step navigation through the model. As the figure, SAP HANA Live Stacked Calculation Views, shows, the models are highly stacked, so take care as you open more and

Unit 4: SAP Supplied Virtual Data Models

more source views; it is easy to become overwhelmed with so many open views occupying multiple tabs.



Hint:

Inside the view, use the expand icon in the SAP HANA Studio to expand all nodes with one click so you can easily see all sources at a glance. By default the nodes are usually collapsed.

The Tools of SAP HANA Live

SAP HANA Live Browser



What is SAP HANA Live Browser?

- SAP UI5 based web application
- Browse, search, tag HANA Live models
- Develop personal lists of models
- Business User or Developer mode
- Launch model in Lumira or Analysis
- Identify 'broken' models

SAP SAP HANA Live

ALL VIEWS MY FAVORITES SEARCH INVALID VIEWS

Views	Views in HANA
» BC - Basic Components	
» CA - Cross-Application Basis Components	
» CO - Controlling	
» EC - Enterprise Controlling	
» FI - Financial Accounting	
» FIN - Financials	
» FS - Financial Services	
» LE - Logistics Execution	
» LO - Logistics - General	
» MM - Materials Management	
» PA - Personnel Management	
» PM - Plant Maintenance	
» PP - Production Planning and Control	
» PS - Project System	

Figure 115: SAP HANA Live Browser

As the figure, SAP HANA Live Browser shows, SAP HANA Live Browser is an SAPUI5 web application built in SAP HANA XS that allows you to easily and quickly browse, search, and tag SAP HANA Live views and try them out in SAP BusinessObjects Lumira or Analysis, edition for Microsoft Office.

With the SAP HANA Live Browser, you can browse SAP HANA Live views organized by application component or by SAP HANA modeling package. This makes it easy to find the views you are looking for.

You can also search for specific views. The search considers not just the title of the view but even the column names within the views. So, for example, you could easily locate all SAP HANA Live views that contain the column "order reason".

SAP HANA Live views are always dependent on underlying views and tables being available (not just present but also active). Broken models (listed under the tab Invalid Views) are those SAP HANA Live views where active, dependent models or source tables are missing. If a SAP HANA Live view is missing dependent objects, it cannot be activated. In this case, the SAP HANA Live view is listed in Invalid Views until its dependent objects are made available.

Features of SAP HANA Live Browser

SAP HANA Live Browser can also display all customer-created views, not just those belonging to SAP HANA Live. So you can use the tool to help you catalog and explore your own views.

Custom views appear under the UNDEFINED node in the Application Component hierarchy. Your custom views are not assigned to SAP Application Components, but you can still browse your custom views by SAP HANA packages.

The application can be used by anyone who wants to explore SAP HANA Live views without using SAP HANA studio. One of the unique features of SAP HANA Live Browser is the ability to obtain a complete and easy to interpret overview of an entire SAP HANA Live model including all views and tables that are involved. This is lineage at the view or table level. This cannot easily be done with the SAP HANA studio. You cannot view lineage at the column level using SAP HANA Live Browser, so to trace the journey of a column through a model you would need to use SAP HANA studio, which provides a more detailed view of a model.

SAP HANA Live Browser is a tool in the SAP HANA Live family and must be installed separately. The installation procedure is documented at help.sap.com. Downloading and installing the package provides the XS application code to run the tool. When the tool is installed, you can see a new package: SAP → HBA → Explorer. You also see the schema (SAP_EXPLR) where the tables and views used by the tool (to search, manage tags, and favorites) are located. When you have fully installed the tool you are ready to log on to the SAP HANA Live Browser.

Versions of SAP HANA Live Browser

SAP HANA Live Browser comes in two versions: a limited-feature version that supports business users who do not need all technical options, and a full-feature version that supports developers. To use your version, you assign either the business role or the developer (or both) to an SAP HANA user, as follows:

- In the business-user version, the user can explore models, preview data, and launch the view in SAP BusinessObjects Lumira or Analysis using the following information:

- **http://<SAP HANA Server Host>:80<SAP HANA Instance>/sap/hba/explorer/buser.html**
- Technical name of the required business user role:
sap.hba.explorer.roles::Business

- In the developer version, you can also display broken models, create a list of tables needed for replication in a side-car deployment, and add additional metadata to models, such as tags that aid searching. You access the developer version as follows:

- **http://<SAP HANA Server Host>:80<SAP HANA Instance>/sap/hba/explorer**
- Technical name of developer role: **sap.hba.explorer.roles::Developer**



Note:

The official name for this tool is SAP HANA Live **Browser**. But often you hear the tool incorrectly referred to as SAP HANA Live **Explorer**. This probably originates from the technical component name which is HCO_HBA_EXPLORE.

Unit 4: SAP Supplied Virtual Data Models

Features of SAP HANA Live Browser



Definition

- Explore the metadata of each model

Name	Package
MaterialValuatedSpecialStockQuery	sap.hba.ecc

Content

- Display the actual live data through the model

SAPClient	CompanyC...	Plant	Material	Inv
800	1000	1300	T-20100	
800	1000	1300	T-20210	
800	1000	1300	T-20220	
800	1000	1300	T-20230	
800	1000	1300	T-20300	
800	1000	1300	T-20500	
800	1000	1300	T-20600	
800	1000	1300	T-20610	
800	1000	1300	T-20620	
800	1000	1300	T-20630	

Figure 116: Explore the Metadata and Content of a Model

To preview metadata in an open view, select the Open Definition tab. To preview data, use the Open Content tab, as shown in the figure, Explore the Metadata and Content of a Model.

Use the Search tab in the top menu to search for views, tables, view description, view column names, or private tags. When you expand a view, matched tags, tables, and columns display. Upon enabling the auto complete feature, the name of the view displays. You can also choose to view the definition or content of each view depending on your analytical privileges.

When you enter a word, views that consume the table directly or indirectly are listed.

Explore the Object Relationships and Generate List of Tables for a Model

For efficiency, SAP HANA Live views display in a tree view format on the left side. When you select any view, details of the respective view display on the right side. For views that are open, you can choose to view metadata (Open Definition tab) or Data (Open Content tab), as well as choose to view tables and views used as cross references (Cross Reference tab).

To display all views and tables within a model, select the Open Cross Reference tab. You can then choose Graph View to view cross references as a graph. All the information can be exported in XML format.

SAP HANA Live Browser – Features

Cross reference

- Explore relationships between objects

Generate CSV file of required tables

- All required tables are identified for SLT replication

Figure 117: SAP HANA Live Browser – Features

As the figure, SAP HANA Live Browser – Features shows, you can quickly generate a CSV file for a view, entire package, or application component by right-clicking the respective package or application component and then choosing **Generate SLT file**. You can also choose **CSV** on the top menu. The SLT file option generates a CSV file containing a list of the tables that are needed by the SAP HANA Live views.

One of the benefits of using an SLT Replication Server to replicate source suite tables in a side-car deployment is that you can import the generated list of tables directly into SLT Replication Server. This means that you do not need to manually enter each table name in the replication console; this list can be very long.

To generate connection files for SAP BusinessObjects Lumira, select a view from the SAP HANA Live Browser and choose **Open View in SAP BusinessObjects Lumira**. This feature generates a .svic file. If you open the file, you are prompted to enter the username and password to grant access to the SAP HANA system.

To open the view in Analysis for Microsoft Excel, select a view from the SAP HANA Live Browser and choose **Open View in Analysis Office**.

Unit 4: SAP Supplied Virtual Data Models

SAP-Supplied Analytical Content Built on SAP HANA Live

Rapid Deployment Solutions (RDS) for SAP HANA Live



- Pre-build reporting content
- Covers many SAP BI reporting tools
- Extensible by customers



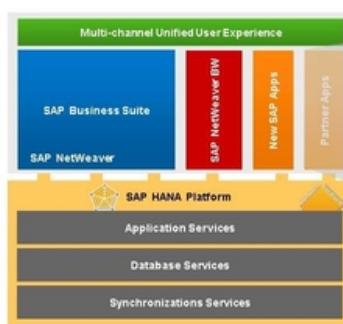
Figure 118: Rapid Deployment Solutions for SAP HANA Live

As the figure, Rapid Deployment Solutions for SAP HANA Live illustrates, SAP HANA Live rapid-deployment solution includes ready-made reports for various SAP BI reporting tools, such as SAP Crystal Reports and Dashboards. SAP HANA Live rapid-deployment solution requires a separate purchase and installation and is not included with SAP HANA Live. It is used when customers want a fast start with ready to run reports. A key difference between SAP BW Business Content and SAP HANA Live is that SAP BW Business Content includes the VDM and many reports built and supplied by SAP to get started. With SAP HANA Live, you get the data model but you need to begin the report building process yourself. Unless, of course, you decide to purchase SAP HANA Live rapid-deployment solution.

Real-Time Analytical Applications



- SAP-built applications with SAP HANA Live content
- Built on top of VDM



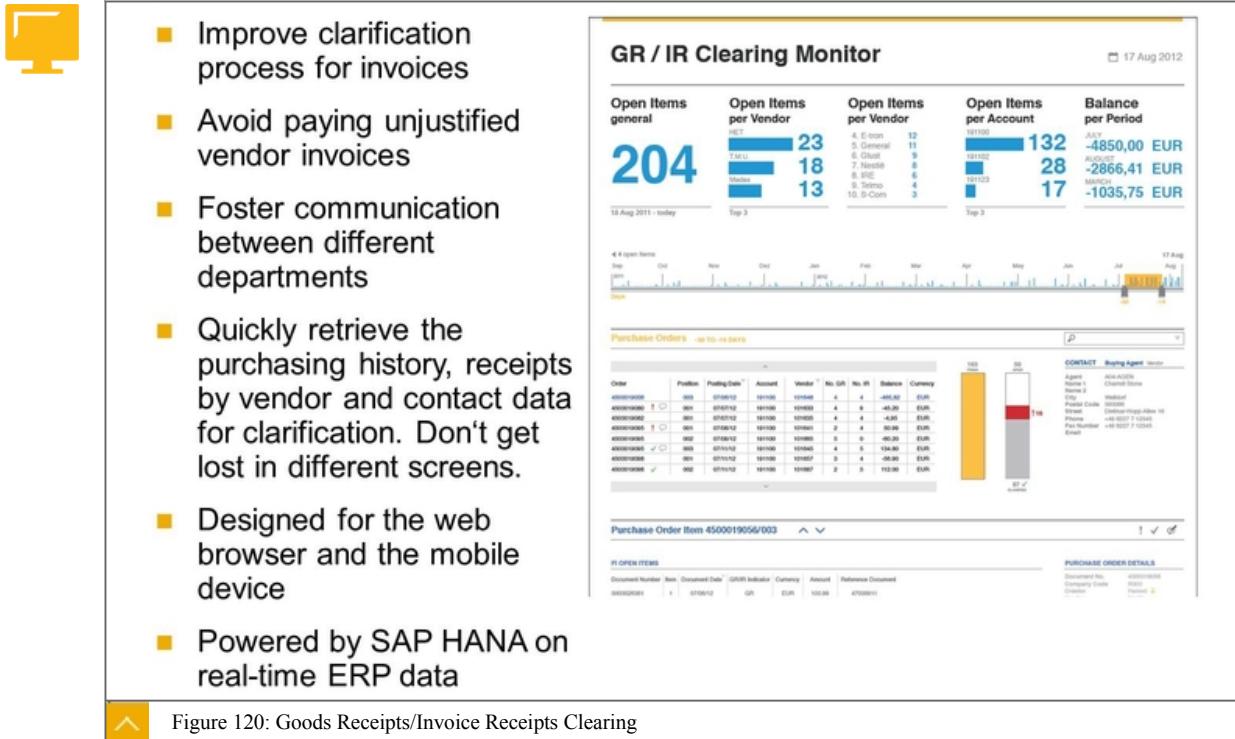
Real-time Applications



Figure 119: Real-Time Analytical Applications

As the figure, Real-Time Analytical Applications, shows, SAP HANA Live features several SAP delivered real-time applications as standard. These applications are created using the SAPUI5 development kit (HTML5, OData, SAP NetWeaver Gateway Services) to create a consumer-grade experience and really show off what can be achieved with SAP HANA Live, beyond reporting. Also referred to as Smart Business Cockpits, the applications rely heavily on the VDM delivered with SAP HANA Live. These applications help customers to understand how to build applications on top of SAP HANA Live.

Goods Receipts/Invoice Receipts Clearing



The figure Goods Receipts/Invoice Receipts Clearing lists some of the features of this application, which is delivered with SAP HANA Live (not SAP HANA Live RDS) to showcase how applications can be built on top of SAP HANA Live.

Unit 4: SAP Supplied Virtual Data Models

SAP HANA Live Customization

Modify SAP HANA Live Models



- You could modify **any** SAP HANA Live model
- Modify and test using **standard** SAP HANA tools in SAP HANA Studio (modeler, debug, trace)
- Detailed SAP HANA modelling **knowledge** is required
- **Non-disruptive** to the use of standard delivered HANA Live models
- New tool available for easy view extensions
- If you will be referring to new tables, make sure they are replicated if using a side by-side scenario

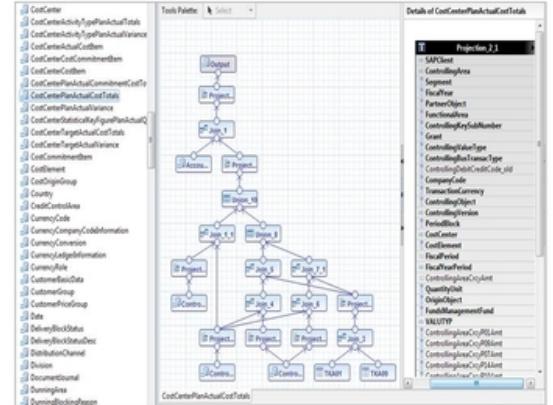


Figure 121: Modifying SAP HANA Live Models

SAP HANA Live views are technically the same as other views, so you can modify them using the standard SAP HANA modeling tools and techniques, as the figure, Modifying SAP HANA Live Models indicates.

SAP also supplies a free tool, called the SAP HANA Live Extension Assistant, to support adjustments. Adjustments you make using this tool are non-disruptive to standard delivered SAP HANA Live models.

No special knowledge is required, just apply your standard SAP HANA modeling knowledge.

You can use all standard modeling, testing, tracing, and debug tools with SAP HANA Live views.

When to Modify SAP HANA Live Views



- Remove / add user prompts to the standard models
- Rename attributes to use business user terminology
- Apply custom filters to reduce the data volume (security + performance)
- Standard models may be missing attributes from underlying Suite tables
- Standard models may expose too many attributes from Suite tables
- Add additional calculated attributes to the standard models
- Build custom hierarchies and combine with standard models
- Combine custom tables / models to the standard models
-and many more reasons



Figure 122: Reasons to Modify SAP HANA Live Views

The figure Reasons to Modify SAP HANA Live Views lists some of the most common reasons for modifying SAP HANA Live views. SAP HANA Live views do not expose all attributes and columns from SAP tables — there are just too many. Only the most important attributes and measures were chosen by SAP, but you can modify the views to suit your needs.

For example, SAP did not know where to apply fixed filters to suit customer requirements, but you can add filters to improve performance and business meaning. You can also develop variables and add them to the model where necessary.

SAP only applied filters to give the view basic meaning (for example, filter order processing status to 'A' and 'B' for Open Sales Orders). If your source system has been customized, you will need to consider additional status codes.

Consider renaming attributes and measures to suit your business requirements.

You can also combine custom tables and views with SAP HANA Live views.

Adjustment Guidelines and Best Practices



- Making changes to Private, Re-Use and Query Views is not recommended
- Changes will be lost when a new VDM release is installed
- Unlike SAP BW, when you install the new content, it will **not** prompt you to use your content or the new standard version
- If a change is required the customer should first copy views manually or use the supplied tool to manage view extensions (which always copies the original view)

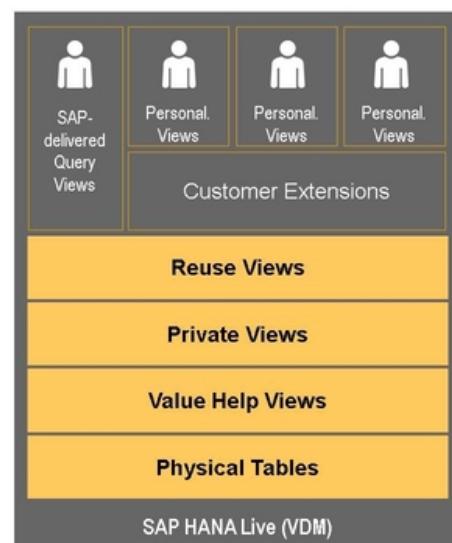


Figure 123: Modifying SAP HANA Live Views – Guidelines

The figure, Modifying SAP HANA Live Views – Guidelines, lists key guidelines to follow when undertaking modifications.

Do not change SAP-delivered SAP HANA Live views directly. You should always work on a copy of an SAP HANA Live view in the customer name-space, otherwise you will lose your changes when you re-import views from new versions of the SAP HANA Live packages. If you just need to add missing columns from the underlying views, then use the supplied Extension Assistant which automatically generates a copy of the view you wish to extend.

Avoid combining SAP HANA Live views with custom attribute and analytical views unless there is a requirement to use features that are only available in these types of views. These types of views can cause the join, OLAP, and calculation engines to be invoked, and when mixed with the SQL engine, this can lead to sub-optimal performance.

If you are tempted to completely remove an attribute or measure from the SAP HANA Live model, remember that it might be referenced by other objects (for example, calculations, restrictions).

Hiding the attribute is best practice and ensures that it can still be referenced by other objects now and in the future, even when it is hidden from the output view for direct consumption.

Unit 4: SAP Supplied Virtual Data Models

Using the SAP HANA Live Extension Assistant

The SAP HANA Live Extension Assistant is part of the SAP HANA Live toolset and must be installed separately. The installation steps are described in detail in the documentation at help.sap.com.

An easy way to determine if the Extension Assistant is already installed is to check if the Extend View menu option appears at the bottom of the context menu for any view (custom or SAP HANA Live view).



Name of extended view	SALESORDERHEADER_EXT																																					
Package of extended view	student00																																					
<input style="width: 100%; height: 20px; margin-bottom: 5px;" type="text"/> type filter text																																						
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 10%;">Type</th> <th style="width: 40%;">Name</th> <th style="width: 50%;">Label</th> </tr> </thead> <tbody> <tr><td>AB</td><td>HeaderBillgIncompletionStatus</td><td>Header Billg Incompletion Status</td></tr> <tr><td>AB</td><td>OverallPricingIncompletionSts</td><td>Overall Pricing Incompletion Sts</td></tr> <tr><td>AB</td><td>HeaderDelivIncompletionStatus</td><td>Header Deliv Incompletion Status</td></tr> <tr><td>AB</td><td>OvrIitmGeneralIncompletionSts</td><td>Ovr Itm General Incompletion Sts</td></tr> <tr><td>AB</td><td>OvrIitmBillingIncompletionSts</td><td>Ovr Itm Billing Incompletion Sts</td></tr> <tr><td>AB</td><td>OvrIitmDelivIncompletionSts</td><td>Ovr Itm Deliv Incompletion Sts</td></tr> <tr><td>AB</td><td>CompleteDeliveryIsDefined</td><td>Complete delivery defined for each sales order?</td></tr> <tr><td>AB</td><td>TotalCreditCheckStatus</td><td>Total Credit Check Status</td></tr> <tr><td>AB</td><td>PhoneNo</td><td>Customer Phone Number</td></tr> <tr style="outline: 2px solid orange;"><td>AB</td><td>TotalNetAmount</td><td>Total Net Amount</td></tr> <tr><td>AB</td><td>SalesOrderCount</td><td>Sales Order Count</td></tr> </tbody> </table>			Type	Name	Label	AB	HeaderBillgIncompletionStatus	Header Billg Incompletion Status	AB	OverallPricingIncompletionSts	Overall Pricing Incompletion Sts	AB	HeaderDelivIncompletionStatus	Header Deliv Incompletion Status	AB	OvrIitmGeneralIncompletionSts	Ovr Itm General Incompletion Sts	AB	OvrIitmBillingIncompletionSts	Ovr Itm Billing Incompletion Sts	AB	OvrIitmDelivIncompletionSts	Ovr Itm Deliv Incompletion Sts	AB	CompleteDeliveryIsDefined	Complete delivery defined for each sales order?	AB	TotalCreditCheckStatus	Total Credit Check Status	AB	PhoneNo	Customer Phone Number	AB	TotalNetAmount	Total Net Amount	AB	SalesOrderCount	Sales Order Count
Type	Name	Label																																				
AB	HeaderBillgIncompletionStatus	Header Billg Incompletion Status																																				
AB	OverallPricingIncompletionSts	Overall Pricing Incompletion Sts																																				
AB	HeaderDelivIncompletionStatus	Header Deliv Incompletion Status																																				
AB	OvrIitmGeneralIncompletionSts	Ovr Itm General Incompletion Sts																																				
AB	OvrIitmBillingIncompletionSts	Ovr Itm Billing Incompletion Sts																																				
AB	OvrIitmDelivIncompletionSts	Ovr Itm Deliv Incompletion Sts																																				
AB	CompleteDeliveryIsDefined	Complete delivery defined for each sales order?																																				
AB	TotalCreditCheckStatus	Total Credit Check Status																																				
AB	PhoneNo	Customer Phone Number																																				
AB	TotalNetAmount	Total Net Amount																																				
AB	SalesOrderCount	Sales Order Count																																				

■ SAP HANA Live models do not expose all standard Suite fields from ABAP tables as attributes (or measures), customers can choose to add more
■ It is possible to easily achieve this using the View Extension tool delivered with SAP HANA Live

Figure 124: Extension Assistant

Using the Extension Assistant, you can extend either a query view or a reuse view but not a private view.

As the figure, Extension Assistant, shows, when extending a query view, the tool automatically exposes all unused columns from all underlying reuse views. It does not expose unused columns from the source tables.

When extending a reuse view, the tool automatically exposes all unused columns from all underlying tables of the reuse view.

When extending query or reuse views, you can only add columns as attributes and not measures.

Extension Assistant

The tool cannot be used to remove standard attributes, but you can use the tool to remove attributes that you added earlier with this tool.

You must be assigned the specific role

sap.hba.tools.extn.roles::ExtensibilityDeveloper to extend any view using the Extension Assistant.

Also, assign the developer to the following permissions on the packages where new extended views are created:

- REPO.READ

- REPO.EDIT NATIVE OBJECTS
- REPO.ACTIVATE NATIVE OBJECTS

You can extend a view multiple times.

If a standard view has already been extended at least once, a dialog displays, and you must choose to create a new view or edit an existing one.

The Extension Assistant only works on standard SAP HANA Live views, not customer created views (the Extend menu option appears but it is inactive).

You cannot use the tool to extend again a generated extension view (the menu option appears but is inactive if it is not available).

You cannot extend query views with Unions . This is because the tool does not provide opportunities to update any union node with the new columns added.

It is not possible to extend query views that have aggregation nodes anywhere other than the top node (the node just below the semantics node). This is because only a final aggregation on all nodes is possible. Otherwise, the measures aggregated may not make sense if you then combine nodes with the newly added fields at a higher level than the aggregation that already took place.



LESSON SUMMARY

You should now be able to:

- Describe and use SAP HANA Live

Unit 4

Lesson 2

Embedded Analytics and CDS

LESSON OVERVIEW

In this lesson, we will introduce you to Core Data Services (CDS) views.



LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Understand the Virtual Data Model that is based on CDS views

Core Data Services (CDS)

We learned that the virtual data model (VDM) for SAP Business Suite is called SAP HANA Live and is built using calculation views. However, SAP HANA Live is not used with SAP S/4HANA.

SAP does provide a comprehensive VDM for SAP S/4HANA but it is shipped within a family of tools and content called **Embedded Analytics**. The VDM is built using a special data modeling language called Core Data Services (CDS). Calculation views are not used.

Just like SAP HANA Live, the CDS based VDM sits on top of the SAP S/4HANA application tables and provides a comprehensive set of business views that covers all business areas. As well as the VDM, Embedded Analytics also provides a tool to explore the VDM called the View Browser, (very similar to SAP HANA Live Browser). SAP Fiori query tools are provided for users so that they can launch and navigate the CDS-based models to view their business data. Just as with SAP HANA Live, the CDS-based VDM of Embedded Analytics exposes data from the transaction and master data tables to enable **live operational** reporting.

One of the motivational factors for moving away from SAP HANA Live is that Embedded Analytics offers more use cases over SAP HANA Live and this was seen as important for SAP S/4HANA. SAP HANA Live has only one use case, albeit a very strong one, and that is:

Business Intelligence (BI). SAP HANA Live is based on calculation views, the sole purpose of which are to expose and enrich data for BI consumption. Calculation views are further limited by being read-only. In contrast, the more flexible multi-purpose CDS-based model has read and write capabilities, and can be used in multiple SAP S/4HANA scenarios for transaction and analytical purposes. Here are some examples:

- To build individual KPI tiles with drill down navigation.
- To build search-based applications.
- To build planning applications.
- To build fact sheets (a new SAP S/4HANA type of innovative application).
- To expose SAP S/4HANA data to ETL tools for extraction.
- To expose live, operational data for BI.
- To provide full OLAP models for dimensional analysis (star or snowflake schemas).

The CDS approach also allows you to extend the same data model for multiple cases. For example, a CDS model could start out being used to expose data for live operational reporting and then be easily extended to provide data extraction services for loading data warehouses with the same live operational data. They can also continue to be adapted to more scenarios as they come up.

But as well as defining data models that sit on top of the tables (virtual layer), CDS can also be used to define the tables. We call this the persistency layer. This means that you could define everything you needed in the database from tables to views all with CDS.

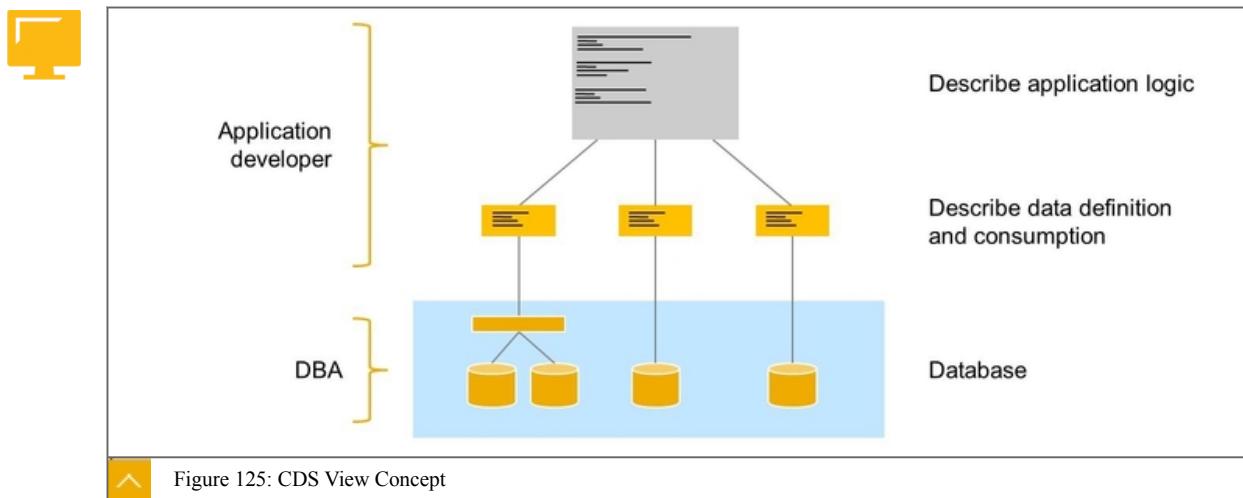
Separating Application Development From Database Development

CDS is built at the application layer and is considered an application developer language. It is not built in the database. Consider the following: Before application developers can begin coding, they first need to manually define all tables and views to which their application code will refer. They must log on to the database and use the database tools (or SQL) to create tables and views (or contact the DBA and request that this is done). Only then can they create their application code on these tables. Now, suppose that the application developer realizes they need to add an additional column to a table. They return to the database, drop the table and add the column, and refill the table (or ask the DBA to do this), before returning to the application code. As well as being cumbersome and time consuming and potentially involving many people, this approach also means the database objects are separate from the application development layer. That is an issue when it comes to transporting everything you need in your application to other systems. It means that you have to transport the application code and then transport the database objects separately and try to ensure that these are synchronized in the target systems.

With CDS, you combine the development of the application code with the development of the database objects all in one layer. In simple terms, a developer is able to define a table and views using CDS without even touching the database. They stay in the application layer and **describe** what they need in the database using the declarative CDS language. The database objects are automatically generated from the CDS definitions. This means that you ensure that both application and database artifacts are kept together for transports because you move the application code and CDS code under one project.

In contrast, calculation views are part of the database layer and not the application layer. This means that if an application or business report consumes a calculation view, then you have to manage the objects and the transportation of these separately.

CDS View Concept



Unit 4: SAP Supplied Virtual Data Models

CDS is a high level abstract language for defining the **persistent** data model (tables) and also the **logical** data model (views), as illustrated by the figure, CDS View Concept. CDS also allows you to define precisely how the models are allowed to be consumed. For example, you can build a rule in the CDS model to state that it may be used for OLAP purposes but not extraction (ETL). These CDS definitions sit between the application code and the database. With CDS you simply describe what you need from the database using a data definition language (DDL) to create tables or views. CDS is a declarative language, so you simply state intent and do not technically describe how the data should be processed. This is exactly the way SQL works, and in fact CDS is largely based on the SQL language. This makes them easy to work with.

CDS is written in source files in SAP Web IDE (for XSA/HDI) and Studio (for XS/classic). A single CDS source file can be used to create multiple database objects. Each object is known as a **CDS entity**. A table is a CDS entity and so is a view. Just as with calculation views, you build the CDS file to generate the database objects. But you never consume the generated database objects directly, you always consume the CDS entity. When you adjust the CDS entity and reactivate, the database objects are automatically regenerated, even if they contain data.

Major Feature of CDS

A key feature of CDS is the ability to include technical and business semantic information to enrich the models. You insert special instructions throughout the CDS code to describe the meaning or rules of the various elements. These are called **Annotations**. An Annotation is simply a line of code that describes rules such as:

- How the CDS view can be consumed (for example, by other CDS views, by reporting clients, by Odata)
- How a measure should be aggregated
- The overall purpose of a view (for example, dimension or fact)
- That a column is currency and the currency code is fixed to YEN
- The purpose of a column (for example, a description for another column, a currency)

CDS is built in a stack. You begin by building CDS entities that describe basic fields such as street and city. You then build CDS entities to combine these, for example address which refers to street and city and so it goes. Once you have finished building entities that make up the persistence layer, you then move on the logical layer and build CDS views on top.

Even CDS views are built in layers, just like SAP HANA Live Calculation Views. You define the elementary CDS views at the lowest level and then stack them until you reach the top level which you then make available for consumption.

CDS views can be grouped with other CDS views in an association to create a relational model. This is similar to database table joins, but the association is defined at an abstract, not a hard, physical level.

When you adjust a CDS view, all other CDS views that refer to it will be adjusted too. For example, if you add a column to a CDS entity that describes a table, you don't have to change all the other CDS views that refer to it. This facilitates extensibility and supports customer extensions to standard delivered CDS views.

Another key difference between SAP HANA Live and CDS is that SAP HANA Live relies entirely on business data access control through SAP HANA Analytic Privileges. Within CDS views, you can define the access controls individually. You can also reuse ABAP authorizations. However, you do not use SAP HANA Analytic Privileges with CDS.

Types of CDS View

You need to be able to distinguish clearly between the two types of CDS that are available:

SAP HANA CDS and **ABAP CDS**. They are similar in concept and design, but the technical implementation differs.

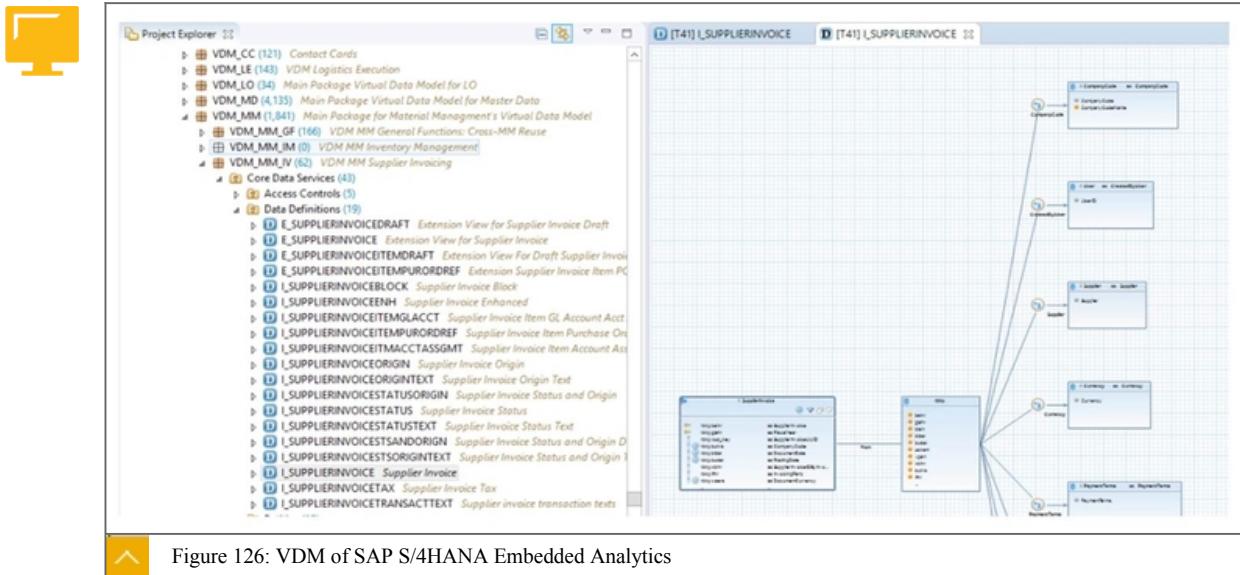
SAP HANA CDS is built using SAP HANA Studio or SAP Web IDE and is stored in SAP HANA. Developers who build native SAP HANA applications work with these types of CDS views. When a developer activates a CDS view in SAP HANA, the SAP HANA database objects, such as tables and views, are then created. When the SAP HANA application code is transported from one SAP HANA system to another, the CDS views are also included and both the application code and the CDS code are activated on the target system together. These CDS views are native to SAP HANA, processed in SAP HANA and can only be used with SAP HANA applications.

ABAP CDS is built and stored in the ABAP data dictionary of the ABAP server and not in SAP HANA. Developers who build ABAP applications work with these types of CDS views. They work with ABAP CDS using the ABAP Perspective of Eclipse (SAP HANA Studio). Just as an ABAP developer activates the definition of an ABAP table from transaction SE11, which then generates a physical table in the database, the activation of an ABAP CDS view in the ABAP dictionary also generates a physical database object which is an SQL view. Another benefit of ABAP CDS is that data security can be managed using the classic ABAP-based authorizations. You can either refer to existing ABAP authorization objects, or assign the business values directly in the CDS views. This is seen as a major benefit of simplification — you do not need to have two layers of security, one in the application layer and then another in the database layer. ABAP CDS views are combined with other ABAP artifacts, such as function modules, and tables can be transported and activated together in the target systems. ABAP CDS views belong to the ABAP layer and are not native to SAP HANA. This means ABAP CDS can be redeployed to other databases. The range of available functions and syntax in ABAP CDS views is similar, though not identical to SAP HANA CDS views. SAP HANA CDS views expose more native features of SAP HANA and can only be used with SAP HANA.

SAP S/4HANA VDM is Built with ABAP CDS Views

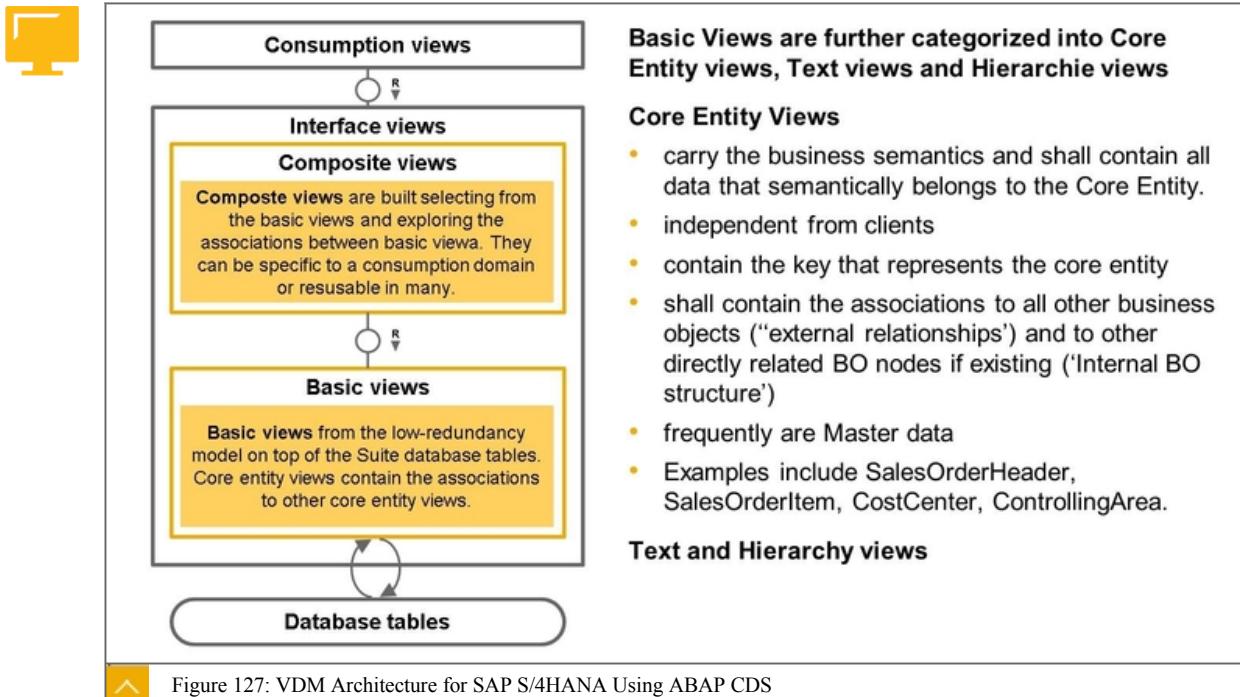
The VDM built by SAP for SAP S/4HANA is built with ABAP CDS. SAP HANA CDS can be used to define both the persistence model (tables) and also the logical model (views). However, ABAP CDS is used to develop only the logical data model. ABAP data dictionary elements such as domains, data elements, tables are still built using the ABAP DDIC toolset (for example, SE11).

Unit 4: SAP Supplied Virtual Data Models



Just like calculation view used with SAP HANA Live, the CDS views are built in layers to support high reusability. As the figure VDM of SAP S4/HANA Embedded Analytics shows, the lowest layers of the VDM expose the basic table elements and combine them to denormalize the schema. The higher layers add semantics and expose the data to any consumption applications. The consumption views layer (prefix is C_) is the equivalent of the query view layer of SAP HANA Live. The composite view layer is the equivalent of the reuse layer of SAP HANA Live. The basic view is the equivalent of the private layer of SAP HANA Live. Composite and basic views are both types of interface views (prefix is I_).

VDM Architecture for SAP S/4HANA Using ABAP CDS



The figure, VDM Architecture for SAP S/4HANA Using ABAP CDS, lists the features of ABAP CDS views. Like SAP HANA Live, the ABAP CDS views can also be extended by customers, and the extensions are upgrade-proof (CDS prefix is E_).

However, unlike SAP HANA Live, you must create ABAP CDS using a text editor. There is no graphical editor.



Note:

If you see something that looks like a graphical representation of ABAP CDS, it is simply a read-only view of the model in a graphical format that can be launched from the text editor. You cannot use this tool to make changes.

CDS View Creation

An Example of a Simple View Built with CDS



```
Annotations explained:
@EndUserText.label: 'Financial Statement' ← The label that is shown in the UI
@Analytics: {dataCategory: #FACT } ← Used by BI tool to identify
                                         transaction data (inc. a measure)

define view FinancialStatementQuery as

    select from FinancialStatement
    {
        key ChartOfAccounts,
        key GLAccount,

        @Semantics.currencyCode: true ← Indicates the following element
                                         is a currency code
        key CurrencyCode,

        @Semantics.amount.currencyCode: 'CurrencyCode' ← Indicates where to find the currency
        @DefaultAggregation: #SUM ← Specifies default aggregation behavior
        @EndUserText.label: 'Amount in Company Currency'
        AmountInCompanyCodeCurrency
    }
```

Figure 128: Example of a View Built with CDS

The figure, Example of a View Built with CDS, shows a CDS view that a developer has created for consumption by business users in their various reporting tools. The view has been classified as a fact table so the reporting tool knows how to handle this and what to look for (measures and aggregation rules). Its purpose is to sum the G/L amounts and present them in the currency of the company. Notice how the label Amount in Company Code Currency has been defined so that the business user understands the business meaning of the element. This CDS view is reading from another CDS view or entity called FinancialStatement. If you want to see how the data is sourced, you would have to open that CDS view by double-clicking it.

The @ symbol appears in the CDS view at the start of each annotation. Annotations define the technical and business semantics. You can add a great many different annotations to the CDS view to give it meaning. Some annotations relate to security, filters, aggregation, and the types of consumers allowed.

Unit 4: SAP Supplied Virtual Data Models

When opening a reporting tool, this CDS view could appear alongside calculation views for selection by the business user.

CDS Views and Core SAP HANA Modeling

So with their superior flexibility it might appear that SAP is moving away from calculation views and towards CDS. But this is simply not true and SAP recommends the use of calculation views for **pure BI cases** as opposed to application development. We are aware that calculation views cannot define the persistence layer, they only describe the logical or consumption layer. But calculation views come back strong in the areas of BI:

There are some differences to be aware of:

- Not all calculation view settings and features are available in CDS
- Calculation views provide more opportunities for optimization than CDS
- BI modelers tend not to work with text editors and are more familiar with graphical modeling tools
- Modeling with CDS requires a solid understanding of SQL. This is desirable with calculation views, but not essential



Note:

The key reason for covering CDS briefly in this HA300 course is to simply raise **awareness** amongst BI modelers that an alternative approach to building a virtual data model exists and this was used for SAP S/4HANA. There are dedicated SAP training courses and other resources that cover ABAP and SAP HANA CDS in detail.



LESSON SUMMARY

You should now be able to:

- Understand the Virtual Data Model that is based on CDS views

Unit 4

Learning Assessment

1. The relational model of SAP HANA Live is built using which type of views?

Choose the correct answer.

- A** Scripted calculation views
- B** Calculation views of data category cube with star join
- C** Calculation views of data category cube without star join
- D** Analytic views and attribute views

2. Why is an integrated deployment approach preferred over the side-car approach to SAP HANA Live?

Choose the correct answers.

- A** It provides a quick start for customers using Business Suite on non-SAP HANA databases.
- B** There is no data movement.
- C** Data is available in real time using SLT.
- D** It offers a simplified landscape.

3. How should I ensure that my users do not see values in a sensitive column that is part of a standard SAP HANA Live view?

Choose the correct answer.

- A** Mark the column as ‘hidden’ in the semantic layer of the view.
- B** Use the Extension Assistant to remove the column.
- C** Delete the column from the private view.
- D** Delete the column from the query view.

Unit 4: Learning Assessment

4. Which version of CDS is used by SAP S/4HANA embedded analytics?

Choose the correct answer.

- A** ABAP CDS
- B** SAP HANA CDS

5. What are two reasons why SAP recommends calculation views over CDS for modelers?

Choose the correct answers.

- A** More use cases are available with calculation views than CDS
- B** Not all features and settings relevant to BI are available with CDS.
- C** More optimization possibilities are available with calculation views
- D** Calculation views can define the persistence layer

Unit 4

Learning Assessment - Answers

1. The relational model of SAP HANA Live is built using which type of views?

Choose the correct answer.

- A** Scripted calculation views
- B** Calculation views of data category cube with star join
- C** Calculation views of data category cube without star join
- D** Analytic views and attribute views

Correct — SAP HANA Live is built using only calculation views of data category cube with star join.

2. Why is an integrated deployment approach preferred over the side-car approach to SAP HANA Live?

Choose the correct answers.

- A** It provides a quick start for customers using Business Suite on non-SAP HANA databases.
- B** There is no data movement.
- C** Data is available in real time using SLT.
- D** It offers a simplified landscape.

3. How should I ensure that my users do not see values in a sensitive column that is part of a standard SAP HANA Live view?

Choose the correct answer.

- A** Mark the column as ‘hidden’ in the semantic layer of the view.
- B** Use the Extension Assistant to remove the column.
- C** Delete the column from the private view.
- D** Delete the column from the query view.

Correct — Mark the column as ‘hidden’ in the semantic layer of the view so that it cannot be seen but is still part of the view in case other columns depend on it.

Unit 4: Learning Assessment - Answers

4. Which version of CDS is used by SAP S/4HANA embedded analytics?

Choose the correct answer.

A ABAP CDS

B SAP HANA CDS

Correct — SAP S/4HANA embedded analytics uses ABAP CDS

5. What are two reasons why SAP recommends calculation views over CDS for modelers?

Choose the correct answers.

A More use cases are available with calculation views than CDS

B Not all features and settings relevant to BI are available with CDS.

C More optimization possibilities are available with calculation views

D Calculation views can define the persistence layer

Correct — Not all features and settings relevant to BI are available with CDS and also more optimization possibilities are available with calculation views

UNIT 5

Using SQL in Models

Lesson 1

Introducing SAP HANA SQL

164

Lesson 2

Defining the Persistence Layer using CDS

184

Lesson 3

Working with SQLScript

187

Lesson 4

Query a Modeled Hierarchy Using SQLScript

195

Lesson 5

Creating and Using Functions

198

Lesson 6

Creating and Using Procedures

204

UNIT OBJECTIVES

- Describe SAP HANA SQL
- Define a simple table using CDS
- Work with SQLScript
- Explain the SQLScript implementation logic
- Query a modeled hierarchy using SQL
- Work with functions
- Create and use procedures
- Create a procedure
- Call a procedure

Unit 5

Lesson 1

Introducing SAP HANA SQL

LESSON OVERVIEW

This lesson provides an introduction to the SQL Language in SAP HANA, and covers the following topics:

- Overview of SQL language elements
- Identifiers
- SQL data types
- Predicates and operators
- Functions and expressions
- SQL statement examples

Business Example

As an SAP HANA consultant, you should learn how to perform tasks such as querying source data out of a table or an information model, granting or revoking privileges, copying tables, or altering table column types. Knowledge of SQL will let you perform these tasks. Also, in order to develop sophisticated models, you sometimes need to write SQL and add this to the calculation view in the form of functions.



LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Describe SAP HANA SQL

SQL - Definition and Terminology

Structured Query Language (SQL) is a hugely popular and standardized **database language**, defined by the American National Standards Institute (ANSI). SQL is used to build a database and its tables, to read, update and insert data, and to define security to relational databases.

Importance of SQL in SAP HANA Modeling

SQL plays an important role in SAP HANA modeling and can be implemented in different SAP HANA modeling objects, such as the following:

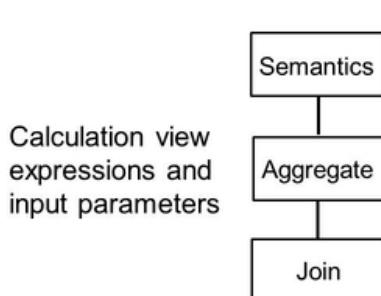
- SQL expressions in calculation views
- Procedures
- Functions
- SQL-based analytic privileges

Also, in order to debug calculation views you should know how to interpret the generated SQL.

So it is very important for SAP HANA modelers to acquire skills in SQL so that they can be fully effective in projects.



Where can we implement SQL in SAP HANA modeling?



Functions



Procedures



Analytic privileges



Figure 129: Where SQL is used in HANA modeling



Note:

In this course, we will not go deeply into the language of SQL but will focus more on where and how we implement SQL in SAP HANA modeling. For a deep dive on the SQL and SQLScript language, refer to the SAP course HA150 – SAP HANA SQL.

Classifications of SQL Statements

For those who are new to SQL let's explore the basics. SQL statements are grouped into the following classifications:

Data definition language (DDL)

DDL is a group of SQL statements to create, modify, and delete database objects such as tables, views, and procedures. You use DDL statements to define the physical layer (create tables) and logical layer (create views).

Data manipulation language (DML)

DML is a group of SQL statements to insert, update, delete, and read the data records, in other words, working on the data and not the database objects.

Data control language (DCL)

DCL is a group of SQL statements to define the database users, roles, and to grant security clearance to the database objects. (For example, who can access a particular table?)

Your SQL code can mix statements from any of the three classifications. For example, in one block of SQL, you could create statements that do the following:

Unit 5: Using SQL in Models

1. Create a table. (DDL)
2. Fill the table with records (DML)
3. Read the records (DML)
4. Grant permissions to other users to read the table (DCL)

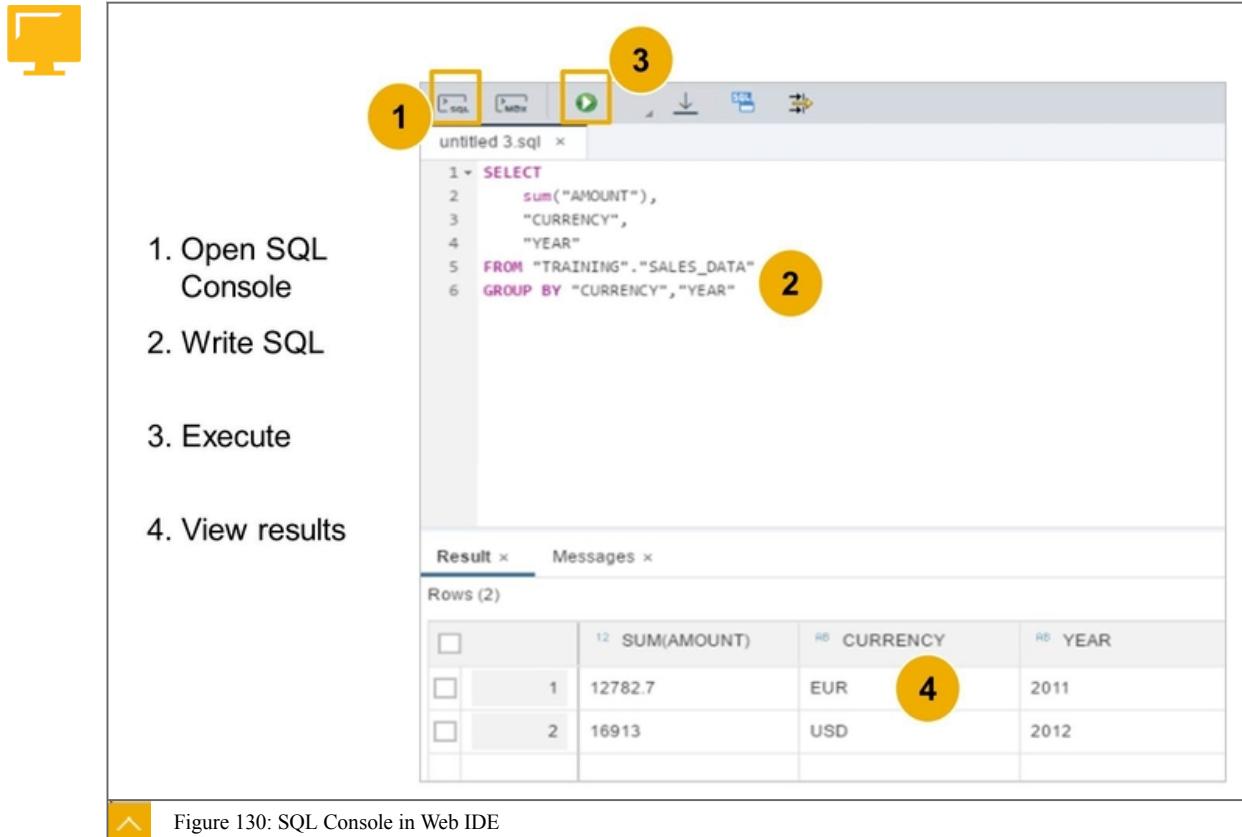
SQL is a descriptive, or sometimes called, declarative language. With SQL you describe what you would like to achieve (the intent), and not how to explicitly achieve it. When executed, the SQL is optimized automatically to achieve the very best performance, which is always its main goal. Depending on the growth of data and even the re-configuration of the hardware (for example, partitioning, scale-out), the optimization may change over time. But your SQL code does not usually need to be changed.

In contrast, procedural languages provide opportunities to tightly control the flow of execution by explicitly stating how something should be achieved. In other words, you are taking over the control of optimization and leaving little or nothing for the optimizer to work on. Procedural languages are sometimes referred to as imperative languages. These are often associated with application development languages such as C++ or JAVA.

Launching the SQL Console

The SQL Console, which is available in Studio and also Web IDE, can be used to write SQL statements and directly execute them.

Although the SQL Console is ideal for executing one-time instructions such as creating a temporary table or executing a test query, it is not recommended for generating official reusable development objects. For this, we recommend you use Core Data Services (CDS) for table creation and we always recommend building calculation views for the data modeling layer.



You can launch the SQL Console from the Database Explorer of SAP Web IDE. Each time you launch the SQL console a new tab appears. You can open as many tabs as you wish.

When you launch an SQL Console in SAP Web IDE from the Database Explorer view, you need to be aware of which database connection you are using because the executed SQL will run against that connection and also the design-time errors will be based on the connection you are using. The database connection for your console is selected in a number of ways:

- Highlight an entry in the database connection list and then press  Open SQL Console button in the toolbar
- Right-click an entry in the database connection list and then choose  Open SQL Console in the context menu
- Expand a database connection and locate a database object then right-click the object and choose an SQL expression from the context menu such as  select

You can clearly see which database connection you are using as this is displayed at the top of the screen.

Regardless of how you chose the connection, you can easily swap the connection by using the  button in the toolbar.

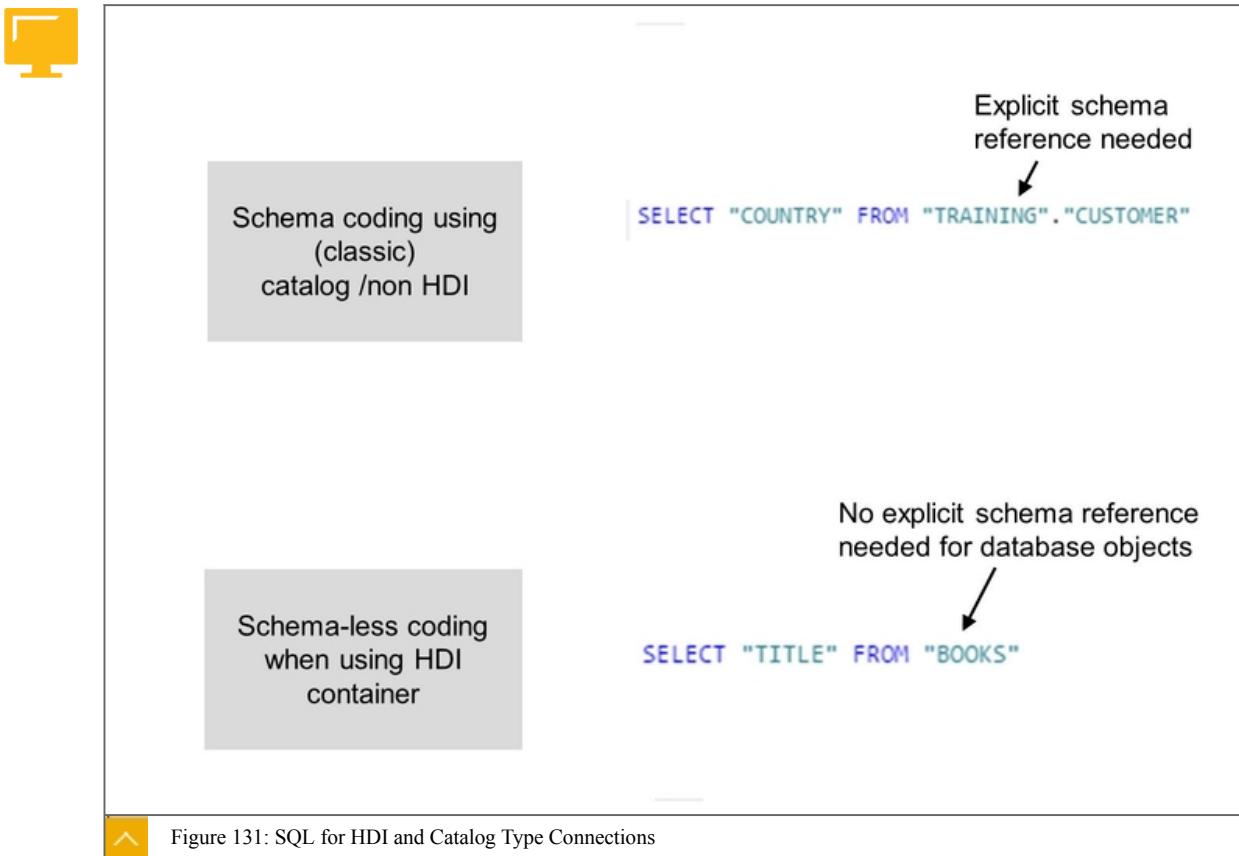
HDI or Classic Database Connections

There are two main types of database connection:

- HDI Container
- SAP HANA Database

It is important that you understand which type you are using for your SQL Console because this affects how you write the SQL code.

Unit 5: Using SQL in Models



When using a HDI Container type, you do not make references to database schema. This is called schema-less coding. Of course the SQL is always executed against a database schema, but the schema is determined automatically from the HDI container. It is essential that you **not** refer to any schema when using HDI containers because the schema might change in future and your code will no longer work.

do

If you are using a connection that is a SAP HANA database type, then you must explicitly reference the schema, unless you are using the user default schema.

SQL Console Auto-Complete and Other Productivity Aids

There are a number of productivity aids built into SAP Web IDE.



1. Type part of an SQL statement

```
untitled 3.sql x
1 ~ sel
```

2. CTRL + SPACE

untitled 3.sql x
1 ~ sel

SELECT STATEMENT

- SELECT LIMIT OFFSET STATEMENT
- SELECT INNER JOIN STATEMENT
- SELECT LEFT JOIN STATEMENT
- SELECT RIGHT JOIN STATEMENT
- SELECT FULL JOIN STATEMENT
- SELECT ROW_NUMBER STATEMENT
- SELECT SYSUUID STATEMENT
- WITH SELECT
- WITH SELECT MULTI

3. Code template is presented with placeholders

```
untitled 3.sql x
x 1 SELECT * FROM <TABLE>
```



Figure 132: SQL Code Completion in SAP Web IDE

As the figure SQL Code Completion in SAP Web IDE shows, the SQL Console includes an auto-completion feature, which can be activated by pressing **Ctrl + SPACE** while entering a command.

The auto-complete feature is also available in the code editor when you create functions and procedures. This speeds up the writing of SQL by building the basic statement structure for you, and ensures that you include all important parameters. Simply enter an expression such as `select` or a function such as `daysbetween` and then press **Ctrl + SPACE** to use auto-complete so that you see the placeholders that represent the required parameters. You don't even need to type the entire word – for example if you type the letters `le` the selection list will include multiple options including `Left()`, `Left Outer Join`, `Length()` and so on.

There is no validation button or menu option in the SQL Console because SQL code is constantly checked in real-time for correctness as you type. If there are design-time errors, a red alert icon appears on the left of the line in error. You can hover over the icon and read the error text. Simply fix the problem and the red alert icon disappears. For example, a missing bracket would cause a syntax design-time error. The console is even smart enough to know that a statement, although syntactically correct, is still invalid if it refers to a table that does not exist. Also, even if there are no design-time errors, you may still receive run-time errors when you execute the SQL statement. For example, you may have forgotten to add an attribute to the GROUP BY clause that is used in a SELECT statement which contains a sum on a measure.

At run-time, if errors are encountered, at the first error you will be prompted and given three options to choose how you wish to proceed:

1. Ignore this statement and continue to the next statement
2. Ignore this statement and all other statements with errors

Unit 5: Using SQL in Models

3. Abort the execution immediately



Hint:

Sometimes you only want to execute a snippet of code or a single statement in the console. If so, then simply highlight a section of SQL code and the hit F8 or use the execute button in the toolbar. Then, only that highlighted code is executed. If you do not highlight any code then all code in the SQL console is processed when you execute. This is a great way of testing single statements.



Hint:

When you need to include the name of a database object in your code, rather than type it in manually, just drag and drop it from the database explorer connection on the left panel.

Example of an SQL Statement

**-- Find a list of SAP courses about HANA**

```
SELECT name, description, cost
FROM courses
WHERE company = 'SAP'
      AND description LIKE '%HANA%';
/*
```

Look at the comments, SELECT statement, identifiers (name, description, cost, company), predicates (= and LIKE), and operators (AND)

***/**

Figure 133: Example of an SQL Statement

Let's have a look the example SQL statement in the figure, Example of an SQL Statement:

- All text after a double hyphen (--) or between the /* and */ text is seen as comments. In this case, you specified in the top line what the statement does.
- The SELECT statement is the most used statement when writing SQL code. It allows you to read data from a data source, as follows:

- You first specify which columns (fields) you want to read from the data source. In this case, you want to read the name of the course, the course description, and the cost of the course. You can read ALL the columns (fields) from a data source by writing SELECT *. This is not recommended, especially for columnar databases like SAP HANA because it can degrade performance.
- You can specify the name of the data source in the FROM clause, which in this case is a table named courses.
- You can then restrict the amount of data returned by specifying a WHERE clause. In this case you only want to see courses from SAP.
- You can further restrict the data returned by additional clauses using the AND and OR operators.
- Finally, you can specify that you only want SAP courses where the description field contains the word HANA somewhere in the text. You do this using the LIKE predicate.
- The field names in this statement can be referred to as identifiers.

SQL Language Elements

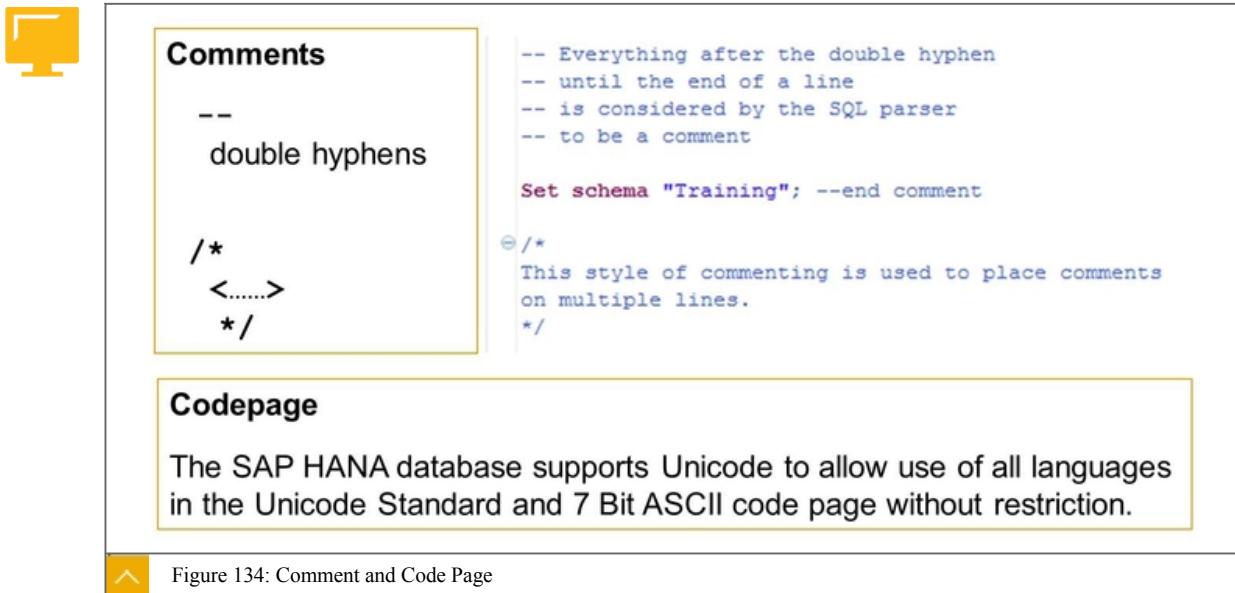
The table SQL Language Elements defines the elements that make up SQL.

Table 11: SQL Language Elements

Element	Description
Identifiers	Used to represent object names used in SQL statements such as table names or column names
Data type	Specify the characteristics of a data value such as NVARCHAR or DECIMAL.
Expressions	Clause that can be evaluated to return values for example SELECT or GROUP BY
Operators	Used in expressions to perform operations for example >=, , *, <>
Predicates	Combine one or more expressions to return either TRUE, FALSE or UNKNOWN, Use predicates such as BETWEEN, IN, CONTAINS, LIKE
Functions	Used in expressions to return information from the database using pre-programmed logic for example daysbetween().

Unit 5: Using SQL in Models

Comment and Code Page



The diagram illustrates two sections: **Comments** and **Codepage**.

Comments:

- double hyphens: `--` Everything after the double hyphen until the end of a line is considered by the SQL parser to be a comment.
- /* <.....> */: This style of commenting is used to place comments on multiple lines.

Codepage:

The SAP HANA database supports Unicode to allow use of all languages in the Unicode Standard and 7 Bit ASCII code page without restriction.

As the figure, Comments and Code Page shows, comments are delimited in SQL statements as follows:

- Everything from a double hyphen (`--`) to the end of a line is parsed as a comment.
- Everything between `/*` and `*/` is also parsed as a comment. This style of commenting is used to place comments on multiple lines.

The SAP HANA Database supports Unicode to allow use of all languages in the Unicode Standard and 7 Bit ASCII code page without restriction.

Identifiers

You use identifiers to represent names used in SQL statement including the following examples:

- Table names
- View names
- Column names
- Synonym names
- Index names
- Function names
- Procedure names
- User names
- Role names

Delimited and Undelimited Identifiers



Undelimited Identifiers	<ul style="list-style-type: none"> must start with a letter cannot contain any symbols other than digits or an underscore _ 	TRAINING Training (treated with upper case!) 1TRAINING TRAINING%
Delimited Identifiers	<ul style="list-style-type: none"> enclosed in the delimiter double quotes "<identifier>" can contain any character including special characters. 	"TRAINING" "Training" (case sensitive) "1Training" "Training%"
Limitations	<ul style="list-style-type: none"> _SYS_ is reserved exclusively for database engine Role name and user name must be specified as undelimited identifiers. Maximum length for the identifiers is 127 characters. 	

Figure 135: Identifiers

As the figure Identifiers shows, there are two kinds of identifiers: undelimited and delimited.

Undelimited table and column names must start with a letter and cannot contain any symbols other than digits or an underscore _.



Note:

Undelimited identifiers are implicitly treated as upper case. When quoting identifiers, it is possible to use spaces and lower or mixed case in identifiers.

As the figure shows, delimited identifiers are enclosed in the delimiter double quotes, and the identifier can contain any character, including special characters. For example, →AB\$%CD→ is a valid identifier name.



Note:

Delimited identifiers are case sensitive. SAP HANA is case sensitive on database objects such as the names of tables and fields. For example, you can have a table called "DATA" and another called "data". These will be treated like two different tables.

Delimiters can be delimited by single quotation marks, or double quotation marks, as follows:

Single quotation mark

Single quotation marks are used to delimit string literals and a single quotation mark itself can be represented using two single quotation marks.

Double quotation mark

Double quotation marks are used to delimit identifiers and double quotation mark itself can be represented using two double quotation marks.

Unit 5: Using SQL in Models

You can see the different delimiters in the following example:

```
select 'String', 'SAP''s current offering' from "DUMMY"
```

Data Types

Data types are used to specify the characteristics of the data stored in the database, for example, a string can be stored as a NVARCHAR data type. The table, SQL Data Types, lists some of the data types available in SQL. Data types specify the characteristics of a data value.

Table 12: SQL Data Types

Classification	Standard ANSI-92 Data Type
Date/Time	DATE, TIME, TIMESTAMP
Numeric	DECIMAL, INTEGER, SMALLINT, FLOAT, REAL, DOUBLE
Character String	CHAR, NCHAR, VARCHAR, NVARCHAR
Binary	BINARY, VARBINARY



Note:

A special value of NULL is included in every data type to indicate the absence of a value.

Predicates

Predicates are typically used in the WHERE clause of a SELECT statement.



Comparison Predicates	<expression> { = != <> > < >= <= } [ANY SOME ALL] { <expression_list> <subquery> }
BETWEEN Predicate (range)	<expression1> [NOT] BETWEEN <expression2> AND <expression3>
IN Predicate	<expression> [NOT] IN { <expression_list> <subquery> }
EXISTS Predicate	[NOT] EXISTS (<subquery>)
LIKE Predicate	<expression1> [NOT] LIKE <expression2> [ESCAPE <expression3>]
NULL Predicate	<expression> IS [NOT] NULL



Figure 136: SQL Predicates

A predicate is specified by combining one or more expressions or logical operators and returns one of the following logical or truth values:

TRUE, FALSE, or UNKNOWN.

The figure, SQL Predicates, shows the correct syntax for predicates.

The LIKE Predicate

The `LIKE` predicate is used for string comparisons. A value, `expression1`, is tested for a pattern, `expression2`. Wildcard characters (`%`) and (`_`) may be used in the comparison string `expression2`. `LIKE` returns true if the pattern specified by `expression2` is found. The percentage sign (`%`) matches zero or more characters and underscore (`_`) matches exactly one character. To match a percent sign or underscore in the `LIKE` predicate, an escape character must be used. Using the optional argument `ESCAPE expression3`, you can specify the escape character that will be used so that the underscore (`_`) or percentage sign (`%`) can be matched.

Operators



Unary	operator operand	unary plus operator(+) unary negation operator(-) logical negation(NOT)
Binary	operand1 operator operand2	multiplicative (<code>*</code> , <code>/</code>), additive (<code>+</code> , <code>-</code>) comparison operators (<code>=</code> , <code>!=</code> , <code><</code> , <code>></code> , <code><=</code> , <code>>=</code>) logical operators (<code>AND</code> , <code>OR</code>)
Arithmetic Operators	<code>-< expression ></code> <code>< expression > operator < expression ></code>	Negation, Addition, Subtraction Multiplication, Division
String Operator	<code>< expression > < expression ></code>	String concatenation
Comparison Operators	<code><expression> operator <expression></code>	<code>>=</code> Greater or equal to <code><=</code> Less than or equal to <code>!=</code> , <code><></code> Not equal
Logical Operators	Search conditions can be combined using <code>AND</code> or <code>OR</code> operators. You can also negate them using the <code>NOT</code> operator.	<code>AND</code> , <code>OR</code> <code>NOT</code>
Set Operators	Set operators perform operations on the results of two or more queries.	<code>UNION</code> , <code>UNION ALL</code> , <code>INTERSECT</code> , <code>EXCEPT</code>

Figure 137: Operators

You can perform operations in expressions by using operators. Operators are grouped into categories, such as Comparison Operators or Logical Operators. Operators can be used for calculation, value comparison, or to assign values.

Unit 5: Using SQL in Models

SQL Functions



**-- Find a list of SAP HANA courses
-- for the next 4 weeks (28 days)**

```
SELECT name, description, cost, date,  
DAYS_BETWEEN(CURRENT_DATE, Date) AS  
in_x_days  
FROM courses  
WHERE company = 'SAP'  
AND description LIKE '%HANA%'  
AND DAYS_BETWEEN(CURRENT_DATE, Date) <= 28;
```



Figure 138: An Example of SQL Functions

The figure, An Example of SQL Functions, gives an example of a simple SQL function.

SQL Functions

Table 13: Functions

Classification of the Functions	Examples
Data type conversion	CAST, TO_APLHANUM, TO_BIGINT, ...
Date and Time	ADD_DAYS, ADD_MONTHS, ADD_YEARS, DAYS_BETWEEN, DAYNAME, CURRENT_DATE, ...
Number	ABS, ACOS, ASIN, ATAN, COS, SIN, TAN, ...
String	CONCAT, LEFT, LENGTH, TRIM, ...
Miscellaneous	IFNULL, CURRENT_SCHEMA, ...

Functions provide a reusable method to evaluate expressions and return information from the database. They are allowed anywhere an expression is allowed. Functions use the same syntax conventions used by SQL statements. The table Functions lists the main functions available in SQL.

Expressions

An expression is a clause that can be evaluated to return values, as shown in the figure, An Example of an SQL Expression Using COUNT.



-- Find out how many unique SAP HANA courses
-- are available

```
SELECT COUNT(DISTINCT name) as num_courses
FROM courses
WHERE company = 'SAP'
AND description LIKE '%HANA%';
```



Figure 139: An Example of an SQL Expression Using COUNT

Types of Expressions in SQL

The table Expressions lists the types of expressions used in SQL.

Table 14: Expressions

Expression Type	Description
Case Expressions	Allows the user to use IF ... THEN ... ELSE logic without using procedures in SQL statements.
Function Expressions	Allows SQL built-in functions to be used as an expression.
Aggregate Expressions	Uses an aggregate function to calculate a single value from the values of multiple rows in a column.
Subqueries in expressions	A SELECT statement enclosed in parentheses. The SELECT statement can contain only one select list item. When used as an expression, a scalar subquery is allowed to return only zero or one value.

Unit 5: Using SQL in Models

Expressions: Examples

**Case expression**

You can use IF ... THEN ... ELSE logic without using procedures in SQL statements.

```
<expression> ::= CASE <expression>
                  WHEN <expression>
                  THEN <expression>, ...
                  [ ELSE <expression> ]
                  { END | END CASE }
```

Aggregate Expressions

```
<aggregate_expression> ::= COUNT(*) | <agg_name> ( [ ALL | DISTINCT ] <expression> )
<agg_name> ::= COUNT | MIN | MAX | SUM | AVG | STDDEV | VAR
```



Figure 140: Examples of Expressions

As the figure, Examples of Expressions shows, in a case expression, if the expression following the CASE statement is equal to the expression following the WHEN statement, then the expression following the THEN statement is returned. Otherwise the expression following the ELSE statement is returned, if it exists.

As the figure also shows, an aggregate expression can contain the following functions:

- COUNT: Counts the number of rows returned by a query. COUNT (*) returns the number of rows, regardless of the value of those rows and including duplicate values, whereas COUNT (<expression>) returns the number of non-NULL values for that expression returned by the query.
- MIN: Returns the minimum value of an expression.
- MAX: Returns the maximum value of an expression.
- SUM: Returns the sum of an expression.
- AVG: Returns the arithmetical mean of an expression.
- STDDEV: Returns the standard deviation of a given expression as the square root of the VARIANCE function.
- VAR: Returns the variance of an expression as the square of a standard deviation.

Creation of Tables

The figure, SQL Statement: Create Table, shows the syntax you use to create a table.



```

CREATE [<table_type>] TABLE <table_name> <table_contents_source>;
::= COLUMN | ROW | HISTORY COLUMN | GLOBAL TEMPORARY | LOCAL TEMPORARY
::=(<table_element>, ...)[(column_name, ...)]|<like_table_clause>|<as_table_subquery>|[WITH [NO] DATA]
::= column_definition column_constraint | table_constraint (column_name, ...)
like_table_clause ::= LIKE like_table_name
as_table_subquery ::= AS (<select_query>)
column_definition ::= column_name data type [<column store data type>]<ddic data type>[DEFAULT default_value][GENERATED ALWAYS AS <expression>]

```



Figure 141: SQL Statement: Create Table

Table Types

The figure, Create Table – Table Types, lists the types of table that you can create with SQL.



COLUMN	COLUMN-based storage should be used, if the majority of access is through a large number of tuples but with only a few selected attributes.
ROW	ROW-based storage is preferable, if the majority of access involves selecting a few records with all attributes selected.
HISTORY COLUMN	Creates a table with a particular transaction session type called <i>HISTORY</i> . Tables with session type <i>HISTORY</i> support time travel ; the execution of queries against historic states of the database is possible.
GLOBAL TEMPORARY	Table definition is globally available while data is visible only to the current session. The table is visible in the catalog and is dropped at the end of the session.
LOCAL TEMPORARY	The table definition and data is visible only to the current session. The table is not visible in the catalog and is dropped at the end of the session. Table name must begin with the hash (#) symbol.



Figure 142: Create Table – Table Types

The history column table is particularly interesting because this allows SAP HANA to store all previous values of a record. Instead of updating a record when you edit it and thereby losing the previous values, SAP HANA keeps the previous record if the table type is a history column table. It makes the previous record with a `valid_to` time and date field, and inserts the record with the new values with a `valid_from` date and time field.

This allows you to go back in time and see what the table looked like at a specific point in the past. In future, you can expect to see many more features in the business systems where SAP will leverage this functionality, for example, for slowly changing dimensions or month-end processes. You can see an example of a history column table in the figure, Use a History Column Table.

Unit 5: Using SQL in Models

Use a History Column Table



-- Show the SAP HANA courses that
-- were available on January 1st, 2015

```
SELECT name, description, cost, date
FROM courses_history
WHERE company = 'SAP'
AND description LIKE '%HANA%'
AS OF UTCTIMESTAMP '2015-01-01
12:00:00';
```



Figure 143: Use a History Column Table

Syntax Elements



```
SQL
1 -- Preparation
2 -- Replace ## with your group number (e.g. STUDENT##)
3 SET SCHEMA STUDENT##;
4
5
6 -- STEP 1
7 drop table publishers;
8
9 CREATE COLUMN TABLE publishers
10   ( pub_id INTEGER PRIMARY KEY,
11     name VARCHAR (50),
12     street VARCHAR (50),
13     post_code VARCHAR (10),
14     city VARCHAR (50),
15     country VARCHAR (50));
16
17 drop table "publishers";
18
19 CREATE COLUMN TABLE "publishers"
20   ( pub_id INTEGER PRIMARY KEY,
21     name VARCHAR (50),
22     street VARCHAR (50),
23     post_code VARCHAR (10),
24     city VARCHAR (50),
25     country VARCHAR (50));
```



Figure 144: Create Table – Syntax Elements

The figure, Create Table – Syntax Elements, shows the syntax used in creating a table. Note the following:

```
column_constraint ::= NULL | NOT NULL | UNIQUE [BTREE | CPBTREE] |
PRIMARY KEY [BTREE | CPBTREE]

table_constraint ::=UNIQUE [BTREE | CPBTREE] | PRIMARY KEY [BTREE | CPBTREE]
```

SQL Statement: Insert



```

INSERT INTO <table_name> [ ( column_name, ... ) ]
{ VALUES (expr, ... ) | <subquery> };

15einsert into publishers VALUES
16      ( 1, 'Oldenburg Wissenschaftsverlag GmbH',
17      'Rosenheimer Strasse 145',
18      '81671',
19      'Muenchen',
20      'Germany');
21einsert into publishers VALUES
22      ( 2, 'Pearson Education Deutschland GmbH',
23      'Martin-Kollar-Strasse 10-12',
24      '81829',
25      'Muenchen',
26      'Germany');

insert
into "DOM_STATV"
select
domvalue_1
from dd071
where domname = 'STATV'
and as4local = 'A';

```

 Figure 145: SQL Statement: Insert

The figure, SQL Statement: Insert, gives an example of this statement.

SQL Statement: Select

In the following example of a select statement, note that anything in square brackets is an optional clause of the SELECT statement.

```

SELECT [TOP number] [ALL | DISTINCT]
<select_list>
<from_clause>
[<where_clause>]
[<group_by_clause>]
[<having_clause>]
[<order_by_clause>]
[<limit_clause>]
[<for_update_clause>]
[<time_travel_clause>];

```

In particular, note the operation of the following optional clauses:

- `<for_update_clause> ::= FOR UPDATE [OF <column_name>,]`

The FOR UPDATE keyword locks the selected rows so that other users cannot lock or update the rows until end of this transaction.

- `<time_travel_clause> ::= AS OF [COMMIT ID|TIMESTAMP] [<commit_id> | <timestamp>]`

Can be used for statement level time travel to go back to the snapshot specified by commit_id or timestamp.

Implementing SQL in Calculation Views

As the figure Use of SQL in Calculation View Expressions shows, SQL can be used to create expressions in calculation views to define the following:

Unit 5: Using SQL in Models

- Calculated columns
- Filters
- Restricted columns

In addition, SQL can be used to return values for input parameters. This is very powerful because input parameters are used in many places throughout calculation views as dynamic placeholders for filters, ranking, user prompts, calculations and more.



Note:

Variables do not make use of SQL to return values.



Use of SQL in Calculation View expressions



Figure 146: Use of SQL in Calculation View Expressions

It is essential for modelers to grasp the basics of SQL because they will encounter SQL often throughout their modeling activities. Pay particular attention to the very long list of available functions that are available with SQL. These functions can provide significant additional options when you are trying to develop some logic where data must be manipulated. For example, SQL provides many string manipulation functions that are useful for re-formatting a field, or extracting some characters from it. Also, there are many predefined data functions available in SQL that can help you to calculate with dates, for example, to find the number of days between two dates.

You may be left wondering if you could even abandon calculation views completely and instead write the data models using only SQL. To a large extent this would be possible but remember, calculation views carry a lot of valuable metadata including many flags and settings that provide the optimizer with run-time hints for optimization and to ensure accurate results under different query conditions. Plus there are graphical tools for lineage

and impact analysis when using calculation views, to support developers. And of course, who wants to read through lines of code in order to figure out what a model is actually doing?



Note:

SQL expressions in calculation views uses plain SQL and not SQLScript.



LESSON SUMMARY

You should now be able to:

- Describe SAP HANA SQL

Unit 5

Lesson 2

Defining the Persistence Layer using CDS



LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Define a simple table using CDS

Building the persistence layer using design time objects

What is the persistence layer?

Before we begin, let's first define the persistence layer .

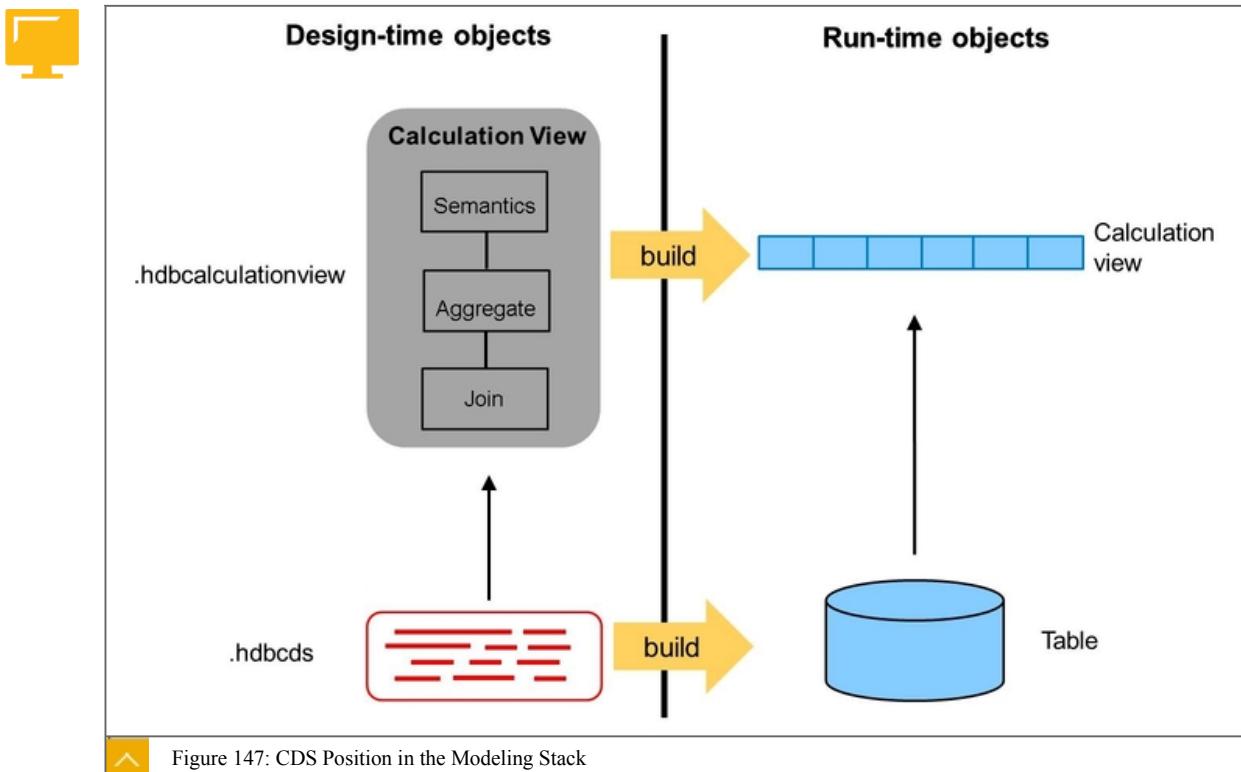
The persistence layer of SAP HANA refers to the physical data storage layer that is typically built using tables. As a point of reference, if you imagine a modeling stack, the persistence layer sits at the bottom of the stack (assuming you don't include the ETL design in that stack). On top of the persistence layer, we build the data modeling layer, where we introduce calculation views.

We know that it is possible to define the SAP HANA persistence layer using standard SQL data definition language (DDL) such as **Create Table**. In the SQL Console we enter these SQL statements and execute them directly. But when you close the SQL Console the SQL statements are gone. Whilst this approach produces the desired end result, the code you enter into the SQL Console cannot be stored in a way that enables managed transportation to another SAP HANA system to recreate the same database objects. In this case you would have to re-enter the SQL code again in the SQL Console of each SAP HANA system and reactivate in order to generate the same persistence object.

As well as being inefficient, there is another problem that comes with creating persistence objects directly using the SQL Console. Persistence objects should be part of an SAP Web IDE project so that they are tied closely with other objects in the same project. For example, we should define the tables, calculations views, functions and all other development artifacts in a single project. By doing this we ensure better integrity as all dependant objects are checked and built together as a single unit.

You need a way to describe the persistence objects using source code and then put this code in a HANA-recognized file that is part of an overall development project, so that it can be transported and activated in any SAP HANA target system.

Creating Tables with CDS



CDS allows you to describe the persistence layer starting with the most primitive elements. These elements can include a simple `type` that represents a generic re-usable field such as `amount`, or `product` where you define basic technical parameters such as length and data type. At this point the `type` has no business context.

Once you have built the primitive types, you build semantic types, such as `tax amount`, that are defined by consuming the primitive types. In this case, the type `tax amount` consumes the primitive type `amount`, but many more semantic types could also consume the same primitive type `amount`.

Finally, with CDS, you can build tables that consume the semantic types. For example, a table `sales orders` consumes the semantic type `tax amount` and many other related semantic types such as `shipping amount` and `requested product`.

A key benefit of this consumption approach is that changes to the lowest level of the model, in this case the primitive type `amount`, affect all consumers of this type. Changing the length of the type in just one place instead of making the change separately in each table of the database is more time efficient, and gives greater consistency across entire applications.

But you could also keep things simple by defining a single CDS file that defines the table, explicitly specifying each column, data type and length and avoid consuming any other pre-defined `types`. However, if you change the definition of a field, you then need to make this change in all tables where this field exists. So a normalization approach using consumption is always recommended for CDS.

In order to drop a table built using CDS, you should never directly delete the runtime object. You simply delete the design-time object and the next time you build the project the table is dropped.

Unit 5: Using SQL in Models

Another thing to note: when you define a table using a CDS design-time file in an XSA or HDI project, you never specify the database schema. The generated table actually belongs to the project's container which in turn has its own database schema. You can check the generated table if you know the name of the container's schema. The schema name is usually includes your project name plus the words 'hdi-container'.

Note:

Detailed coverage of CDS is not in the scope of this course. Modelers often do not need to define the persistence layer, so this may interest only a minority of students. You can find online resources to cover all aspects of CDS, including openSAP courses and the SAP course: HA450 SAP HANA Application Development.

There is some history behind the creation of tables using source file definitions. A few years ago, SAP originally introduced a design-time file with the extension **.HDBTABLE** to support the creation of source file definitions of tables. When activated, it generated the required table in the specified schema of the SAP HANA catalog. This approach was only valid for classic XS or repository developments. **.HDBTABLE** is not supported by HDI or XSA developments, and in fact SAP Web IDE does not recognize this file type. A little later, SAP then introduced another design-time file with the extension **.HDBDD** that was based on the core data services framework. This was the new recommended file type to use, instead of **.HDBTABLE**. But again, this file type was only for classic XS or repository developments and not HDI or XSA projects. Again, SAP Web IDE does not recognize this file type.

So We Never Use SQL Console to Create Tables?

Creating tables and other persistence objects directly using the SQL Console approach can be useful in a few cases. For example, you might need a simple table for a one-time purpose, perhaps in testing or checking a concept. Building a complete, reusable CDS based persistence model may be overkill in these cases. Many modelers will already be very familiar with the SQL syntax for creating, altering, truncating, and dropping tables and will be comfortable using that. But for all tables that will be used in production, we recommend always using CDS.

CDS Usage Beyond the Persistence Layer

We have described our recommendations for the use of CDS to create the persistence layer in SAP HANA modeling. We then build our calculation views on top of this layer. But as well as defining the persistence layer, CDS even goes on to support the building of the modeling layer too. Now this is where core modeling with SAP HANA collides with CDS based modeling.

We can build modeled views using CDS that can be built in stacks. But for pure BI cases, SAP recommends using calculation views which provide better performance optimization possibilities compared to CDS views. Also, graphical calculation views provide more support for pure BI type roles with well developed tooling such as graphical editors, performance analysis mode, lineage and so on. Building the data model layer using CDS views is more likely to appeal to application developers who are familiar with a code level development environment.

**LESSON SUMMARY**

You should now be able to:

- Define a simple table using CDS

Unit 5

Lesson 3

Working with SQLScript

LESSON OVERVIEW

This lesson will give you a general introduction to SQLScript. You will learn about the benefits of SQLScript and how you can use it in the context of SAP HANA modeling which includes functions and procedures.

Business Example

You have created graphical calculation views, but you now require additional functionality that can only be found in SQLScript.



LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Work with SQLScript
- Explain the SQLScript implementation logic

Why Do We Need SQLScript?

SQLScript is language developed by SAP that takes standard SQL, and adds additional capabilities to allow developers to move more of the data-intensive logic away from the application server and into the database. The figure, SQLScript: Concept, lists some of the key features of SQLScript.

Unit 5: Using SQL in Models



- **SQLScript** extends ANSI-92 SQL. Just like other database vendors extend SQL in their databases, SAP HANA extends SQL using SQLScript.
- **SQLScript** enables you to access SAP HANA-specific features like column tables, parameterized information views, delta buffers, working with multiple result sets in parallel, built-in currency conversions at database level, fuzzy text searching, spatial data types, and predictive analysis libraries.
- **SQLScript** allows developers to push data intensive logic into the database
- **SQLScript** encourages developers to maintain algorithms using a set-oriented paradigm, instead of a one record at a time paradigm
- **SQLScript** does however allow looping through results and using if-then-else logic
- **SQLScript** allows you to break complex queries into multiple smaller statements, thereby simplifying your SQL coding, and enhancing parallelism via the optimizer.



Figure 148: SQLScript: Concept

SQLScript exposes many of the in-memory features of SAP HANA to the SQL developers. These features include column tables, parameterized information views, delta buffers, working with multiple result sets in parallel, built-in currency conversions at database level, fuzzy text searching, spatial data types, and predictive analysis libraries. The only way to make these features available via SQL queries, even from other applications, is to provide them as extensions to the standard SQL language.

SQLScript Extends SQL

In traditional client-server approaches, the business logic is executed in the application server. With SAP HANA, much of this logic and execution is pushed down into the SAP HANA database. This approach is very different from the standard SQL way of working. SQLScript caters for these requirements.

SQLScript allows you to use variables to break a large complex SQL statement into smaller, simpler statements. This makes the code much easier to understand, and it also helps with SQL HANA's performance because many of these smaller statements can be run in parallel.

Let's have a look at an example.

```
books_per_publisher = SELECT publisher, COUNT (*) AS num_books FROM BOOKS GROUP BY publisher;

publisher_with_most_books = SELECT * FROM :books_per_publisher WHERE num_books >= (SELECT MAX (num_books) FROM :books_per_publisher);
```

You normally write this example as a single SQL statement using a temporary table, or by repeating a sub-query multiple times. In this example, however, you break this into two smaller SQL statements by using table variable.

The first statement calculates the number of books each publisher has and stores the entire result set into the table variable called `books_per_publisher`.

This variable, containing the entire result set, is used twice in the second statement.

Notice that the table variable is prefixed in SQLScript with a colon (':') to indicate that this is used as an input variable. All output variables just have the name, and all input variables have a colon prefix.

The second statement uses `:books_per_publisher` as inputs, and uses a nested `SELECT`.

The SQLScript compiler and optimizer determine how to best execute these statements, whether by using a common sub-query expression with database hints or by combining this into a single complex query. The code becomes easier to write and understand, more maintainable, and developer productivity increases.

By breaking the SQLScript into smaller statements and filling table variables, you also mirror the way in which you have learned to build your calculation views. Just like when you start building calculation views in layers, starting from the bottom up, you do the same with your SQLScript code. Keep in mind that the precise sequence of your SQLScript steps is not necessarily the runtime order, because the optimizer will always decide on the best execution plan. However, the optimizer will never alter your desired outcome. This is also true for calculation views.

To a large extent, what you create in calculation views, you can also create using SQLScript.

For example, in graphical calculation views you create union nodes. But you can also create unions with SQLScript, using the `UNION` operator, as shown in the following example:

```
SELECT author, title, price FROM BOOKS
UNION
SELECT author, title, price FROM BOOKS_OUT_OF_PRINT;
```

 Note:

You can also use the `UNION ALL` operator instead of `UNION`. The `UNION` operator returns only distinct values, whereas the `UNION ALL` will also return all duplicate records.

SQLScript Data Types

SQLScript adds extra data types to help address missing features from standard SQL, for example, spatial data types and text data types. These data types are listed in the table ANSI-92 and SQLScript data types.

Unit 5: Using SQL in Models



Classification	Standard ANSI-92 Data Types	SQLScript-specific Data Types
Date/Time	DATE, TIME, TIMESTAMP	SECONDDATE
Numeric	DECIMAL, INTEGER, SMALLINT, FLOAT, REAL, DOUBLE	TINYINT, BIGINT, SMALLDECIMAL
Character string	CHAR, NCHAR, VARCHAR, NVARCHAR	ALPHANUM, SHORTTEXT
Binary	BINARY, VARBINARY	
Large Object		BLOB, CLOB, NCLOB, TEXT, BINTEXT
Boolean		BOOLEAN
Spatial		ST_POINT, ST_GeOMETRY



Figure 149: ANSI-92 and SQLScript Data Types

Table Type Extension

**Table Type:**

SQLScript's datatype extension also allows the definition of table types. These table types are used to define parameters for a procedure.

A table type is created using the **CREATE TYPE** and delete using **DROP TYPE** statement.

Syntax:

**CREATE TYPE [schema.]name AS TABLE
(name1 type1 [, name2 type2,...])**

DROP TYPE [schema.]name [CASCADE]



Figure 150: SQLScript: Data Type Extensions

SQLScript also allows the definition of table types, as shown in the figure, Data Type Extensions. Table types are used to define tabular structures used by procedures and functions to pass data during input or output. Table types represent tabular results. Unlike a regular table, no data persists in a table type; they are used to pass tabular data in a data flow using SQLScript. It might help to think of them as internal data structures. The runtime objects associated with table types can be created and dropped as required using the Create Type and Drop Type syntax.

Create

The Use of Variables

The use of variables in SQL statements is not specified by the ANSI-92 SQL. In SAP HANA, SQLScript implements the concept of variable as an extension to ANSI-92 SQL.

Variables can be declared, assigned, and reused within a script.

The list, Using Variables in SQLScript, gives an overview of how variables are used within SQLScript.

Using Variables in SQLScript



- Declare a variable

```
DECLARE <variable_name> <type> [NOT NULL] [= <value>]
```

Example 1: `DECLARE a int;`

Example 2: `DECLARE b int = 0;` (the value is assigned when creating the variable)

- Assign a variable (define the value of a variable)

```
<variable_name> = <value> or <expression>
```

Example 1: `a = 3;`

Example 2: `b = select count(*) from baseTable;`

- Use a variable in an expression

`:<variable_name>` (with a colon) returns the current value of the variable

Example: `b = :a + 5;` (assigns the value of `a + 5` to `b`)



Note:

These examples are simplified and do not cover the complete SQLScript syntax for variables. You can find more information in the SAP HANA SQLScript Reference Guide, available at <http://help.sap.com/hana>

Processing Input Parameters Defined in Calculation Views with SQLScript

When you use a `select` statement in SQLScript to query a calculation view that has input parameters, you use the `PLACEHOLDER` keyword with the name of the input parameter wrapped in `$$` symbols, and the value you want to pass to the input parameter. For example:

Using Input Parameters in SQLScript

Use the `PLACEHOLDER` keyword to pass input parameters to the calculation view in a `select` statement:

```
SELECT "PRODUCT", sum("TAXED_AMOUNT")
    FROM "_SYS_BIC"."MyPackage/SalesResults"
        ('PLACEHOLDER' = ('$$Tax Amount$$', '17.5'))
GROUP BY "PRODUCT"
```

For functions and procedures you refer to the input parameter using a colon:

```
Sales with tax = Sales Revenue * :Tax_Amount
```

Imperative Logic

Occasionally, you might prefer to receive the answers from a database query one record at a time. This is most often the case when running SQL queries from application servers. In these cases, you will use imperative logic, meaning that you will loop through the results one record at a time. We can also use conditional statements such as `If-then` to control the flow logic direction.

This is (mostly) not available in ANSI SQL, and therefore SAP HANA provides this as part of SQLScript to provide flow control.

Unit 5: Using SQL in Models



Hint:

Remember that imperative logic cannot have the same performance as declarative logic because the SQL optimizer is not free to decide on the sequence of steps and has to take care to follow your logic which might destroy optimizations. If you require the best performance, try to avoid imperative logic.

Control Statements and Dynamic SQL



```
IF <bool-expr1>
  THEN
    {then-stmts1}
  {ELSEIF <bool-expr2>
  THEN
    {then-stmts2}}
  {ELSE
    {else-stmts3}}
END IF
```

Example:

```
SELECT count(*) INTO found
  FROM books WHERE isbn = :v_isbn;
  IF :found IS NULL THEN
    CALL ins_msg_proc
      ('result of count(*) cannot be NULL');
  ELSE
    CALL ins_msg_proc
      ('result of count(*) not NULL - as expected');
  END IF;
```



Figure 151: The IF Statement

The figure, The IF Statement, shows an example of this statement, which consists of a Boolean expression `bool-expr1`. If this expression evaluates to true then the statement `then-stmts1` in the mandatory `THEN` block is executed. The `IF` statement ends with `END IF`. The remaining parts are optional.

If the Boolean expression `bool-expr1` does not evaluate to true the `ELSE` branch is evaluated. In most cases this branch starts with `ELSE`. The statement `else-stmts3` is executed without further checks. After an else branch no further `ELSE` branch or `ELSEIF` branch is allowed.

Alternatively, when `ELSEIF` is used instead of `ELSE`, another Boolean expression `bool-expr2` is evaluated. If it evaluates to true, the statement `then-stmts2` is executed. You can add an arbitrary number of `ELSEIF` clauses in this way.

This statement can be used to simulate the switch-case statement known from many programming languages.

WHILE and FOR Loops

The figure, WHILE and FOR Loops, shows the syntax of these two loops.



WHILE <bool-stmt> DO

{stmts}

END WHILE

FOR <loop-var> IN {REVERSE} <start> .. <end> DO

{stmts}

END FOR



Figure 152: WHILE and FOR Loops

The WHILE loop

The WHILE loop executes the statement stmts in the body of the loop as long as the Boolean expression at the beginning bool-stmt of the loop evaluates as true.

The FOR loop

The FOR loop iterates a range of numeric values – denoted by start and end in the syntax – and binds the current value to a variable (loop-var) in ascending order. Iteration starts with value start and is incremented by one until the loop-var is larger than end.

Therefore, if start is larger than end, the body loop will not be evaluated. For each enumerated value of the loop variable the statements in the body of the loop are evaluated. The optional keyword REVERSE specifies to iterate the range in descending order.

The EXEC Statement

The EXEC statement helps to create dynamic SQL statements. For example, let's say that you want to find the travel arrangements for users. A web form is requesting the user's passport number. You insert this passport number into a partially prepared SQL statement, and then execute this SQL statement using the EXEC statement to get the travel plans.

EXEC '<SQL statement goes here>'

Dynamic SQL statements are used to construct SQL statements at runtime in a procedure.

The EXEC statement executes the SQL statement passed in a string argument. This statement allows for dynamically constructing an SQL statement at execution time of a procedure.

For example, a SELECT statement could reference a table whose name is not known until runtime of the procedure. Perhaps the table name is retrieved from a lookup based on a condition.

Therefore, on the positive side, dynamic SQL allows the use of variables where they might not be supported in SQLScript, or more flexibility in creating SQL statements.

On the negative side, dynamic SQL comes with the following additional costs at runtime:

- Opportunities for optimizations are limited.
- The statement is potentially recompiled every time the statement is executed.

Unit 5: Using SQL in Models

- It is not possible to use SQLScript variables in the SQL statement (but when constructing the SQL statement string).
- It is not possible to bind the result of a dynamic SQL statement to a SQLScript variable.
- SQL injection bugs could harm the integrity or security of the database.



Note:

SAP HANA SPS11 introduced features to secure Dynamic SQL. However, we recommend that you avoid dynamic SQL because it might have a negative impact on security or performance.

Security of Dynamic SQL

Built-In Procedures to Secure Dynamic SQL



- For variables containing a SQL string literal, use the following:
`ESCAPE_SINGLE_QUOTES
(string_var)`
- For variables containing a delimited SQL identifier, use the following:
`ESCAPE_DOUBLE_QUOTES (string_var)`
- To check that a variable contains safe, simple SQL identifiers (up to num_tokens, where the default is 1), use the following:
`IS_SQL_INJECTION_SAFE(string_var[,
num_tokens])`

You can find details on these procedures in the SAP HANA SQLScript Reference Guide, available at <http://help.sap.com/hana>.



LESSON SUMMARY

You should now be able to:

- Work with SQLScript
- Explain the SQLScript implementation logic

Unit 5

Lesson 4

Query a Modeled Hierarchy Using SQLScript



LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Query a modeled hierarchy using SQL

Query Hierarchies using SQLScript

A hierarchy that is defined in a calculation view can also be accessed directly by SQLScript statements. For example, you may need to return the total number of absence days for the Service line of business that can only be found by rolling up many departments and sub-departments in a company hierarchy that is modeled in a calculation view. A simple SQLScript `select sum(days)` statement with a `where hier_node = 'service'` clause could easily provide this value.

But before you can write the SQLScript query, you need to work on some setup. Let's cover the basic rules:

1. The hierarchy you wish to query must be defined in a **dimension** calculation view which then must be consumed in the **star join** node of a **cube** calculation view because only **shared** hierarchies can be read by SQLScript
2. The calculation view of type cube with star join must allow the shared hierarchy to be exposed to SQLScript by setting a special parameter in the **Properties**
3. You should check the name that is given to the node column because you will need to refer to this in SQLScript

Reading hierarchies using SQLScript is supported for both level and also parent child types.

Let's look in detail at the settings in the calculation view type cube with star join.

In the **View Properties** tab of the **Semantics** node you need to set the flag **Enable Hierarchies for SQL Access** in order to give permission for this calculation view to expose its shared hierarchies to SQLScript.

Unit 5: Using SQL in Models

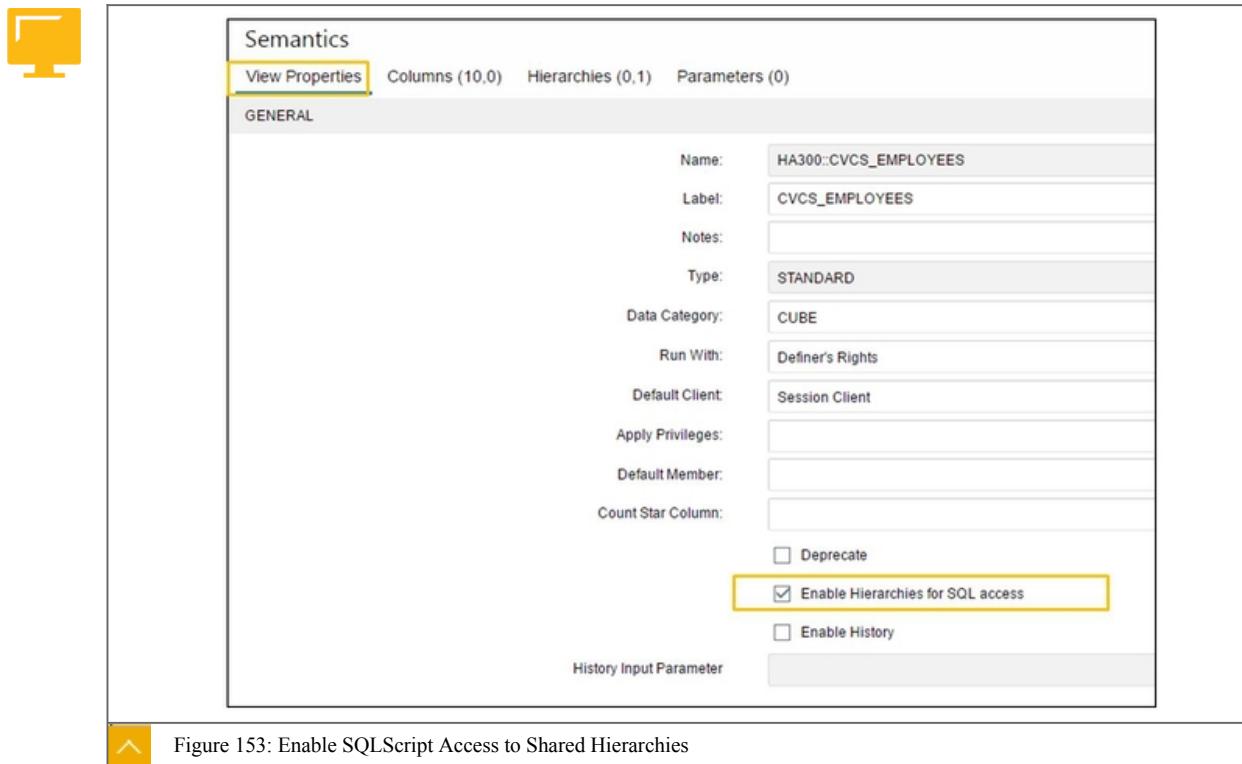


Figure 153: Enable SQLScript Access to Shared Hierarchies

Once this is set, you can then review the settings in the Hierarchy tab of the Semantics node.

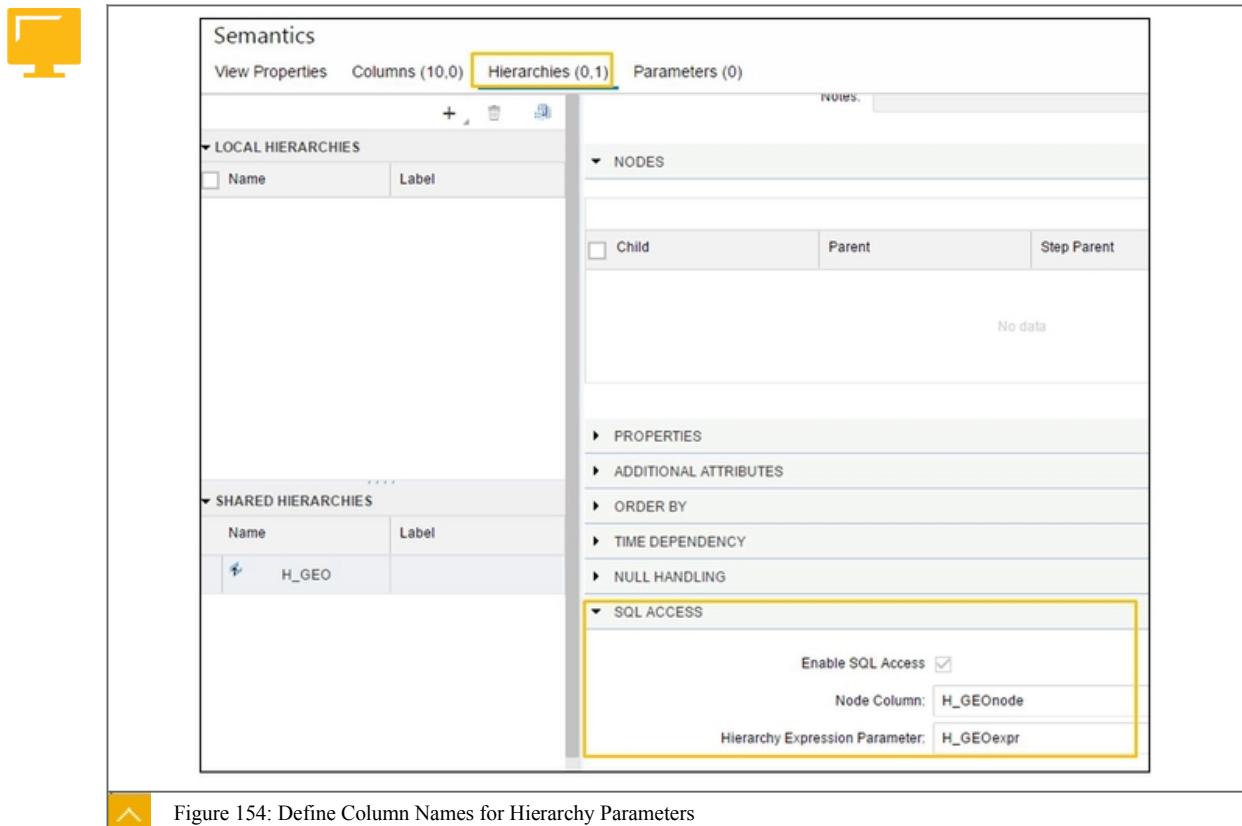


Figure 154: Define Column Names for Hierarchy Parameters

Here you will see, for each shared hierarchy, the node column name that is proposed. This name can be overwritten so that a more meaningful name can be supplied.



Note:
The Hierarchy Expression Parameter is not used in the current release of SAP HANA.

Aggregating Values in a Hierarchy Via SQL

The figure, Aggregating Values in a Hierarchy Via SQL, shows how the SQLScript is written and also how the result would appear. Simply refer to the generated hierarchy node column name as if it were a regular column in the data source..



Aggregation in a Hierarchy via SQL

The node column can be used in the GROUP BY clause

SQL		Result
	H_GEONode	SUM(SALARY_AMOUNT)
1	(all)	3,565,994
2	[DE]	124,618
3	[DE].[Berlin]	124,618
4	[EN]	478,449
5	[EN].[London]	478,449
6	[FR]	0
7	[FR].[Paris La Défense Cedex]	0
8	[US]	2,962,927
9	[US].[Antioch, Illinois]	1,767,002
10	[US].[New York]	619,509
11	[US].[San Francisco]	576,416

The result shows the aggregated amount for each node of the hierarchy



Figure 155: Aggregating Values in a Hierarchy via SQL



LESSON SUMMARY

You should now be able to:

- Query a modeled hierarchy using SQL

Unit 5

Lesson 5

Creating and Using Functions



LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Work with functions

Introducing Functions

Functions, or User Defined Functions (UDF), as they are more formerly known, allow developers to capture a reusable block of SQLScript code that provides a result after working through some data processing logic. Very often, functions allow input parameters to pass key information to the processing logic so that the function can be reused in different scenarios. Functions are relevant to SAP HANA modeling because they offer a way to extend the standard capabilities provided with graphical calculation views to reach more complex data processing possibilities. Functions are written in SQLScript and are built using design-time files. After a successful build, a runtime object is generated in the SAP HANA database. When executed, functions always run in the in-memory SAP HANA database and so performance is optimized.



Note:

Functions are relevant for classic repository or XS development and also the newer HDI or XSA development. In this lesson we focus on the HDI or XSA approach.

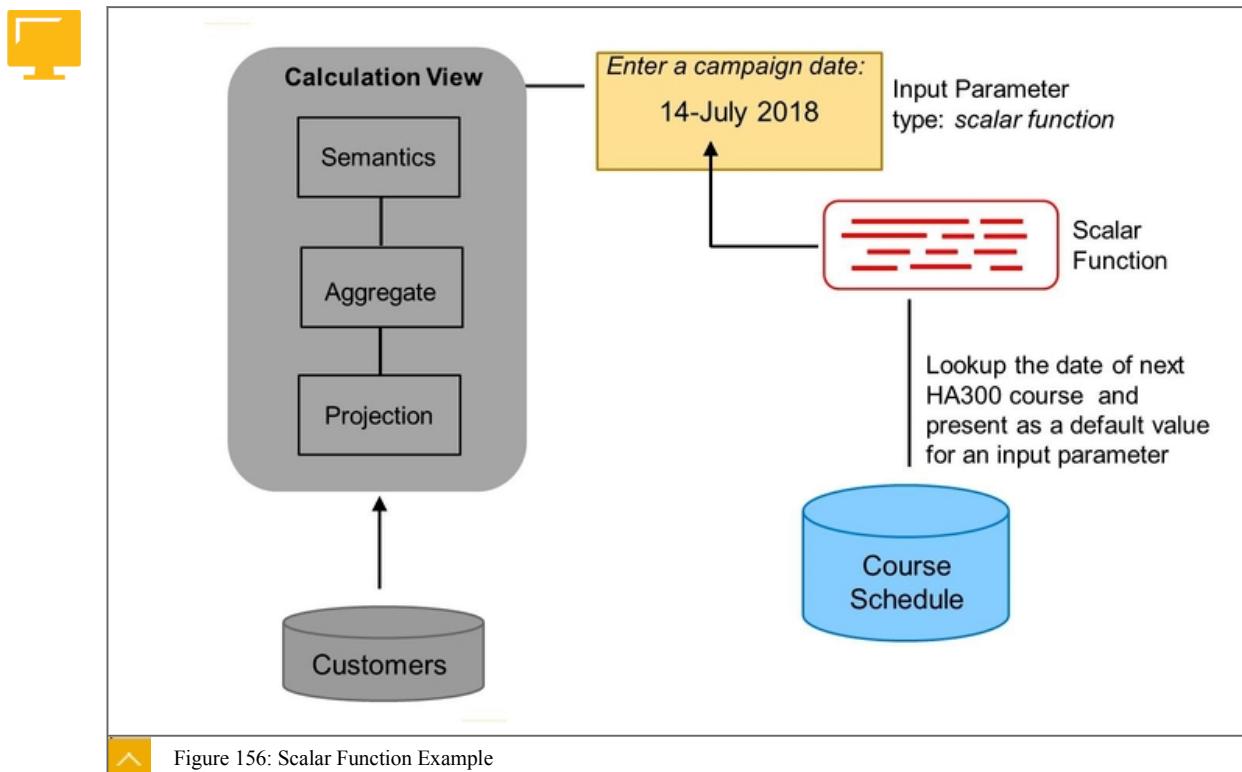
SAP HANA supports **scalar** and **table** functions. Table and scalar functions are created as design-time objects in an SAP Web IDE project folder. The design-time object has the extension **.hdbfunction** and the same extension is used for table and scalar functions.

Functions, both scalar and table, are always **read-only**. It is not possible to alter the database by using data definition statements such as **create table**, or data manipulation statements such as **update**. If you need to do such things, then consider the use of **Procedures**.

Functions can call other functions which encourages a modular approach to ensure high reuse.

Scalar Function

One of the common uses of a scalar function in a calculation view, is to derive values for input parameters, either as a default proposal in a popup that can be overwritten by a user, or as a fixed value that cannot be changed by a user.

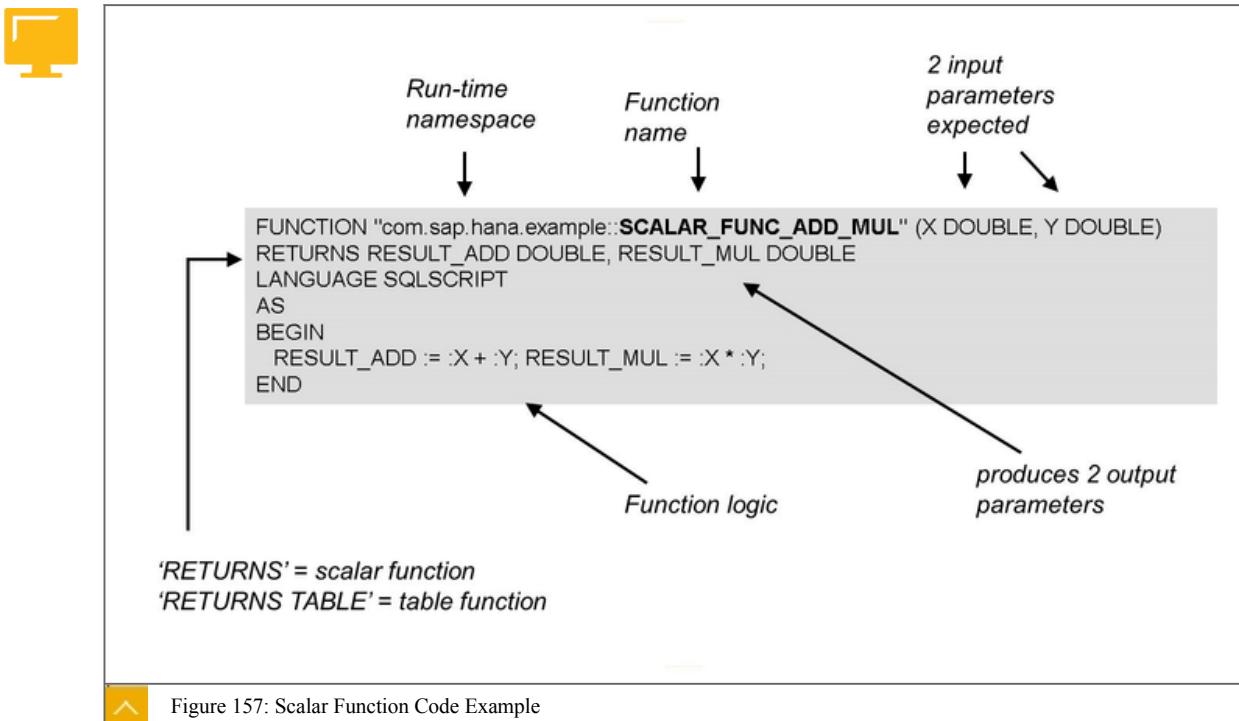


A simple example of a scalar function could be to return the current date. But scalar functions can be more complex and can read tables and call other database objects (including other user defined functions).

A scalar function returns one or more parameters but each parameter always has only a single value. Scalar functions often have one or more input values, but these are not mandatory. For example, returning the current date requires no input parameter, whereas if you wanted to return the date that was x days earlier than today, then you would need to provide x as the input parameter.

Here is a breakdown of a simple example of a scalar function's code:

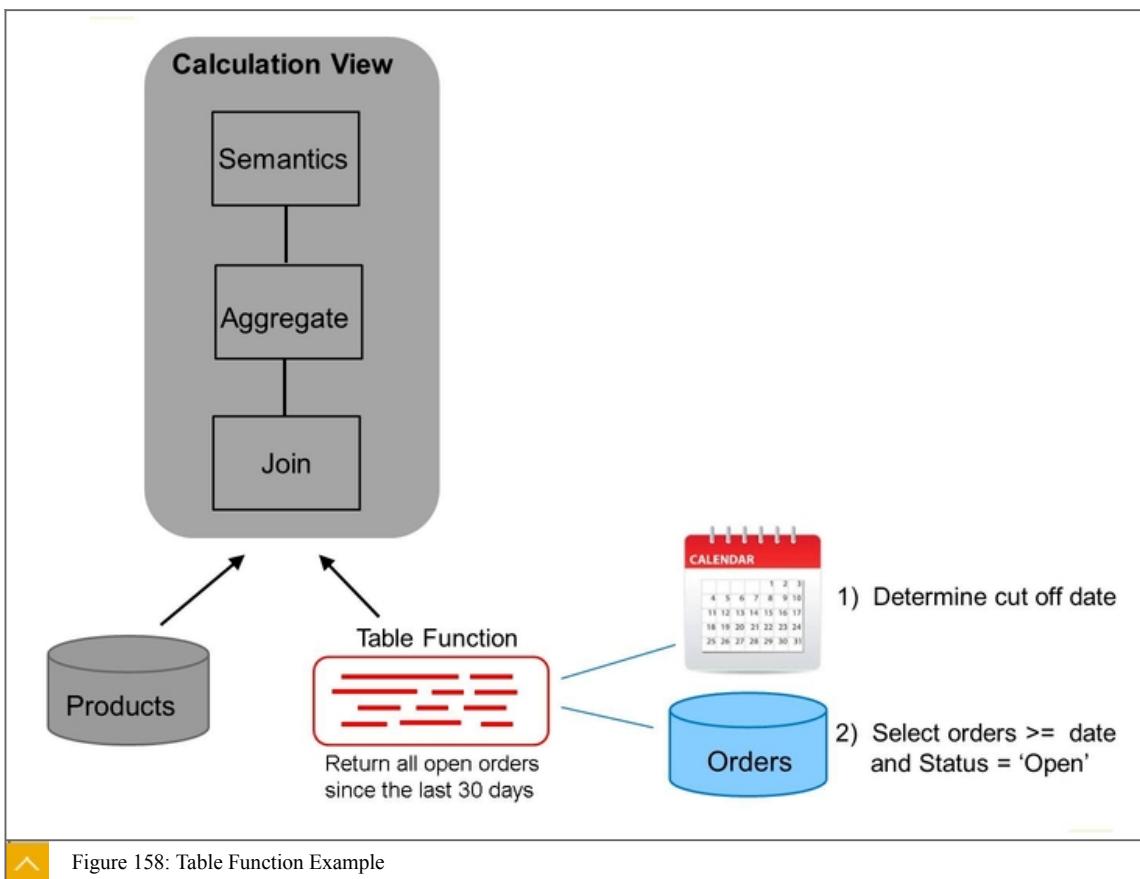
Unit 5: Using SQL in Models



Notice the use of the keyword **RETURNS** for scalar functions. The keyword **RETURNS TABLE** is used for table functions.

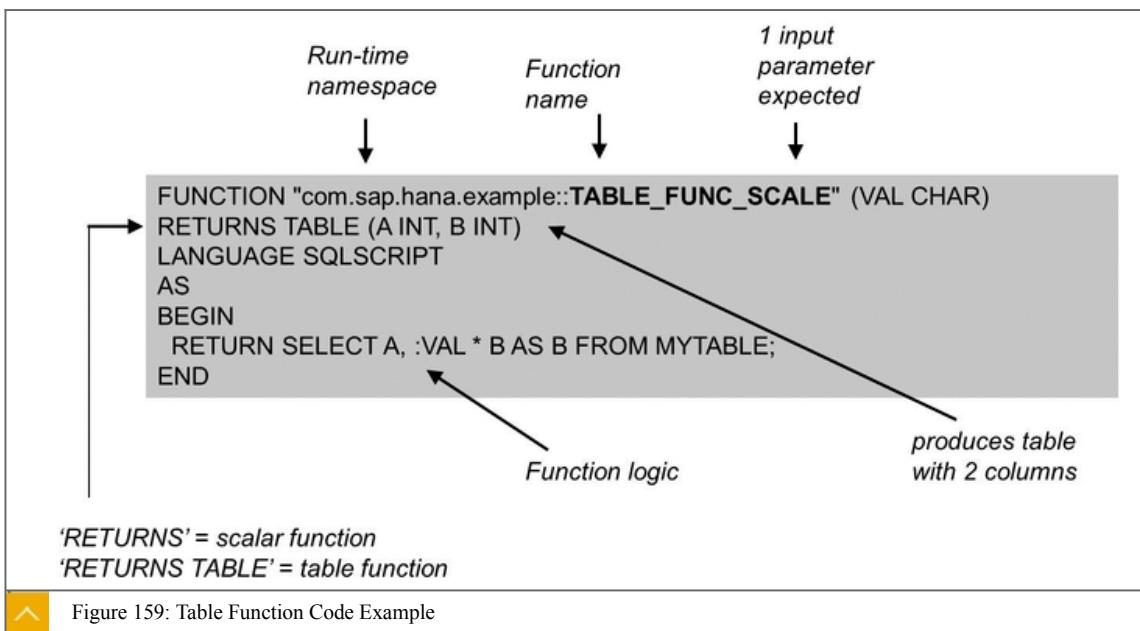
Table Functions

Table functions are used to return a **tabular** data set as opposed to a single value parameter. A tabular data set can consist of one or more columns and usually has multiple values (rows). Table functions are commonly used by modelers to define a customized data source in a calculation view. Refer to the function wherever a data source can be defined, such as in a calculation view **Projection** node or **Join** node just as you would include a table. In fact it might help to think of them as dynamic tables that are created at run time.



As an example of a table function, consider that you may need to read through a table that contains sales orders so that you can find open orders that have been recently created. The date used for the determination of the required orders is based on an input parameter passed from the calculation view. The resulting open orders are passed to the calculation views as a data source.

Here is a breakdown of an example table function's code:



Unit 5: Using SQL in Models

Notice the use of the keyword **RETURNS TABLE** for table functions. The keyword **RETURNS** is used for scalar functions.

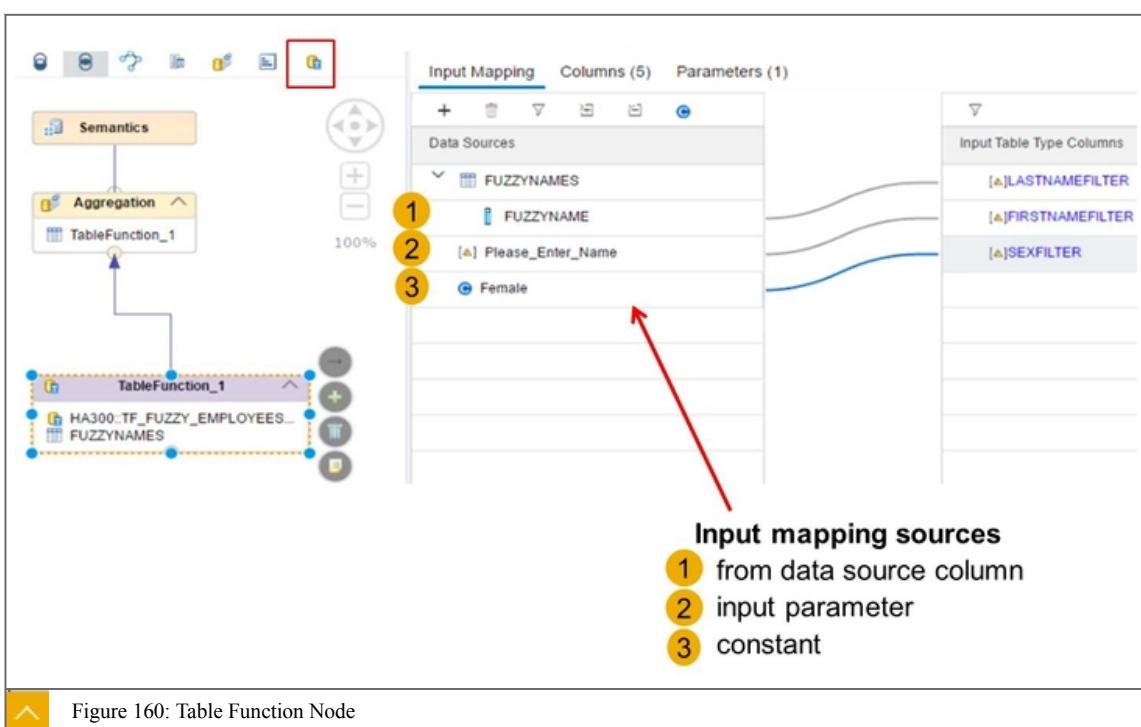


Note:

Introduced in the first release of SAP HANA, there was a type of calculation view that could be built using 100% SQLScript instead of using the graphical approach. They were called **Scripted Calculation Views**. It is still possible to create these using SAP HANA Studio under the classic/repository framework, however SAP no longer recommend these because they do not provide the best performance optimizations. You should avoid these and instead build your SQLScript into functions and then consume the functions in graphical calculation views. SAP provide a migration tool in SAP HANA Studio to convert scripted calculation views to functions and also a wrapper graphical calculation view is generated to consume the functions. So the outcome is the same.

All output parameters of the table function are offered as columns to the calculation view node. If your table function requires input parameters you can first define input parameters in the calculation view and then use the input parameter mapping function to map them.

Since SAP HANA 2.0 SPS01, a new calculation view node is available – **Table Function** – especially dedicated to the handling of the different types of mapping of the input parameters in a table function. It was possible to achieve the same outcome using other node types but this new node makes things a lot easier to define the parameter mapping.



There are three types of input mapping available in the **Table Function** node:

- **Data Source Column** — First add a data source to the Table Function node then map one of its columns to one or more table function input parameters.
- **Input Parameter** — Choose an input parameter from the calculation view and map it to one or more table function input parameters

- **Constant** — Manually define a single fixed value and map it to one or more table function input parameters



LESSON SUMMARY

You should now be able to:

- Work with functions

Unit 5

Lesson 6

Creating and Using Procedures

LESSON OVERVIEW

In this lesson, you will get an introduction to procedures and learn how to define a procedure and how to call the procedure.

Business Example

As part of your SAP HANA implementation project, you have some business requirements for reporting that cannot be fulfilled with only analytic or graphical calculation views.

You have decided to use SQL Calculation Views, but you would like to get the best possible flexibility by creating procedures that you can easily reuse in several calculation views.

You want to know more about procedures and how to use them in the most relevant way.



LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Create and use procedures
- Create a procedure
- Call a procedure

When to Use a Procedure

Procedures and functions have some similarities, in that they are both written in SQLScript and are used to create reusable blocks of processing logic passing input and output parameters. A key difference, however, is how and where they can be used. For example, you can add a function as data source to a calculation view, just like a table, but you cannot add a procedure as a data source. You can also use a function in a FROM clause of a SELECT statement, but not a procedure. So where do we use procedures?

Introduction to Procedures



- Procedures can have multiple input parameters and output parameters (which can be of scalar or table types).
- Procedures describe a sequence of operations on data passed as input and database tables.
- Procedures can in turn call other procedures.
- Read-only procedures can only call other read-only procedures.



Figure 161: Procedures

Procedures can be used in calculation views to return results to input parameters and also to determine a user's permissions in analytic privileges. Procedures can also be called from

functions. A procedure is able to write data and well as read. A table function can only have one tabular output but a procedure can produce multiple, different tabular outputs.

SQLScript and Procedures



- Procedure is a reusable processing block. It is implemented using SQL Script.
- A procedure can be created as read only (without side-effect) or read-write.
- Procedure can be implemented using SQL Script, L or R language.



Figure 162: Procedures - Key Properties

The figure, Procedures – Key Properties, lists some of the features of procedures.

The body of a procedure consists of a sequence of SQLScript statements separated by semicolons.

An intermediate variable, inside a procedure, does not need to be defined before it is bound by an assignment. You simply state the variable name and fill it.

A variable name is prefixed by a colon (:) when it is used as an input to other statements.



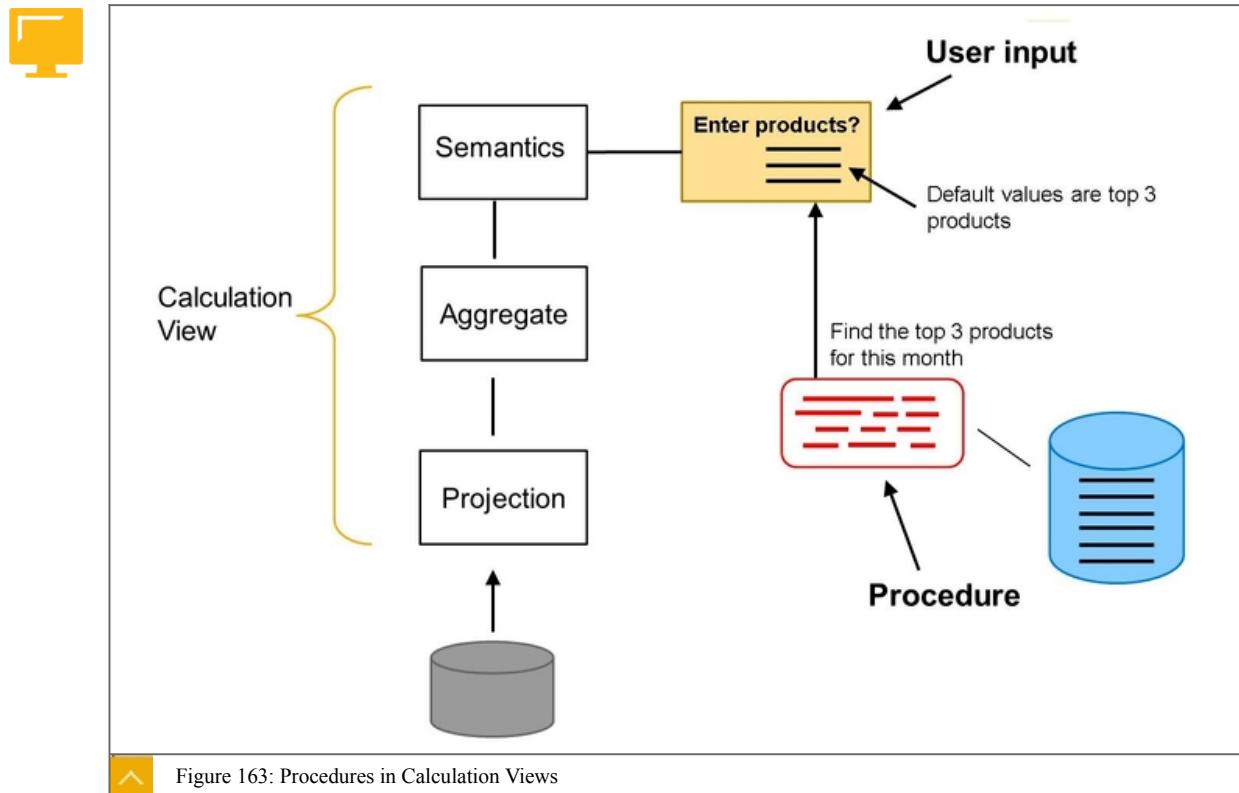
Note:

In SAP HANA, the L language is used by SAP internally to develop internal functions in AFL libraries. It is not supported for customer developments.

Procedures in Calculation Views

Procedures can be used to derive results to be used in input parameters. For example you could define a procedure to determine the date of the next campaign or to calculate the number of target days to use in a calculation.

Unit 5: Using SQL in Models



Procedures in Analytic Privileges

Analytic privileges are a type of security artifact used to define data access authorization for calculation views. We will cover these later.

A typical use case for procedures is to define dynamic analytic privileges. Dynamic analytic privileges do not have fixed data values (for example, US, France) the values are determined at run time, often based on the user id or user's role when a calculation view is queried. This means only one analytic privilege needs to be created that will be used by many users to determine automatically the countries allowed Sales calculation view for all the users.

Creation of a Procedure

```

1 PROCEDURE "HA300::P_BOOKS_VAT_00"
2 (IN tax DECIMAL(5,2),
3 OUT out_with_tax table
4 (ISBN VARCHAR(20),
5 TITLE VARCHAR(50),
6 PUBLISHER INT,
7 PRICE DECIMAL(33,2),
8 PRICE_VAT DECIMAL(33,2),
9 CURRENCY VARCHAR(3)
10 )
11 )
12 LANGUAGE SQLSCRIPT
13 SQL SECURITY DEFINER
14 READS SQL DATA AS
15
16
17 /****** Begin Procedure Script *****/
18 BEGIN
19
20 out_with_tax = SELECT ISBN,
21           TITLE,
22           PUBLISHER,
23           PRICE,
24           PRICE * :tax AS PRICE_VAT,
25           CURRENCY AS CURRENCY
26   FROM "HA300::BOOKS";
27
28 END;
29 /****** End Procedure Script *****/

```

Figure 164: Example of a Procedure

You create a procedure by writing your SQLScript code in a dedicated source file in SAP Web IDE, which has the extension **.hdbprocedure** and then building it to create the runtime object in the HDI container. It is also possible to create procedures directly using the **CREATE PROCEDURE** statement in the SQL console but this is not recommended because this does not follow the HDI approach.

Language

The default implementation language is **SQLSCRIPT**. It is best practice to define the language in every procedure definition. Other implementation languages, such as the **R** language and **L** language (SAP use only), and more recently **GraphScript** are supported, but not covered here.

Security Mode

With security mode **Definer's rights**, which is the default, the execution of the procedure is then performed with the privileges of the procedures definer. On build, the technical user of the container creates the runtime procedure and is then considered as the definer of the procedure. The other alternative is mode **Invoker**. In this case, privileges are checked at runtime with the privileges of the caller (application) of the procedure.



Note:

Analytic privileges of the user are always used to check the data permissions of the database objects being accessed regardless of the security mode for the execution of the procedure.

READS SQL DATA

`READS SQL DATA` is used to define a procedure as read-only and marks a procedure as being free of side-effects. Neither DDL nor DML statements are allowed in its body, and only other read-only procedures can be called by the procedure. The advantage to this definition is that certain optimizations are only available for read-only procedures.

WITH RESULT VIEW

`RESULT VIEW` can be specified for read-only procedures (those identified by the `SQL DATA` keyword). It is used to specify a view that is used as the output of the procedure. If a result view is specified for a procedure, the procedure can be called by an SQL statement in the same way as a table or a view. The name of the result view is no longer bound to a static name scheme but can be any valid SQL identifier. For backward compatibility reasons the old static name scheme will be only supported for procedures that are generated with the deprecated `CREATE FUNCTION` syntax.

READS

CREATE FUNCTION

Input parameters are not mandatory but in most cases you will define some to ensure your procedures are modular and allow better reuse.

Procedure Calls



CALL - Procedure Called From Client

A procedure (or table function) can be called by a client on the outer-most level, using any of the supported client interfaces.

Syntax

```
CALL [schema.]name (param1 [, ...])
```

```
SQL Result
call "_SYS_BIC"."student03/MICHAEL_PROC" (input_int => 3, output_int => ?);
```

For table output parameters it is possible to either pass a table or '?'.
 '?' can be used to represent an empty parameter binding.



Figure 165: Calling a Procedure

As the figure, Calling a Procedure, shows, when calling a procedure using the `CALL` statement, the procedure behaves in a consistent way. The SQL standard semantics, for example, Java client's can call a procedure using a JDBC Callable statement. `CALL` returns an iterator over result sets. Each output variable of the procedure is represented as a result set.

CALL WITH OVERVIEW



CALL...WITH OVERVIEW From Client

This CALL statement returns one result set that holds the information of which table contains the result of a particular table's output variable.

This is used to populate an existing table by passing it as parameter.

When passing '?' to the output parameters, temporary tables holding the result sets will be generated.

Syntax

CALL [schema.]name (param1 [, ...]) WITH OVERVIEW

SQL SQL Result

```
call "_SYS_BIC"."student03/MICHAEL_PROC"
  (input_int => 3, output_int => ?)
  WITH OVERVIEW;
```

	variable	table
1	OUTPUT_INT	"STUDENT03"."OUTPUT_INT_50AB74A47460572CE10000000A1671F8"

Figure 166: Calling a Procedure with Overview

The CALL statement displayed in the figure, Calling a Procedure with Overview, returns one result set that holds the information of the table, which contains the result of a particular table's output variable.

When passing existing tables to the output parameters, the result set tuples of the procedure into the given tables.

CALL WITH OVERVIEW inserts

Procedure Call from a Script-Based Calculation View



CALL – Internal Procedure Call

For internal calls, i.e. calls to a procedure by a calculation view or another procedure, IN variables are bound by literals or variable references, new OUT variables are bound to the result of the call.

Syntax

CALL [schema.]name (:in_param1, out_param [, ...])

```
BEGIN
  it_pubs =  SELECT PUB_ID AS PUBLISHER, NAME
              FROM PUBLISHERS;

  CALL "_SYS_BIC"."STUDENT00/P_BOOKS_VAT_00" (:INP_TAX_RATE, it_books);

  var_out =   SELECT P.PUBLISHER, P.NAME, B.ISBN, B.TITLE, B.PRICE, B.CURRENCY, B.PRICE_VAT
              FROM :it_pubs P, :it_books B
              WHERE P.PUBLISHER=B.PUBLISHER;
```

Figure 167: Calling a Procedure

Procedures can be called with this syntax by another procedure or a calculation view. In the case of a script-based calculation view, it is possible to pass the IN parameters to the

Unit 5: Using SQL in Models

procedure by hard-coding the values of these parameters in the procedure call, or by requesting these parameters to the end user like in any information view with input parameters.



Note:

In the figure, Calling a Procedure, :INP_TAX_RATE refers to an input parameter defined in the calculation view.



LESSON SUMMARY

You should now be able to:

- Create and use procedures
- Create a procedure
- Call a procedure

Unit 5

Learning Assessment

1. Why is knowledge of SQL important to an SAP HANA modeler?

Choose the correct answers.

- A** So that they can avoid creating calculation views
- B** So that they can understand how to read a function or procedure
- C** So that they can extend the capabilities of a calculation view

2. When must I avoid specifying a schema in my SQL?

Choose the correct answer.

- A** When writing SQL against a database connection of the type HDI Container
- B** When writing SQL against a database connection of the type SAP HANA Database.

3. Why should I define my SAP HANA tables using CDS?

Choose the correct answer.

- A** The syntax is much simpler than the standard SQL Create Table expression.
- B** A table that is created with CDS performs better than one created with the standard SQL Create Table expression.
- C** The table definition code is easily transported as part of the overall application code to ensure integrity.

4. What are the types of user defined function that you can include in a calculation view?

Choose the correct answers.

- A** Table
- B** Column
- C** Scalar

Unit 5: Learning Assessment

5. When would you use a function?

Choose the correct answer.

- A** As a replacement for calculation views when you need to best possible performance
- B** When you need to write results back to a table
- C** When you need additional data processing functions that the graphical calculation view cannot provide

6. How should you create a procedure?

Choose the correct answer.

- A** Create a source file in the SAP Web IDE with the extension .hdbprocedure
- B** Use the CREATE PROCEDURE statement in SQL

7. Procedures can be used in the FROM clause of a SELECT statement.

Determine whether this statement is true or false.

- True
- False

Unit 5

Learning Assessment - Answers

1. Why is knowledge of SQL important to an SAP HANA modeler?

Choose the correct answers.

- A** So that they can avoid creating calculation views
- B** So that they can understand how to read a function or procedure
- C** So that they can extend the capabilities of a calculation view

Correct — SQL is important for modelers because they need to know how to read the logic written in SQL inside functions and procedures and also how they might extend the calculation view capabilities, for example, by writing expressions that call standard SQL functions.

2. When must I avoid specifying a schema in my SQL?

Choose the correct answer.

- A** When writing SQL against a database connection of the type HDI Container
- B** When writing SQL against a database connection of the type SAP HANA Database.

Correct — You must never specify a schema when writing SQL against a HDI Container type database connection.

3. Why should I define my SAP HANA tables using CDS?

Choose the correct answer.

- A** The syntax is much simpler than the standard SQL Create Table expression.
- B** A table that is created with CDS performs better than one created with the standard SQL Create Table expression.
- C** The table definition code is easily transported as part of the overall application code to ensure integrity.

Unit 5: Learning Assessment - Answers

4. What are the types of user defined function that you can include in a calculation view?

Choose the correct answers.

- A** Table
- B** Column
- C** Scalar

5. When would you use a function?

Choose the correct answer.

- A** As a replacement for calculation views when you need to best possible performance
- B** When you need to write results back to a table
- C** When you need additional data processing functions that the graphical calculation view cannot provide

6. How should you create a procedure?

Choose the correct answer.

- A** Create a source file in the SAP Web IDE with the extension .hdbprocedure
- B** Use the CREATE PROCEDURE statement in SQL

Correct – You should always create procedures using source files in the SAP Web IDE of the type .hdbprocedure

7. Procedures can be used in the FROM clause of a SELECT statement.

Determine whether this statement is true or false.

- True
- False

Correct — You CANNOT use procedures in the FROM clause of a SELECT statement

UNIT 6

Optimization of Models

Lesson 1

Implementing Good Modeling Practices

216

Lesson 2

Using Tools to Maximize Optimization

233

UNIT OBJECTIVES

- Implementing good modeling practices
- Using Tools to Maximize Optimization

Unit 6

Lesson 1

Implementing Good Modeling Practices

LESSON OVERVIEW

There are many choices to make when developing SAP HANA models. The wrong choice can severely affect the function and performance of your model. In this lesson, we will guide you towards good modeling approaches.



LESSON OBJECTIVES

After completing this lesson, you will be able to:

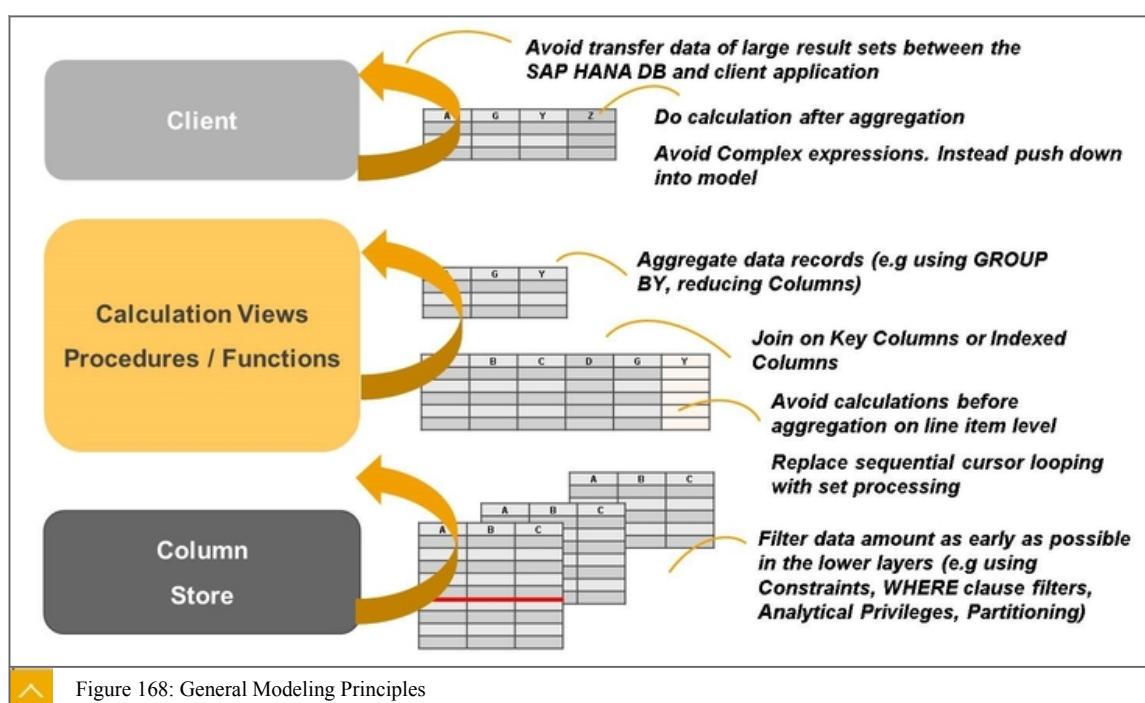
- Implementing good modeling practices

Best Practices for Modeling

Throughout this course we have encouraged a general design approach that provides a basis for good performance. In this lesson, we will introduce some specific recommendations that will ensure best performance.

General Modeling Principles

The figure, General Modeling Principles, lists some of the basic principles that you should keep in mind to maximize performance.



Let's take a look at some of the recommendations in detail. Starting from the bottom of the modeling stack, you should do the following:

- Filter data as early as possible in the lowest layers. This means that you must prune results that are not needed. Do not carry unwanted results to the top of the stack to then ignore them. Lose the data early using expression filters, analytic privileges, WHERE clauses, table partitions, union pruning rules, input parameters, and so on.
- When writing SQL, always try to use set-based processing, and avoid procedural constructs such as cursors and loops that can damage performance by breaking optimizations.
- Calculate after aggregations where possible so that the calculation does not have to be performed at individual row level.
- Join data sources on indexed or key columns.
- Aggregate data as early as possible so that you reduce the number of rows that have to travel through the stack. Remove unwanted columns early so that they are not part of the aggregation that is not needed.
- Push as much processing to SAP HANA as possible by defining the complex logic in the model and not in the front-end logic. Make SAP HANA work hard on the data processing, not the front-end client tool.
- Avoid moving large data sets between SAP HANA and the front-end client. Try to use input parameters that prompt users for data selections for each query so that the filters are pushed back to SAP HANA.

Avoid Calculation Plan Operators (CE functions)

In the first releases of SAP HANA, SAP introduced a dedicated calculation engine and a set of functions to speed up common data processing tasks such as joins, unions, and other calculations. The calculation engine was invoked by writing SQL that used special calculation engine functions (CE functions). These functions – approximately 10 – were easily identifiable because they were prefixed with the characters `CE_`. CE functions are delivered as part of the SQLScript language and provided an alternative to writing the data processing instruction using plain SQL. To proceed, you would write standard SQL but wherever a CE function was available, SAP encouraged you to use that instead of a standard SQL function to achieve the best performance. Using a CE function, forced the calculation engine to be invoked instead of the SQL engine. In early releases of SAP HANA the difference between plain SQL and CE functions in terms of performance was very noticeable.



SQLScript approaches

- Standard SQL versus Calculation Engine (CE Functions) Plan Operators
 - Recommendation! When creating functions or procedures with SAP HANA SP09+ use standard SQL syntax instead of CE Functions. The SQL optimizer is able to leverage alternative execution engines and are not restricted only to the Calculation Engine.

Recommended approach	
SELECT on column table	SQL
	<code>out = SELECT A, B, C from "COLUMN_TABLE"</code>
GROUP BY	<code>out = SELECT A, B, C, SUM(D) FROM "COLUMN_TABLE" GROUP BY A, B, C</code>
CE function	
	<code>out = CE_COLUMN_TABLE("COLUMN_TABLE", [A, B, C])</code>
GROUP BY	<code>col_tab= CE_COLUMN_TABLE("COLUMN_TABLE"); out = CE_AGGREGATION(col_tab, SUM(D), [A, B, C])</code>



Figure 169: Avoid Calculation Engine Operators

Unit 6: Optimization of Models

Since SPS09, SAP recommends that you no longer use CE functions and that you should only use plain SQL. There are a few reasons for this:

- The SQL engine is now capable of providing excellent performance when processing plain SQL expressions. This means that we can use the industry standard SQL language to encourage more openness and compatibility.
- In most cases, you would have needed to use a mixture of plain SQL and CE functions because not everything could be done in CE functions. This meant that both engines (SQL and CE) would be invoked and optimization opportunities would be limited. Swapping data between engines is bad for performance and should be avoided.

The calculation engine still exists and could be invoked whenever the optimizer considers this to be the way to achieve best performance. But when you use CE functions, you do not allow the optimizer to make this decision and you would always force the calculation engine to be invoked. CE functions are still supported to ensure backward compatibility with older models and many customers will still have embedded these in their SQL code, but SAP recommends that you change the CE functions to use standard SQL to achieve the best possible performance.

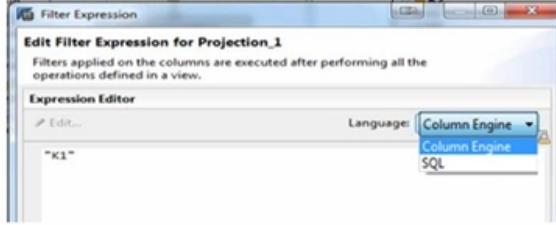
Write Calculation View Expressions in SQL, not Column Engine Language

Expressions are used in many places within calculation views including calculated columns, restricted measures, and filter expressions. When you build expressions, you can choose to use either the **Column Engine** expression language or **SQL** expression language. The language is chosen using a dropdown selector in the Expression Editor.



General SQL support within Expression Editors

- **Explicit expression language selection**
 - Explicit distinction between Column Engine expression and SQL Expression was introduced for Calculation Columns since SAP HANA SPS10
 - Filter expressions
 - Restricted measure expressions
 - Default value expressions for Variables/Parameters
- **Expression language conversion behavior**
 - Harmonized execution modes between Column Engine- and SQL expression language
 - No result differences between use of a SQL vs Column Store expression due to respective semantic functions conversation



Example – Add semantic using a NULL-value

Within a column store expression, you want to use the column store semantic addition behavior

1 + NULL = 1

This may implicitly get converted into a SQL expression

ADD_NAZ(1, NULL) = 1

Within a column store expression, you want to use the SQL semantic addition behavior

SQLADD(1, NULL) = NULL

This may implicitly get converted into a SQL Function

1 + NULL = NULL

Figure 170: Use SQL for Expressions

SAP recommends that you use SQL expressions wherever possible. SAP will continue to develop more functions using SQL only and it will become the default expression language going forward. Whether you choose to use SQL or column engine expressions makes no difference to the expression result. At run time, column engine expressions are converted to SQL expressions. But we recommend that you write the expression using SQL where possible for best performance and future compatibility. SAP only includes the column engine language to ensure backward compatibility with older models that may still have expressions written in that language.

But even if you choose to use the column engine language, SAP HANA converts this to plain SQL at run time so that more of the query can be unfolded for better performance.



Note:

Unfolding is a word that an SAP HANA modeler should become familiar with in the context of performance optimization. Unfolding is the action by the optimizer of converting the steps of a calculation view query plan to plain SQL. Think of it as flattening the steps to the lowest level of SQL execution so that a single native SQL statement can be generated from all steps required in the entire calculation view stack. The optimizer tries to unfold all steps in a query plan but sometimes it is not able to do this for all the steps. In that case, the result is a partially unfolded query plan. A query plan that could not be completely unfolded would include steps that refer to the database column view (the runtime object of the calculation view) and not to the base tables. Unfolding is preferred so that the entire query can be executed completely in the SQL engine and not have to pass data to the column engine for some of the steps. Executing the entire query in the SQL engine optimizes performance. In early releases of SAP HANA, SQL unfolding was fairly limited but today there is very little in a calculation view that cannot be unfolded by the optimizer to plain SQL. SAP continues to eliminate any steps that are found to be not unfolding. Using the query plan tools, you can actually identify when a query step is unfolded or not. If the query step still refers to the column view, then it is not unfolded.



Note:

SAP has documented, in quite some detail, the topic of SQL optimization (including unfolding) and provide deeper insight and various recommendations. Refer to SAP Notes 1857202, 2291812, 2223597 for the latest information.

Optimized Execution of SQL against Calculation Views

Since SPS12, you can consume all of the functional capabilities of attribute and analytic views into calculation views, so we no longer need to build these types of calculation views and they are now known as **deprecated**. Deprecated means that they are still supported and are working, but SAP will not develop them any further. You should now only build calculations views and in fact, SAP Web IDE and the new HDI container framework only supports calculation views and not analytic or attribute views. But the specialist SAP HANA engines such as the OLAP engine (which was heavily used by analytic views) or join engine (which was heavily used by attribute views) are still valid and their operators will be called from the calculation views as required to process OLAP and join workloads. But now the optimizer is free to decide which engine to call when it creates a query plan to processing the calculation view. For example, if you use a star join in your calculation view, the OLAP engine provides excellent performance over multi-dimensional expressions for large volumes of data versus standard SQL.

Unit 6: Optimization of Models

**Optimized execution for SQL-queries against Calculation Views**

- The **initial Calculation Engine optimization** generates a single SQL statement across a stacked model, which is then passed to the SQL optimizer
- The **SQL optimizer adds additional optimizations** and delegates operations to the best database execution operator, e.g. delegate star join-nodes to the OLAP processing engine where possible

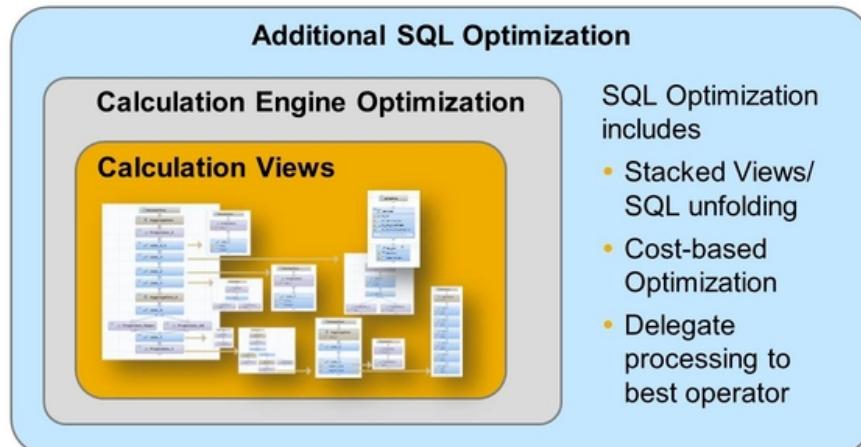


Figure 171: SQL Optimization on Calculation View

When you allow the SQL optimizer to make the delegation choice, it finds the very best performance plan, making use of the best engines for the job, as shown in the figure, SQL Optimization on Calculation View. Perhaps plain SQL is enough for some operations, and no special SAP HANA engine is required. Allowing the SQL optimizer to choose means it doesn't use special engines unless absolutely necessary. The SQL optimizer evaluates the entire stacked model from top to bottom, including all dependent calculation views. A single SQL statement is created across the entire stacked model and then the SQL optimizer decides how to add optimizations and also how to delegate the various operations. Unlike the dedicated engines such as OLAP, JOIN and CE engines, the SQL optimizer has the benefit of sophisticated cost-based optimization techniques. This means that it uses statistics to decide on the best execution plan.

In early versions of SAP HANA, you could set a calculation view manually to use SQL optimization, especially if your calculation View did not use any special CE expressions or features that needed special engines. This manual switching was called explicit optimization and guaranteed that only the SQL engine would be used. Today SAP HANA optimizes the execution automatically by default, and tries to unfold all operations to plain SQL. We call this implicit optimization. There is no longer a switch to manually control this. If SAP HANA does not unfold the operation, it is because either the operation is not possible in SQL or performance is not improved by unfolding to SQL.

General Performance Considerations for Calculation Views

As you model calculation views, keep the ideas from the figure, General Guidelines for Calculation View Modeling, in mind. It is not always possible to follow these guidelines due to functional requirements, but it is good to be aware of what could contribute to poorly performing models. So as you model, keep in mind these guidelines and be sure to focus on these should you experience problems with performance.



Performance considerations when modeling Calculation Views

- Set (correct) Join Cardinality
- Leverage Union Pruning
- Ensure filter pushdown
- If required, ensure value conversion for filter values
- Aggregate early, try to avoid static aggregations
- Avoid stacking of different view types, thus avoid unnecessary materialization of intermediate results
- Avoid filtering, joining and aggregation on calculated columns
- Use statistics for Smart Data Access virtual data sources
- Validate execution optimization and SQL unfolding
- Use Calculation View debugger to validate query scenarios



Figure 172: General Guidelines for Calculation View Modeling

Always set cardinality to provide extra metadata to the optimizer so that it can make the best decisions.

Define union pruning to provide extra hints for the optimizer when building plans around unions.

General Guidelines for Calculation View Modeling

Let dive deeper into some of the recommendations:

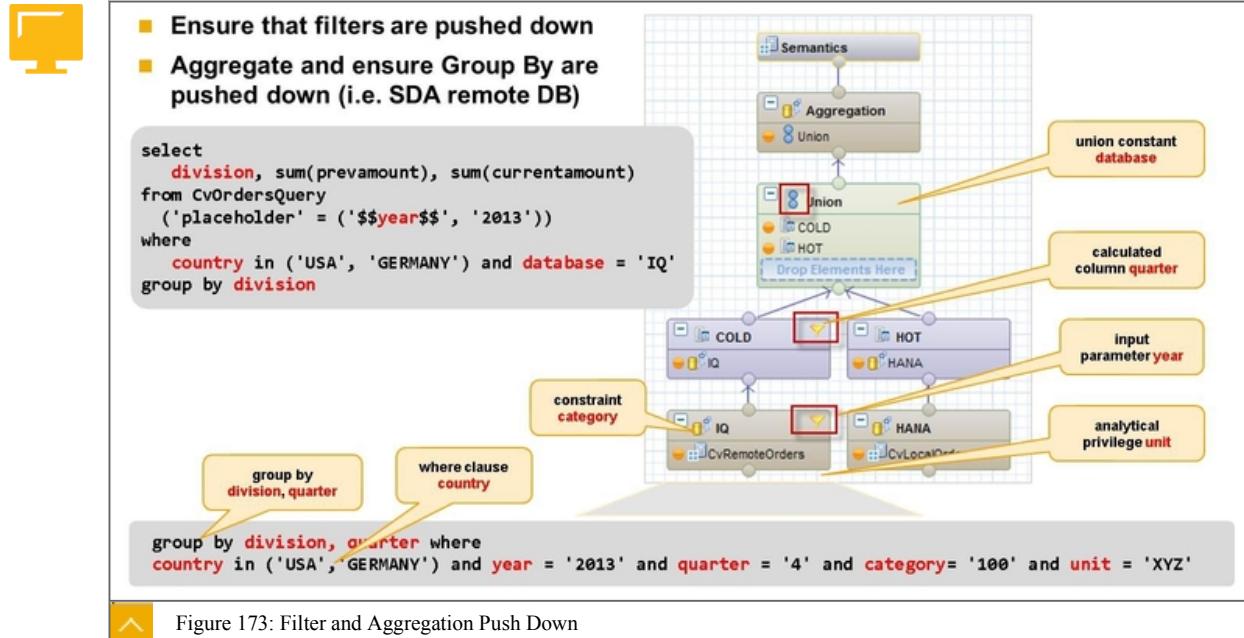
- Filter as low down in the stack as you can without compromising the function and the reusability of the components of the model. Remember sometimes the lower calculation views are reused in other views, so fixing filters too low could mean that you can't re-use the lowest calculation views for different scenarios. Consider the use of input parameters that can be passed down the stack to provide the best compromise.
- Convert input values used in Where clauses to the correct length and type for the column that is being filtered to ensure best performance
- Avoid static aggregations by using the keep flag only when required. The keep flag is like a rule that says 'regardless of what the query asks for, we will always need to force an aggregation to be calculated at this level in order to provide correct results'. But if the forced aggregation will never be used the operation and also the aggregated data set is wasteful.
- Don't stack different view types (attribute, analytic, calculation) to avoid materializing and passing of result sets between engines. Stick to calculation views and you can stack them without this concern.
- Try to avoid using calculated columns in filter expressions, join expressions, and aggregates.
- If using virtual tables, collect statistics so the optimizer can decide whether to push down the queries to the remote source or select the raw data from the remote source and process the query in SAP HANA.

Unit 6: Optimization of Models

- Use tools to check the unfolding of operations to plain SQL and try to figure out what is stopping complete unfolding.
- Use the calculation view debug mode to check how your calculation views reacts to various query scenarios. For example, this can show you how pruning is taking effect (or not).

Filter and Aggregation Push Down

We recommend that you perform filtering and aggregation as early as possible in the calculation scenario. Use the calculation view debug mode to identify how filters are being pushed down, or not. You can view the SQL at each node to inspect the code and find out what is happening in detail, as shown in the figure Filter and Aggregation Push Down.



Use Dedicated Views or Tables for ‘List of Values’

For performance issues, we generally recommend that you create dedicated calculation views or tables to generate the lists of help values for variables and input parameters.



- Faster value help dialogs
- Consistent Lists of Values across consuming views
- Dependencies between value help views
- Support for Analytic Privileges

```
select
    teamname, playername, rank,
    sum(playergoals), sum(teamgoals)
from
CvSoccerPlayers(
    'placeholder'='$$top_n_players$$', '2'),
    'placeholder'='$$ip_team$$', 'paderborn')
)
group by teamname, playername, rank
order by teamname
```

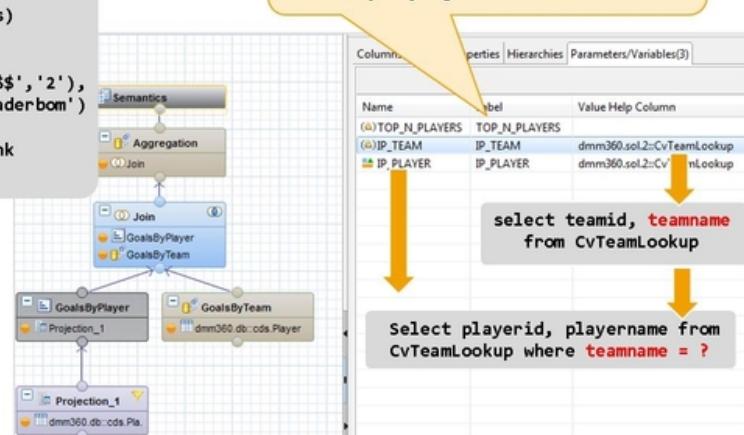


Figure 174: Using Dedicated Value Help Views or Tables

This approach enables a faster display of the list of values, without reading the entire view or table on which the calculation view is based.

This is also best practice so that the list of values is re-usable and consistent from one calculation view to another so that users see the same selection values.

The list can be driven by simple fixed filters in the help calculation view or even by analytic privileges that offer a dynamic list restricted to what each user is allowed to choose from.

Pruning Union Nodes

Be aware of the various pruning techniques when using union nodes, as shown in the figure, Pruning Unions: Implicit or Explicit. Efficient pruning can improve performance considerably by elimination unwanted data sources from a union.



Explicit pruning

```
select
    country, sum(previous), sum(current)
from CvSalesQuery
where year = 2015 and
country in ('usa', 'germany') and
SOURCE = 'Hot'
group by country order by country
```

Query execution direction relies on the Union Constant filter



Implicit pruning

```
select
    country, sum(previous), sum(current)
from CvSalesQuery (
    'placeholder' = ('$$ip year$$',
    '2015')
)
Where country in ('usa', 'germany')
group by country order by country
```

Query execution direction relies on the Date Input parameter

Figure 175: Pruning Unions: Implicit or Explicit

You can define a constant value for any source column that is part of a union to explicitly tag each source with some meaning such as plan or actual, and hot or cold. You can set the constant value against an existing column coming from a data source, or you can add an extra column and then set the constant value against it. If a query does not match the constant value for a specific source of a union then the source is ignored. This is called explicit pruning,

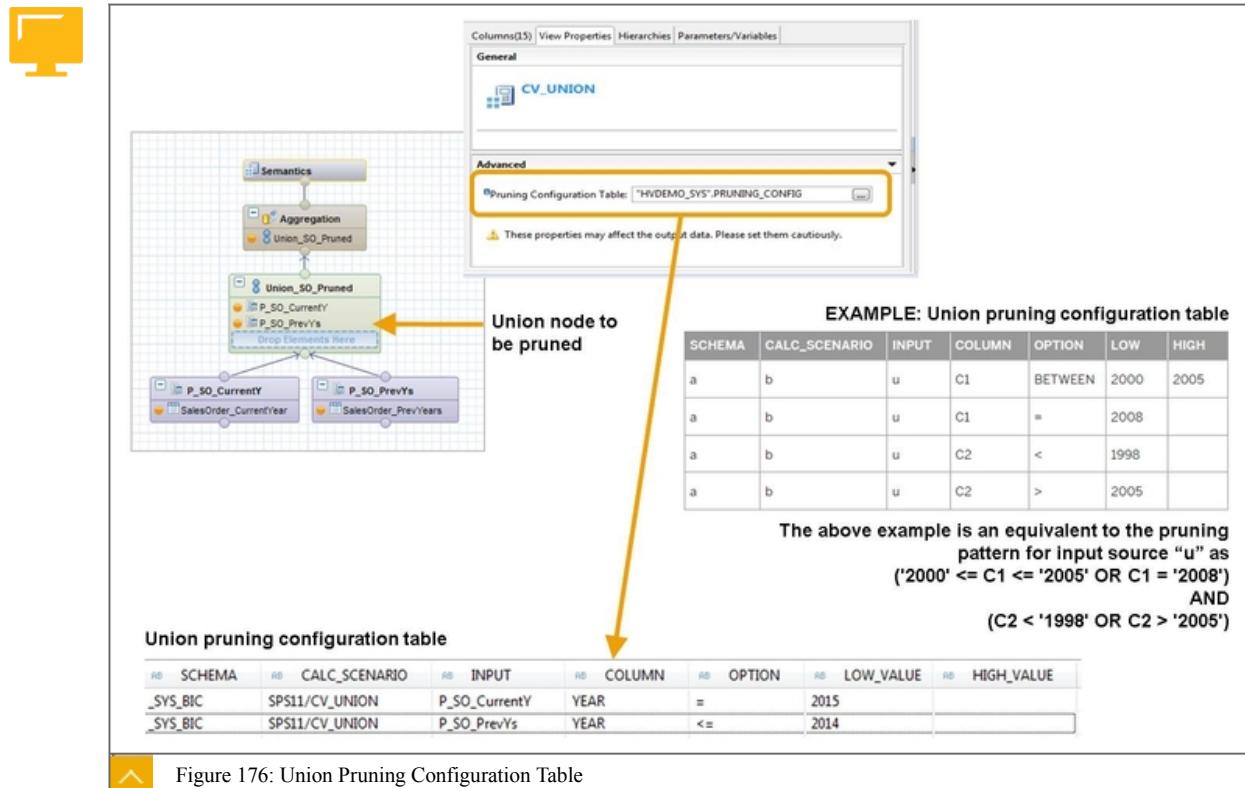
Unit 6: Optimization of Models

and it helps with performance by not accessing the source when it is not needed. A constant value is typed in manually to the union node and for each source of the union. In the example in the figure, the union node has a new column, SOURCE (not visible). This column was first created in the union node, and then had a fixed constant value assigned to it in each data source. ‘Hot’ for one data source and ‘Cold’ for the other data source. Basically, you explicitly fix a single value in a column that can then be used as a filter. This is a simple way to filter out sources to unions where the rules are simple and rarely change, and also where you want to select one union source over another, as with the hot or cold example. Although a union node usually has just one constant column, you can create and map multiple constant columns. But a key point is that if you don't refer to the constant columns in the query, then the source is always included in the union.

In addition to this approach, since SAP HANA 1.0 SPS11, SAP added a union pruning configuration table. This pruning configuration table allows you to describe in much more detail the rules that determine when each source of a union is valid. This implicit pruning approach does not need constant values. For example, instead of explicitly defining one fixed value per column inside the union node, as with constants, with a union pruning configuration table you can define multiple single values per column which are treated with ‘OR’ logic. You can also define values across multiple columns in a rule. Values across columns are treated with ‘AND’ logic.

Union Pruning Configuration Table

From SAP HANA SPS11 on, you can optimize the execution of union nodes by specifying pruning conditions in a dedicated table called the **Pruning Configuration Table**.



In the example in the figure, Union Pruning Configuration Table, the pruning is defined for a specific calculation view (in the column CALC_SCENARIO). It specifies the pruning conditions to be applied to one or several nodes or columns for this calculation view, as follows:

- The possible operators are =, <, >, <=, >= and **BETWEEN**
- The BETWEEN operator refers to a closed interval (including the LOW and HIGH values).
- If several pruning conditions exist for the same column of a view, they are combined with OR.
- On different columns, the resulting conditions are combined with AND.

For example, you could define that a source to a union should only be accessed if the query is requesting the column YEAR = 2008 **OR** YEAR = 2010 **OR** YEAR between 2012 – 2014 **AND** column STATUS = ‘A’ **OR** STATUS = ‘X’.

This means that if the requested data is 2008 and status ‘C’ then the source is ignored. If the request is for 2013 and status ‘A’ then the source is valid.

We can use input parameters to direct the query to use only the union sources that contain the data required.

Whichever technique you choose, you are providing the optimizer with important information that it uses when making decisions regarding the union. This ensures the best possible performance by avoiding processing sources that are not needed.

Table Partitioning

Data in column store tables is separated into individual columns to support high performance on queries (query pruning) and also for better memory management (load only required column to memory). But it is also possible to sub-divide the rows of column tables into separate blocks of data. We call this **table partitioning**. If column separation is vertical sub-division, think of partitioning as horizontal sub-division.



Partition tables for better load balancing and performance

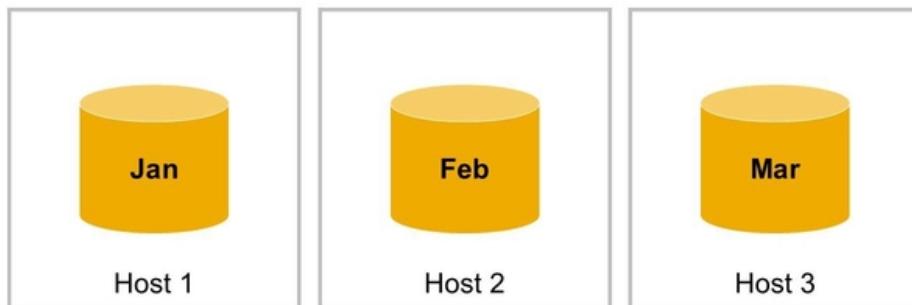


Figure 177: Partition Tables for Better Load Balancing and Performance

Reasons for Partitioning a Column Store

There are many reason for partitioning a column store table including:

- Load balancing in a distributed system** – Spread the data equally over nodes so that each server process a partition in parallel to improve performance
- Parallelization** – Partitioning allows operations to be parallelized by using several execution threads for each table to improve query performance
- Pruning to improve query performance** – Query only hits the partition where the requested data resides. This improves query performance.

Unit 6: Optimization of Models

- **Overcoming the size limitation of column-store tables** – Non-partitioned tables have a 2 billion row limit, partitions also have a 2 billion row limit but you can define up to 16,000 partition per table



Note:

Partitioning tables also helps with more efficient delta merge management as you can perform merging on partitions.

Table partitioning is typically used in multiple-host systems, but it may also be beneficial in single-host systems especially on very large tables where only small sets of data are requested. Table partitioning is typically evaluated in the wider context of overall SAP HANA data management. Often modelers will not be responsible for creating and maintaining partitions but their input is incredibly important to the decisions being made about partitioning. That is why modelers must be aware of partitioning.

Table Partitioning

SAP HANA support three types of table partitioning:

- **Range** – Define partitions explicitly using single values or ranges based on values found in one or more columns. For example, 2010–2013, 2014–2017, 2018. You need to know the table data very well to determine the best columns to use. The table can have primary key or no primary key. If the table has a primary key then the partitioning columns must be part of that primary key.
- **Hash** – Allow SAP HANA to build partitions, you simply choose one or more columns on which HANA computes the hash values and HANA distributes the data evenly. Hash partitioning does not require an in-depth knowledge of the actual content of the table. The table can have primary key or no primary key. If there is a primary key then all partition columns must be part of that key.
- **Round Robin** – New rows are assigned to partitions on a rotation basis. You do not specify the columns to use. The table must not have primary keys

For all these partitioning types the number of partitions can be either explicitly defined or can be automatically determined based on the number of configured servers that host the database.



Note:

Be careful not to confuse table partitioning with table distribution. The latter is where **entire tables** are carefully distributed across hosts for better load balancing. Table partitioning is where we distribute the data from one table across partitions in single or multiple hosts systems.

So how do we partition a column store table? There are a number of methods to create table partitions:

- SQL
- CDS
- SAP HANA Studio wizard

Specifying table partitions with SQL



Specifying table partitions with SQL

```
CREATE COLUMN TABLE MY_TABLE (a INT, b INT, c INT, PRIMARY KEY (a,b))
PARTITION BY RANGE (a)
(PARTITION 1 <= VALUES < 5,
PARTITION 5 <= VALUES < 20,
PARTITION VALUE = 44, PARTITION OTHERS)
```



Figure 178: Specifying table partitions with SQL

Table partitions can be defined, altered and deleted using SQL.

Specifying Table Partitions with CDS



Specifying table partitions with CDS

```
entity MyEntity {
    key id : Integer;
    a : Integer;
    b : Integer;
    t : String(100);
} technical configuration {
    partition by hash (id) partitions 2,
        range (a) (partition 1 <= values < 10,
                    partition values = 10,
                    partition others);
}
```



Figure 179: Specifying Table Partitions with CDS

Table partitions can also be defined using Core Data Services (CDS). This is the recommended approach because the partitioning information is saved in a source file inside a Project. It can then easily be transported and redeployed many times.

Defining table partitions in SAP HANA Studio

One of the easiest ways to define table partitions is to use the partitioning wizard which you find in the SAP HANA Studio. This is found under the Catalog node of the Systems view.



Note:

The partitioning wizard is not available in SAP Web IDE.

Unit 6: Optimization of Models



Simply right-click either the schema name, or first expand a schema and then right-click the Tables node. Then select the menu option Show Table Distribution . You will then see a list of all tables and, if any, the partitioning information for each table. To define partitions for the first time you right-click any unpartitioned table in the list and select Partition Table . You then define the partitions.



Note:

The partitioning wizard cannot be used to add extra partitions to an already partitioned table or to redefine existing partitions or remove individual partitions.

The wizard can only be used to create partitions on an unpartitioned table. But you can easily delete all partitions using the menu option **Merge Partitions** and then begin again.

Distribution of records over partitions



Partition Details of Table "TRAINING"."BIGGER_BANKING"				
Host:Port/Partition/Sub-Partition	Part ID	Range	Record Count	
wdflbmt7215:30003				
1	1	20-30	3,667	
2	2	30-40	4,430	
3	3	40-50	4,127	
4	4	51	387	
5	5		12,071	

Partitioning is transparent for SQL query definitions. This means the SQL code does not refer to partitions. However, the key purpose of partitioning tables from a modeler perspective is to improve performance of the queries. Knowing how a table is partitioned is essential for the modeler. With this knowledge they can then carefully design calculation views, functions or procedures to exploit the way the partitions are formed. For example, a modeler that is aware of partitions should ensure that filters are applied as early as possible on the partitioned column. This means that only data for the partition is loaded to memory.

Partitions are usually defined when a column store table is created, but it is also possible to partition a column store table that was created but never partitioned. You do not have to drop the table to create partitions. You can partition an existing table that is empty or already contains data.

As well as creating partitions, it is also possible to:

- Re-partition an already-partitioned table, for example to change the partitioning specification (hash to round-robin and so on) or to change the partitioning columns or to increase or decrease the number of partitions
- Merge partitions back to one table
- Add/delete individual partitions
- Move partitions to other hosts

Multi-Level Partitions

An easy way to spot if a table is partitioned is to observe the icon alongside the table name. You will see that the icon looks similar to a column table icon but the columns are separated in a partitioned column table. You can also open the **Runtime Information** tab of the table metadata and in the lower part of the screen you can click the **Parts** tab and expand the table node to see the partitions (if they exist).

Finally, we should briefly mention **multi-level partitions**. With multi-level partitions, you can create additional partitions on each partition. It might help to think of this as a hierarchy of partitions. This is typically used when you first want to distribute large tables over multiple hosts (use hash partitioning to load balance) and then for each of those partitions you apply the next level of range partitions (break up by year). So now you have host/year partitioning. Also, multi-level partitioning can be used to overcome the limitation of single-level hash partitioning and range partitioning, that is, the limitation of only being able to use key columns as partitioning columns when you are dealing with keyed tables. Multi-level partitioning makes it possible to partition by a column that is not part of the primary key.



Note:

You cannot define partitions on a row store table



Note:

We have covered only the very basics of table partitioning here. Refer to the official SAP Help documentation for full information.

Caching View Results

In SAP HANA, it is possible to cache the results of a calculation view, in order to speed up the time needed to get the results when executing a query against the view. This feature is useful when you have complex scenarios or large amounts of data.

Unit 6: Optimization of Models

Caching view data involves two types of settings, at the SAP HANA database level, and at the individual view level.

General Settings for View Results Cache

The possibility to cache query results must be turned on by the SAP HANA database administrator.

The system administrator defines some key options, such as the minimum duration of a query that triggers caching for a view. Indeed, if the query results are extremely fast to generate, it might not be useful to use caching.

In addition, because caching view data can use a lot of space on the database system, it is possible to define a size limit for the cache.

Specific Cache Settings in the Calculation View

In the semantics node of the calculation view, switch to the View Properties tab. You will find two settings that relate to the use of cache:

- Cache – Set this flag to enable cache invalidation
- Cache Invalidation Period – Choose when the cache become invalid:

- Hourly

The cache is no longer valid after one hour.

- Daily

The cache is no longer valid after one day.

- Transactional

The cache is invalidated whenever a database transaction affects one of the source tables for the calculation view.

Best Practices When Writing SQL

One of the benefits of using graphical modeling tools is that you don't have to concern yourself with writing SQL code. However, there are many times when you do need to write the SQL code directly, such as when you are developing a function or procedure.

Writing the SQL directly offers the most flexibility to control exactly what you want to achieve and how you want to achieve it, but you may write code that is suboptimal for SAP HANA.

Even experienced SQL coders should pay attention to the following guidelines to avoid applying techniques that do not work well with SAP HANA.



Reduce complexity of SQL statements

- Use variables to break up statements

```
books_per_publisher = SELECT publisher, COUNT (*) AS cnt
FROM :books GROUP BY publisher;
largest_publishers = SELECT * FROM :books_per_publisher
WHERE cnt >= (SELECT MAX (cnt))
```



Figure 182: SQL Best Practices – Variables

As the figure, SQL Best Practices Variables, suggests, you can reduce complexity of SQL statements by using table variables. Table variables are used to capture the interim results of each SQL statement and can then be used as the inputs for subsequent steps. This makes understanding the code much easier and does not sacrifice performance. Using variables also means the calculated data set can be referred to multiple times within the script, avoiding repeating the same SQL statement. So we highly recommend that you use table variables.

Apply Multi-level Aggregation



Use multi-level aggregation

- Use GROUPING SETS

```
SELECT publisher, name, year, SUM(price)
  FROM :it_publishers, :it_books
 WHERE publisher=pub_id AND crcy=:currency
 GROUP BY GROUPING SETS ((publisher, name, year), (year))
```



Figure 183: SQL Best Practices Multi-Level Aggregation

As the figure, SQL Best Practices Multi-Level Aggregation shows, you can apply multi-level aggregation by using SQL grouping sets. Grouping sets aggregate source data at multiple

Unit 6: Optimization of Models

levels all from only one SQL statement. This means that you only read the data once but produce multiple aggregation results that subsequent SQL steps can read.

Reduce Dependencies



Reduce dependencies

- Avoid imperative statements

```
IF <bool_expr1>
THEN
    <then_stmts1>
[ {ELSEIF <bool_expr2>
THEN
    <then_stmts2>}... ]
[ ELSE
    <else_stmts3>]
END IF
```



Figure 184: SQL Best Practices Dependencies

As the figure, SQL Best Practices Dependencies illustrates, you can reduce dependencies by ensuring that your SQL steps are independent from each other. When you wrap SQL expressions in looping constructs or use IF/THEN/ELSE expressions to determine the flow, you inhibit the SQL Optimizer's ability to produce the best plan. This is because you have declared a specific flow that the SQL optimizer cannot easily break. Also, using expressions that control the logic makes it harder to create a plan that can be parallelized and therefore achieve the best performance. Sometimes you cannot help using this technique but it is important that you are aware of the performance implications when doing so.



LESSON SUMMARY

You should now be able to:

- Implementing good modeling practices

Unit 6

Lesson 2

Using Tools to Maximize Optimization



LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Using Tools to Maximize Optimization

Calculation View — Performance Analysis mode

When building a calculation view it is possible to have SAP HANA automatically highlight areas that might lead to poor performance. To do this, you use the Performance Analysis mode.

This tool is launched from inside the calculation view editor in the toolbar (the icon looks like a speedometer). You do not need to activate or build a Calculation View in order to use this tool.

The basic idea is that you are able to react at **design time** to warnings and other information that is being passed to you that might lead to poor performance so that you can consider alternative approaches.



Note:

Some optimization tools require you to first build the calculation view. This is not true for Performance Analysis mode, which analyzes the design-time object.

Performance Analysis Detailed Information



Join_1

JOIN DETAILS for NODE "Join_1"

Left Table:	com.sap.data::SO.Item	Right Table:	com.sap.data::SO.Header
Join Type:	Inner	Referential Integrity:	Maintained
Cardinality:		Proposed Cardinality:	n:1

DATA SOURCE DETAILS for NODE "Join_1"

Data Source	Table Type	Partition Type	Number of Rows
com.sap.data::SO.Item	COLUMN	Not partitioned	6046
com.sap.data::SO.Header	COLUMN	Not partitioned	1029

PARTITION DETAILS for DATA SOURCE com.sap.data::SO.Item

Partition ID	Host	Port	Columns	Number of Rows

Figure 185: Performance Analysis Detailed Information

Unit 6: Optimization of Models

You switch your calculation view to Performance Analysis mode by pressing the button that resembles a speedometer in the toolbar. Once you do this, the Performance Analysis tab appears for all nodes except the semantics node. Choose this tab to view detailed information about your calculation view, in particular the settings and values that can have a big impact on performance.

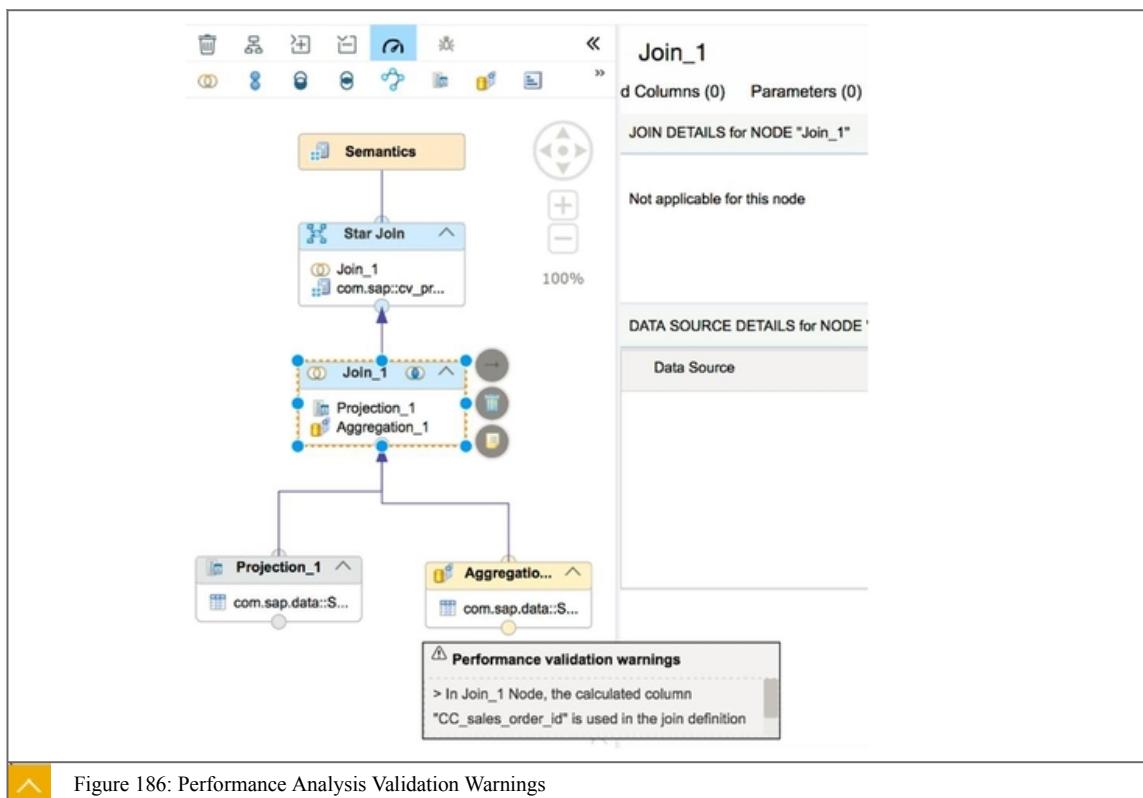
Examples of the information presented in Performance Analysis mode tab include:

- The type of tables used (row or column)
- Join types used (inner, outer, and so on)
- Join cardinality of data sources selected (n:m and so on)
- If tables are partitioned and also how the partitions are defined and how many records each partition contains

**Note:**

In SAP Web IDE Preferences you can set a flag that causes all calculation views to open in Performance Analysis mode by default. This can be useful to ensure design violation warnings are not missed by the modeler.

When you leave Performance Analysis mode, the Performance Analysis tab disappears from all nodes.

Performance Analysis Validation Warnings

As well as information about the calculation view, Performance Analysis mode also provides you with warnings such as the following:

- The size of tables with clear warnings highlighting the very large tables
- Missing cardinalities or cardinalities that do not align with the current data set
- Joins based on calculated columns
- Restricted columns based on calculated columns

This information enables you to make good decisions that support high performance. For example, if you observe that a table you are consuming is extremely large, you might want to think about adding some partitions and then apply filters so you process only the partitions that contain data you need.

**Note:**

This tool first appeared in SAP HANA Studio and is feature complete. For SAP Web IDE, the Performance Analysis mode is not currently 100% complete. Whilst the SAP Web IDE version provides very useful information, we are still missing a few things (for example, join information) This is expected to be developed in future releases. Since the last release of this course we did see the table partitions info added plus table sizes and so on, so we are moving in the right direction. The screen shots in this lesson were taken from the SAP product documentation and this doesn't align exactly with what we have in our landscape, but it gives an idea of what we can expect.

Setting Number of Rows Threshold



The screenshot shows the SAP Modeler Performance Analysis preferences dialog box. The title bar says "Modeler" and "Performance Analysis". There is a checkbox labeled "Always open Calculation Views in performance analysis mode". Below it is a "Threshold Value" input field containing "10000". A note at the bottom states: "Note: Warning icon is displayed next to those table names, which have rows more than the above threshold".

As the figure, Setting Number of Rows Threshold shows, you set the threshold in the Modeler preferences of SAP Web IDE.

Calculation View — Debug Query mode

To examine the behavior of a calculation view when it is being called by a query, you use the Debug Query mode. In Debug Query mode, you can interrogate each node in the calculation view to see how the SQL is generated.

When you switch to Debug Query mode, a generic query is automatically created to call the calculation view. This generated query simulates a reporting tool or application that is sending a query to the calculation view. The generated query requests all columns of the calculation view and does not have filters. In other words, the generated query requests everything from the calculation view. But you can modify the generated query if you wish, for example, to

Unit 6: Optimization of Models

remove columns or add filters to simulate a user requesting different view of the data. Once you execute the generated query on the calculation view, you can then view the SQL that is generated for each node of the calculation view and even execute the SQL at each node if you wish to see the interim result. This helps to ensure that your calculation view is behaving as you'd expect, or perhaps to investigate a performance or output issue that has been reported. One of the most powerful features of Query Debug mode is that you can drill down to the lowest level of the entire modeled stack in the graphical editor of the calculation view to interrogate all the SQL code.

One of the most useful things you can discover when using Query Debug mode is to see how each node is pruned of columns and even complete data sources under various query conditions. For example, this mode is a great way of testing if union pruning is working as you would expect.

Debug Query mode

```

1 SELECT "Age", "Income", "churn_flag" FROM "MA300_A_HDI_CONTAINER_1"."MA300::CVC_BANK_CHURN/dp
/Join_1"("PLACEHOLDER"="$client$$", "$client$$"), "PLACEHOLDER"("$$language$$", "$$language$$"
)) WHERE (( "Age" >= 21) AND ( "Age" <= 26))

```

Age	Income	churn_flag	
23	76850	0	1
24	109479	1	2
22	49583	0	3
21	53878	0	4
25	117020	0	5
24	127569	1	6
23	45330	1	7

Figure 188: Debug Query mode

To get started with Debug Query mode:

1. Open a calculation view, and from the toolbar, choose the button that resembles a bug. This switches the calculation view to Debug Query Mode. When a calculation view is in Debug Query Mode, it is read-only. This means changes to the calculation view definition cannot be made. If you want to make changes you must switch off Debug Query mode.
2. Select the Semantics node.
3. Select the tab Debug Query . You will see the generated default query which simulates a query coming from a reporting tool that includes all attributes and measures and uses a TOP 1000 constraint. You can edit this query if you wish to simulate a particular query execution from a reporting tool that might pass filters in a WHERE clause or PLACEHOLDER for example or remove some columns you do not need.
4. Run the query by choosing the Execute button. Now when you select any node you will see how the SQL appears at that node.
5. Run the SQL for a particular node by choosing Execute for that node. The results for the individual node appears below the SQL.

You can reset the debug query at any time using the reset button which is adjacent to the Execute button.

**Note:**

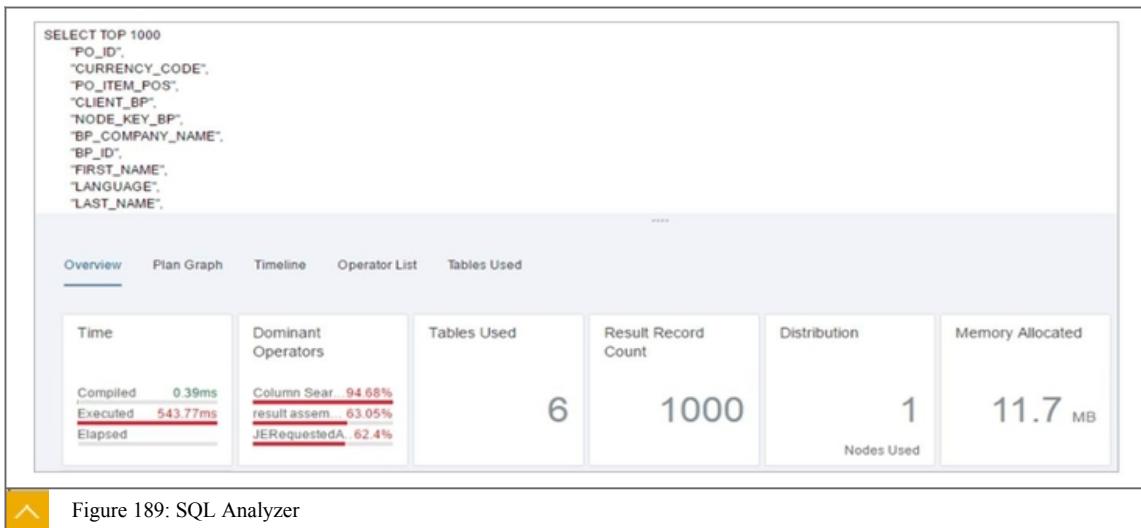
Remember that the Debug Query mode calls the **active** version of the calculation view. This means that if you have made changes to the calculation view but not yet re-built it, then you switch on Debug Query mode, the debug results will not reflect those latest changes.

Analyzing SQL Statement Performance

Although your calculation view may appear well designed, ultimately it is the performance of the calculation view that really counts. You have seen how to display the generated SQL in a calculation view using the Debug Query mode. While this is helpful to see how filters are pushed down and data is pruned, this does not tell us how the SQL actually performs. We really need to know the answers to the following questions:

- What was the total execution time?
- How long did each step take?
- Are there any steps that are taking significantly longer than other steps?
- Are there any bottlenecks?
- How many records did each step process?
- In what sequence are the steps carried out?
- Which operators are being called?
- To what extent was SAP HANA able to parallelize the query?
- Which processing engines are being invoked?
- Were all the query plan steps unfolded?

To help us learn more about the performance of the SQL we can use the SAP Web IDE tool **SQL Analyzer**, as shown in the figure SQL Analyzer.



The SQL Analyzer can be launched from various places in SAP HANA including from the Database Explorer of the SAP Web IDE.

Unit 6: Optimization of Models

In addition to analyzing the SQL of calculation views, the SQL Analyzer can be used wherever SQL is defined, for example in procedures or functions.

To get started with SQL Analyzer, take the following steps:

1. Go to the Database Explorer view of SAP Web IDE.
2. Locate the column view that corresponds to the calculation view that you want to analyze.
You will find this by expanding the runtime container that corresponds to the project where the calculation view was built and then choose **Column Views**. The column views then appear in the lower pane.
3. From the context menu of the column view, choose **Generate SELECT statement**.
4. Right-click anywhere in the SQL code and select the option **Analyze SQL**. A new tab appears with the SQL analysis results.



LESSON SUMMARY

You should now be able to:

- Using Tools to Maximize Optimization

Unit 6

Learning Assessment

1. Where possible, always use CE functions to ensure best performance.

Determine whether this statement is true or false.

- True
- False

2. Identify the principles that you need to follow during modeling.

Choose the correct answers.

- A You should perform calculations before aggregation.
- B You should create joins on key columns.
- C You should push data processing to the client as much as possible.
- D You should reduce data transfer between the calculation views by applying filters as low down as possible.

3. To work with calculation view debug query, you first need to build the calculation view?

Determine whether this statement is true or false.

- True
- False

4. When do you use the SQL Analyzer?

Choose the correct answers.

- A To identify the longest running SQL statements.
- B To investigate the generated SQL for each node in my calculation view.
- C To highlight syntax errors in my SQL code.

Unit 6

Learning Assessment - Answers

1. Where possible, always use CE functions to ensure best performance.

Determine whether this statement is true or false.

True

False

Correct – SAP no longer recommend the use of CE functions as these severely limit the optimization possibilities.

2. Identify the principles that you need to follow during modeling.

Choose the correct answers.

A You should perform calculations before aggregation.

B You should create joins on key columns.

C You should push data processing to the client as much as possible.

D You should reduce data transfer between the calculation views by applying filters as low down as possible.

Correct – You should create joins on key columns and you should reduce data transfer between the calculation views by applying filters as low down as possible.

3. To work with calculation view debug query, you first need to build the calculation view?

Determine whether this statement is true or false.

True

False

Correct — The debug query mode calls the runtime object of the calculation view and does not work on the design-time object.

4. When do you use the SQL Analyzer?

Choose the correct answers.

- A** To identify the longest running SQL statements.
- B** To investigate the generated SQL for each node in my calculation view.
- C** To highlight syntax errors in my SQL code.

Correct — You can use SQL Analyzer to identify the longest running SQL statements and many other issues that might contribute to poor performance.

UNIT 7

Management and Administration of Models

Lesson 1

Working with Modeling Content in a Project

244

Lesson 2

Creating and Managing Projects

262

Lesson 3

Working with GIT within the SAP Web IDE

273

Lesson 4

Using SAP Enterprise Architecture Designer

294

Lesson 5

Migrating Modeling Content

299

UNIT OBJECTIVES

- Analyze and document information models
- Explain the structure of a project
- Build modeling content
- Modify and move modeling content
- Set up collaboration on modeling projects
- Define the key settings of a project
- Set up access to external data
- Manage the lifecycle of a project
- Use the Native Git Integration of the SAP Web IDE
- Understand how SAP Enterprise Architecture Designer supports the design phase of your project
- List the deprecated modeling artefacts

- Explain how to migrate modeling content

Unit 7

Lesson 1

Working with Modeling Content in a Project

LESSON OVERVIEW

In this lesson, you will learn about several features related to the lifecycle of an information model, such as validation, version comparison, documentation, and performance analysis.

Business Example

During the development phase of your project, you are involved in troubleshooting of some AnalyticViews .

You want to use different types of model validation rules to narrow down the list of issues and perform a more specific analysis.



LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Analyze and document information models
- Explain the structure of a project
- Build modeling content
- Modify and move modeling content
- Set up collaboration on modeling projects

Auditing Dependencies between Information Models

Two features are available in the SAP Web IDE to analyze modeling content within a project.

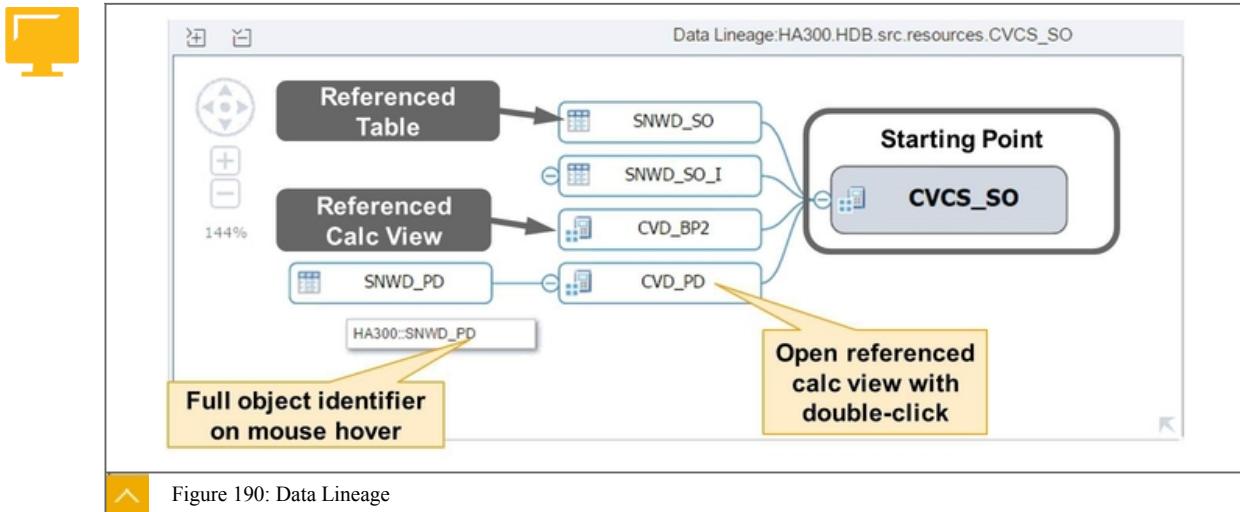
- Data lineage
- Impact analysis

They work in a symmetrical way. For a calculation view you choose, you can show the dependencies with other modeling content.

To use these features, right-click a calculation view and choose
or Other Action → Impact Analysis .

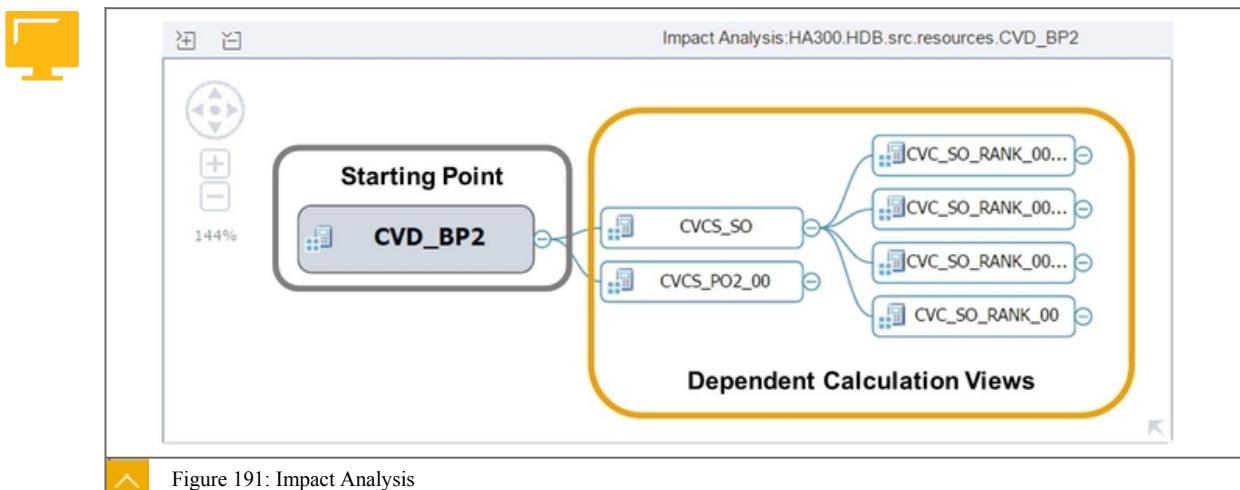
Other Action → Data Lineage

Data Lineage



Data lineage shows all the calculation views and source tables on which a given calculation view depends.

Impact Analysis



The purpose of impact analysis is to show all the chain of calculation views that depend on a given calculation view.



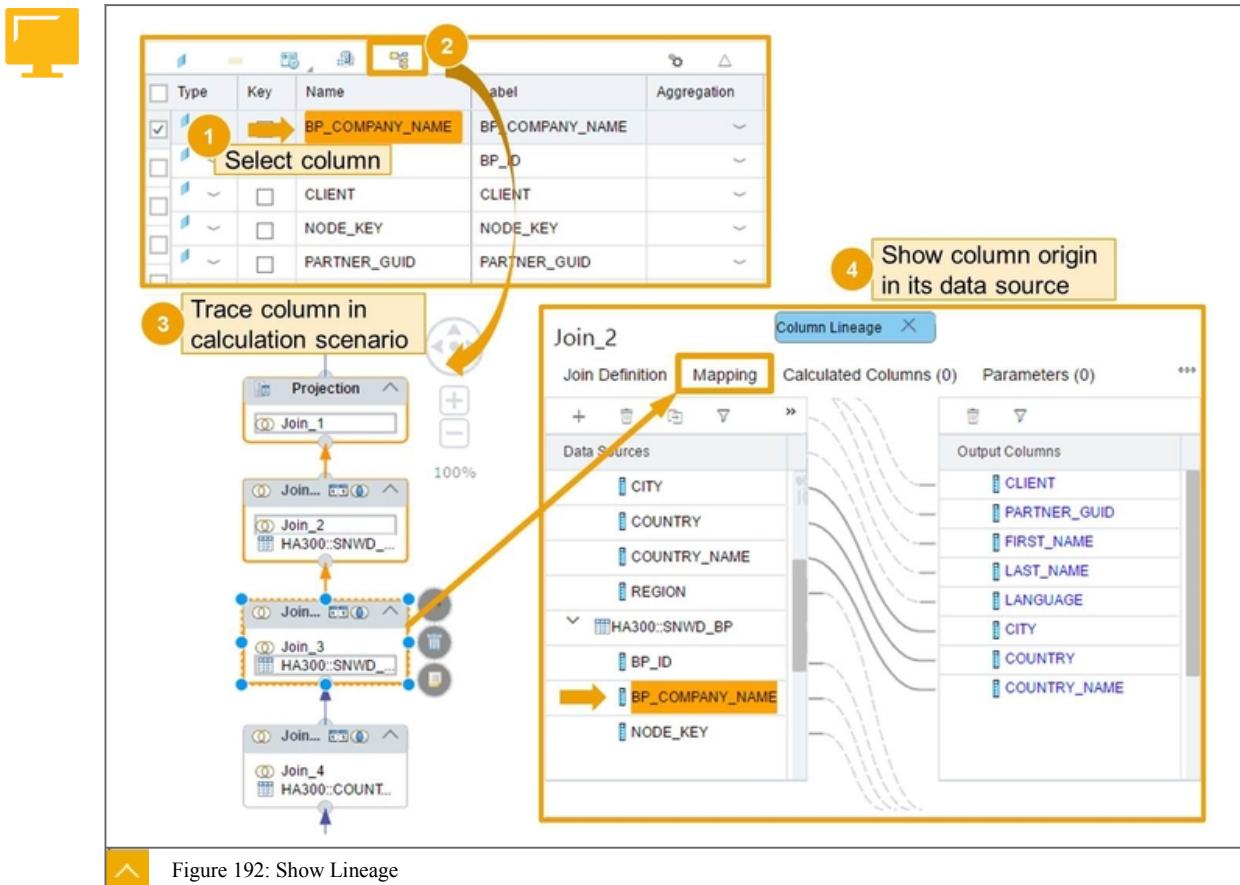
Hint:

From the Data Lineage or Impact Analysis view, you can directly open a calculation view from the dependency tree by double-clicking it.

Tracing the Origin of a Column in a Calculation View Scenario

Another powerful auditing feature is available within a calculation view to show the origin of a column.

Unit 7: Management and Administration of Models



From the Semantics node, you can choose a column and trace its origin with **Column Lineage**.

The column lineage shows, within the scenario of the opened calculation view, all the nodes where this column exists. At the bottom of the calculation view scenario, you find the origin of the column. Opening the **Mapping** tab of this node allows you to identify the source column name in the data source.

Note:

In a cube with star join calculation view, the **Show Lineage** feature only works for private columns, not shared columns from dimension calculation views.

Column Lineage

Column lineage is very useful when columns are renamed within the calculation scenario, or when you want to see quickly where a calculated or restricted column originates from.

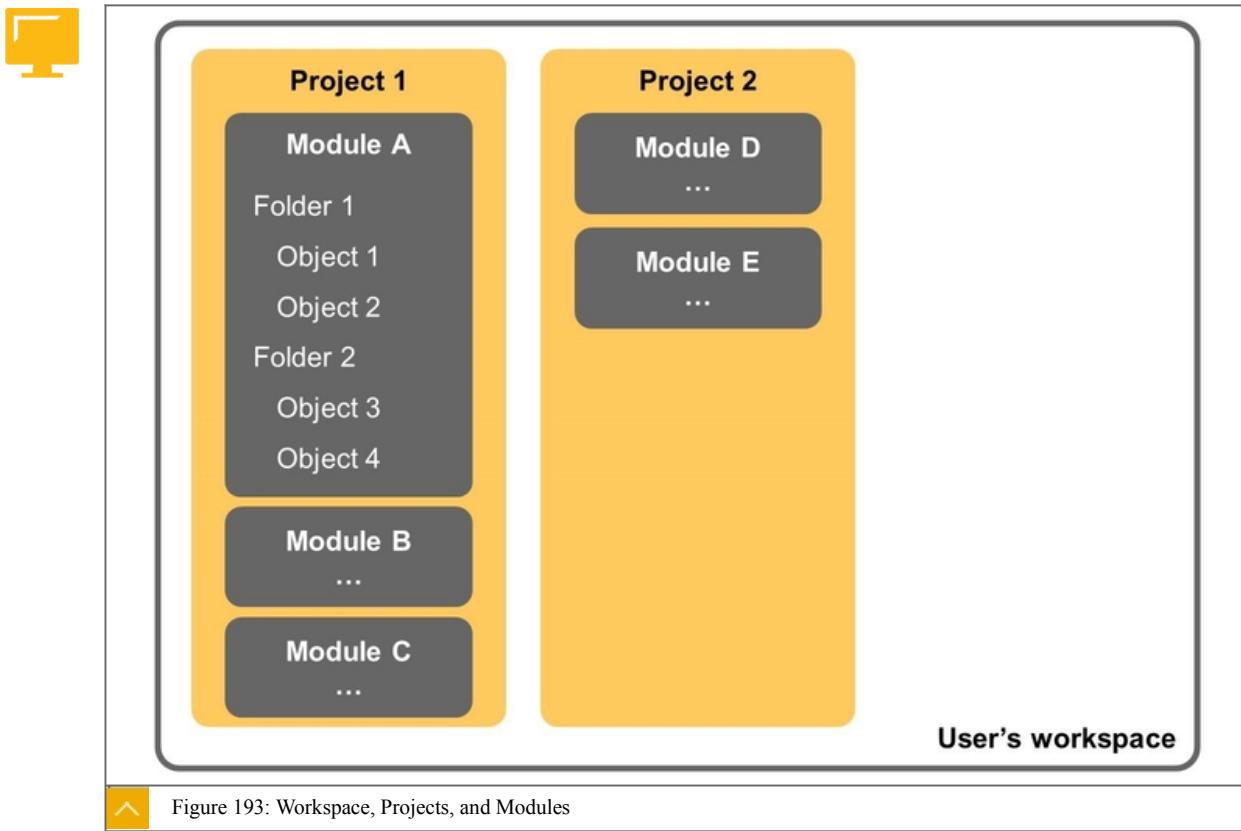
It also helps you to avoid mixing up columns with the same name, but not necessarily the same data that are present in several nodes of a calculation view.

For example, the **NODE_KEY** column in an SAP ERP system is used in many tables to join the master data-defining attributes. You might want to make sure that the **NODE_KEY** column in the output of a dimension calculation view originates from the correct table (for example, the table **SNWD_BP** containing business partners) and not from another table, also used in the calculation view, in which a **NODE_KEY** column is present but does not identify business partners; for example, the **NODE_KEY** in the table **SNWD_AD** containing Contact addresses).

Workspace, Projects, and Modules

The workspace in the SAP Web IDE is a structure where you can store one or several projects. Each user of the SAP Web IDE for SAP HANA has their own workspace.

The workspace of a user can contain as many projects as needed, but each project must have a different name.



When you create a project, you choose a project template, depending on the type of application you want to create. Currently, the SAP Web IDE comes by default with one template, which is called the Multi-Target Application (MTA).

An MTA is comprised of multiple parts (modules), created with different technologies and deployed to different targets, but with a single common lifecycle.

Main Types of Modules

Table 15: Main Types of Modules

Module Type	Key Role in Application
SAP HANA Database (HDB) module	Definition of database objects that provide persistence layer, virtual data model, data access, data processing (functions, procedures)
Java module	Business logic
Node.js module	Business logic
Basic HTML5	User interface (UI)
SAPUI5 module	UI based on the SAPUI5 standard

Unit 7: Management and Administration of Models

Module Type	Key Role in Application
SAP Fiori launchpad site module	UI: SAP Fiori launchpad site with entry point (host) and site's content and configuration
SAP Fiori master-detail module	UI: Typical split-screen layout (one of SAP Fiori design patterns)

In this course, the focus is exclusively on the HDB Module.

**Note:**

The other types of modules are introduced in another course,
Application Development for SAP HANA .

HA450 –

HDB Modules

A HDB module contains all the design-time files that define the database objects that must be deployed together with the application.

Main Database Artefacts Defined in a HDB Module



- Tables
- Table data
- Calculation views
- CDS views
- Table functions
- Procedures
- Analytic privileges
- Synonyms

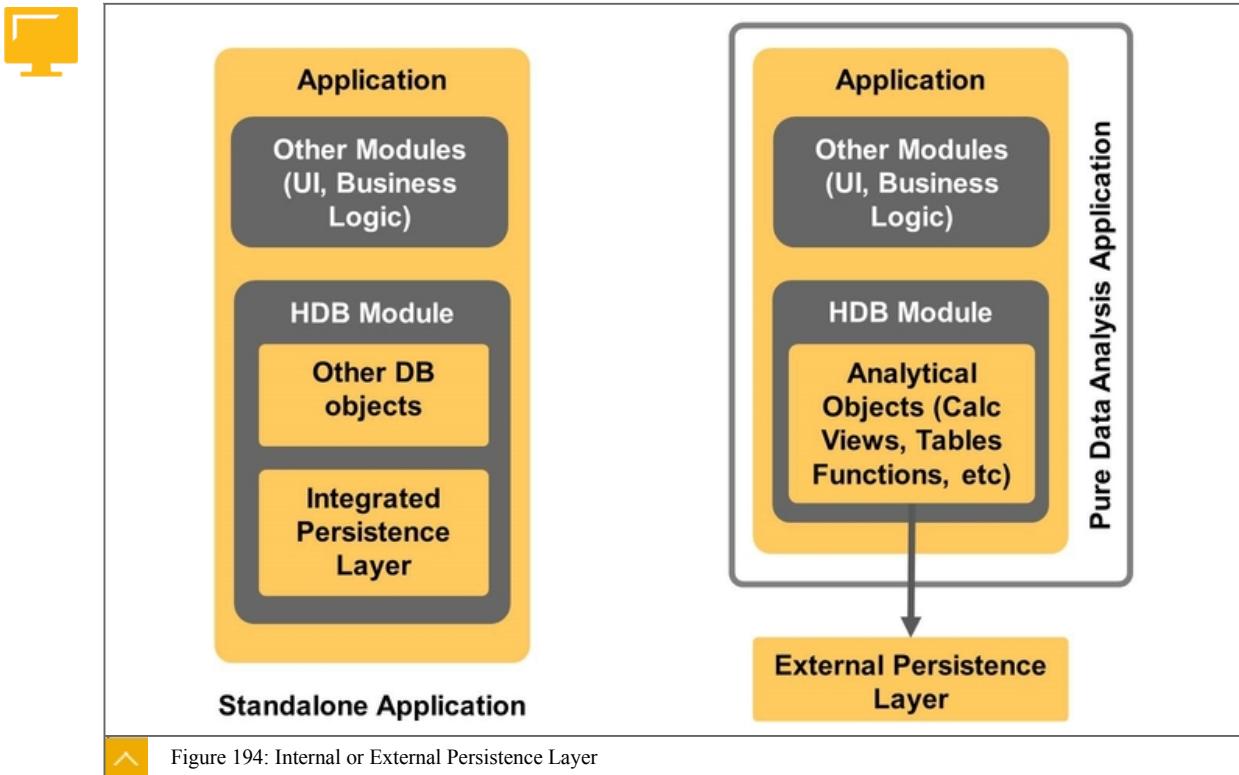
A project can contain several HDB modules. For example, the SHINE for XSA application (SHINE stands for SAP HANA Interactive Education) is comprised of several modules of different types (HDB, node.js, and so on), including two HDB modules.

Persistence Layer: Inside the HDB Module or Outside?

Depending on the type of application, the main data managed by the application can be stored in different places.

A major design option for an application is to decide whether this data is stored in a persistence layer defined by the application itself, or if it is stored in another location, such as a classical database schema. Let's consider the two following examples to illustrate these approaches:

Internal or External Persistence Layer



- **Standalone Application**

With the SAP Web IDE and XSA-based application development, it is very easy to define the entire set of components of an application, including the persistence layer. With this approach, this persistence layer will be defined in the HDB module, mainly by tables defined as CDS entities.

- **Pure Data Analysis Application**

In this scenario, the data sources are not defined by the application itself, but located elsewhere, for example in a classical database schema. A number of application artefacts defined in the HDB module, such as calculation views, table functions, and so on, consume these external data sources. For example, when building a virtual data model on top of an existing application such as an ERP, the data persistence layer is managed by the ERP application layer, so you do not define it in your reporting application.

**Note:**

There is sometimes a lightweight persistence layer built inside the application, in order to manage some application-related information, such as metadata associated with the virtual model itself, or information on the way users consume the views. For example, the history of views queried by each user, or a favorite flag/personal flags that users can assign to the views they use on a regular basis.

Objects Identifier and Namespace

In XS Advanced, each runtime object has an object identifier, which is used to reference this object within the container/schema. To provide a simple example, if the calculation view CV1

Unit 7: Management and Administration of Models

references the calculation view CV2, the execution of CV1 will trigger a “call” to view CV2 by using its object identifier.

A typical structure of a runtime object identifier is as follows:

<Namespace>: :<Runtime Object Name>

- The **Runtime Object Name** is mandatory.
- The **Namespace** is an optional part of the object identifier, and is mainly used to organize logically the runtime objects within the container.

When you work with XS Advanced projects, the naming of runtime objects is distinct from the way design-time files are organized. The main advantage of this approach is that you can more easily relocate design-time objects from one folder to another without impacting the corresponding runtime object name.



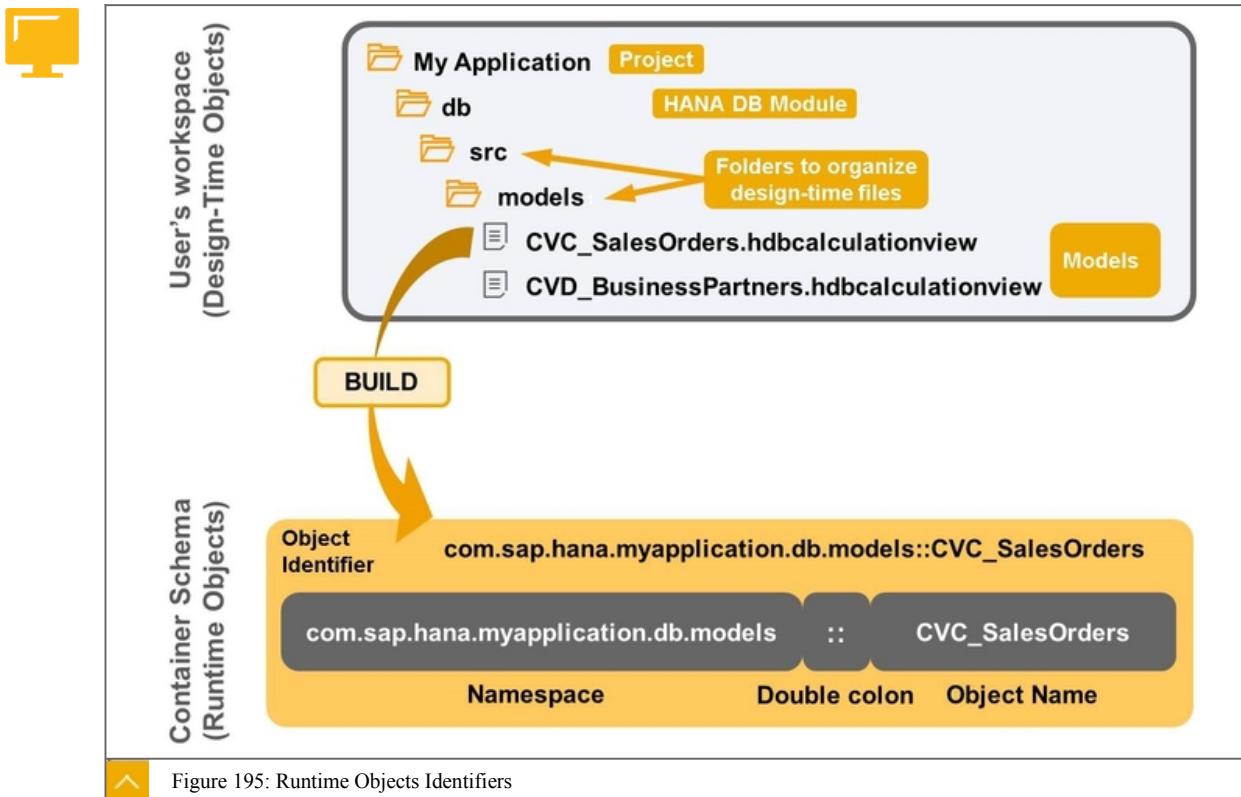
Note:

In this course, the namespace option we have chosen for the main application is to have a namespace prefix **HA300**, applied uniformly to all objects, regardless of the folder location of design-time objects. In other words, the namespace is the same for all the runtime objects defined in our HDB module. So, these objects all have the identifier pattern **HA300 : :<Runtime Object Name>**.

Runtime Objects Identifiers

The object identifiers (including the namespace) are always specified in the design-time objects. That is, the design-time objects must define without ambiguity how the runtime objects will be identified.

Let's take an example of a modeling object, a calculation view **CVC_SalesOrders**, with its design-time version in the SAP Web IDE workspace, and its runtime version (created during build) in the container schema.



The figure, Runtime Objects Identifiers, explains how folders are used to organize the design-time files in the HDB module of your application, and shows what the runtime object identifier could be. In this scenario, the calculation view identifier has a namespace. Let's now discuss the main rules and possible options to define the namespace.

Namespace

The following rules apply to the namespace:

- The namespace is optional. Some objects can be defined with a namespace in their identifier, and others without.
- An HDB module can have no namespace defined at all, or can specify any number of different namespaces.
- The namespace is always identical for all the design-time objects stored in the same folder.
- The namespace must always be explicitly mentioned inside the design-time files, and must correspond to the namespace defined explicitly in the containing folder, or implicitly cascaded from the parent folder(s).

The .hdinamespace File

The definition of a namespace is possible in the src folder of an HDB module, and also in any of its subfolders. This is done with a file called .hdinamespace .

When you create an HDB module, a .hdinamespace file is created automatically in the folder <hdbmodulename> → src and contains a default value "<application_name>.db". You can modify this file to apply different rules than the default to your project's namespace(s).

Unit 7: Management and Administration of Models



```
.hdinamespace x
1 {
2   "name": "MyApp1.db",
3   "subfolder": "append"
4 }
```

Figure 196: The .hdinamespace File

Elements of .hdinamespace File

The content of any .hdinamespace file is always comprised of two elements, `name` and `subfolder`.

Element	Possible values	Default Value
"name"	<ul style="list-style-type: none"> <code>"<Any syntactically correct namespace of your choice>"</code> <code>""</code> [no namespace specified] 	<code>"<application_name>.db"</code>
"subfolder"	<ul style="list-style-type: none"> <code>"append"</code> <code>"ignore"</code> 	<code>"append"</code>

To structure your runtime modeling content in a way that suits your needs, you can add a .hdinamespace file to any sub-folder of the `src` folder. The `"subfolder"` setting determines whether the sub-folder names should be added implicitly to the namespace for objects located in sub-folders (`append`) or not (`ignore`).



Caution:

When you modify the specifications of the .hdinamespace file (or create a new .hdinamespace file in a folder), the new namespace rules are taken into consideration only after the .hdinamespace has been built.

As a general recommendation, the namespace rules should be defined before you create modeling content. Indeed, if you modify these rules after creating modeling content, you have to adjust the object identifiers (which includes the namespace) before you can build your models. And this is even more complicated if dependencies exist between models.

Object Name

The way to define a runtime object name in design-time files is governed by one of the two following rules, depending on the object type.

Table 16: Defining the runtime object names

Type of modeling object	Number of objects per design-time files	Naming rule
Calculation views, table functions, procedures, analytic privileges	Each design-time file defines only one runtime object (*)	The runtime object name is defined inside the object content and should match the design-time file name (without file extension).
Synonyms, CDS tables, CDS views, roles	Each design-time file can define one or several runtime object(s)	The runtime object name is defined only in the object content and is not linked to the design-time file name.

(*) Here, we mean, only one “core” runtime object. As you might have observed, even a simple calculation view generates a core column view, plus other “child” column views for the attributes, hierarchies, and so on.



Note:

In this table, we are not covering the entire typology of models, but only the main ones that are discussed in this course.

Let's take two examples, one for each of the two approaches.

Object Name Example: Calculation View



Design-time file
(Web IDE workspace)

```
CVC_SO.hdbcalculationview
1 <?xml version="1.0" encoding="UTF-8"?>
2 <Calculation:scenario ..... id="HA300: CVC_SO" applyP
3 <descriptions defaultDescription="Summary of Sales Order"
4 <localVariables/>
5 ...
6 ...
7 ...
8 ...
9 ...
10 <DataSource id="HA300::SALES_DATA">
11   <resourceUri>HA300::SALES_DATA</resourceUri>
12 </DataSource>
```

Recommendation:
Filename and Runtime Object name should match

Runtime object
(Database Explorer)

- My HA300_03 Container
- Public Synonyms
- Column Views
- DataStores
- Functions
- Indexes
- Procedures
- Sequences
- Synonyms

B
BUILD

Figure 197: Object Name Example: Calculation View

Technically, the design-time file name for a calculation view can be different from the runtime object name.

© Copyright. All rights reserved.

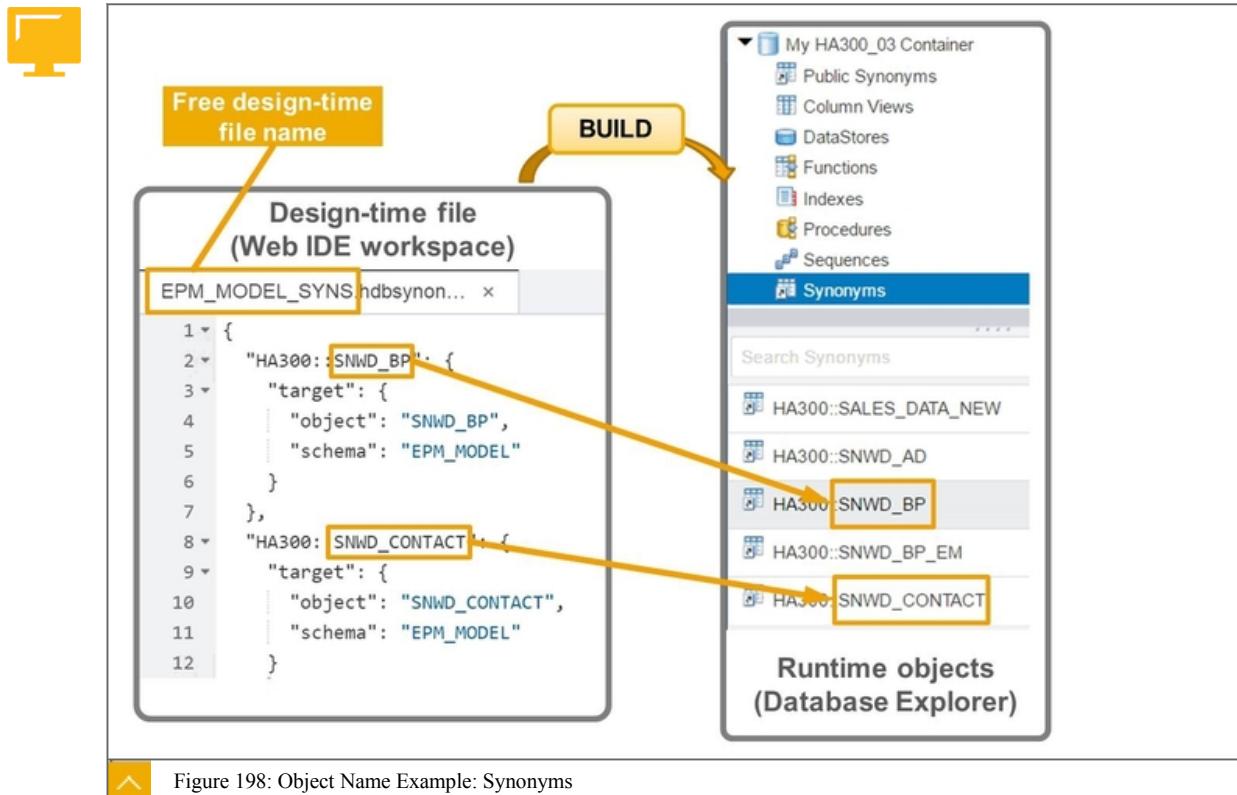
253

For Any SAP / IBM / Oracle - Materials Purchase Visit : www.erpexams.com OR Contact Via Email Directly At : sapmaterials4u@gmail.com

Unit 7: Management and Administration of Models

However, we recommend that you always keep these names in sync to make it easier to find a design-time object based on the runtime object name.

Object Name Example: Synonyms



This second configuration example shows that a design-time file for synonyms can define several synonyms. In that case, there is no specific rule or recommendation for the design-time file name.

Note:
In your HA300_## project, we have chosen an approach where each .hdbsynonym file contains synonyms for objects located in the same schema (for example, EPM_MODEL or TRAINING).

The SAP HANA Deployment Infrastructure

The SAP HANA Deployment Infrastructure (HDI) is a set of services specific to XS Advanced that allows the independent deployment of database objects. It relies on HDI containers.

HDI containers are a special type of database schemata that manage their contained database objects.

These objects are described in the HDB modules of XSA projects, in design-time artifacts that are deployed by the HDI Deployer application. HDI takes care of dependency management and determines the order of activation; HDI also provides support for upgrading existing runtime artifacts when their corresponding design-time artifacts are modified.

One key concept of HDI containers is the fact that the entire lifecycle (creation, modification, and deletion) of database objects is performed exclusively by the HDI. Therefore, you cannot

create database artifacts, such as tables or views, with Data Definition Language (DDL) statements in SQL.

Indeed, if you create a table in a container schema with a plain SQL statement, this means there is no design-time object for this table. So, upon deployment of the container, this table will not be recreated.

 Note:

This is very different from other types of applications that use different persistency frameworks that rely on plain schemata. These frameworks can directly execute DDL statements.

Key Properties of HDI Containers



- A container is equal to a database schema.
- Database objects are deployed into the schema.
- Definitions must be written in a schema-free way.

The name of the container schema is determined only when deploying the container.

- Direct references to external schema objects are not allowed.

These objects must be referenced using database synonyms created within the container.

 Note:

You will learn more about synonyms later on in this unit.

The Database Explorer

Inside the SAP Web IDE for SAP HANA, you can use the **Database Explorer** perspective to view the database artifacts (tables, views, procedures, column views) of one or several containers that you add to the list.

When you do this, even if you are logged on to the SAP Web IDE with your usual SAP Web IDE user, access to the container is done by the technical user **SBSS_xxxxxx** that is created transparently when you add the container to the Database Explorer. This technical user interacts with the database objects on your behalf, for example to view the content of a table or preview the data of a calculation view.

If your container consumes data from an external schema, the technical user must also be granted authorizations to the external schema objects, for example to view the data of an external table referenced by a synonym. You will learn more about this in the unit, **Security in SAP HANA Modeling**.

 Note:

The Database Explorer also allows you to connect to classic database schemas, not managed by the HANA Deployment Infrastructure, to show the database objects and the content of tables, execute queries in an SQL console, and so on.

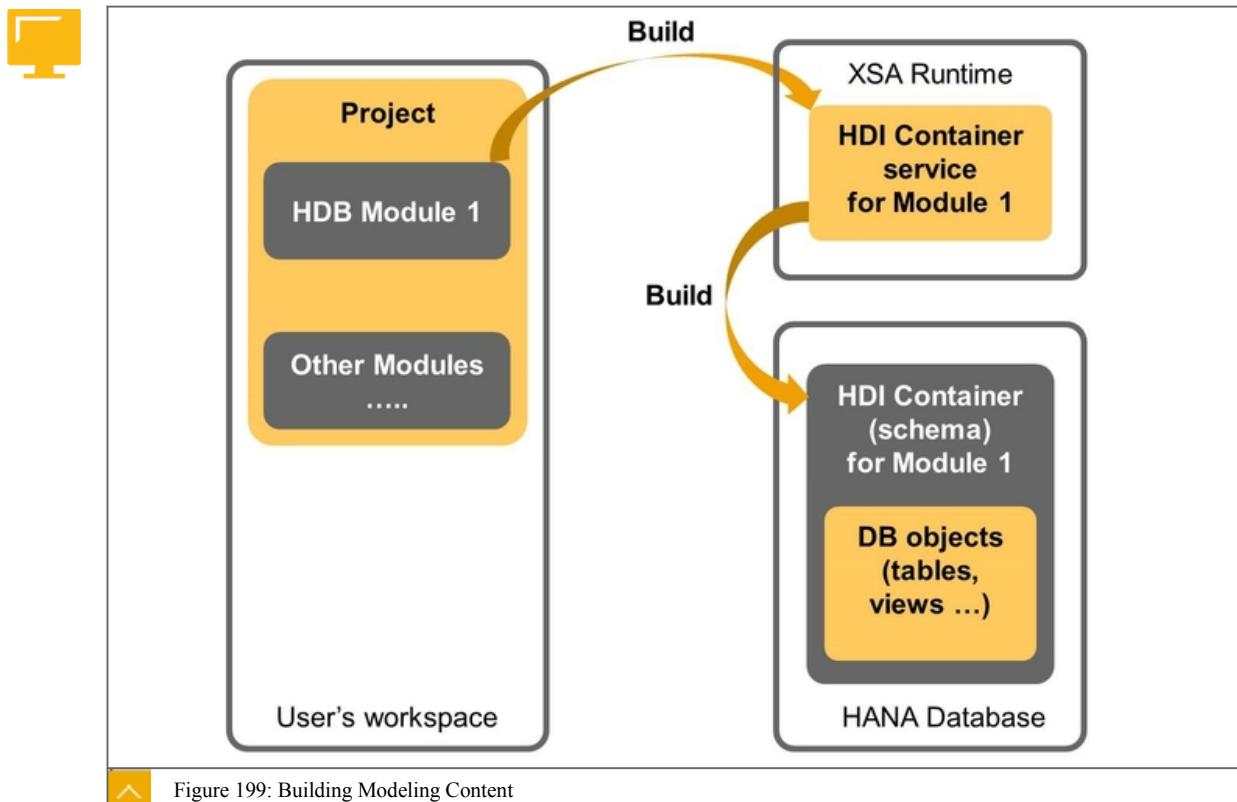
Unit 7: Management and Administration of Models

Building Modeling Content

Building modeling content means creating the runtime objects from the design-time objects, and deploy them to the container (schema).

A container service is also created in XS Advanced to manage the interaction between the XSA environment (in particular the SAP Web IDE, the HANA Deployment Infrastructure, and so on) and the SAP HANA database.

The first build of an HDB module (or some of its design-time content, such as CDS entities defining the persistence) also creates the corresponding container schema.

Structure of Building Modeling Content

A build operation can be executed at the following levels of a project structure:

- An entire HDB module
- A subfolder of the HDB module
- One or several individual objects

Caution:

In SAP HANA 2.0 SPS01, building the entire XSA Project does NOT build the HDB module(s) contained inside the project.

The behavior was different in HANA 2.0 SPS00.

Regardless of what you build (HDB module, subfolder, or single object), there is always an end-to-end dependency check applied to each object that is part of the build scope. So, even

if you build a single object, there will be a recursive check of all the objects it references to make sure these objects already exist as runtime objects or are defined in a design-time object and built at the same time.

Build Errors

When you build modeling content defined in a HDB module, you might come across different issues which are described in the build log that you can consult in the console. Some are related to the dependency checks we have just discussed, others have to do with the consistency of design-time objects (for example, the namespace settings of design-time folders). Another classical build error is when two different design-time files provide a definition for the same runtime object.

Let's try to list a number of frequent root causes for build errors.

Most Common Errors during Build Operations



- The project has just been imported but is not yet assigned to a space.

Indeed, a build always triggers the deployment to a container. A condition is that the space in which to deploy the container service must be specified. Your SAP Web IDE user must have the `Developer` role in the space.

- The definition of an object on which another object depends is not provided, or it is provided but this object has not been built yet.
- The definition of a runtime object is provided by several design-time files.
- There is a namespace inconsistency between the content of a design-time file and the namespace property of the folder in which it is located.
- The service to access external schemas is not defined and running in the target space.
- The synonyms for external schema access are not built yet.
- There is an object-specific design error.

For example, no measure is defined in a Cube calculation view.

- There is an inconsistency between the object you build and a referenced object.

For example, there is a mismatch in a column name between two objects with a dependency.

Import and Export of Modeling Content

The SAP Web IDE for SAP HANA offers import and export features within the workspace.

These features can be used to keep several versions of your modeling content, and re-import all or part of this content if needed.

They are also useful for quick sandboxing. Indeed, even in your own workspace, you cannot copy/paste an entire project. Instead, you export this project and import it. You will only need to change the imported project name in the `Import to` field of the Import dialog box.

Additionally, import/export can be used to share in a simple way your modeling content with another developer. For example, when a series of synonyms to external database objects have been defined in your project, you can easily share the synonyms definitions with another developer by sending the relevant design-time files for synonyms.

Correspondingly, you can import an individual file or an archive. In that case, most of the time, the archive needs to be unzipped.

Unit 7: Management and Administration of Models

Exporting Modeling Content

An export of modeling content can be executed at different levels of a project structure, and applied to either a single file or a folder.

- A single design-time file is exported as is, with the source filename and extension.
You cannot export a multi-selection of individual files.
- A folder within the project structure (up to the project folder itself) is always exported as a .zip archive.
The .rar standard is not supported for project content export/import. This standard is used in .mtar (MTA archive) files, but these files are used for application deployment, which is different from design-time content export/import.

Importing Modeling Content

Corresponding to the export, you can either import an individual file or a .zip archive. In the latter case, most of the time, the archive needs to be unzipped.

Two specific cases always require an intermediate step, after the import, before you can build the imported objects:

- When you import an entire XSA project, you must assign the project to a space before building the project or any of its components (in particular the HDB module).
In the workspace, right-click the project and choose **Projects Settings**. In the **Space** section, assign a space and choose **Save**.



Note:

Only the spaces to which you are assigned as a developer are listed.

- If you import a HDB module into a new project (for example, after exporting this HDB module from another project), you must first identify the imported content as an HDB module in the project.
In the workspace, right-click the new (imported) module —it is not yet known as an HDB module even if its name is something like **HDB**, **dbmodule** or **db**— and choose **Convert to → HDB Module**.
The module entry is added to the MTA descriptor file **mta.yaml** and, from this point on, is considered as an HDB module.

Copy, Rename, Move, and Delete Modeling Content

The structure of a project in the SAP Web IDE for SAP HANA looks like a classical file structure. And indeed, it is. But the changes you make to this structure by copying, renaming, moving or deleting files (or folders) can have strong impact on the consistency of your project. And this impact does not necessarily show up immediately during the file structure modification: it generally pops up when you build the modeling content.

So, whenever you copy, rename, move, or delete modeling content, you must keep in mind the key rules that govern the build of database artifacts:

Main Rules for a Consistent Management of Modeling Content Files



- In an entire HDB module, the definition of a given runtime object (`<namespace>::<object_name>`) cannot be provided more than once.
- The namespace defined in the design-time file of a database object must correspond to the namespace setting applied to the folder in which it is located
- A build operation always checks the end-to-end dependency between modeling content across all the HDB module, but only builds the design-time files you have selected for the build operation.
- During a build operation, the checks apply to all the runtime objects that are already built, and also all the objects included in the build scope (that is, in case of a partial build, the design-time files you have selected).

A design-time file that has never been built and is not part of the build operation is ignored.

Copying Modeling Content

When you copy/paste a design-time file within the same folder, you must provide a different name for the copy. This is not the case if you copy/paste a file to a different folder where there is not a file with the same name.

However, in any case, the content of the design-time file is not modified at all. Which means you have in your HDB module two design-time files with identical content.

At least, you must ensure that you rename all the runtime objects defined in the design-time file to make these objects unique in the whole HDB module. And, if needed, you must also change the namespace according to the target folder namespace settings.



Note:

If you copy a design-time file that was already opened in a code or graphical editor, keep in mind that copy/pasting does not open the new file (the copy). The file visible in the editor is still the original one, when—in many circumstances—you might want to actually modify the copy.

Moving Modeling Content

When moving modeling content, one key rule that you must think about is the namespace setting of the target folder.

- If the namespace settings for source and target folder are the same, moving the file has no particular impact on objects that reference the moved object.

However, a build of the moved object can be successful only if the build scope includes the source folder, in order to execute/acknowledge the “deletion” of the moved file in its source folder. If not, the build will fail.

- If the namespace settings for source and target folder are different, you must at least adapt the namespace. In this case, you must first perform an impact analysis to check whether some existing objects reference the one you are about to move.

Renaming Modeling Content

For the HDI builder, renaming a design-time file is like deleting a design-time file and creating a new one with the same content. You must be aware of the following rules and recommendations:

Unit 7: Management and Administration of Models

- A design-time file that creates only one “core” runtime object (for example, a calculation view or a table function) should always have the same name as the runtime object it defines. For example, after renaming the design-time of a calculation view, it is recommended to also change its ID in the code by opening the design-time file with the Code Editor.

This is not a technical requirement, but rather a best practice to keep your design-time content readable.

- You must check if database objects reference the object you plan to rename. Generally speaking, renaming objects that have dependencies entails manual modifications of the references to this object in other objects. Ideally, you should do this only on exception, because it is error-prone, or limit this practice to test objects, typically when you copy — and rename — an existing object in order to test changes you make to this object while keeping the source object.
- You must be particularly careful with partial builds. If you rename an object that was already built and had dependencies, the renamed file is seen by the HDI Builder as a new design-time. The original design-time file will be seen as deleted by the HDI Builder only when you build either the entire HDB module, or any of its sub-folders that contains the renamed file.

SAP HANA 2.0 SPS02 introduces a new feature, **Rename & Refactor**. This allows the following operations:

Rename & Refactor Functionality

- Rename a runtime object (e.g. a calculation view)
- Check which other calculation views reference the renamed object, and adjust the reference accordingly

This does not work in artefacts where the references are located in SQL or SQL Script code (for example in a procedure).

- Optionally, rename also the design-time file.

This is currently only supported if the initial design-time file name and runtime object name were identical (setting apart the runtime object namespace and design-time file extension).

There are two ways to invoke this functionality:

- From the workspace

Right-click the object you want to rename and choose **Other Actions → Rename & Refactor**

- From the calculation view editor

In the **View Properties** tab of the **Semantics** node, click the **Edit** icon next to the runtime object **Name**.

Deleting Modeling Content

Deleting modeling content can have surprising effects, especially in case of partial builds. Indeed, the deleted design-time file is only seen as actually deleted by the HDI Builder when you build the entire HDB module, or any of the subfolders that contained the deleted file.

In other words, the smallest build scope you can think of, in order to validate the deletion of a design-time file, is the folder in which the deleted file was located. So, even if it is empty, this folder is extremely precious.

Indeed, if you delete this folder as well, you can only validate the deletion of the design-time files it contained by building a parent folder. Suppose this parent folder is the `src` folder and it contains one (or several) models that you know cannot be built at the moment (for whatever reason).

So, when deleting modeling content, you should apply the following best practices:

- Before deleting a design-time file, always perform an impact analysis.
- Before deleting a folder, always perform a partial build at this folder level. If the build is successful, it means that all the runtime objects that were defined in design-time files from this folder have actually been (successfully) deleted.

Using the Search/Replace Feature

The SAP Web IDE for SAP HANA offers several find/search and replace features that are useful to manage the renaming of objects and the code adjustments that must be performed manually.

Within the Code Editor, you can use Find (**Ctrl+F**) or Replace (**Ctrl+H**) to locate easily (and replace if needed) the ID of an object.

On the right toolbar, you can also use the Advanced Repository Search to look for a particular text string, either in the name of design-time files (within a folder, a project, or the entire workspace) or within the content of the design-time files. You can directly open a design-time file from the search results.



LESSON SUMMARY

You should now be able to:

- Analyze and document information models
- Explain the structure of a project
- Build modeling content
- Modify and move modeling content
- Set up collaboration on modeling projects

Unit 7

Lesson 2

Creating and Managing Projects

LESSON OVERVIEW

In this lesson, you will learn about how to create a brand new project and how to define the key settings of a project. You will also get the main information on how to consume data from external data sources.

Like in the entire course, we will only focus on modeling within an HDB Module, so we will not cover any other type of module (Java, Node.js, HTML5, UI5, and so on).

Business Example

You are working as a Modeler at a client site and need to create a project that will contain the modeling content. You need to understand the key options that you need to define to set up the project.

Your project will mainly consist in Calculation Views modeling based on data located in an external schema. So you want to understand the key steps needed to enable access to this external data.

Finally, you also want to understand the key difference between developing/building an application in a space, and deploying a ready-made application to a specific target space for testing or production.



LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Define the key settings of a project
- Set up access to external data
- Manage the lifecycle of a project

MTA Project

A Multi-Target Application (MTA) project is comprised of one or several modules of different types. You learned about modules types in a previous lesson. Here, we focus on the SAP HANA Database (HDB) module type.

However, it is important to understand some key settings of an MTA project, and some of them do not have to do only with HDB modules, but are related to the lifecycle of a project in general.

When you create a new project, you have to define a few parameters. Let's review these parameters.



Table 17: Key Settings of an MTA Project

Setting	Description	Example
Project name	The name given to the new project in your workspace	
Application ID	The identifier of the multi-target application, used during deployment. It must be unique in the target environment, and uses the reverse-URL dot-notation.	com.sap.mta.sample
Application version	A three-level application version number (Major.Minor.Patch), used to handle successive deliveries of the same application to runtime environments.	1.0.3
Description	A free text describing the purpose of the MTA.	
Space	The space where you want to build your project content during the development phase.	DEV

All these settings are stored in the project descriptor file, which is named `mta.yaml`. This file can be edited with the code editor, or with a new MTA editor available from SAP HANA 2.0 SPS01 onwards.

HANA Database Module

As you have already seen, a Multi-Target Application can include one or several modules of the type HDB (SAP HANA Database). In the context of modeling, an HDB module can define a local persistence (a persistence layer that is defined and deployed in the application itself).

When you build a virtual data model based on calculation views —even if the source data is stored outside of the application— the HDB module contains the database artifacts that define this virtual data model.

When you create a new HDB module, you must provide a number of properties for the module. Most of these properties are stored in the descriptor file of the application: (`mta.yaml`), except for a few. In particular, the namespace property of the module is not stored in the descriptor file but is materialized by the `.hdinamespace` file located in the `src` folder.



Table 18: Key Settings of an HDB Module

Setting	Description	Example
Name	The name of the module. It should be unique within the entire descriptor file (<code>mta.yaml</code>).	core-database-module
Type	<code>hdb</code> is the reserved module type for a HDB module.	hdb

Unit 7: Management and Administration of Models

Setting	Description	Example
Path	The relative location of the module root within the file structure of the project.	db
Schema Name	A specific schema name for the HDB module container, used during build and deployment.	SAMPLE_SCHEMA
SAP HANA Database Version	The SAP HANA database version against which the artefacts of the HDB module must be validated.	_schema-version: "2.0"

Default Schema Name of the HDB Module Container

As you might have noticed in the HA300_## project, specifying a schema name is optional.

From SAP HANA 2.0 SPS01 onwards, the default schema name for the container generated during the build of a project is the following string, after capitalization and replacement of some special characters (for example, any dash “-” is replaced with an underscore “_”):

```
<project-name>_<hdi-container-service-name>_<numeric_increment>
```

 Note:

The default schema name generated during a deployment with the XSA Command-Line Interface (CLI), that is, outside of the SAP Web IDE, is different: <project-name> is replaced with <application-ID> (because a project name makes sense only in the context of the SAP Web IDE). It is also possible to specify the schema name at deployment time with an mta extension descriptor (a .mtaext file).

Specifying a Schema Name for the HDB Module Container

You can specify a schema name for your HDB module. This is done in the descriptor file `mta.yaml` of your project, and more specifically in the parameters of the `hdi-container` service that is defined as a resource used by the HDB module.

The following sample shows the additional `parameters` section of the `mta.yaml` file where the schema name is defined:

```
resources:
- name: hdi-cont
  parameters:
    config:
      schema: <YOUR_SCHEMA_NAME>
    properties:
      hdi-container-name: ${service-name}
    type: com.sap.xs.hdi-container
```

Then, the schema generated during the first build of the HDB module is named as follows:

```
<YOUR_SCHEMA_NAME>_<numeric_increment>
```

**Caution:**

It is possible to force the schema name to be exactly what you specify in the `mta.yaml` file, by adding as follows:

```
config:
  schema: <YOUR_SCHEMA_NAME>
  makeUniqueName : false
```

However, this can generate schema naming conflicts in case several projects use the same schema name.

- 1
- 2
- 3

To Find the Actual Container Service Name and Schema Name of an HDB Module
You have built an HDB module from the SAP Web IDE for SAP HANA, and want to know the actual name of the HDI container service, and the name of the corresponding database schema.

1. Add the container to the Database Explorer.
 - a) Choose Tools → Database Explorer .
 - b) Choose the + (Add a Database to the Database Explorer) icon.
 - c) In the Add Database dialog box, select the database type project name (or a part of it) in the search field. The corresponding container(s) is (are) displayed.
 - d) Select the relevant container.
The entry in the Name column corresponds to the actual XS Advanced service that manages the container.
 - e) In the Name to Show in Display field, enter a meaningful display name for the container.
 - f) Choose OK.
2. Open an SQL console connected to the container.
 - a) If needed, select the container in the top-left list (it should be already selected after the previous step).
 - b) Choose Open SQL Console .
3. Execute the following SQL query:

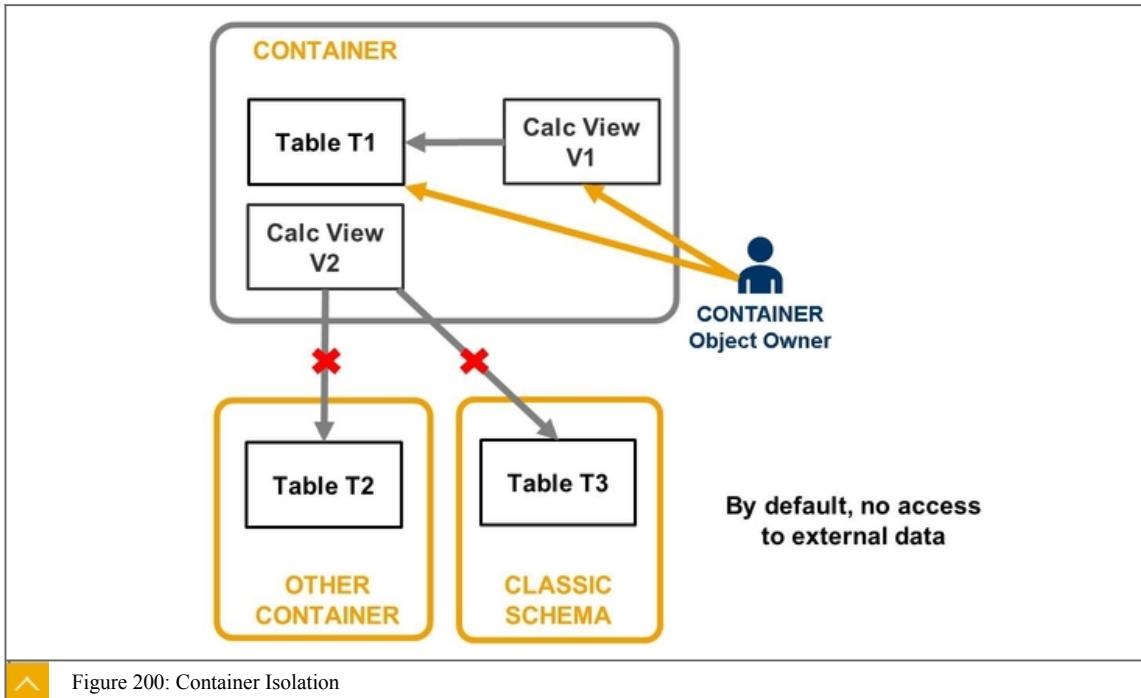

```
select CURRENT_SCHEMA from DUMMY
```

 - a) Enter the SQL statement.
 - b) Choose Run (or press F8).
The schema name of the container is displayed in the SQL query results.

External Data Access Setup

The SAP HANA Deployment Infrastructure (HDI) relies on a strong isolation of containers and a schema-free definition of all the modeling artifacts.

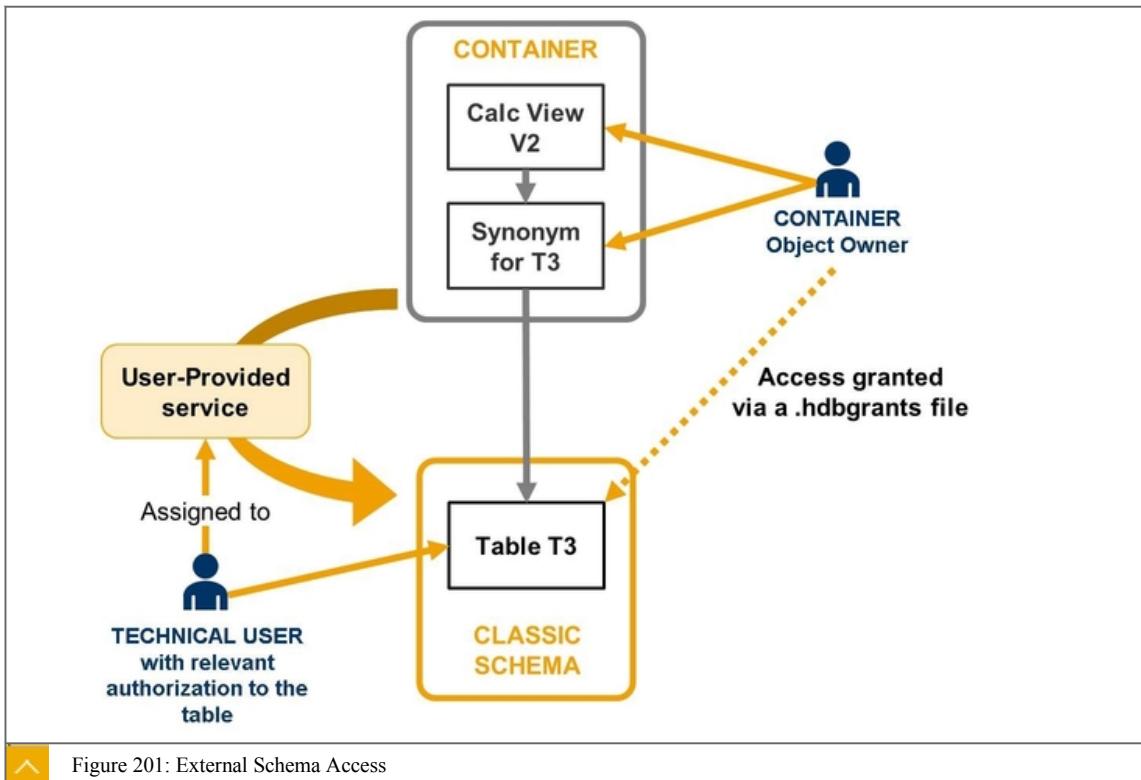
Unit 7: Management and Administration of Models



By default, a container (that is, the database materialization of an HDB module) is fully aware of the database objects it contains (tables, calculation views, and so on), but has no access at all to other database schemas, whether it is another container or a classic database schema.

Defining Access to an External Schema – End-to-End Scenario

Let's consider the end-to-end setup of an access to an external schema.



First of all, you need a **user-provided service**. It is an XS Advanced service that an administrator must set up in the space to which your project is assigned. The main properties of this service are the following:

Properties of a User-Provided Service for External Schema Access



Table 19: Properties of a User-Provided Service for External Schema Access

Parameter	Description
host	The identifier of the SAP HANA system in which the target schema is located Example: wdf1bmt7215.wdf.sap.corp
port	The port used to access the database (SQL port of the indexserver) Example: 30015
user	The name of the technical user assigned to the user-provided service
password	The password of this technical user
driver	com.sap.db.jdbc.Driver
tag	hana
schema	(optional) The name of a schema to which the user-provided service will give access. If blank, the service can be used to provide access to several schemas of the target database.

Technical User

The **technical user** assigned to the user-provided service must have the relevant authorizations to the target objects—and with the grant option—so that some (or all of) these authorizations can be granted to the object owner of your container.



Note:

The authorizations of the technical user define the maximum set of authorization that can be, in turn, granted to the object owner and application user.

The next step is to include a **reference to this service in the project descriptor file** `mta.yaml`. This is done in the `resources` section, with the following code:

```
resources:
- name: <logical_name_of_user_provided_service>
  type: org.cloudfoundry(existing-service)
  parameters:
    service-name: <actual_name_of_user_provided_service>
```



Note:

It is possible to omit the `parameters` section if you put in the `name:` key the actual name of the user-provided service that is running in your project space.

Unit 7: Management and Administration of Models

You must also, in the `mta.yaml` file, **register the user-provided service as a dependency of the HDB module**. This is done in the `requires` section of the HDB module definition, with the following code:

```
requires:
  - name: <logical_name_of_user_provided_service>
```

Then, you **can grant the relevant authorizations** to the object owner and application user. This is done in a `.hdbgrants` file.


Note:

You will learn more about this in the unit, Security in SAP HANA Modeling.

Creating Synonyms

The final step is to **create synonyms to the target objects** of the external schema. The synonyms declaration is done in a `.hdbsynonym` file. There are two ways to provide the definition of the synonyms:

- In the same `.hdbsynonym` file (like in the example below),
- In a dedicated `.hdbsynonymconfig` file

These file types can be edited either with the code editor, as in the example below, or a dedicated synonym editor.

Extract of an EPM_MODEL.hdbsynonym File



```
{
  "HA300::SNWD_BP": {
    "target": {
      "database": "H00",
      "schema": "EPM_MODEL",
      "object": "SNWD_BP",
    }
  },
  "HA300::SNWD_PO": {
    "target": {
      "database": "H00",
      "schema": "EPM_MODEL",
      "object": "SNWD_PO",
    }
  }
}
```

The `"database"` field is optional, needed only in cross-database access scenarios.


Caution:

Public synonyms CANNOT be used in XSA projects to access external objects such as tables. Local synonyms must be defined INSIDE the container to access the data.

When adding a data source to a node, there are mainly two possible scenarios:

- Your data source is defined locally in your container or a synonym for an external data source (in another schema or container) already exists in your local container

- In the **Find Data Source** dialog box, select the existing data source and choose **Finish**.
- You need to access an external data source that is not yet referenced by a local synonym.

You must first define the synonym (as described above) in a **.hdbsynonym**, and maybe adjust the **.hdbgrants** file to make sure the external object will actually be accessible.

From SAP HANA 2.0 SPS02 onwards, the synonym can be created automatically at the time you need to access the external data source from the Calculation View editor. To do that, in the **Find Data Source** dialog box, you must first select the external service and then enter a search criteria. Then, select the object and choose **Next**. You can adjust the synonym name (without changing the namespace). Choose **Finish**.

Two files, **grantor.hdbgrants** and **synonyms.hdbsynonym**, are created (if needed) or adjusted in the same folder where the consuming calculation view is located. An entry in the **.hdbgrants** file is created for each external object.

 Note:

The location of the automatically generated **.hdbgrants** and **.hdbsynonym** files might not suit your requirements, in particular if you prefer to keep all synonyms defined in a more central place (for example, a dedicated folder). In that case, we recommend that you go on creating the synonyms manually.

Dynamic Target Schema Identification

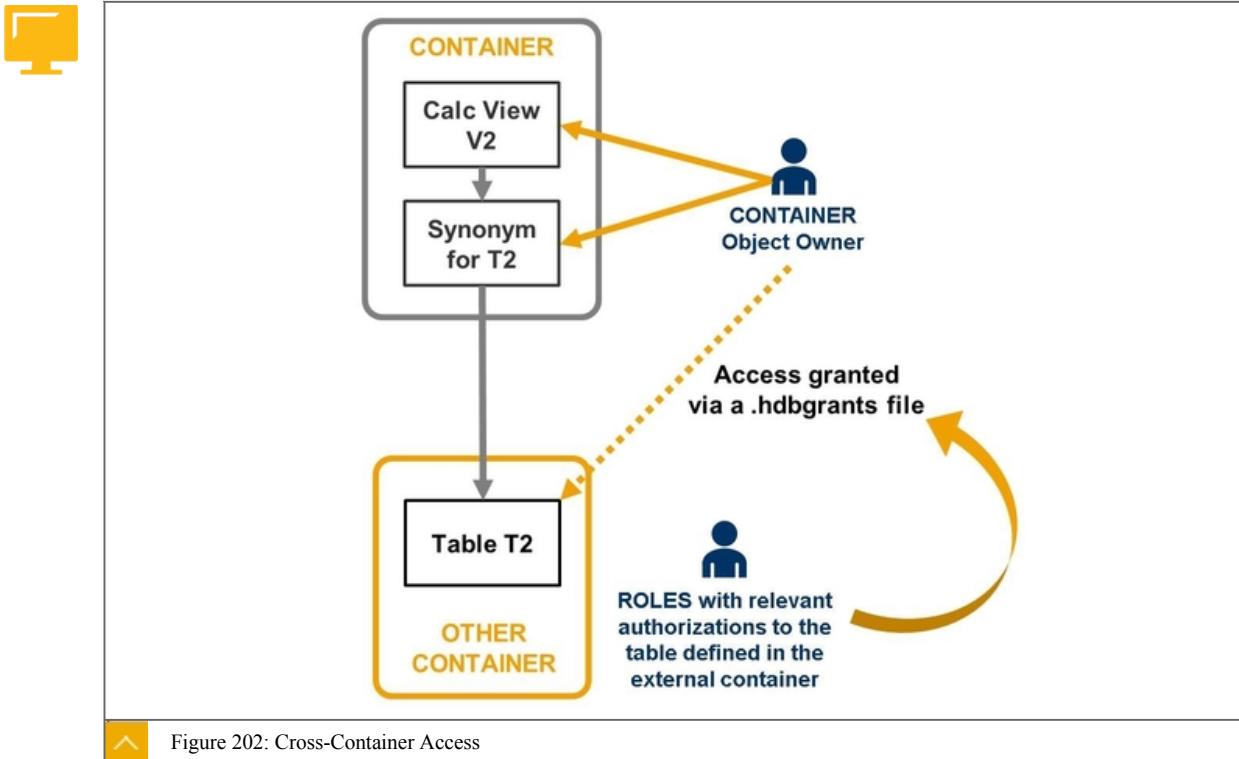
If you use a **.hdbsynonymconfig** file, this gives you additional possibilities to provide the actual target schema name on deployment time. There are two different ways to achieve this:

- You can retrieve the schema name based on what is specified in the user-provided service (or an existing HDI container in case of cross-container access)
- For example, you can replace `"schema": "EPM_MODEL"` with `"schema.configure": "<logical_name_of_user_provided_service>/schema"`
- You can use a logical schema name instead of a real schema name.
- For example, you can replace `"schema": "EPM_MODEL"` with `"logical_schema": "<logical_schema_name>"`
- The mapping is done in a **.hdblogicalschema** file.

 Note:

You can learn more about these two techniques in the SAP HANA Developer Guide for SAP HANA XS Advanced Model. Search respectively for **Templates for HDI Configuration Files** and **hdblogicalschema**.

Cross-Container Access



When you need to access data from another HDI container, the setup is relatively similar to what you have just learned for an external (classic) database schema. Let's point out the main differences:

- There is no need for a user-provided service if the external container service is running in the same space as the one your project is assigned to.
Indeed, you can directly reference in your `mta.yaml` file the external HDI container service.
- You must create relevant roles inside the external container.
- The `.hdbgrants` file does not refer to database object authorizations of the technical user assigned to the user-provided service, but to the dedicated roles created inside the external container (see the previous point).

If you want to learn more about synonyms, external schema and cross-container access, in addition to the SAP HANA documentation, you might want to consult a series of three detailed blog posts, starting at the following URL: <https://blogs.sap.com/2017/01/06/synonyms-in-hana-xs-advanced-introduction/>.



Note:

The blog posts are about SAP HANA version 2.0 SPS00, but most of the information they provide is still relevant for SAP HANA 2.0 SPS01.

Lifecycle of a Modeling Project in XS Advanced

You have already learnt how to manage some important stages of a modeling project's lifecycle.

- Create a project and define some key settings.
- Import and export modeling content, including an entire project.
- Build selected objects or an entire HDB module.

You also got an overview of how to collaborate on projects with repositories such as GitHub.

Let's now discuss the deployment of an application.

Application Deployment

We will only consider the deployment of a Multi-Target Application made up of one or several HDB modules. That is, we will not discuss the other tiers such as the user interface or application logic.

Deploy Versus Build

Let's first make a clear difference between building and deploying an application.

When you work on a project, you build the modeling content on a regular basis in order to test it, check dependencies, and so on. This is something you trigger from the SAP Web IDE for SAP HANA. To do that, you need to have a modeler role in the space to which you want to assign your XSA project.

When you build an HDB module (or some of its design-time content), if the build is successful, the corresponding runtime objects are deployed to the corresponding space.



Note:

Remember that triggering a build at the project level does not build the HDB module(s) that the project contains.

On the other hand, deploying an application is something you do in a target space at a specific point in time, for example to test the application in a QA environment or set the application to production. Deploying an application can be done outside of the SAP Web IDE and the XS Advanced user who deploys the application does not need to have a developer role in the target space.

Source Files for Deployment

The key source file for application deployment is generated by a build operation at the project level in the SAP Web IDE. This creates a .mtar (MTA archive) file, which is technically a .rar compressed file. This file is named <Application ID>_<version>.mtar and can be found in your workspace, in the folder mta_archives . Among other, this archive contains the file mtad.yaml , which is an equivalent of the mta.yaml file but specific to application deployment.

Another important (though optional) file used for deployment is the .mtaext file. This is an “extension” file in which the space administrator who deploys the application can specify additions or modifications to the mtad.yaml file in order to pass the relevant parameters for the target environment during deployment. Typical examples of what you can specify in the extension file include the following:

- The name you want to give to the HDI container schema
- The actual name of a user-provided service in the target environment
- The actual name of another HDI container service in the target environment

The name of the HDI container(s) created during the deployment of your application is also an important parameter. This is the name of the HDI container resource declared in the mta.yaml

Unit 7: Management and Administration of Models

file (by default, when you create the HDB module, `hdi-container`). You probably want to make it specific to your application, in order to identify it more easily among the XS services.

For example, the SHINE application installed in your training environment defines its two container services as follows:

- `shine-container`
for the HDB module `core-db`
- `shine-user-containe r`
for the HDB module `user-db`

Deleting a Project

When you delete a project from the workspace in the SAP Web IDE for SAP HANA, the corresponding HDI container is not removed. This means there is still a service running in the XS Advanced runtime, and the corresponding schema still exists in the database.

To clean up the XSA runtime and the database, you must execute an `xs delete` command in the XS Advanced Command-Line Interface (XSA CLI).



LESSON SUMMARY

You should now be able to:

- Define the key settings of a project
- Set up access to external data
- Manage the lifecycle of a project

Unit 7

Lesson 3

Working with GIT within the SAP Web IDE

LESSON OVERVIEW



LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Use the Native Git Integration of the SAP Web IDE

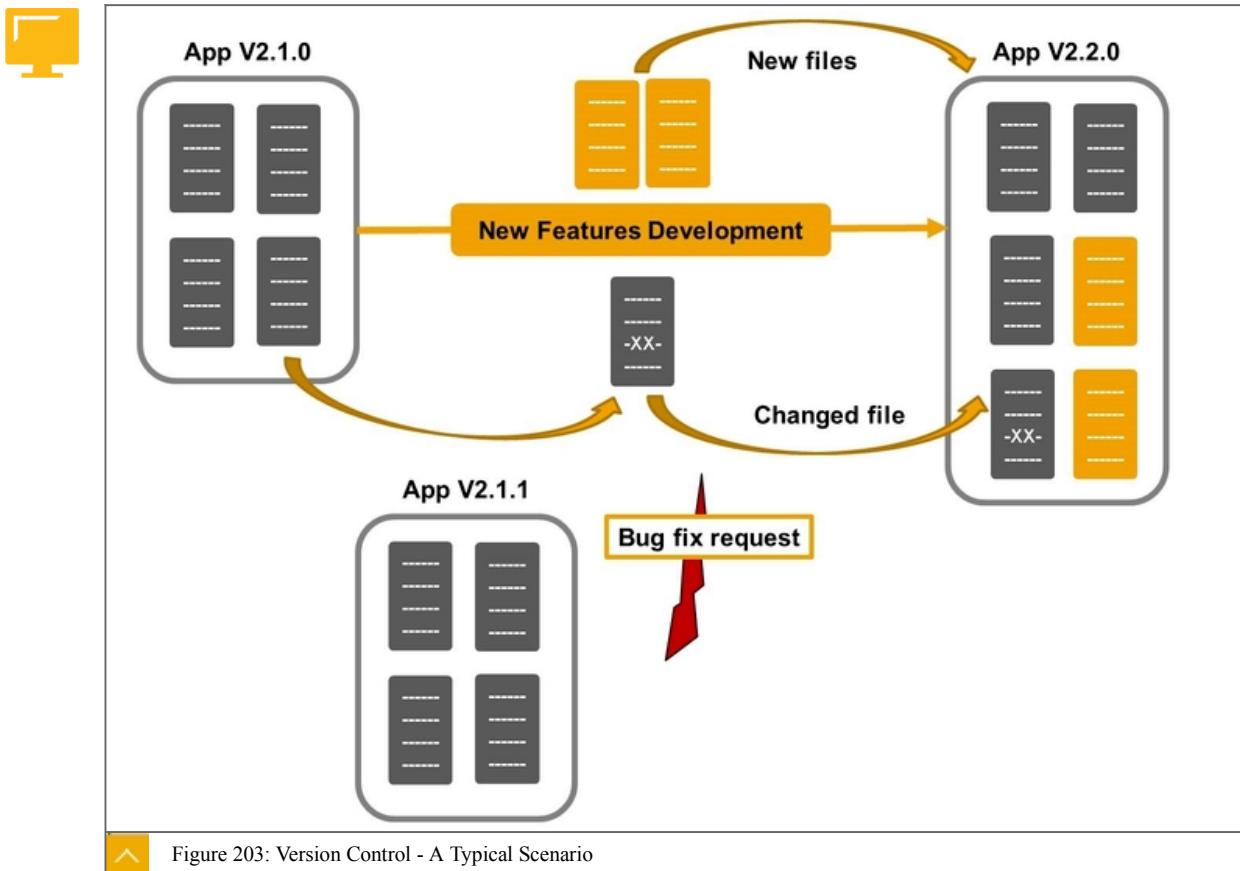
Overview of Git

Why Should You Use a Version Control System?

Think about an XSA application. This application, with its different modules (SAP HANA Database Modules, UI, and control flow modules), consists of a number of files, organized in different folders.

Now, suppose the last released version of the application is V2.1.0, and you're currently preparing new features for the upcoming minor release V2.2.0, which is scheduled in a few weeks. You might have imported the MTA project in your SAP Web IDE workspace and started developing the new features that are planned for V2.2.0. This development means essentially new files, as well as modifications to existing files. Maybe the deletion of a few ones as well.

Unit 7: Management and Administration of Models



At some point, you receive an important e-mail from support saying that a bug in V2.1.0 needs an urgent fix. This cannot wait until the next minor version, and requires a patch (V2.1.1).

So, how do you handle this request? At the moment, your current development is not finished, not tested, but you need to patch your version 2.1.0. Of course, you want to start from the last release (you do not want to include any part of the future functionality into the patch), but your patch might affect some files that you already modified as part of the development of new features.

This is where a version control system comes into play. It allows you to keep a complete change history by using kind of milestones during the development of your code, at a very fine-grained level if needed. And you can also branch your code, which means, you can develop and test different features in different parallel development threads (branches). If a feature branch is good to go, you can merge this branch with the main branch. If it is not, you can continue your development, or even get rid of this branch if you realize that a development option for a feature was not relevant and you need to think about it again.

Key Benefits of a Version Control System



- Source code backup
- A complete change history
- Branching and merging capabilities
- Traceability (for example, connecting a change to project management software)

To learn more about version control systems, you can have a look at this page: <https://www.atlassian.com/git/tutorials/what-is-version-control>.

<https://>

Git in a Nutshell

Git is a Distributed Version Control System (D-VCS) used to manage source code in software development, or more generally to manage the lifecycle of any set of files.

It allows one or several developers to work locally with their own copy of the Git repository, which contrasts with a traditional client/server architecture.



Note:

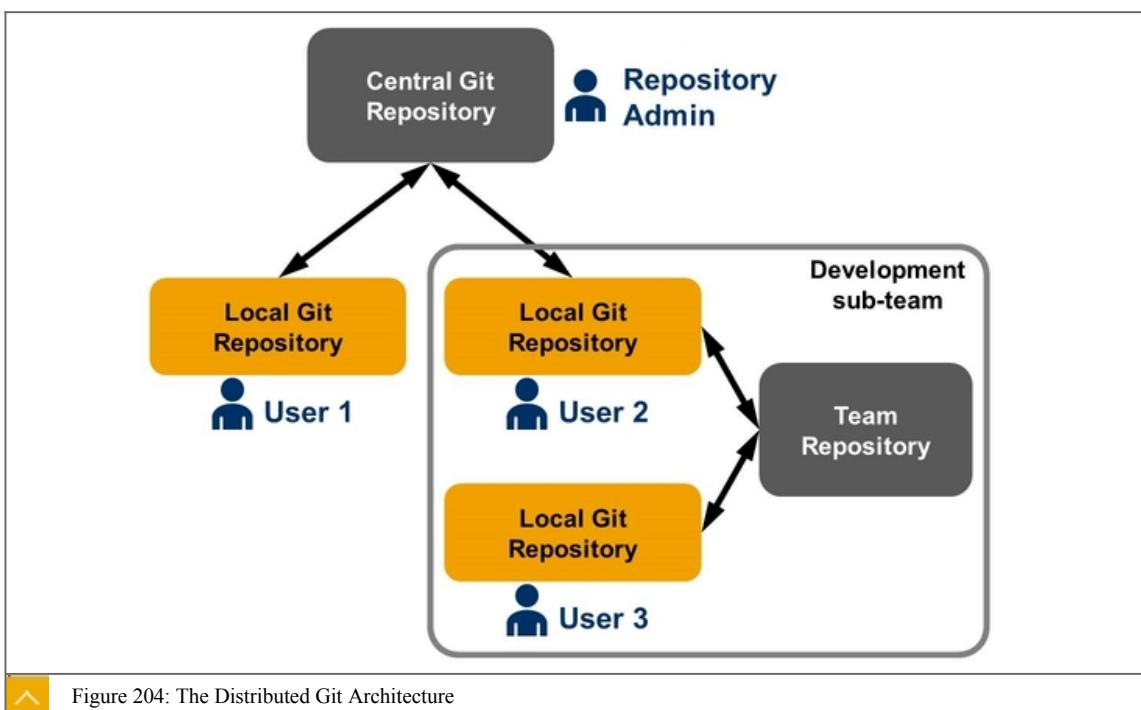
Git can also be used even out of a collaboration context, to help you control the development of a project on which you work alone, thanks to a number of capabilities.

The architecture of Git is distributed. It is somewhere between purely local and purely central. A local architecture would make it difficult for several developers to collaborate. A centralized one allows collaboration, but in some cases, a developer needs to block a piece of code on the central repository while working on it.

Instead of this, Git is designed so that every developer can keep the entire history of a project (or only a part of it, depending on their needs), locally on their computer.

Git is a free software distributed under GNU GPL (General Public License).

The Git Architecture



In a classical Git architecture, developers work in a local repository on their computer, and connect on a regular basis to a central repository to share their contribution and get the contribution from other developers on the project.

Git also supports easily several remote repositories for a single project. If needed, it is possible for a sub-team of developers to have their own sub-team repository to collaborate, and synchronize their changes with the central repository when needed.



Note:

In Git, it is even possible for a developer to define the local repository of another developer as a remote repository and to synchronize his development. This requires a network access and the relevant authorizations.

The shared Git repositories can be hosted on the own company's IT infrastructure, or on Git hosting services such as GitHub –one of the most popular–, Helix (Perforce), Bitbucket (Atlassian), and many more. A lot of companies offering Git hosting services also provide additional services such as code review or issue tracking.

The Key Benefits of Git

Git has been designed with Security, Flexibility and Performance in mind. Almost every operation is performed locally. The branching model provides an extreme flexibility.

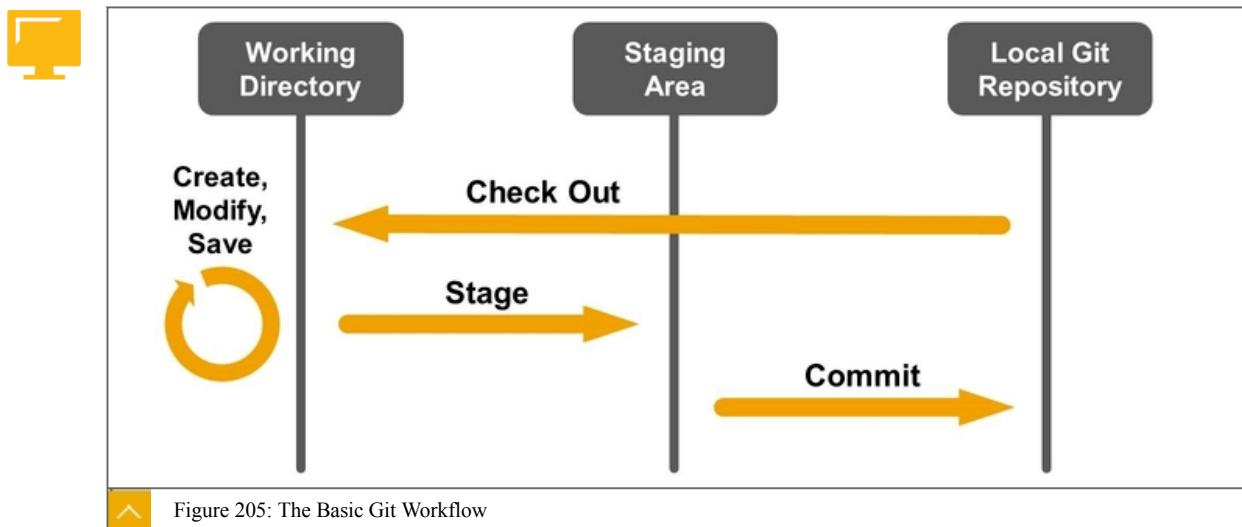
To learn more about Git, goto: <https://git-scm.com/>.

The Lifecycle of Files in Git

When you work with files locally in Git (the case of remote Git repositories will be discussed later on), this involves three major “logical” areas.

- The Working Directory
- The Staging Area, also known as INDEX.
- The (local) Git Repository

These are not all real areas, in the sense that a given file is not necessarily materialized in each of them. Let's explain this with a diagram.



The way to manage files in a local Git repository is very straightforward, relying on a small number of actions.

When you create a file, this file is initially stored in your **Working directory**. You can also modify (or even delete, if needed) a file that you have previously exposed in your **Working Directory** with a **Check Out** command.

At this stage, your changes –even if they are saved in your working directory– are not yet part of the Git history. To update the Git history, you must perform what is called a **Commit**. This is

what will create a new point in Git history (a so-called Commit), referencing a number of changes since the previous commit.

But what changes exactly do you want to include in your next commit? This is where the **Staging Area** comes into play. By staging a file, you mark this new or modified file so that it is included in the next Commit. You can of course stage several files (this is very common), or even stage all the current changes of your working directory.

Let's put it another way: The staging area is the “virtual” place where you put all the modifications that you want to include in your next commit.

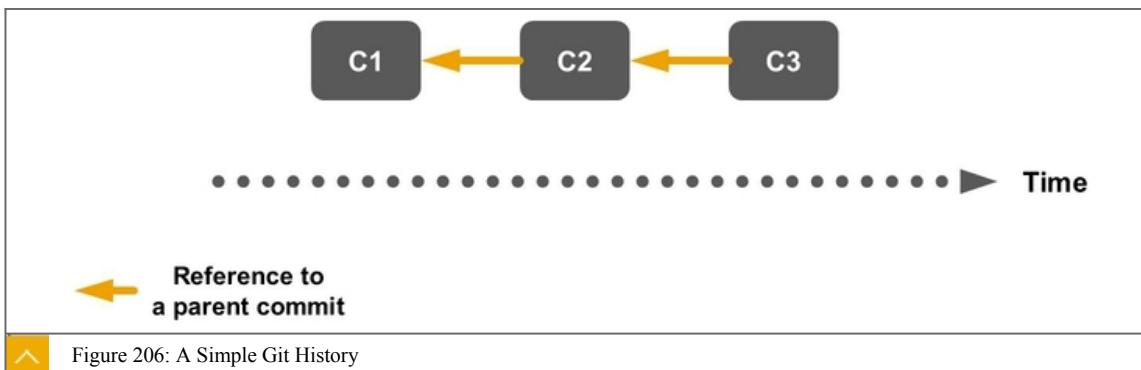


Note:

So, when your working directory contains changes that you do NOT want to include in your next commit, you just need to make sure you do NOT stage the corresponding files before committing.

The Git History

Git is very good at representing the history of a project in a simple way, as the succession of commits.



Over time, all the commits you execute in your project are added to the history, and each commit (except the initial one) references its parent commit.



Note:

Actually, you will see later on that in the case of a Merge operation, a commit can have two parent commits.

With each commit, Git keeps record of the commit date, the identity of the developer who executed it, and useful information about which files were affected (added, modified, or deleted).

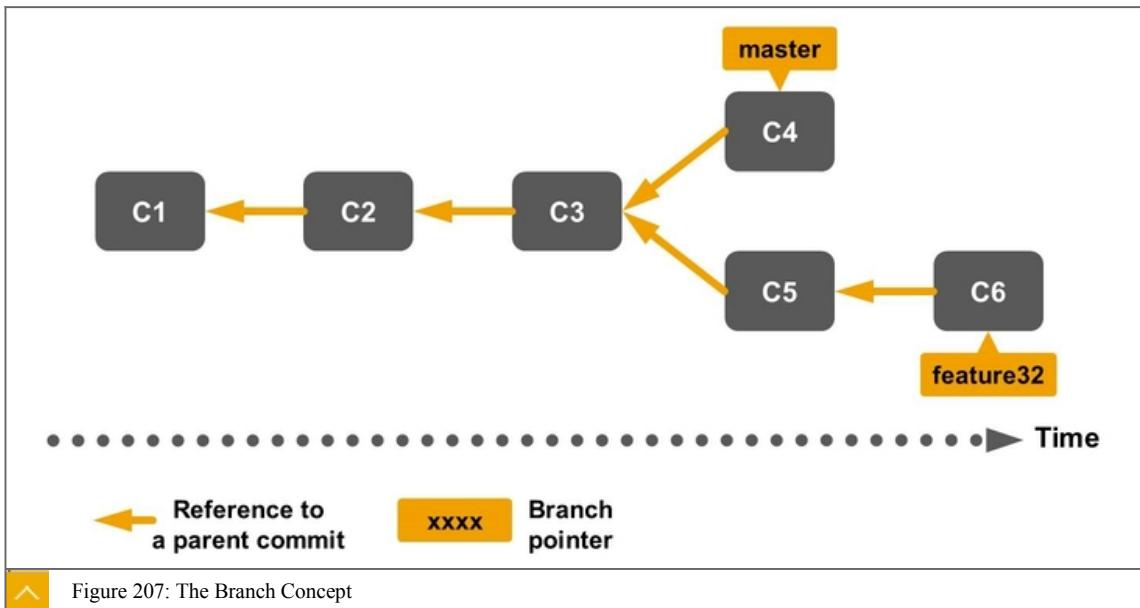
The Branch Concept in Git

Branching is one of the core concepts of Git, which provides a huge flexibility at a very low cost. So what is a branch?

From a conceptual standpoint, a branch is a series of commits. Technically, a branch is just a pointer that references a particular commit of the project history.

Each Git repository always has at least one branch. The default branch is generally called **master**.

Unit 7: Management and Administration of Models



For many different purposes, you can create a new branch and commit your changes to one of the existing branch.

Let's describe the diagram, The Branch Concept. After commit C3, a new branch Feature32 has been created to support the development of a new feature. Two commits, C5 and C6, have been made to this branch. In the meantime, additional changes have been committed in the master branch (commit C4).



"It is easy to shoot your foot off with Git, but also **easy to revert to a previous foot and merge it with your current leg."**

Jack William Bell

Figure 208: Git Flexibility - A Nice Metaphor

Git Integration within the SAP Web IDE for SAP HANA

Git Clients

Since its creation in 2005, Git provides its complete set of features through the command line. Over time, a number of Graphical User Interfaces have been developed. They generally include a subset of the Git functionality available through the command line.

The SAP Web IDE for SAP HANA provides an embedded Graphical User Interface for Git.

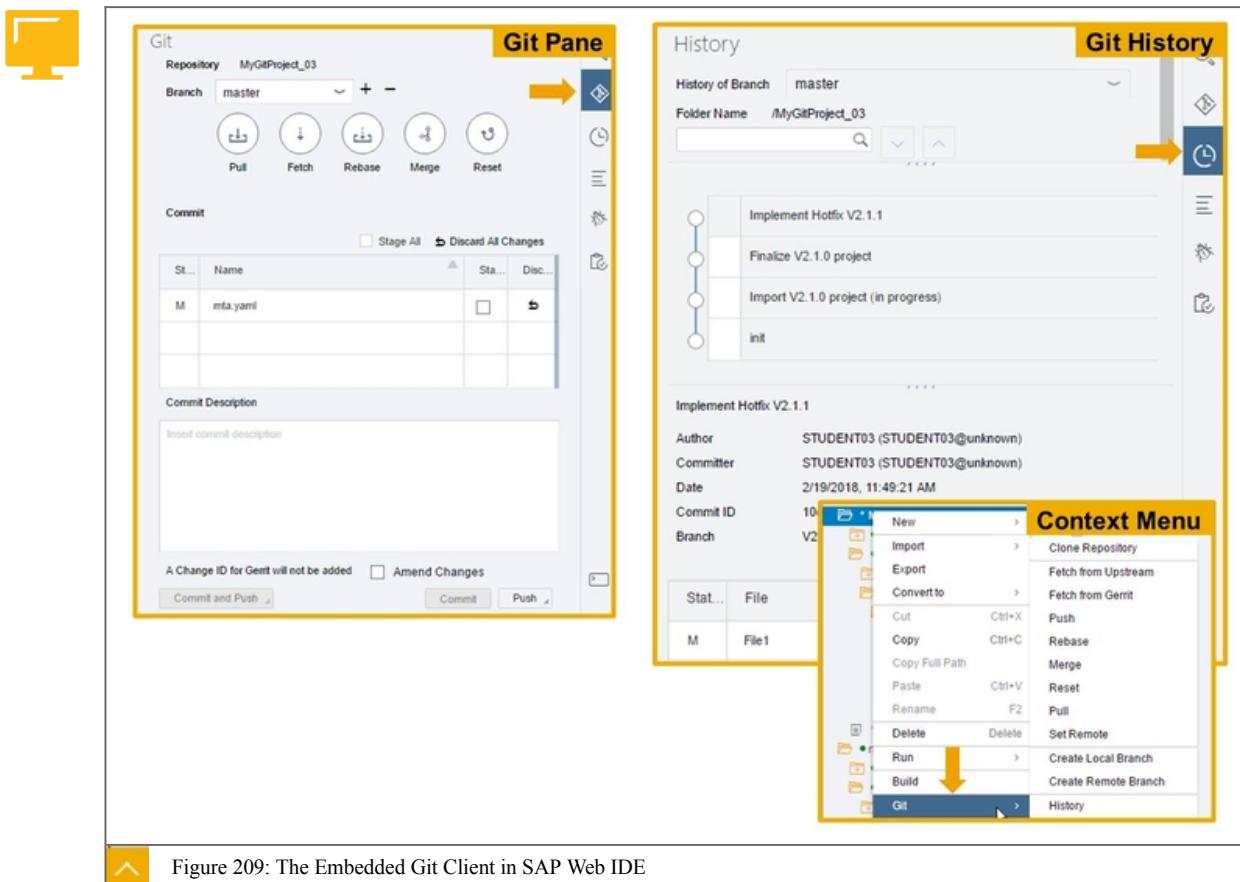


Figure 209: The Embedded Git Client in SAP Web IDE

This client includes two main Git panes, accessible from the right toolbar of the perspective, or the View menu, as well as a context menu and a set of icons displayed next to the files and folders in the workspace.

In addition, in the Project Settings, a dedicated section displays the entries of the Git configuration file. These entries include, among others, the Git user name and e-mail, remote repositories (if any), and so on.



Note:

It is possible to edit the Git configuration file manually, but this requires advanced Git knowledge. In many cases, this is done automatically, based on the actions you perform in the GUI.

The SAP Web IDE includes the core features of Git (commit, history, management of local and remote branches, and so on). The support of additional Git features is included on a regular basis in some new releases of the SAP Web IDE. For example, the Git Stash feature (allowing to set aside uncommitted changes temporarily) will be included in the SAP Web IDE for SAP HANA 2.0 SPS03.

Starting to Use Git in the SAP Web IDE

Working with Git in the SAP Web IDE is something you decide project by project. A project might have Git functionality enabled, when another project does not.

If you decide to activate the Git functionality in the SAP Web IDE, there are different approaches. You will choose the one that is best suited to your needs... keeping in mind that in some use cases, they get to the same result.

Unit 7: Management and Administration of Models

- Clone a Remote Git Repository

This is an easy way to copy an existing project stored on a Git server (or Git hosting service), and if needed, start contributing to the project.

In some cases, you will not contribute to the project, but you just want to get a copy of a project that a development team made available to you, either because you have read access to the remote repository, or because it is available publicly. This is the case of the SHINE for XSA project, which anybody can clone without any credentials.

- Initialize a Local Repository

By default, a project that you create or import in the SAP Web IDE does not have the Git functionality activated. This is the case, for example, for the HA300_## project you've worked on so far.

However, activating Git for a project is extremely quick and simple: via the context menu of the project, you just tell the SAP Web IDE to define the folder where your project is stored as a Git repository. Basically, this adds to this folder a few files that Git will then use to track the history, store snapshots of changes to the repository, and manage settings.

Initializing a local repository is what you need when you want to work with Git, but alone (locally), on a project. That is, without sharing it or having other developers contribute to it (at least at some point). Indeed, you do not need a remote Git repository at all, in this case.



Note:

When you initialize a local repository (which results in an empty initial commit), all your project files are flagged as new and are staged. You are ready to add all your project files to the local Git repository by executing a commit.

- Set Remote Repository

This is what you need to associate your local Git repository (project) with a remote Git repository. For example, when you have already developed content locally and would like to share this content with other developers.



Caution:

This is NOT needed when you clone a remote repository, because in that case the local repository (the clone) is automatically associated to the remote repository you cloned from.

Whenever you work with a remote Git repository, you need the Git Clone URL. In addition, you might need credentials to access the repository if it is not public.

Working with Files in a Git Project

Basically, modifying your project content (with or without Git) means creating, modifying and deleting files. With Git, all these changes are tracked in your working directory as soon as they are done –for example, when you save a file you have just modified– and listed in the File Status area.

Then, for each file, you have the following possibilities:

- Stage the modification so that it will be included in the next commit

- Leave the modification unstaged (it will not be included in the next commit)
- Discard the change

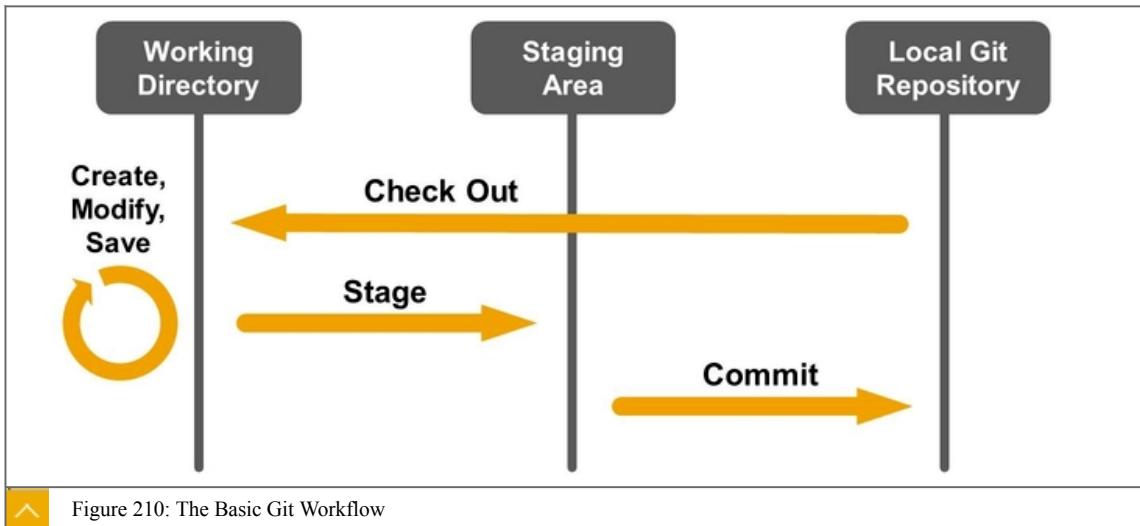
Staging or Discarding can also be done for the entire set of modifications.

The next step is to commit your changes, which will add a next commit to the history of the branch.



Note:

The concept of branches in Git, already introduced, will be discussed in more details later on. For now, let's just consider that you are working on a single local branch, for example, `master`.



To materialize this workflow, each file is assigned a Git status. This status is represented by an icon in the SAP Web IDE workspace and/or the `File Status` area of the `Git Pane`. The list of possible file statuses is as follows:

Unit 7: Management and Administration of Models



Workspace Icon *	Meaning	File Status in the Status Table
	New, not staged	Untracked (U)
	Modified, not staged	Modified (M)
	New/modified and staged	New (N)
		Modified (M)
	Deleted (staged or unstaged)	Deleted (D)
	Committed	[File is no longer listed]
	Conflicting	Conflict (C) – Only during a merge or rebase operation

* Only one icon is displayed for folders that contain files with different statuses
** The icon identifies folders from which files have been deleted

Figure 211: Git Status of Files in the SAP Web IDE



Note:

The Conflict (C) status will be discussed later on.

Committing Changes

When you commit changes, you add these changes to the Git history and provide a commit description. From this moment on, the Git History pane will include this commit, and provide the identity of who committed, the date, the commit description, and details on which files were affected by the commit. Note that committing changes does not modify your working directory. The committed files are still available for further modification, and the new or modified files you haven't committed yet are still here for an upcoming commit. You can also discard the changes to these uncommitted files.



Figure 212: Committing Changes

The commit description is an important info to allow the readability of your changes, in particular when you collaborate with other developers. You can find a number of blogs and tutorials on how to write a good commit message. Here are a few recommendations:

- Start with a relatively short subject (50 characters max), using imperative mood
- Separate the subject and body with a blank line
- Provide info about what the change does, and why (for example, what issue it solves)
- If relevant, provide the url of a related issue or specification in another system (for example, JIRA)

It is possible to **amend** a previous commit. This is a way to replace the very last commit by a new one, after you staged additional files. The original commit description is displayed so that you can modify it before committing again.



Caution:

You must not amend commits that have been shared with other developers, because this would modify a (shared) history on which others might have already based their work.

Working with Git Branches in the SAP Web IDE

Local or Remote Branch

Git allows you to create both local and remote branches. So how to choose?

One key principle is that only remote branches are visible to others: your local branches are for you and you are the only one who works in these branches.

Another important principle in Git is that you never commit changes directly in a remote branch. Instead, you always commit changes in one of your local branches first. Then, if needed, you can transport the commits to a remote branch.

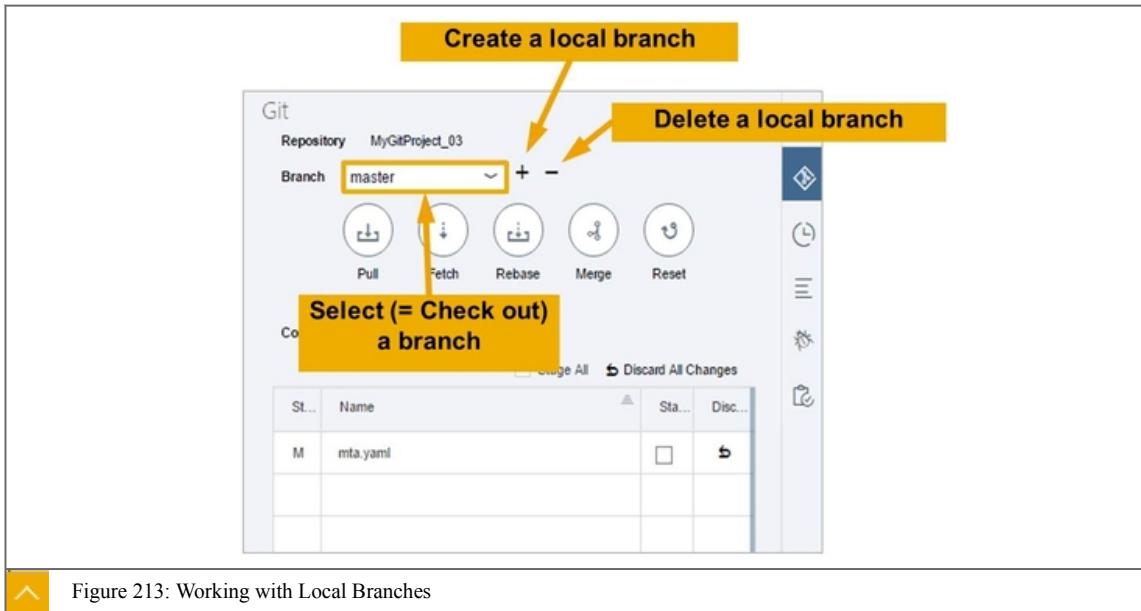
Unit 7: Management and Administration of Models

Let's take an example. You are working on a project with other developers. At some point, you need to test something on your own, without sharing it with other developers. Then you will create a local branch for that. If, later on, you need to share this with others, you have the possibility to create a remote branch based on your local branch.

We will come back to remote branches later on. But for now, let's discuss how you work with local branches.

Working with Local Branches

You can create a new local branch from the **Git Pane** or the Git context menu. The only thing you need is to define a source branch (which could be remote or local) and give the new branch a name. Just after a new branch is created, it is absolutely identical to the source branch.



When you have several branches in your project, you must carefully decide to which branch you want to commit your changes. To do this, you select a branch in the **Git Pane**. In Git terms, this is known as **Check Out**.

Checking out a branch also updates your working directory to reflect the exact current state of files in this branch. In other words, your working directory is likely to change when you switch from a branch to another, except if these branches are identical (that is, if they point to the same commit).



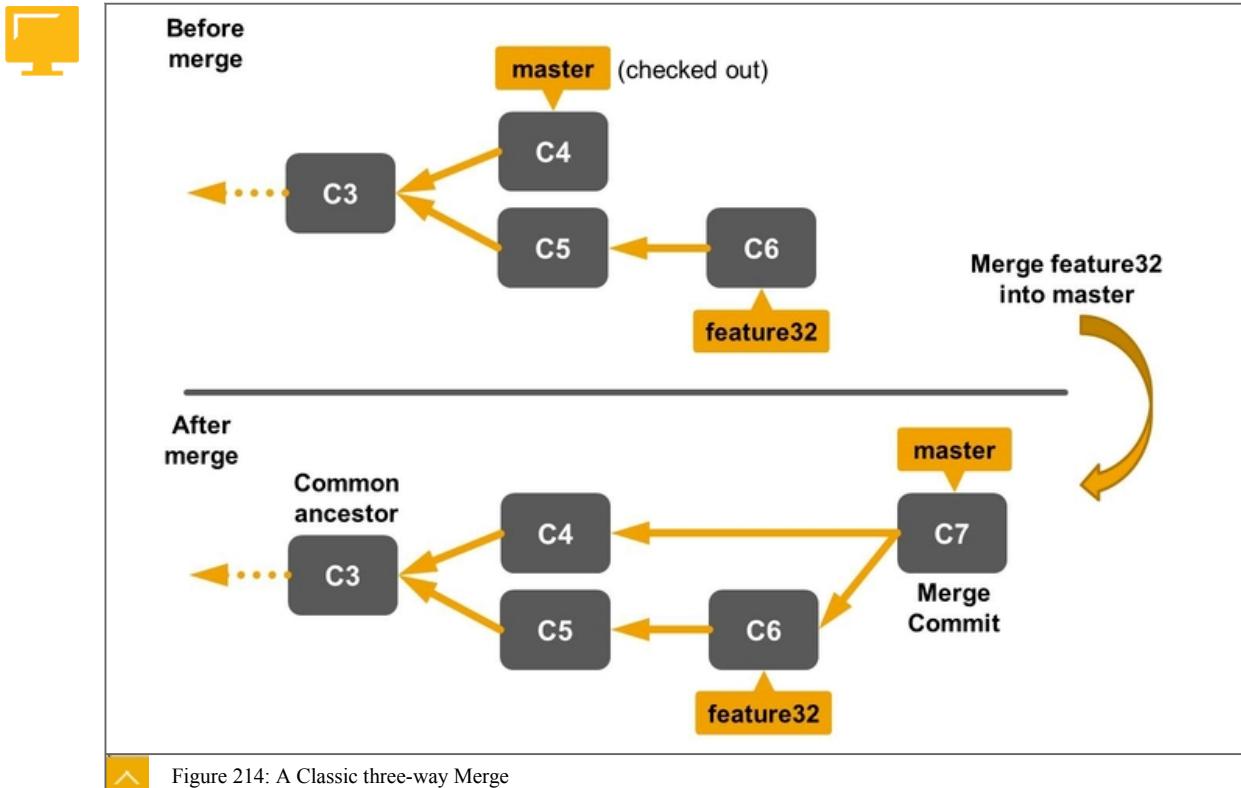
Caution:

Issues might occur when you have changes that are not committed yet and you try to check out another branch. For example, you have made a modification to a file that exists in both branches. To avoid this, the best way is to have a clean working directory without uncommitted modifications before you check out a branch.

Combining Changes from Two Branches

When changes affect several local branches, you might want to combine these changes. For example, to include a hotfix or a new feature in your master.

Let's assume that commits have been executed in two different branches, master and feature32 .



A Git merge operation affects the branch that is currently checked out. Git creates a new commit (merge commit) that will combine the changes of another branch with the changes of the checked out branch since their history diverged. As you see in the figure, A Classic three-way Merge, the merge commit has two parents, which are the last commits of the two branches that you merged together.

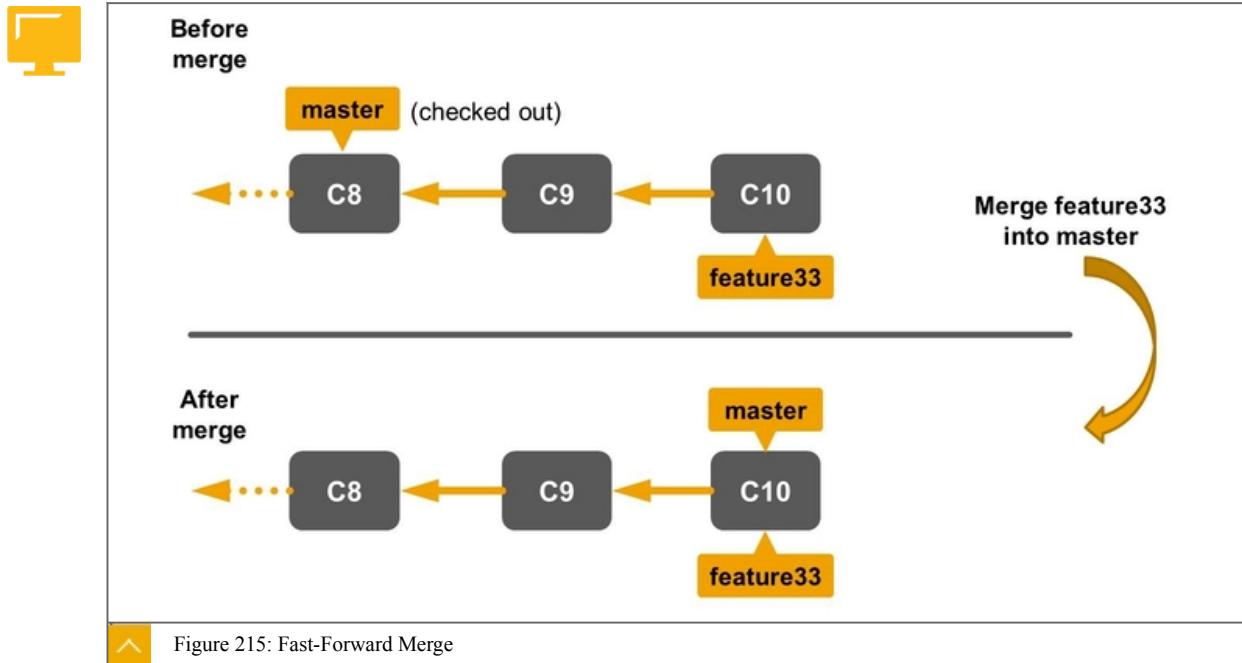


Note:

After the merge, the two branches are NOT identical. In particular, feature32 does not include C4.

A special case for Git merge is when the branch you want to merge into (the checked out branch) points to a commit that exists in the branch you want to merge. Meaning, no additional commit has been made to that branch.

Unit 7: Management and Administration of Models



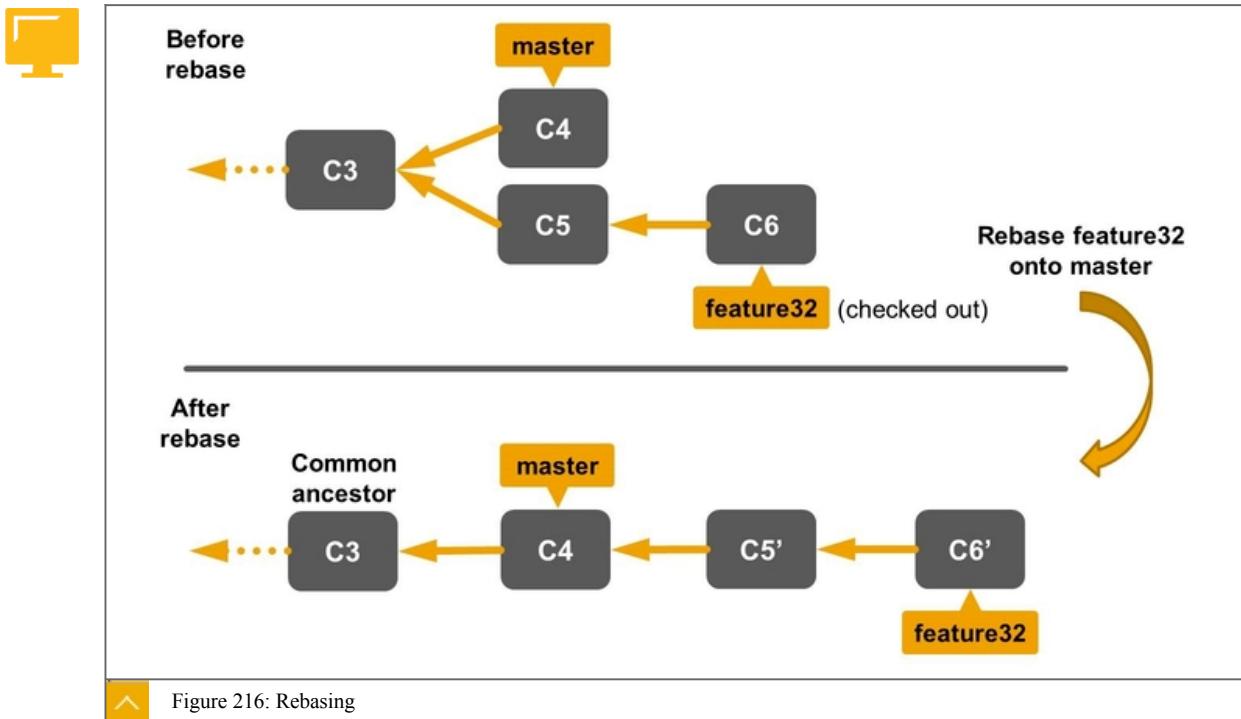
In this case, there is no need for an additional merge commit. Indeed, the pointer of the checked out branch is just pushed forward to the last commit of the other branch. Git calls this a Fast-Forward Merge.



Note:

A fast-forward merge only has effect if you check out the branch that is behind in the commit history. In the scenario of the figure, Fast-Forward Merge, if you check out the branch `feature33` and execute a “merge `master` into `feature33`”, no change will be made at all.

Now, let's introduce an alternative to merging when you want to combine changes from two branches. It is called “Rebase”.



When you rebase a branch B2 (here, `feature32`) onto branch B1 (here, `master`), you basically take the history of commits in B2 and replay these commits on top of B1. Then, you can execute a fast-forward merge on B1 (`master`) so that it points to the same commit as B2 (`feature32`).

As you see, compared with the classic three-way merge we discussed earlier, this keeps the history of the `master` branch linear. The final content of the `master` branch is identical, but the history looks different.

Rebasing is sometimes useful when you work on a purely local project, and also when you want to prepare a clean commit history that will flow naturally on top of a branch to which you contribute.

Caution:

You should always keep in mind the golden rule to use rebase: **You can only rebase changes that you have NOT shared with anyone**. Indeed, a commit that is communicated externally (for example, by pushing changes to a remote Git server) might be the starting point of some work from other developers. If you rebase your changes and push them again to this Git server, this will alter the history and might cause a lot of issues.

Managing Merge / Rebase Conflicts

A conflict occurs during a merge or rebase when one or several files are affected by concurrent modifications from both branches.

In this case, you must analyze the files and determine how to solve the conflict. Which means, decide how to produce a code that suits the requirements that the changes in each branch addressed.

Each conflicting file is identified with a dedicated red icon `> <` in the workspace (project tree). To solve a conflict, you must open the file, in which the content from the two branches are

Unit 7: Management and Administration of Models

presented together, and contains a special markup to identify which changes come from which branch.

Example of a Merge Conflict in a Table Function

```
select
"FIRST_NAME",
"LAST_NAME",
"SEX",
"LANGUAGE"
from "HA300::SNWD_EMPLOYEES"
<<<<< HEAD
where contains("LAST_NAME", :lastNameFilter, FUZZY(0.55));
=====
where contains("LAST_NAME", :lastNameFilter, FUZZY(0.6));
>>>>> hotfix
```

HEAD is the pointer to the checked out branch (the one you're merging INTO) and, here, hotfix is the name of the branch you're merging.

Example of a Rebase Conflict in a Table Function

```
select
"FIRST_NAME",
"LAST_NAME",
"SEX",
"LANGUAGE"
from "HA300::SNWD_EMPLOYEES"
<<<<< Upstream, based on master
where contains("LAST_NAME", :lastNameFilter, FUZZY(0.55));
=====
where contains("LAST_NAME", :lastNameFilter, FUZZY(0.6));
>>>>> 93f6499 Change fuzziness threshold for employee search function
```

In a rebase, the branch on top of which you want to rebase is identified with Upstream, based on <branch name> and the specific commit from the branch that you are rebasing is listed, with its description.

In both cases, you must adapt the code, remove the specific markup, and sometimes include comment lines in your code to explain how you solved the conflict, and then save the file.

When this has been done for each conflicting file, you can then proceed as follows:

- For a Merge: stage the modified file(s), enter a commit description and commit your changes.

This description will replace the default merge commit description (the one that you would get if no conflict occurred).

- For a Rebase: choose Continue .

In the rebased branch history, the conflict resolution is stored in the “replayed” commit. As if there had not been a concurrent change, or to put it another way, as if you had made your change in the hotfix branch based on the modified version from the master branch.

Working with Remote Branches

Working with remote branches is not fundamentally different from what we have discussed for local branches. First of all, because the changes you make to your files are always committed to a local branch. And also because combining the changes from a local branch and a remote one works in a similar way as combining the changes from two local branches. Still, there are a few differences:

- You need to connect to a remote Git server (or a Git hosting service)

- At some point, you will share your changes (commits) with others, and also receive changes (commits) from others.

Connecting to a Remote Git Repository

As discussed earlier, in the section, Starting to Use Git in the SAP Web IDE, there are two ways to connect to a remote Git repository:

- Clone from a remote Git repository
- Set a remote (link your local Git repository to a remote one)

An essential piece of information you need for that is the **remote repository URL**.

In addition, cloning, setting a remote, and then, most of the regular interactions with a remote repository, require **credentials**.

However, some repositories are public, which means that they can be accessed anonymously, without credentials. For example, this allows you to clone in your repository some code that the owners have decided to share publicly.

For security purposes, on the remote Git server side, there is a repository administrator who manages users and authorizations. For example, you might be allowed to see a repository but not to write to it (Guest role). Or some branches might be protected, and only a user with role “Branch Master” can write to these branches.

Interacting with a Remote Git Repository

- Clone from a remote Git repository

If you clone a remote repository, your local Git automatically knows about the branches it contains. By default, it creates the local branch `master` which –at first– corresponds to the remote `master` branch.

- Set a remote (link your local Git repository to a remote one)

In this case, the first thing you need to do is to tell Git to Fetch info from the remote repository. For now, let’s say that a Fetch allows the Git client to know about the branches of the remote repository.

In situations where you connect to a brand new remote repository, you should be aware that some Git hosting services allow the repository creator to include an initial commit (an empty one, or one with just a `readme` file) or not. And connecting to an empty repository will generally work better if there is an initial commit.

Now, let’s discuss what you might want to do.

Copying Branches

You can create a local or remote branch. In any case, you need to specify a source branch.

Table 20: Creating Branches from Other Branches

Task	Purpose / Example	Steps
Create a local branch from a remote branch	You want to contribute to a project, starting from an existing remote branch you don’t have.	Git Pane :Create local branch . Choose the remote branch (source) and define the name of the local branch
Create a remote branch from a local branch	You have created code that you want to share with others developers but without including it in an existing remote branch.	Workspace: Git Context Menu: Create Remote Branch . Choose the local branch (source) and define the name of the new remote branch.

Unit 7: Management and Administration of Models

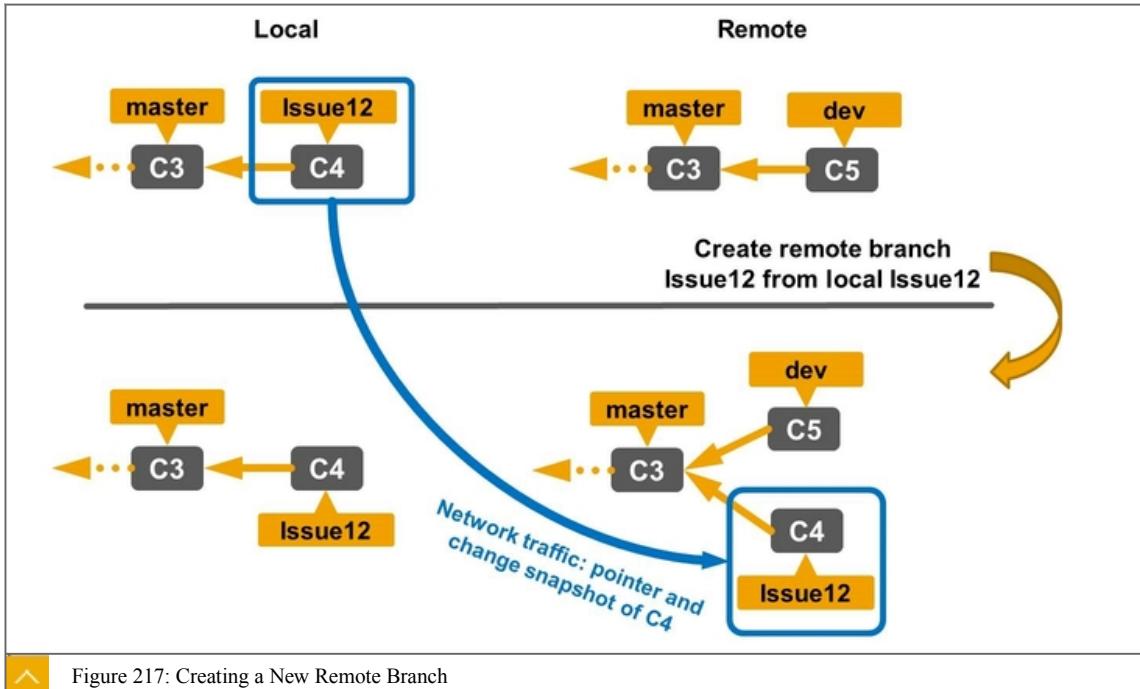
Task	Purpose / Example	Steps
Create a remote branch from a remote branch	You want to keep in a branch V315 the current status of the dev branch, which has just been approved for release	Workspace: Git Context Menu: Create Remote Branch . Choose the source remote branch (dev) and define the name of the new remote branch (V315).

Creating a new branch is a bit like “copying a branch”, but technically, you do not duplicate the entire set of code that the source branch contains. You only create a new pointer, and if needed, Git only transports the changes that correspond to commits that the repository of the new branch does not know about.



Caution:

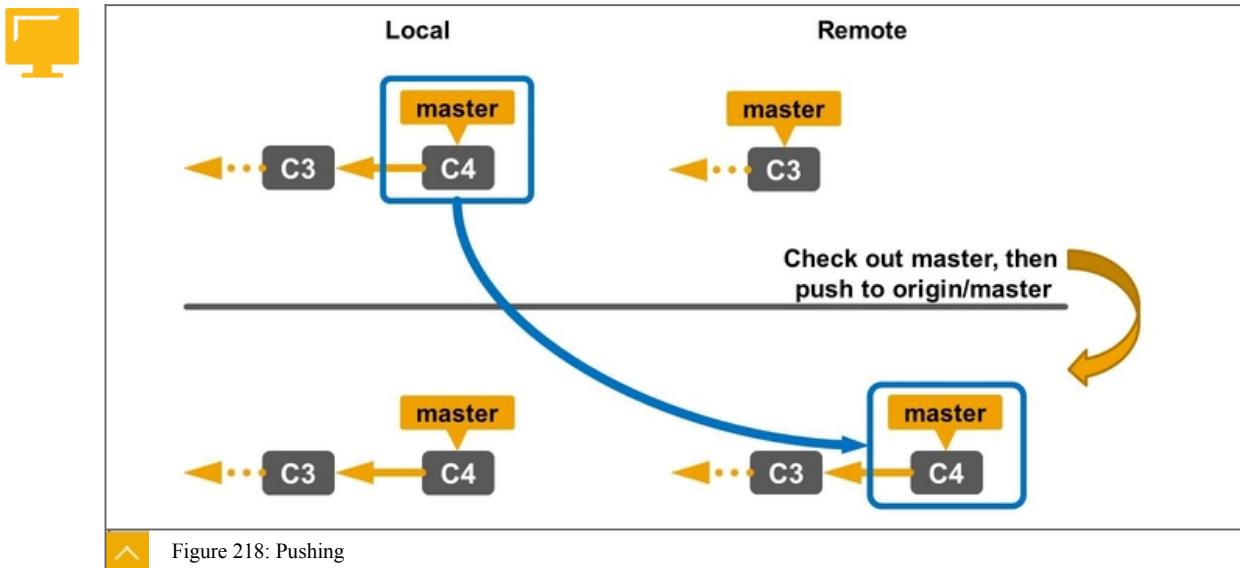
When creating a local branch from a remote one, you must first execute a Fetch to get the very last version of the remote branch.



In the figure, Creating a New Remote Branch, suppose you have been asked to develop a hotfix for issue #12. You need to develop the fix on top of the productive version (master). When the fix is ready, or even before, if you need to discuss it with a colleague, you can copy the corresponding branch Issue12 to the remote repository.

Pushing and Fetching

Now, let's discuss Fetch and Push. To keep it simple, let's say that these commands only work between a local repository and a remote one. These are the two essential commands to send local content to a remote repository (push) and get remote content into your local repository (fetch).



Pushing means that you send commits you have in a local branch to a remote one.



Note:

These might be commits of changes that you have made, or commits from another developer that you have just merged into your local branch after validating them.

To do that, you check out the branch you want to push changes from, and you define which branch you want to push to.

By default, Git allows you to push only if the last commit of the remote branch is already in your local branch (this allows the equivalent of a fast-forward merge of your change into the remote repository). If not, you will have to Fetch the remote branch history, and then rebase your changes onto the remote branch. Only then, you will be able to perform a successful Push.



Note:

The Commit and Push button in the SAP Web IDE triggers first a commit to your current (checked out) local branch, and then a push to a remote branch.

Unit 7: Management and Administration of Models

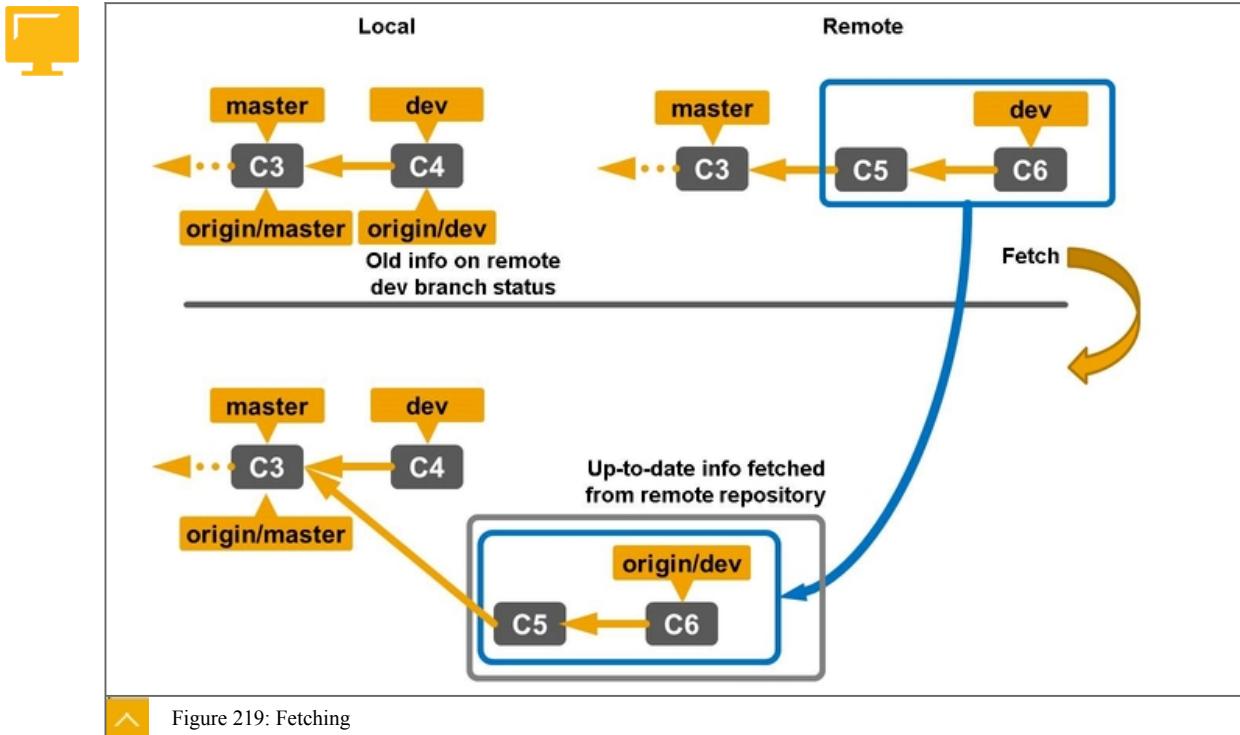


Figure 219: Fetching

With a Fetch, Git downloads from a remote repository the commits and branch pointers. This does not change your working directory or your local branches, but it creates (or adjusts) locally references and change snapshots from the remote branches. This allows you to inspect the commits that occurred on remote branches (date, description, who committed) and, later on, to merge these changes into a local branch, or rebase a local branch onto the remote branch.

Note:

Fetching is also required if you want to get the list of branches available on a remote repository. In particular, if someone else created a new branch on the remote repository, you cannot push commits to this branch before you know it exists.

In the figure, Fetching, we introduce two additional branch pointers: **origin/dev** and (for the sake of consistency) **origin/master**. These are the local pointers to remote branches. They are used to store locally the state of remote branches when you fetch. Then, any merge, rebase, or new branch creation involving a remote branch is executed locally based on these pointers (and the commits they reference). They materialize what Git calls **Remote-Tracking Branches**. That is, local branches that you cannot commit changes into, but are aimed at keeping locally, offline, a recent status of remote branches.

By default, for a given repository that your local repository knows about, Git will create (or update) during a fetch one (local) remote-tracking branch per remote branch. The default naming convention for a remote repository is **origin**, and if a branch in this remote is called for instance **dev**, the corresponding local-tracking branch will be called **origin/dev**. If you want to work on a local copy of this branch, Git will propose the default name **dev** but you can choose another one.

After you fetch, you can see how many commits you are behind of the current remote branch commit. This is displayed in a box next to the current branch name. For example, you might

have made 2 commits locally after cloning the remote repository (you do not need to fetch to know this), while someone has pushed 3 commits to the same remote (you will need to fetch to get this info). In this case, you would say you are “2 commits ahead and 3 commits behind”.

Pulling

Pulling consists of a Fetch, immediately followed by a Merge. It can be useful in some cases

Resetting a Branch

In the SAP Web IDE, it is possible to reset a local branch, which means, to revert this branch to the state of another local or remote branch. In case you reset based on a remote branch, always Fetch before resetting, to make sure you get the very last changes of the remote branch in your local repository.

Two reset modes are proposed:

- MIXED (HEAD and index updated)

This reset mode keeps your working directory as is and unstages the changes. Which means, you can stage and commit again all the changes (or part of them) that the reset rolled back.

- HARD (HEAD, index and working directory updated)

With this reset mode, the commits that the reset rolled back are discarded, even in the working directory.



LESSON SUMMARY

You should now be able to:

- Use the Native Git Integration of the SAP Web IDE

Unit 7

Lesson 4

Using SAP Enterprise Architecture Designer

LESSON OVERVIEW



LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Understand how SAP Enterprise Architecture Designer supports the design phase of your project

The planning and design phase of a project

In this course we are developing skills in SAP HANA modeling, focusing on the development of the virtual data model built with calculation views, that sit on top of an already created persistent data layer. We are using the various interfaces provided with SAP HANA to develop modeling artifacts. But modeling, either at the virtual or persistent layer, in SAP HANA fits into a much bigger picture where **planning** and **design** of the overall business and technical architecture design precedes the **development** and **running** phases.

At the start of a development project, it is usual to define goals. Goals allow you to model your organization's mission, vision, strategy, and objectives, to show how they are related to your organization's business and IT architecture, and how they will be addressed through enterprise architecture initiatives. Then comes the requirements gathering phase, where stakeholders contribute and agreement is reached over the design. Then we move to the design of the business processes and organization to support the processes. Finally we get into more detail with the design of the IT infrastructure, application architecture, and data architecture.



SAP Enterprise Architecture Designer

- Strategy architecture to document goals and projects
- Requirements management
- Organization architecture
- Business process architecture
- Business documents architecture
- IT landscape and application architecture
- Data modelling and data movement architecture
- Reverse engineering
- Impact analysis

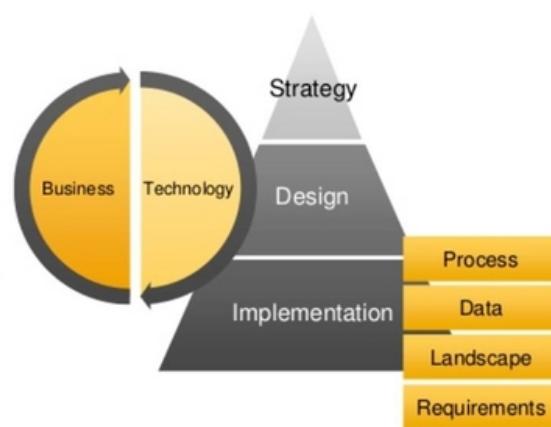


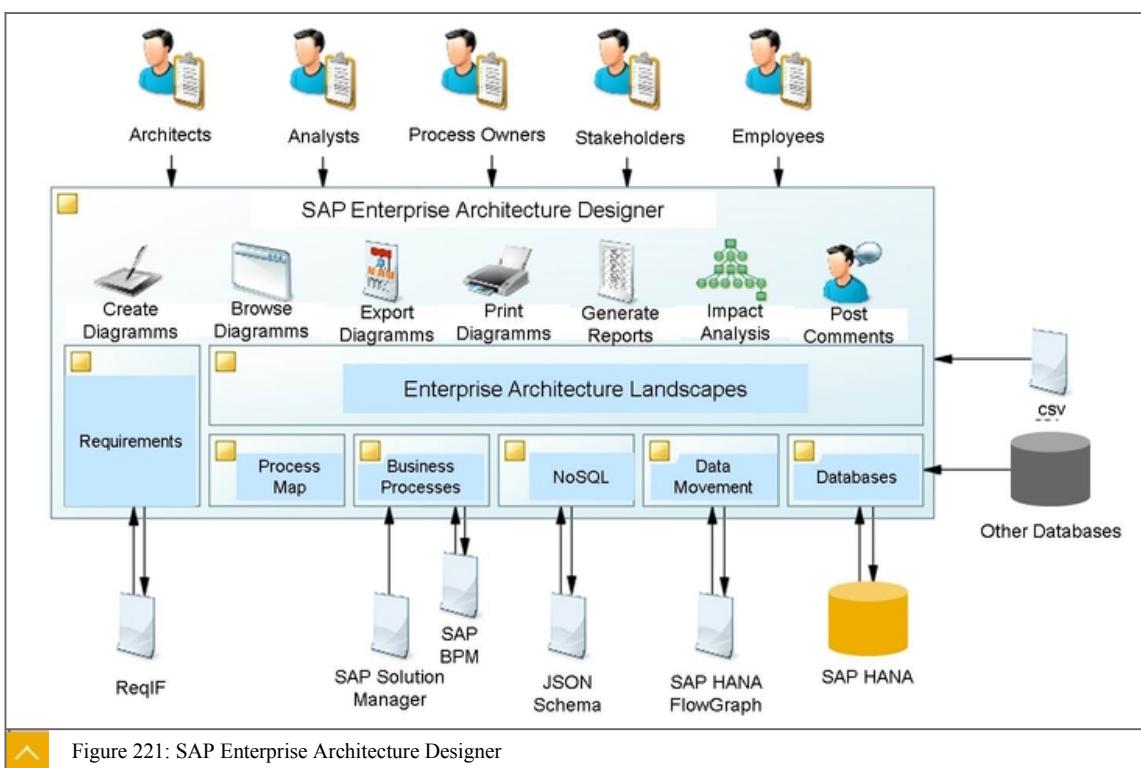
Figure 220: Key capabilities of SAP Enterprise Architecture Designer

SAP Enterprise Architecture Designer (SAP Enterprise Architecture Designer), lets you capture, analyze, and present your organization's landscapes, strategies, requirements, processes, data, and other artifacts in a shared environment. It is a web based application built in XSA on SAP HANA.



Note:

You may have heard of a similar SAP tool called SAP Power Designer. SAP Enterprise Architecture Designer follows the basic principles of SAP Power Designer. Although they have similar design and planning capabilities, SAP Power Designer is not built on SAP HANA and it uses a Windows interface. SAP Power Designer continues to be a relevant product for many customers who may not have an SAP HANA landscape and so SAP Enterprise Architecture Designer should not be regarded as a replacement for SAP Power Designer. SAP Enterprise Architecture Designer is ideal for organizations who have SAP HANA in their landscape because the tool can automate the generation of physical models in SAP HANA from the logical models built in SAP Enterprise Architecture Designer, and vice versa.



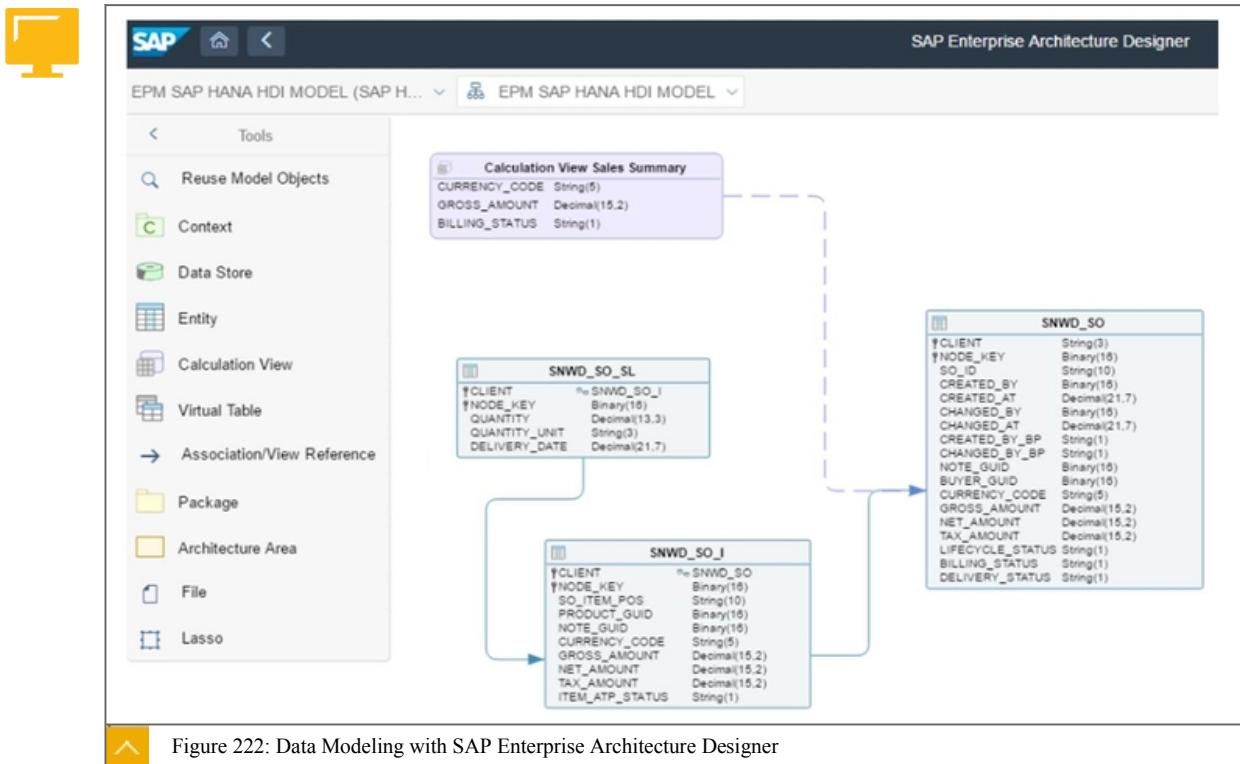
A key capability of SAP Enterprise Architecture Designer is the creation of **diagrams** to visualize various models. SAP Enterprise Architecture Designer supports the creation and editing of the following kinds of diagrams:

- **Requirements** — Requirements documents are organized in a hierarchical and they consist of written requirements. You can link requirements to users and groups who will work on them or are otherwise associated with them, and with any object that contributes to the fulfillment of the requirement or is impacted by it.

Unit 7: Management and Administration of Models

- **Process Maps** — A process map provides a graphical view of your business architecture, and helps you identify your business capabilities and high-level processes, independent of the people and business units who fulfill them. You then model the steps of the processes in business process diagrams (see below).
- **Business Process** — Business process diagrams help you identify, describe, and decompose business processes into workflows. SAP Enterprise Architecture Designer supports importing SAP Solution Manager 7.2 process diagrams. You can share and comment on Solution Manager business process diagrams, link their objects to objects in other models, and include them in impact analysis. SAP Enterprise Architecture Designer supports importing and also exporting from SAP BPM. Both methods use the industry standard BPMN (business process modeling notation) for data interchange.
- **Enterprise Architecture Landscapes** — There are many types of enterprise architecture diagram to help you analyze and document your organization, its capabilities, goals and processes, the applications and systems that support them, and the physical architecture on which they are implemented.
- **NoSQL** — NoSQL diagrams help you design, analyze, and document the JSON structures that are accepted and generated by your systems. JSON is a light-weight, human readable data interchange format (very much like XML) and can be used to describe any type of business document structure. You can reverse-engineer JSON schemas (or derive them from JSON documents) and generate JSON schema files. SAP Enterprise Architecture Designer supports the creation of NoSQL JSON Schema models manually, or by reverse-engineering of a schema or sample data file.
- **Data Movement** — Data movement diagrams help you design and analyze the transfer of data between data stores, and the transformations that they undergo on the way. SAP Enterprise Architecture Designer supports the creation of data movement diagrams manually or by reverse-engineering of SAP Web IDE FlowGraph files. You can model ETL flows and generate FlowGraph files for import to SAP Web IDE for building.
- **Databases** — Physical data models help you design and analyze the structure of your databases. You can reverse-engineer any supported database to create a physical data model in SAP Enterprise Architecture Designer. Generation to SAP HANA, directly to the catalog, or to SAP Web IDE via HDI files is also supported

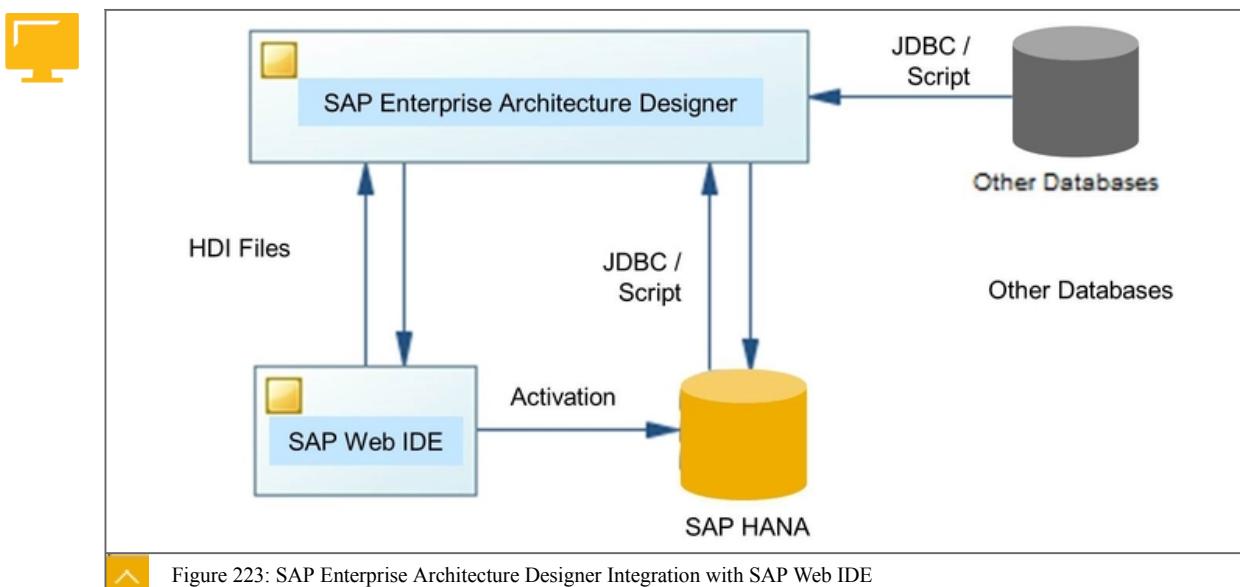
Lesson: Using SAP Enterprise Architecture Designer



A database diagram represents the entities and relationships of all database artifacts, such as tables and calculation views. For a new project you can develop the diagram by dragging and dropping the entities to the canvas and connecting them to create references. You can attach files and other diagrams to any entity and create comments.

If you are starting from an existing database that already exists, you can import the metadata to create a diagram automatically.

Once the diagram is built, you can send it for review before it is accepted.



One of the most powerful features of SAP Enterprise Architecture Designer is the ability to generate from a database diagram, either HDI source files (for HANA 2.0) or a database

Unit 7: Management and Administration of Models

schema (for HANA 1.0). You can also reverse engineer HDI files from SAP HANA or .sql files to create a database diagram in SAP Enterprise Architecture Designer.

It is important to always remember that SAP Enterprise Architecture Designer is not a development tool, but a planning and design tool. So although you can create the high level design of the data model, including tables and calculation views in SAP Enterprise Architecture Designer, you develop the detailed functionality of the calculation views e.g. hierarchies, variables, in SAP Web IDE.



LESSON SUMMARY

You should now be able to:

- Understand how SAP Enterprise Architecture Designer supports the design phase of your project

Unit 7

Lesson 5

Migrating Modeling Content

LESSON OVERVIEW

In this lesson, you will learn about deprecated information model types in SAP HANA Studio and how to convert them into their replacement object types within SAP HANA Studio.

You will also get an overview about the SAP HANA XS Advanced Migration Tool which is used, among others, to migrate SAP HANA Studio-based modeling content (classic repository) to the XS Advanced/HDI infrastructure. The migrated content can then be maintained with the SAP Web IDE for SAP HANA.

Business Example

You work on a project where SAP HANA Studio has been used to create modeling content, including deprecated objects.

As part of a migration project to SAP HANA 2.0 where the SAP Web IDE will be used for modeling, you want to understand the key steps to migrate your existing information views.



LESSON OBJECTIVES

After completing this lesson, you will be able to:

- List the deprecated modeling artefacts
- Explain how to migrate modeling content

Deprecated Modeling Objects in SAP HANA Studio

In the context of SAP HANA Studio modeling, several types of graphical calculation views existed in the past, namely attribute views and analytic views. Historically, calculation views were used only when attribute or analytic views could not fulfill the requirement.

Over time, the successive SAP HANA releases have brought a lot of enhancements to the functional coverage and performance of graphical calculation views. As a result, the use of attribute and analytic views is no longer recommended. All in all, from SAP HANA SPS12 onwards, most use cases should see an equal or better performance of calculation views (even if there can still be a few exceptions).

Besides, it was possible, up to SAP HANA SPS11, to create scripted calculation views. This type of view is now deprecated, and replaced by table functions, which offer better optimization possibilities.

Unit 7: Management and Administration of Models

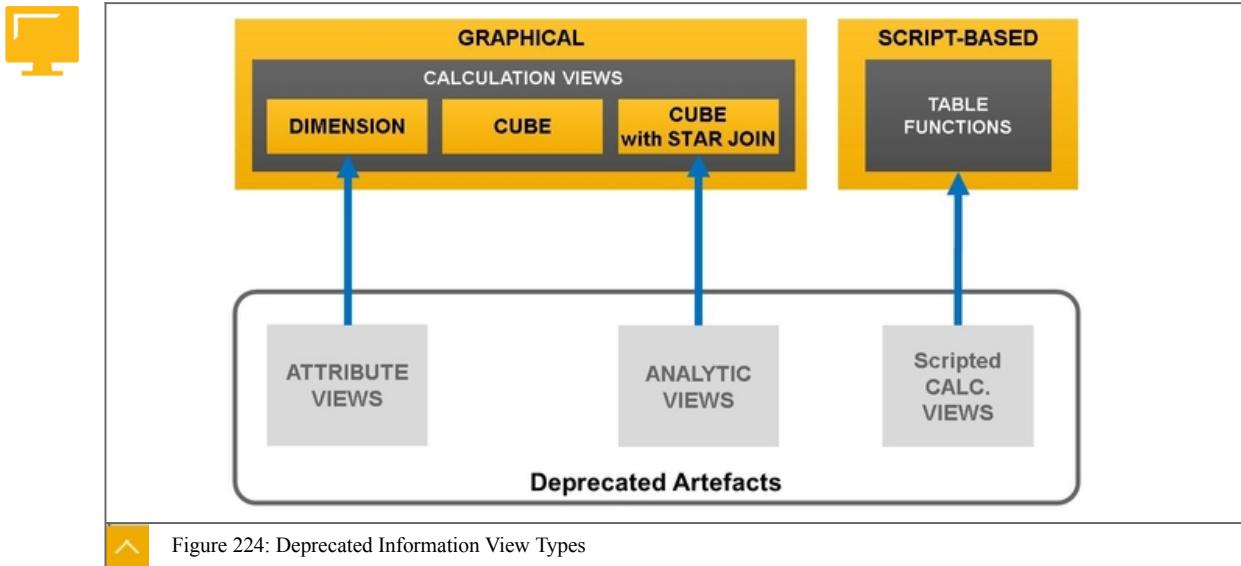


Figure 224: Deprecated Information View Types

The figure, Deprecated Information View Types, shows the deprecated object types and what replaces them.

In addition, to secure the access to the data of information models, it was historically possible to create two different types of Analytic Privileges: XML-based and SQL Analytic Privileges. Now, XML-based (also called “classic”) Analytic Privileges are deprecated.

Migrating Modeling Content

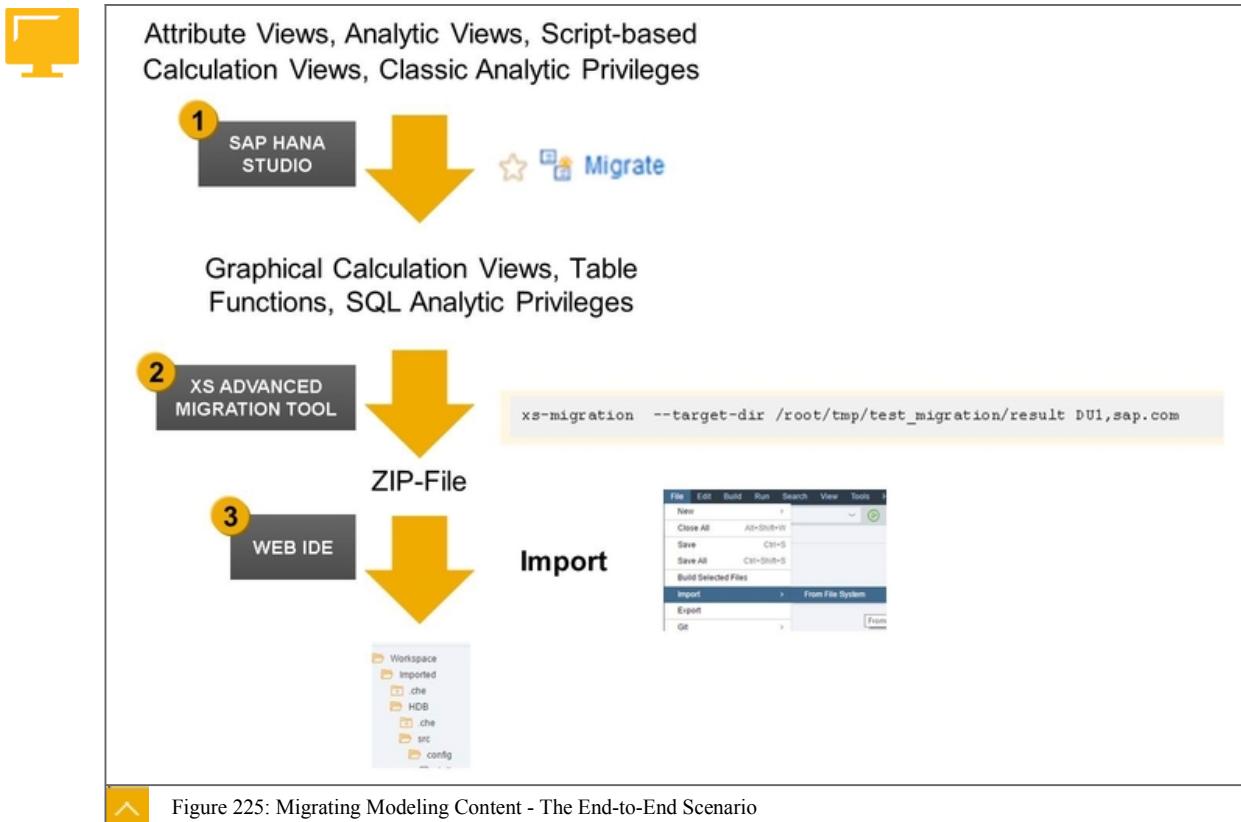


Figure 225: Migrating Modeling Content - The End-to-End Scenario

Migrating modeling content involves two key different steps and tools, before you can import an XS Advanced project into your workspace in the SAP Web IDE for SAP HANA.

The first step, executed in the SAP HANA Modeler of SAP HANA Studio, is aimed at replacing all the deprecated objects types with more recent ones.

The second step is executed in the XS Advanced environment, with a specific XS Advanced Migration Assistant. This assistant is invoked from the command line, and it generates a template of an XS Advance project and places the migrated objects in a specific folder. In addition, a migration report is generated and lists the problems that occurred during the migration. They need to be solved before you trigger the migration assistant again.

Finally, the generated .zip archive containing the migrated content can be imported to your workspace in the SAP Web IDE.

To learn more about the Modeling Content Migration

- The SAP HANA Studio migrate tool is documented in the SAP HANA Modeling Guide for SAP HANA Studio

You can also find additional information in SAP Note [2325817](#), and [2236064](#) in BW contexts.

- The XS Advanced Migration Assistant has its own guide from SAP HANA 2.0 SPS01 onwards.



LESSON SUMMARY

You should now be able to:

- List the deprecated modeling artefacts
- Explain how to migrate modeling content

Unit 7

Learning Assessment

1. Which of the following features analyze dependencies between models?

Choose the correct answers.

- A** Data lineage
- B** Column lineage
- C** Impact analysis

2. Which of the following features of an XS Advanced application are provided by an HDB module?

Choose the correct answers.

- A** Persistence layer
- B** User interface
- C** Business logic
- D** Virtual data modeling

3. An XS Advanced project can contain multiple HDB modules.

Determine whether this statement is true or false.

- True
- False

4. The database objects defined in design-time files located in the same project folder can have different namespaces.

Determine whether this statement is true or false.

- True
- False

5. A HDB module always corresponds to one database schema.

Determine whether this statement is true or false.

- True
 False

6. You have successfully built a project you imported in the SAP Web IDE workspace, but you cannot see the corresponding runtime database objects. What could be the reason?

Choose the correct answer.

- A The project is not assigned to a space
 B Building a project does not generate the database content of its HDB modules
 C You do not have the Developer role in the assigned space.

7. A design-time file F1 has been created in your project but has never been built. Your perform a build of another design-time file F2. The F1 file will be included in the dependency check by the HDI Builder.

Determine whether this statement is true or false.

- True
 False

8. You have copied a design-time file from a source folder to a target folder that has the same namespace setting. In which of the following scenarios will a build of the copy design-time file be successful?

Choose the correct answers.

- A The build of the copy design-time file will always be successful
 B The source design-time file has never been built.
 C The source design-time file has been built once but the built has failed.
 D The source design-time file has been deleted from the source folder in the meantime.

Unit 7: Learning Assessment

9. Which database artifact do you use to access external data from an XS Advanced project?

Choose the correct answer.

- A** Calculation view
- B** Synonym
- C** Logical schema
- D** User-provided service

10. The GIT architecture relies on a central server that stores a unique version of your code and a check-out process that prevents several users from modifying a code file at the same time.

Determine whether this statement is true or false.

- True
- False

11. Which phase of a project does SAP Enterprise Architecture Designer support?

Choose the correct answers.

- A** Development
- B** Design
- C** Planning
- D** Running

12. SAP ED Designer can be used as an alternative to develop calculation views in SAP Web IDE

Determine whether this statement is true or false.

- True
- False

Unit 7

Learning Assessment - Answers

1. Which of the following features analyze dependencies between models?

Choose the correct answers.

- A** Data lineage
- B** Column lineage
- C** Impact analysis

You are correct! Data lineage and impact analysis analyze the dependencies between models, whereas column lineage only shows the origin of a column within a single model.

2. Which of the following features of an XS Advanced application are provided by an HDB module?

Choose the correct answers.

- A** Persistence layer
- B** User interface
- C** Business logic
- D** Virtual data modeling

You are correct! The persistence layer and virtual data modeling are defined in an HDB module. The user interface typically uses HTML5 or UI5 modules, and the business logic uses Java or Node.js modules.

3. An XS Advanced project can contain multiple HDB modules.

Determine whether this statement is true or false.

- True
- False

You are correct! A project can contain more than one HDB module.

Unit 7: Learning Assessment - Answers

4. The database objects defined in design-time files located in the same project folder can have different namespaces.

Determine whether this statement is true or false.

True

False

You are correct! The rules defining namespaces can be defined differently for each folder, but within a design-time folder, all the defined database objects always have the same namespace.

5. A HDB module always corresponds to one database schema.

Determine whether this statement is true or false.

True

False

You are correct! When a HDB module is built for the first time, one schema is created in the SAP HANA database and managed by the corresponding HDI container service.

6. You have successfully built a project you imported in the SAP Web IDE workspace, but you cannot see the corresponding runtime database objects. What could be the reason?

Choose the correct answer.

A The project is not assigned to a space

B Building a project does not generate the database content of its HDB modules

C You do not have the Developer role in the assigned space.

You are correct! Building a project generates an MTA archive for deployment, but does not trigger a build of the HDB module(s). If the project were not assigned to a space, the build would fail. And you can only assign a project to a space where you have the Developer role.

7. A design-time file F1 has been created in your project but has never been built. Your perform a build of another design-time file F2. The F1 file will be included in the dependency check by the HDI Builder.

Determine whether this statement is true or false.

True

False

You are correct! A design-time file that has never been built is ignored by the dependency check performed by the HDI builder if it is not part of the build scope.

8. You have copied a design-time file from a source folder to a target folder that has the same namespace setting. In which of the following scenarios will a build of the copy design-time file be successful?

Choose the correct answers.

- A** The build of the copy design-time file will always be successful
- B** The source design-time file has never been built.
- C** The source design-time file has been built once but the built has failed.
- D** The source design-time file has been deleted from the source folder in the meantime.

You are correct!

9. Which database artifact do you use to access external data from an XS Advanced project?

Choose the correct answer.

- A** Calculation view
- B** Synonym
- C** Logical schema
- D** User-provided service

You are correct! Calculation views can consume external data, but they need a synonym to access this data. A logical schema can be used to define synonyms more dynamically, but this is something defined in XS Advanced, not a database artifact. As for the user-provided service, it is required to access data from an external schema but it is not a database artifact.

10. The GIT architecture relies on a central server that stores a unique version of your code and a check-out process that prevents several users from modifying a code file at the same time.

Determine whether this statement is true or false.

- True**
- False**

Correct. The GIT architecture is not centralized but distributed, which means that several contributors to a project can have the complete project code on their own computer. There is no check-in / check-out process, compared with other types of control-version systems.

Unit 7: Learning Assessment - Answers

11. Which phase of a project does SAP Enterprise Architecture Designer support?

Choose the correct answers.

A Development

B Design

C Planning

D Running

Correct — SAP Enterprise Architecture Designer supports the planning and design phases of a project.

12. SAP ED Designer can be used as an alternative to develop calculation views in SAP Web IDE

Determine whether this statement is true or false.

True

False

Correct — SAP Enterprise Architecture Designer can only create the very basic calculation view and SAP Web IDE is still needed for further development, such as adding input parameters, restricted column etc.

UNIT 8

Security in SAP HANA Modeling

Lesson 1

Understanding Roles and Privileges

310

Lesson 2

Defining Analytic Privileges

322

Lesson 3

Defining Roles

335

UNIT OBJECTIVES

- Understand roles and privileges
- Define analytic privileges
- Create a design-time role

Unit 8

Lesson 1

Understanding Roles and Privileges

LESSON OVERVIEW

This lesson will give you an overview of how security is implemented in SAP HANA.

Business Example

After the SAP HANA System has been installed, you need to define security and give the relevant authorizations to users so that they can administrate the system, provision data, and, in the case of the Modeler role, start creating models.

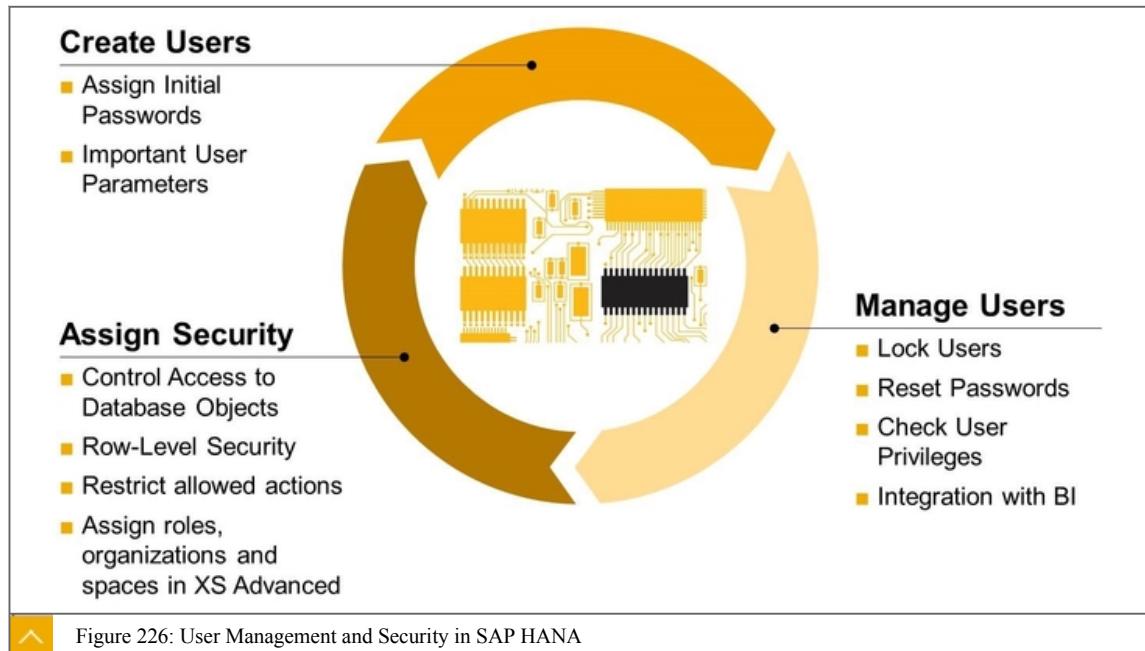


LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Understand roles and privileges

Overview of User Management and Security



Like any other database system, SAP HANA provides features to define and maintain security.

The reasons for implementing security are as follows:

- Restrict database administration to skilled and empowered employees only
- Ensure that each database user has relevant authorization to manipulate database objects
- Separate duties

- Manage access to the system from a variety of front-end tools
- Restrict access to the data stored in the system based on role, geographic, or organizational responsibilities

XS Advanced and SAP HANA Database Security

When you work in SAP HANA with the SAP Web IDE (so, in XS Advanced), there are two aspects of security that you must consider.

- XS Advanced Security

XS Advanced provides its own runtime, which is separate (though heavily connected) to the database. The XS Advanced environment has its own way to implement security. This security is defined in the XS Advanced Administration Tools. For example, when you log on to the Web IDE, a dedicated XS Advanced service checks whether you have the role **Web IDE developer**, and also which space(s) –for example the **DEV** space used for this training– you are allowed to develop in.

- SAP HANA Database Security

The database security is what governs general authorizations on the database system, access to objects (tables, views, procedures), fine-grained access to data with analytic privileges, and so on.

Most of the security artifacts that you build in a HDB module are classic HDB security objects, such as roles, analytic privileges, and so on. Once the roles are built, you can assign them to classic database users.



Note:

This is different when you work in SAP HANA Studio and/or XS Classic, because in this case all the security is relying on classical security artifacts: system privileges, object privileges (schemas and/or objects they contained), analytic privileges, and also, for example, package privileges (for SAP HANA Studio-based modeling) and application privileges (for XS Classic development).

Unit 8: Security in SAP HANA Modeling

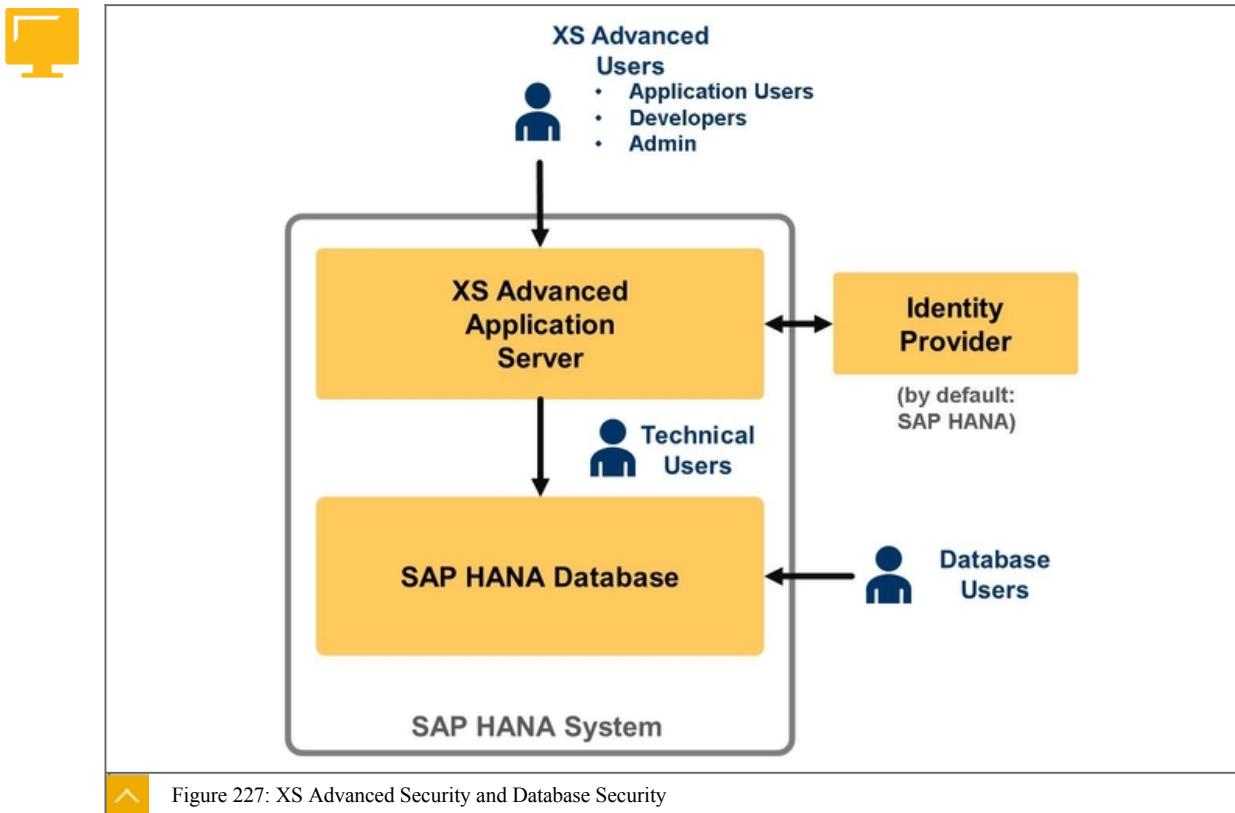


Figure 227: XS Advanced Security and Database Security

The interaction between the XS Advanced Application Server and the SAP HANA database involves a number of technical users. For example, each and every HDI container deployed in the database comes with technical users to manage the ownership of the corresponding schema and the database objects (tables, column views) as well as the key security artifacts such as analytic privileges, roles, and so on.

When you work on a project in the SAP Web IDE for SAP HANA, almost everything you do, for example, access tables from your container or from an external schema, build a calculation view, preview its data, and so on, is done on behalf of your Web IDE user by a technical user that is specific to your container, which is granted the necessary authorizations to the container's schema. In the Database Explorer, viewing the objects of a container's schema is done by this technical user.

On the contrary, when you add a classic database to the Database Explorer and specify a user and password, the queries you execute on this database are executed by this user. In this case, no technical user is involved.

XS Advanced Users

The XS Advanced (XSA) users are listed in the SAP HANA database like classic database users. The main difference is that additional user properties, specific to XSA user management, are used to define the security for these users.

When you create a user in the XSA environment, this is done in the Administration Tool. There are two different possibilities:

XS Advanced

- Create a new user
With this approach, you create a brand new user.
- Reuse an existing database user for XSA,

In this case, you choose an existing database user and enable this user for XSA.

When a user is enabled for XSA, you assign this user a role collection, which contains one or several XSA application roles.

Let's take the following examples:

- Your STUDENT## user is assigned the role collection XSA_WEBIDE_DEV that allows this user to access the Web IDE and create applications.
- The XSA_ADMINuser (super-admin user in the XSA environment) is assigned other role collections to manage users, and so on.

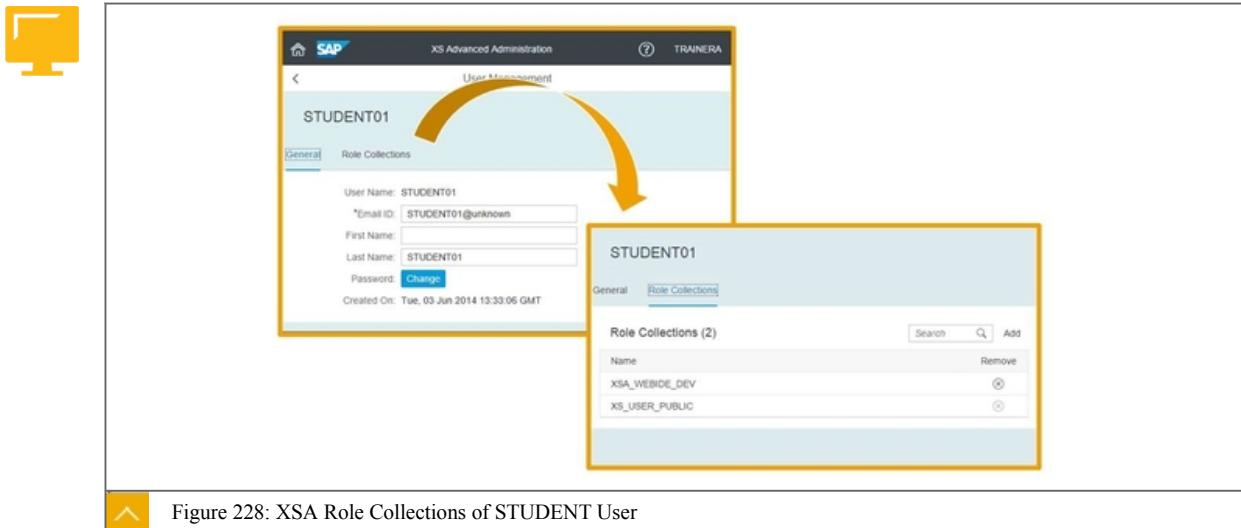


Figure 228: XSA Role Collections of STUDENT User

Note:

During Exercise 1 of this course, you also granted your STUDENT## user specific authorizations to your container. This is what allows your user to directly consume the container's tables and column views with front-end tools such as MS Excel or SAP BusinessObjects Analysis for Excel.

Organizations and Spaces

In addition to the role collections, the security in XSA relies on organizations and spaces.

Unit 8: Security in SAP HANA Modeling

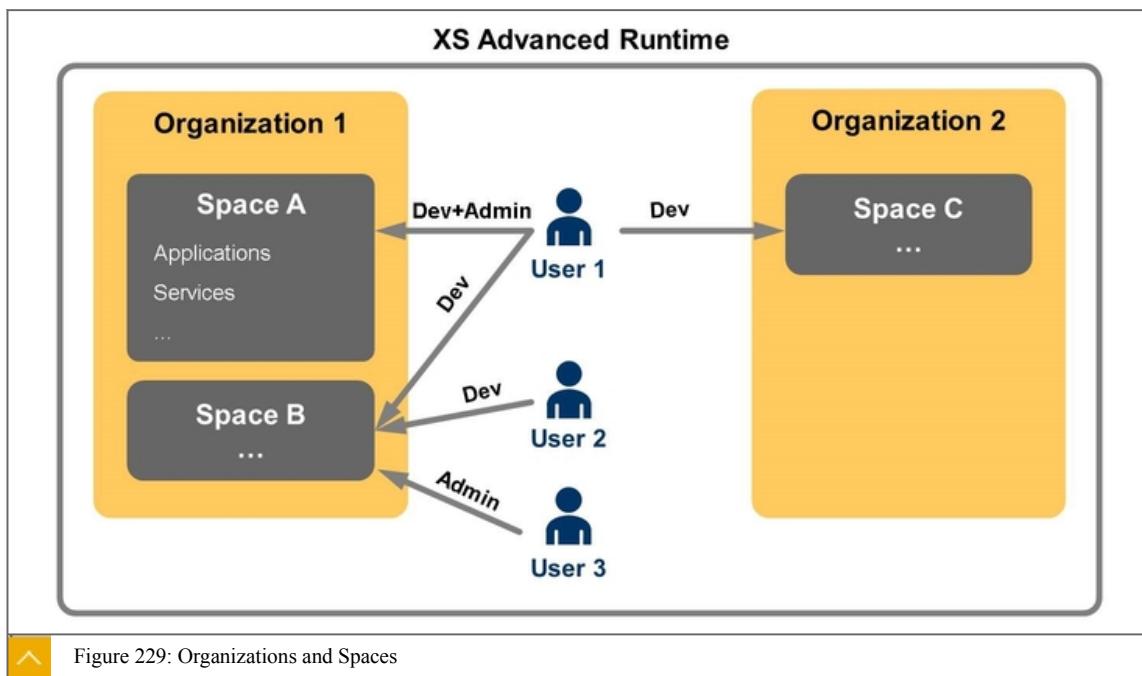


Figure 229: Organizations and Spaces

- **Organizations**

Organizations enable developers to collaborate by sharing resources, services, and applications. Access to the shared resources, services, and applications is controlled by roles, for example, Org Manager or Org Auditor ; the role defines the scope of the permissions assigned to the named user in the organization.

For example, an Org Manager can add new users to organizations; create, modify, or delete organizational spaces; and add domains to the organization.

- **Spaces**

In an organization, spaces enable users to access shared resources that can be used to develop, deploy, and maintain applications.

Access to the resources is controlled by the following roles: Space Manager , Space Developer , and Space Auditor . The role defines the scope of the permissions assigned to the named user in the organizational space. For example, a Space Developer can deploy and start an application.

Organization and Space of STUDENT User

Figure 230: Organization and Space of STUDENT User

The figure, Organization and Space of STUDENT User, shows that the STUDENT## users are assigned as Developers in the DEV space of the SAP organization.

In XSA, a user can be assigned to several spaces, with different roles, in one or several organizations.

The SAP Space

The SAP space is created by default when you install XS Advanced, and this is where most of the applications and services for XSA Developers and Administrators are located. For example, the SAP Web IDE, the XS Advanced Administration Tools, and many others, such as the back-end services used by the Database Explorer, the User Authentication and Authorization services, and the HANA Deployment Infrastructure (HDI) applications used to build your modeling content.

Caution:

For security reasons, it is highly recommended to keep all these applications and services separate from the applications you create and to strictly restrict the authorizations granted to the SAP space.

Unit 8: Security in SAP HANA Modeling

Database Security: Users, Roles and Privileges



- A known user can log on to the database. A user can be the owner of database objects.
- A role is a collection of privileges and can be granted to either a user or another role (nesting).
- A privilege is used to enable a specific operation on one or several specific objects.

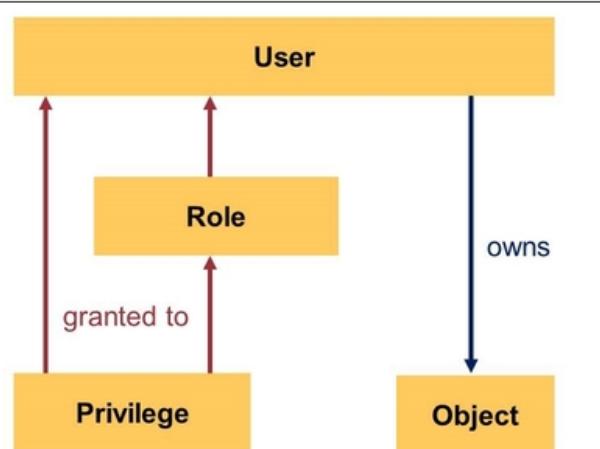


Figure 231: Users, Roles, Objects and Privileges

The figure, Users, Roles, Objects and Privileges, shows how users and roles can be granted privileges.

- Privileges can be assigned to users either directly, or indirectly by using roles.

Roles help you to structure the access control scheme and model reusable business roles. They can be nested, enabling the implementation of a hierarchy of roles.



Hint:

It is highly recommended that you manage authorizations for users by using roles. Assigning a privilege directly to a user is not a good practice.

- All the privileges granted directly or indirectly to a user are combined. Whenever a user tries to access an object, the system performs an authorization check based on the user's roles and directly allocated privileges (if any).
- It is not possible to explicitly deny privileges. In particular, the system does not need to check all the user's roles. As soon as all the privileges required for a specific operation on a specific object have been found, the system ends the check and allows the operation.
- Several predefined roles exist in the SAP HANA database. Some of them are templates (and need to be customized), and others can be used as they are.
- As a best practice, users should only be given the smallest set of privileges required for their role.

Defining Roles

In the SAP HANA database, there are two ways to create roles:

- As pure runtime objects that follow classic SQL principles (**Catalog Roles**)

- By means of design-time files that you create in the HDB module of a project (**Design-Time Roles**)



Table 21: Catalog Roles vs. Design-Time Roles

Feature	Catalog Roles	Design-Time Roles
Transportability	No	Yes
Version Management	No	Yes
Relationship to database users	Each role is owned by the database user that created it.	The roles created when building the HDB Module are owned by the technical user that owns the container's content.

In general, repository roles are recommended, as they offer more flexibility. In particular, they can easily be transported between SAP HANA systems via the build/deployment of XSA Applications.

Design-time roles are created as `.hdbrole` files within a project, with the SAP Web IDE.



Note:

A typical use case where catalog roles can be used is when security is managed in another tool, and the SAP HANA database is only accessed by a very limited number of technical users. But in the context of XSA-based modeling, design-time roles are definitely recommended, so the authorizations stay in sync with the objects they refer to.

How to Assign Privileges

The ways to maintain users and roles in SAP HANA are as follows:

- In SAP HANA Studio, by using the dedicated folder `Security` of the Systems view.
- From an SQL Console, in SAP HANA Studio or the Database Explorer of the SAP Web IDE, by executing SQL statements.

Unit 8: Security in SAP HANA Modeling



- It is possible to assign privileges via an SQL statement or using the SAP HANA Studio GUI.

- Example:
Assigning the generic access role of a container to your student's **TRAINING_ROLE_##**

SQL Console (Database Explorer)

```
SQL Console 2.sql x
1 - GRANT "HA300_00_HDI_CONTAINER_1::access_role" TO TRAINING_ROLE_00;
```

SAP HANA Studio GUI

Systems

H00@H00 - STUDENT00 H00@H00 - TRAINING_ROLE_00

H00@H00 (SYSTEM) My HANA Database wdfibmt7215.wdf.sap.corp 00

User Parameters

Force password change on next logon: Yes

Kerberos External ID:

Valid From: Jun 3, 2014 3:33:06 PM GMT+02:00

Session Client: 800

Granted Roles Part of Roles System Privileges Object Privileges Analytic Privileges Package Privileges Application Privileges Privileges or Details for 'TRAINING_ROLE_00'

TRAINING_ROLE_00

Role Grantor

AFL_SYS_AFL_AFLPAL_EXECUTE SYSTEM
PUBLIC SYS
TRAINING_ROLE_00 SYSTEM
TRAINING_USER_ROLE SYSTEM

Granted Roles System Privileges Object Privileges Analytic Privileges Package Privileges Application Privileges Details for 'TRAINING_ROLE_00'

Grantable to other users and roles:

Figure 232: Assigning Roles by SQL or HANA Studio UI

The figure, Assigning Roles by SQL or HANA Studio UI, shows how to grant the generic access role of a container to your **TRAINING_ROLE_##**. This is what you did as part of Exercise 1.



Note:

This generic access role is also granted to the technical user who works “on your behalf” when you are logged on to the Web IDE and working on an XSA project.

Managing Users and Roles



Define and Create Roles

Create Users

Assign Privileges to Roles

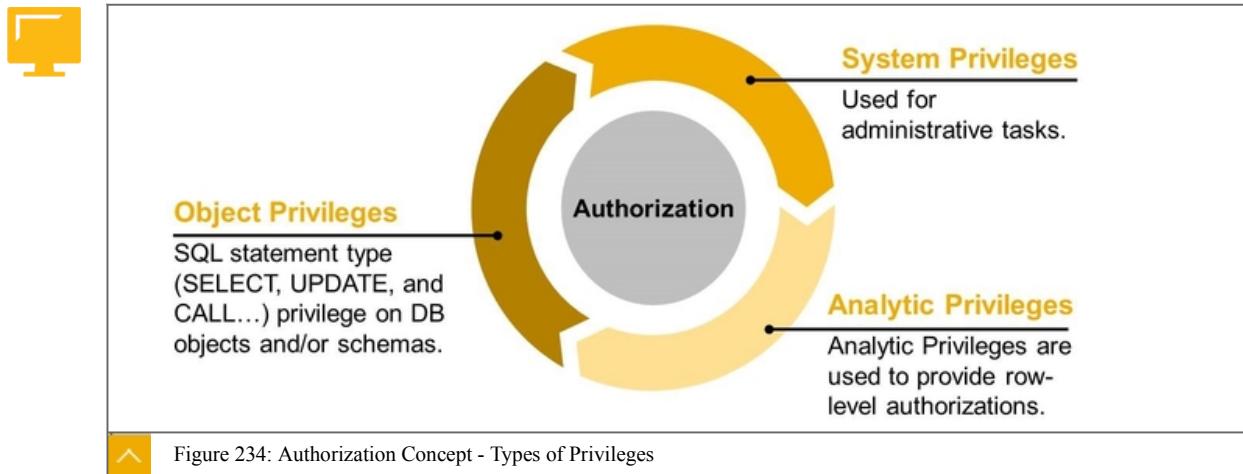
Grant Roles to User

Figure 233: Managing Users and Roles

The figure, Managing Users and Roles, shows the general process to define users and roles and assign privileges. In a typical SAP HANA modeling scenario with the Web IDE, defining the roles and assigning privileges (or other roles) to roles is done in design-time files.

Types of Privileges

Let's have a quick overview of the different types of privileges that can be defined in the database.



When modeling or developing applications in XSA, two types of privileges are not used at all:

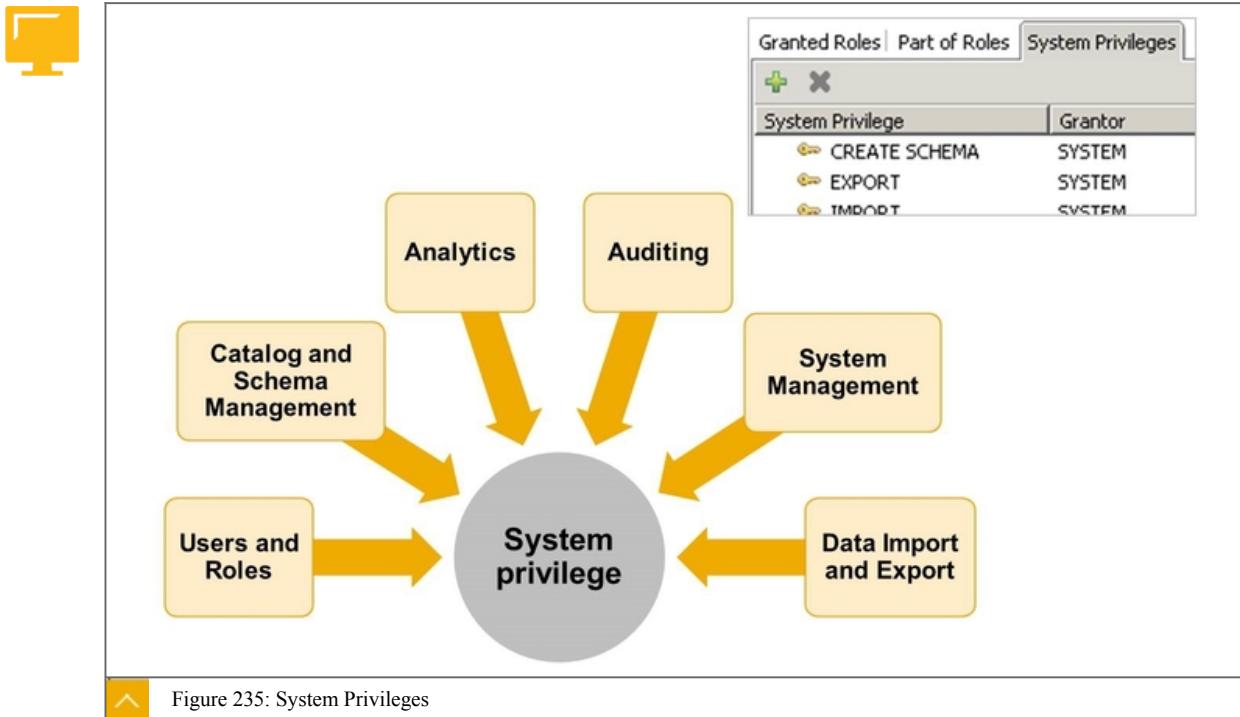
- Package privileges
- Application privileges

System Privileges

System privileges are used to control general system activities. They include general administrative actions, such as creating or deleting schemas, managing users and roles, performing backup, monitoring and tracing, and so on.

System privileges also include several privileges directly related to the modeling activities in SAP HANA Studio (not in the SAP Web IDE), such as maintaining, exporting, and importing delivery units.

Unit 8: Security in SAP HANA Modeling



Object Privileges

Object privileges are used to allow access to and modification of database objects.

Each object privilege is related to one object and, depending of the type of object (schema, table, and procedure) includes the different SQL statement types, which you can grant separately; for example, CREATE, ALTER, DROP, SELECT, INSERT, UPDATE, DELETE, and EXECUTE.



Note:

Most of the object privileges provide an option defining whether the grantee (user or role) has the right to grant it to other users.

Object Privileges for Modular Role

For the modeler role, object privileges are a key building block to define security, in particular because they control the way users read and modify data, both in the Container of a deployed application, but also in external schema or other containers.



- Object privileges are bound to an object, for example, to a database table or schema, and enable object-specific control activities, such as SELECT, UPDATE, or DELETE.

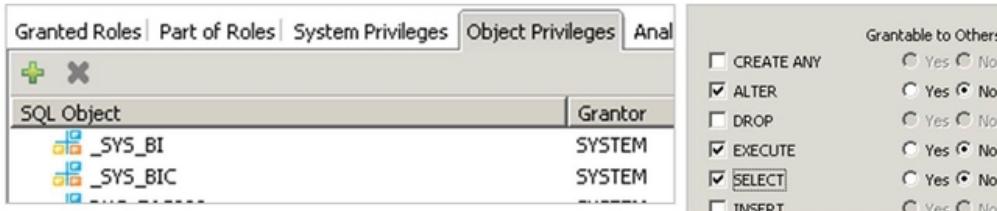


Figure 236: Object Privilege

Analytic Privileges

Analytic privileges are used to enable data access to different portions of data in calculation views, by filtering the attribute values.

Each information view has a dedicated property indicating whether the execution of the view must be restricted by analytic privileges or not. You will learn more about this distinction later.



Note:

An analytic privilege is evaluated when a user executes a query against one of the information views to which the privilege applies.

Related Information

You can find more information about security in the following guide, available on the SAP Help Portal. Go to <http://help.sap.com/hana>:

- SAP HANA Security Guide

You can also refer to the training course **HA240 - SAP HANA Security & Authorization**. Go to <http://training.sap.com> for more details.



LESSON SUMMARY

You should now be able to:

- Understand roles and privileges

Unit 8

Lesson 2

Defining Analytic Privileges

LESSON OVERVIEW

This lesson will describe how to create Analytic Privileges in the SAP Web IDE for SAP HANA, in order to control the access to data based on attributes. Note that two different flavors of Analytic Privileges existed in SAP HANA Studio modeling, but in XSA-based modeling with the SAP Web IDE, only SQL Analytic Privileges are supported. The focus will be put on this type of Analytic Privileges.

Business Example

You work as a Modeler on an SAP HANA Project, and you have been asked to design the data access security. You need to learn about how to define and assign Analytic Privileges.

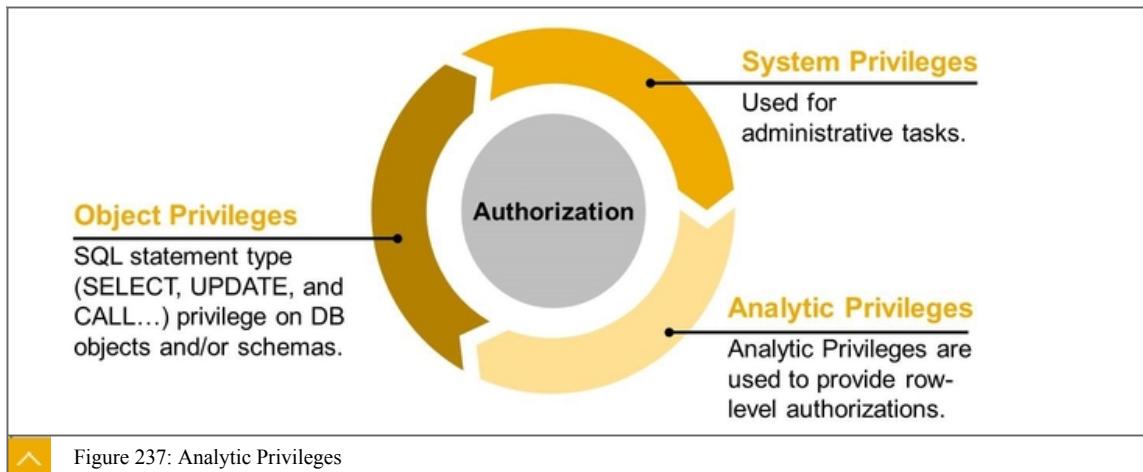


LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Define analytic privileges

Analytic Privileges



Analytic privileges are used to enable data access to different portions of data in information views, by filtering the data based on the values of one or more attributes.

The rationale for analytic privileges is to allow the use of information views by different users who might not be allowed to see the same data.

For example, different regional sales managers who are only allowed to see sales data for their regions could use the same information view, but you would have to define and assign analytic privileges to the view so that each manager sees only the data for their region.

**Note:**

This is different from the behavior of SAP Business Warehouse (BW). While the concept is similar, SAP BW will forward an error message if you execute a query that returns values you are not authorized to see.

With SAP HANA, the query would be executed and, depending on your authorizations, only values that you can see would be returned.

Types of Analytic Privileges

To avoid any confusion, let's explain quickly that there were historically two different types of analytic privileges

Two Flavors of Analytic Privileges



- SQL analytic privileges

This type of analytic privilege provides the highest flexibility in scenarios where filter conditions are complex. In addition, it can be used on a large variety of objects, including calculation views, CDS views, and catalog views.

- Classical analytic privileges (also called XML-based analytic privileges)

This is the historical data access restriction type, and it is NOT supported in XSA-based modeling (in SAP Web IDE).

**Note:**

A migration utility is available in SAP HANA Studio to convert classical analytic privileges to SQL analytic privileges.

In the following sections, we only cover SQL analytic privileges.

SQL Analytic Privileges — The End-to-End Scenario

To secure a calculation view with SQL analytic privilege, the main steps are as follow:

To Create and Assign an Analytic Privilege



1. Start the analytic privilege creation wizard.
2. Assign the calculation view(s) that you want to secure with this analytic privilege.
3. Choose the type of restrictions you want to use and define the restrictions.
4. Set the secured calculation views to check SQL analytic privileges.
5. Save and build the analytic privilege.
6. Assign the analytic privilege to a role.
7. Assign the role to a user.

1. Start Creation Wizard

- Select a folder in your project.

Unit 8: Security in SAP HANA Modeling

- Choose New → Analytic Privilege .
- Provide a name and description and choose Create .

2. Select Information Models

An analytic privilege definition contains the list of information models to which it will apply. You can choose one or several of the following objects:

- Calculation views of type DIMENSION and CUBE (with or without a star join)
- CDS views
- Catalog views



Note:

Defining the list of secured information models is a pre-requisite if you want to create restrictions based on attributes explicitly chosen among the actual attributes defined in these models. You will learn more about restrictions types later on.

The analytic privilege editor lists, at any time, all the models that are secured by the analytic privilege.

Analytic Privilege - Select Information Models

**Select applicable Information Models**

- The views selected to define an Analytic Privilege determine:
 - Which views the Analytic Privilege will apply to
 - What attributes you can choose to define restrictions
- You can add other views to an existing Analytic Privilege at any time

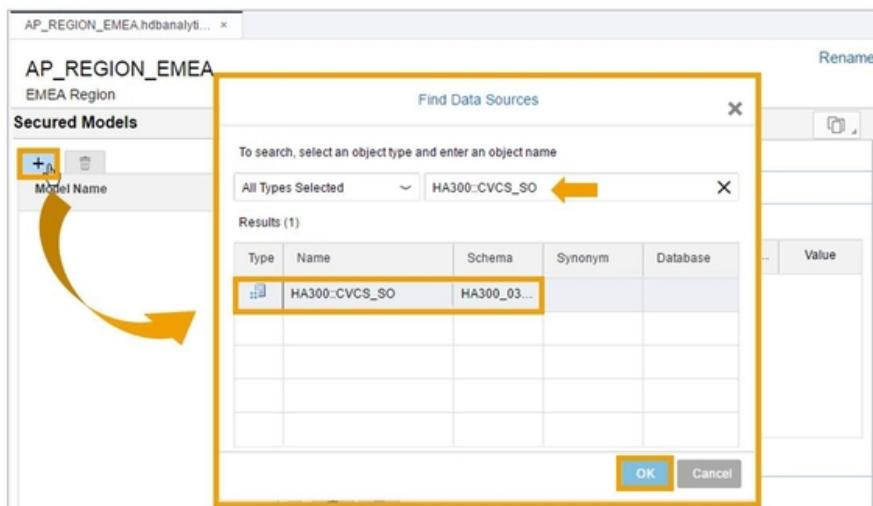


Figure 238: Analytic Privilege - Select Information Models

3. Define the Restrictions

There are three methods available to define the restrictions in an analytic privilege. These methods are exclusive; that is, in one analytic privilege, you use only one of these methods.



Table 22: Restriction Types in Analytic Privileges

Restriction type	How to Use	Restriction Example
Attribute	With the restriction editor, select one or several attributes from the secured views. For each of them, define restrictions.	<ul style="list-style-type: none"> • REGION: EMEA • YEAR: Between 2015 and 2017
SQL Expression	Create a valid static SQL expression that refers to the attributes and the authorized values. This is useful when the Attribute restriction type does not fulfill the requirement.	(“REGION”=’EMEA’ AND “YEAR”=’2015’) (valid SQL expression)
Dynamic	Use a procedure to derive a dynamic SQL expression to restrict the data set. This expression must be similar to a WHERE clause in a select statement.	P_DYNAMIC_AP_FOR_REGION (name of the procedure)

You can change between restriction types, depending on your needs.

Changing Restriction Type



Figure 239: Changing Restriction Type



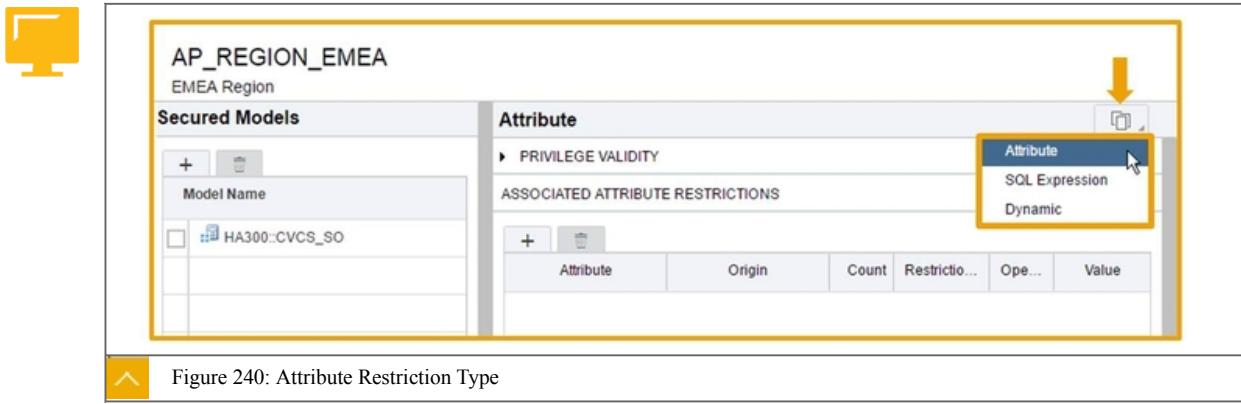
Note:

A restriction of type Attribute can be automatically converted into an SQL Expression restriction, but the other way round is NOT possible.

To define a restriction of type Attribute, you can select among the list of shared and private attributes from the secured calculation views. You can define as many restrictions as needed to define the correct data set.

Unit 8: Security in SAP HANA Modeling

Defining a Restriction of Type Attribute



Defining Values in the Restriction Filter

To define values in the restriction filter, you can use the following operators:

- Between <scalar_value_1> <scalar_value_2>
 - ContainsPattern <pattern with *>
 - Comparison operators: =, <=, <, >, >= with <scalar value>
 - IsNull and Is Not Null

Note:

All filter operators, except **IsNull** and **Is Not Null**, accept empty strings (" ") as filter operands. For example:

- In (" ", "A“, "B“)
 - Between (" ", "XYZ“) (as lower limit in comparison operators)

Only columns of type Attribute (NOT Measure) can be specified in dimension restrictions.

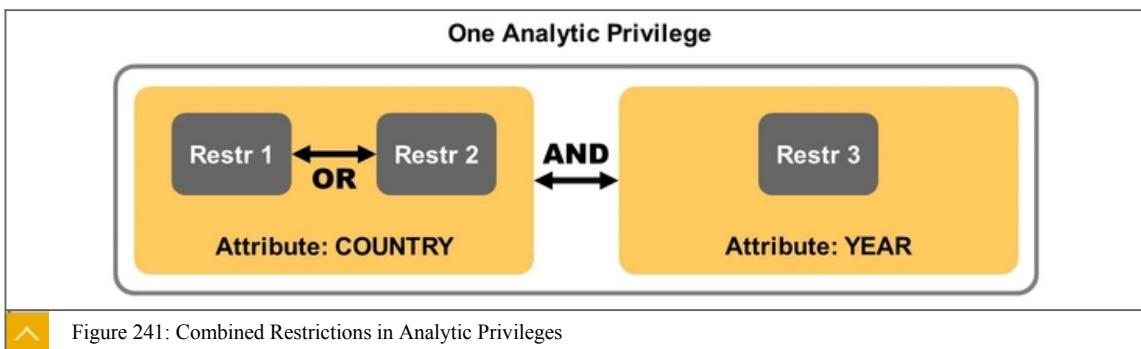
If a DIMENSION calculation view contains a parent-child hierarchy and the hierarchy is enabled for SQL access, it is also possible to define the restriction on a hierarchy node. This is discussed in a dedicated section later on.

Combining Several Attribute Restrictions

Several restriction filters within an analytic privilege are combined in the following way:

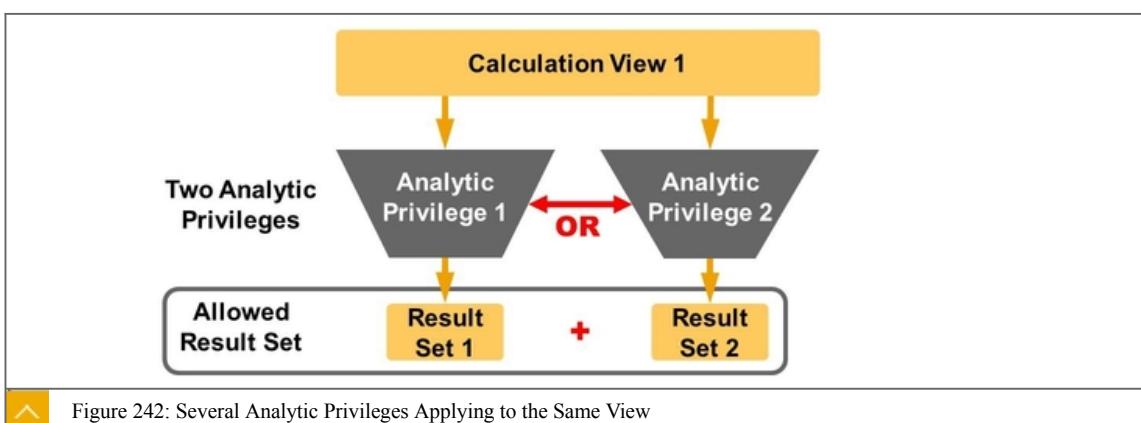
- Within one attribute column, several restrictions are combined with a logical OR.
 - Within one analytic privilege, all dimension restrictions are combined with a logical AND.

For example, if an analytic privilege includes two restrictions on two different attributes (YEAR=2017, COUNTRY=US), the user is allowed to see only data fulfilling the compound condition YEAR=2017 AND COUNTRY=US.



Several Analytic Privileges Applying to the Same View

If two analytic privileges (or more) are defined to apply to the same view, SAP HANA combines the corresponding conditions with a logical “OR”.



Caution:

If the two restrictions from the previous example were defined in two different analytic privileges applying to this user and this view, the user would see more data. Namely all the rows for which `YEAR=2017 OR COUNTRY=US` (that is, any year for `COUNTRY=US` and any country for `YEAR=2017`).

Defining a Validity Period

You can decide to make an analytic privilege of the type `Attribute` valid for only a certain period of time. This restriction applies to the date when users are querying the secured view (`CURRENT_DATE`).



Caution:

This is not related to the time attributes (year, month, date, and so on) defined in your views. To limit the access to the data based on such attributes, you use a classic attribute-based restriction, for example on a `DATE` or `YEAR` column.

Restricting Value with an SQL Expression

This second type of restriction allows you to define a filtering expression that cannot be obtained with restrictions of type `Attribute`. For example, when the precedence of OR and AND logical operators does not correspond to what you want to define.

Unit 8: Security in SAP HANA Modeling

Suppose you want to allow a user to view all the data from the country he's responsible for (US), but also all the historical data for any country. You could create a restriction of type SQL Expression as follows:

```
("COUNTRY" = 'US' OR "YEAR" <= '2016')
```

**Hint:**

Instead of writing your SQL expression from a blank page, you can start defining a restriction of the type Attribute , and then convert the restriction type to SQL expression . The corresponding SQL code will be already generated for you, and you will just need to adjust it to your requirements.

Defining a Validity Period

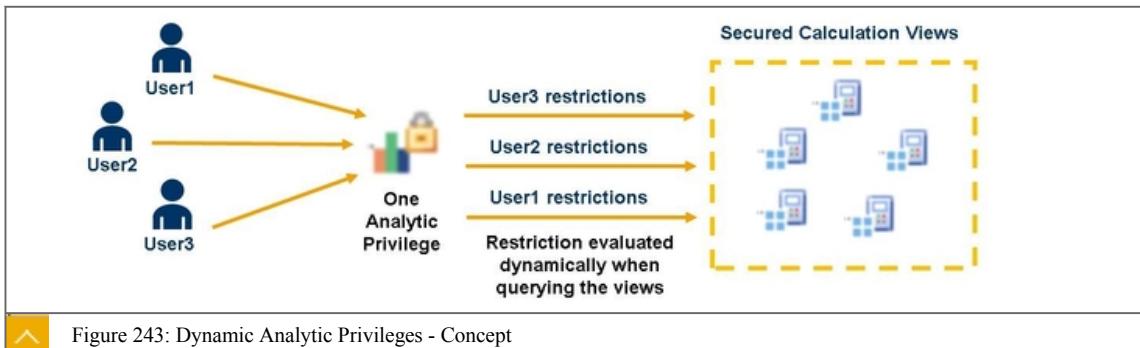
Restrictions of the type SQL Expression also support a validity period, by adding to the SQL expression the relevant SQL code to filter the CURRENT_DATE.

For example:

```
(CURRENT_DATE BETWEEN '2017-07-28 00:00:00.000' AND '2017-07-28  
23:59:59.999')  
AND < SQL expression to define the restriction >
```

Creating a Dynamic Restriction Type

In an analytic privilege, in addition to static values filtering conditions, it is possible to determine the filtering conditions in a dynamic way, which means that the filtering condition is not defined once for all, but is evaluated dynamically when the view is executed. This is called Dynamic Analytic Privilege .



With a dynamic restriction, the filtering conditions that apply for a specific user are determined at runtime. This allows a more scalable approach, where the same analytic privilege can be granted to several users who have different authorization requirements.

For example, the COUNTRY attribute in one or several calculation views can be filtered dynamically based on the actual list of countries that each is allowed to access, depending on his position in the geographical organizational structure.

Technically, to implement a dynamic restriction, you assign to the analytic privilege one procedure, which returns a SQLScript expression to filter data, like in the WHERE clause of an SQL statement.

For example, a procedure could return ("COUNTRY"='US') for User1 and ("COUNTRY"='UK' OR "COUNTRY"='FR') for User2 .

This procedure must have the following properties:

Dynamic Restriction - Procedure Properties



- Procedure must be read-only
- Security mode must be DEFINER
- No input parameters
- Only one scalar output parameter of type VARCHAR(256) or NVARCHAR(256)

4. Set the Secured Calculation Views to Check SQL Analytic Privileges

To actually secure a calculation view with an analytic privilege, you must set the **Apply Privileges** property of the calculation view to **SQL Analytic Privileges**.



Figure 244: Apply Privileges Property

5. Save and Build the Analytic Privilege

During the build of calculation views and analytic privileges, a specific dependency check is triggered to avoid errors that would lead to unsecured calculation views. You get a build error in the following cases:

- If you try to build an analytic view after activating the check for SQL analytic privileges but NO analytic privilege has this view in its Secured Models list.
- If you try to build an analytic privilege but some of the (runtime) calculation views it secures do NOT have the property **Apply Privileges** set to **SQL Analytic Privileges**
- If you try to build a calculation view after deactivating the check for SQL analytic privileges but there is still one or several (runtime) analytic privileges that have this view in their Secured Models list.



Note:

The best approach is to build in parallel analytic privileges and the calculation views that they secure.

When the analytic privilege is built, its runtime version is created in your container schema.

6. Assign the Analytic Privilege to a Role

Once an analytic privilege is built, the calculation views it applies to cannot be viewed until the privilege is granted to the end user.

Unit 8: Security in SAP HANA Modeling

To do so, in your project, you create a design-time role and grants the new analytic privilege to this role.

7. Assign the Role to a User

The last step is to grant the role to the end user. This can be done in the SAP HANA Studio or with an SQL statement.

Summary of the Analytic Privilege Evaluation During a Query on a Calculation View



The Analytic Privilege Check evaluates Analytical Privileges:

- Granted to the User
- Applicable to the queried Information Model
- Currently valid (validity dates)
- With Attribute restrictions covering attributes of the view

If no Analytic Privilege for the user can be found, user queries are rejected with a "...not authorized" error message.

Detailed info can be found in the trace file of the Index Server (Administration → Diagnosis Files)

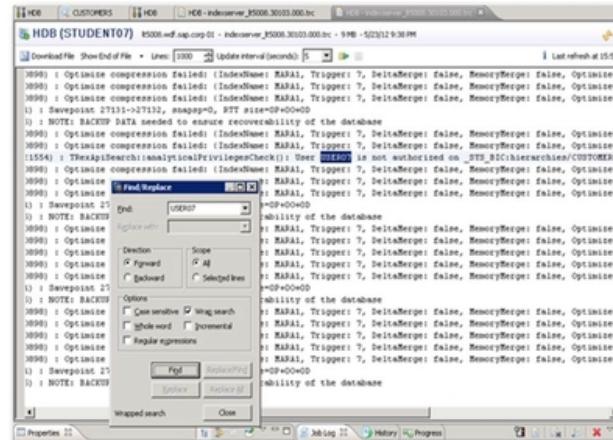


Figure 245: Analytic Privilege Check

The figure summarizes the analytic privilege check process.

If you need to solve issues related to the behavior of analytic privileges, you can find detailed information in the trace file of the index server:

- In the context menu of the SAP HANA system, choose Administration and select the Diagnosis Files tab.
- Locate the file index server_<host name>.<port>.<counter>.trc and double-click to open it.
- Choose Show End of File and search for the user name (press Ctrl+ F).

The trace log displays the analytical privilege check errors.

Securing CDS Views with SQL Analytic Privileges

From SAP HANA 1.0 SPS11 onwards, it is possible to apply SQL analytic privileges to CDS views.

This offers a new possibility in implementation scenarios where CDS views are used and exposed to the end user.

To Secure a CDS View with an Analytic Privilege

1. Enable the CDS view for SQL analytic privilege check.

You include in the .hbcds design-time file of the view the following annotation:

```
@WithStructuredPrivilegeCheck: true
```

2. Create an SQL analytic privilege.
3. Add the CDS view to the list of secured models.
4. Create an Attribute restriction type.

If needed, convert the restriction to an SQL expression.

Defining Restrictions on Hierarchy Nodes in SQL Analytic Privileges

SAP HANA supports attribute restrictions based on a hierarchy node (rather than a list of values, and interval) in SQL analytic privileges.

This is useful in scenarios where a user who is enabled to a certain node of the hierarchy must also be enabled to all the descendants of this node.

With a geographical hierarchy, for example, the manager of the North America area will see all the countries in his area if the analytic privilege sets the attribute restriction to the NA node. You do not have to list all the countries from this area.

To Use a Hierarchy Node in an Attribute Restriction

1. Enable the calculation view for SQL analytic privilege check and SQL access to hierarchies.
2. Create an SQL analytic privilege.
3. Add the calculation view to the list of secured models.
4. Create an Attribute restriction type.
5. In the restriction, choose Hierarchy Node .
6. Select one of the available hierarchies.
7. Choose a hierarchy node.



Caution:

This feature is supported:

- With calculation views of the type CUBE With Star Join that are designed to check SQL analytic privilege and enable SQL access to hierarchy.
- Only for parent-child Hierarchies.

Defining Data Access Security with Nested Information Views

There are many business cases where information views contain references to one another.

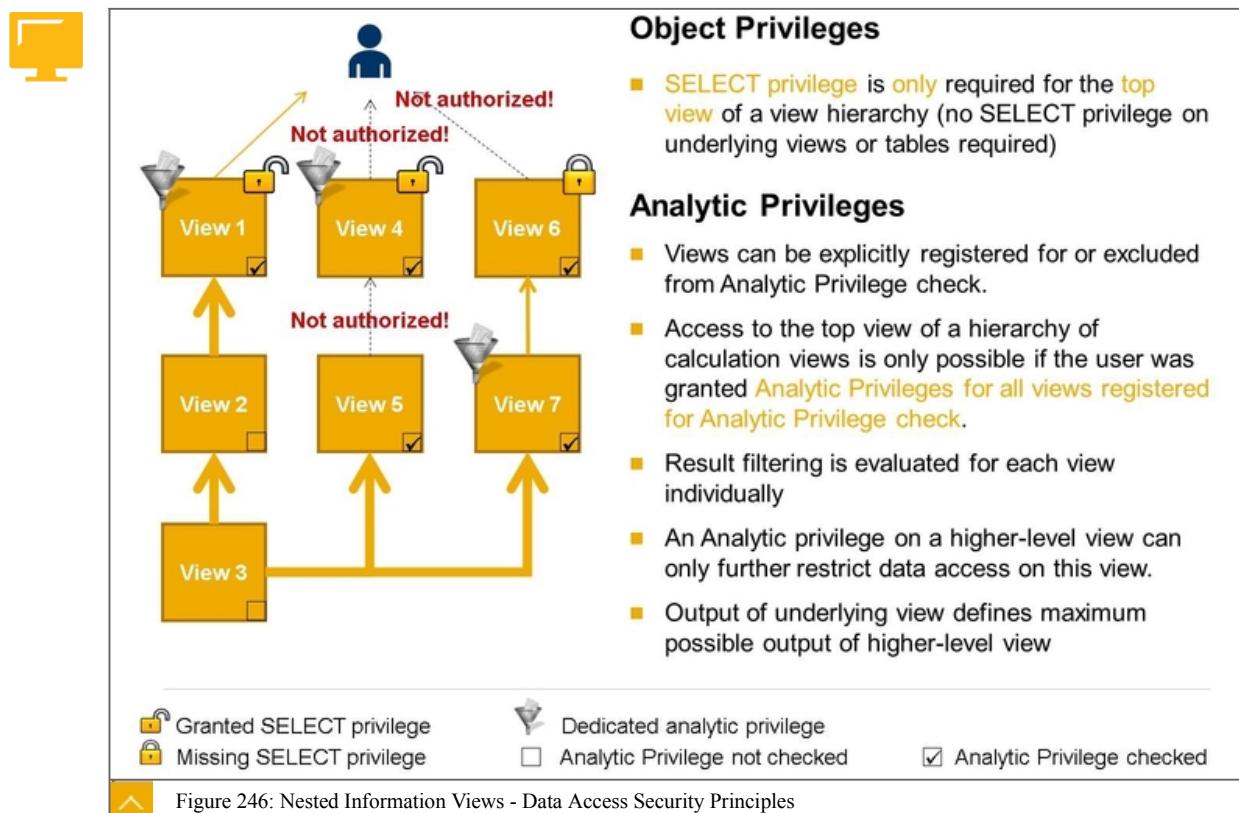
The figure, Nested Information Views - Data Access Security Principles, explains how data access is handled when information views are nested.

For each view, the following criteria are considered:

- Does the user have SELECT privilege on the column view in the container schema?
- Is the view designed to check analytic privileges?
- Is the user granted analytic privileges for the view?

Unit 8: Security in SAP HANA Modeling

Nested Information Views - Data Access Security Principles



The key rules that govern the access to data are as follows:

- Object privileges

There is no need to grant **SELECT** privileges on the underlying views or tables. The end user only needs to be granted **SELECT** privileges on the top view of the view hierarchy.

- Analytic privileges

The analytic privileges logic is applied through all the view hierarchy.

Whenever the view hierarchy contains at least one view that is checked for analytic privileges but for which the end user has no analytic privilege, no data is retrieved (not authorized).

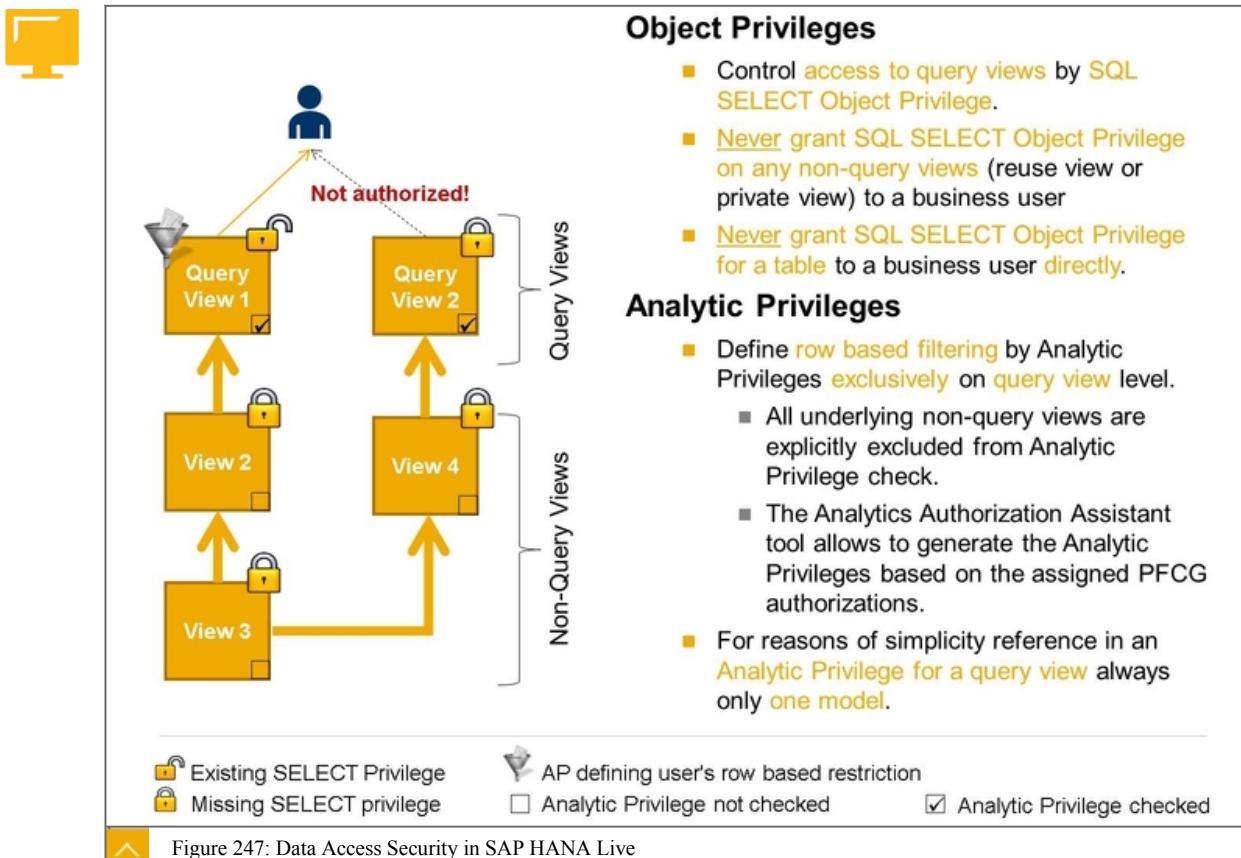


Note:

In the figure, Nested Information Views - Data Access Security Principles, this is the case for View 5, so View 4 will not retrieve any data.

This is also the case for View 6; however, it was not available because the **SELECT** privilege is missing.

Data Access Security in SAP HANA Live



The SAP HANA Live virtual data model relies heavily on views nesting, and in addition, classifies views either as query views (exposed to the end user) or as non-query views (namely, private and reuse Views) which should not be accessed directly by the end user.

The data access security of SAP HANA Live is implemented as follows:

- The end users are granted the object privilege SELECT exclusively on query views. They are never granted any privilege on non-query views or database tables.
- By design, query views perform a check for analytic privileges, and analytic privileges must be defined on these views to enable end users to see the data.

The non-query views do not perform any check for analytic privileges, which is only possible from a security perspective because end-users cannot access them (missing SELECT privileges on the views).

Enabling Access to an External Schema or Another Container

By default, a HDB module is only aware of the objects that are created locally, in the corresponding container schema –for example, tables created with design-time files .hdbcds, calculation views, and so on– and cannot access external data.

This is sufficient when the entire application manages the data persistence inside its own HDB module. However, there are cases where your need to read –or even modify– data located in an external location, such as classic database schema or another container. As an example, all the calculation views you built during this course are using data sources from two main external schemas: EPM_MODEL and TRAINING.

Unit 8: Security in SAP HANA Modeling



LESSON SUMMARY

You should now be able to:

- Define analytic privileges

Unit 8

Lesson 3

Defining Roles

LESSON OVERVIEW

You have already learnt different types of objects that you can create inside the HDB module of an XS Advanced project. For example, Calculation Views, CDS tables for data persistence, Procedures, Analytic Privileges, and so on.

Another object type that you can also create with design-time files, and deploy inside your HDI container, is the Role. As you know, roles are used to distribute in a relevant way authorizations to database objects and data, either located in classic schemas or in the container(s) created as part of the deployment of an HDB Module.

One key benefit of working with design-time roles is that they can be easily transported along with the XSA Project, where pure catalog roles cannot.

Business Example

You work as a Modeler on an SAP HANA Implementation and want to create roles inside your project, in order to easily transport these roles to the production environment and assign them to end-users who will consume your information models.



LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Create a design-time role

Unit 8: Security in SAP HANA Modeling

Design-Time Roles in a HDB Module

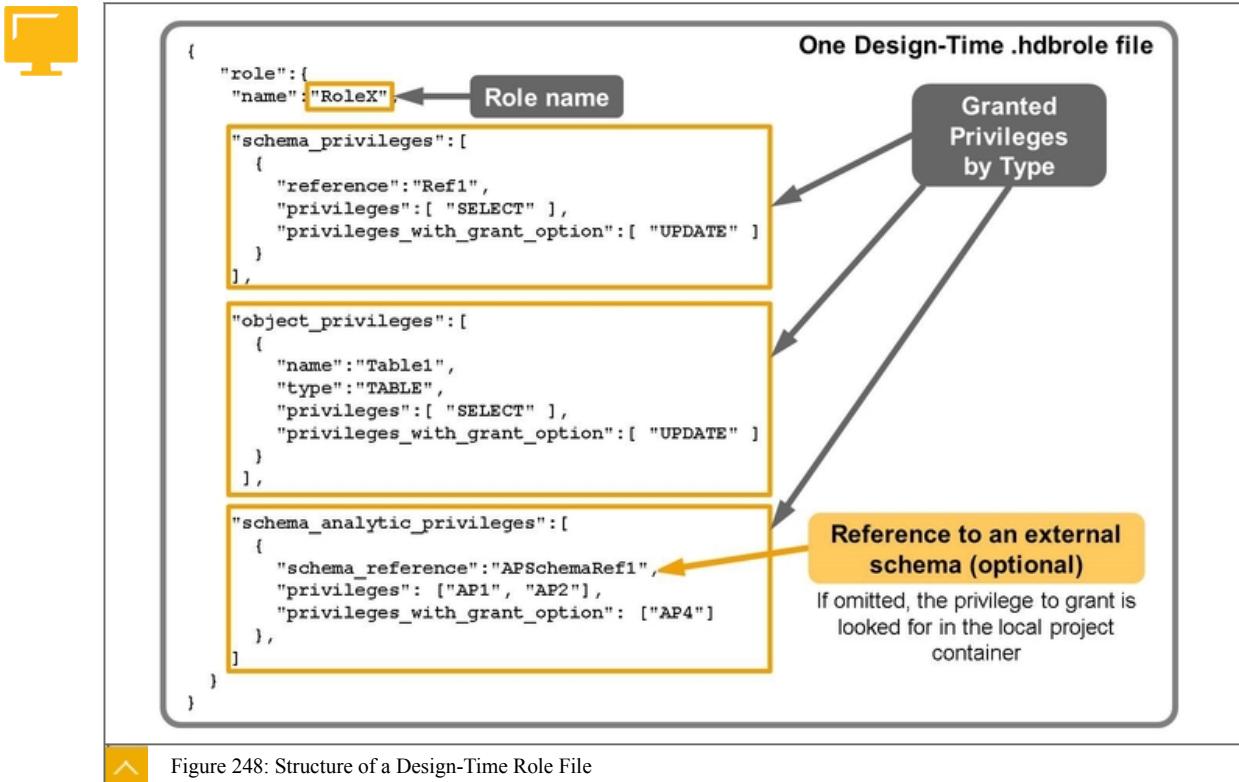


Figure 248: Structure of a Design-Time Role File

The design-time files used to create roles use a Java Simple Object Notation (JSON) syntax, and must have the extension `.hdbrole` in order to be recognized as design-time role files by the HDI build plug-in.



Note:

Each role must be defined in its own .hdbrole design-time file.

It is not possible to create several roles within the same .hdbrole file.

The role ID (including a valid namespace if applicable) must be unique in the HDB module, as for any other object (calculation view, synonym, and so on).

Types of Privileges in a Design-Time Role

A .hdbrole design-time file can include one or several of the following privileges or roles:

Table 23: Types of Privileges in a .hdbrole Design-Time File

Type of Role and Notation in the .hdbrole file	Description
Global Role "global_roles"	Global roles, including a number of built-in roles, defined without a schema, for example: PUBLIC, RESTRICTED_ROLE.
Schema Role (*) "schema_roles"	Schema-local roles defined in your container schema or another schema.

Type of Role and Notation in the .hdbrole file	Description
System Privilege "system_privileges"	System privileges, such as USER ADMIN, ROLE ADMIN.
Privilege on a Schema (*) "schema_privileges"	Privilege that applies to an entire schema, such as SELECT, UPDATE, DELETE, and so on. When granted, these privileges apply to all the objects of the schema (for example all tables, all views).
Object Privilege "object_privileges"	Privileges on objects defined within the local container of your project, such as a table, synonym, procedure or function.
Schema Analytic Privilege (*) "schema_analytic_privileges"	Analytic privileges defined in the local container of your project, or in another container or schema.

(*) For these types of roles or privileges, a reference to a schema must be provided if you want to specify from which external schema the role or analytic privilege must be granted, or to which external schema a schema privilege applies.

If this reference is not specified, these privileges are looked for in the local container (schema) of your HDB module.



Caution:

The .hdbrole file cannot contain references to real schema names, but only logical references that are resolved in another type of design-time file:
the .hdbroleconfig file.

The .hdbroleconfig File

A .hdbroleconfig file must be created to “resolve” the name of external schemas (classic database schemas or other database containers) that are referenced in the .hdbrole file.

The purpose is to maintain the actual name of the external schemas in dedicated files, instead of having many occurrences of the schema names in the .hdbrole files themselves. It makes the maintenance of a project easier when you need to change the references to external schemas.

Unit 8: Security in SAP HANA Modeling

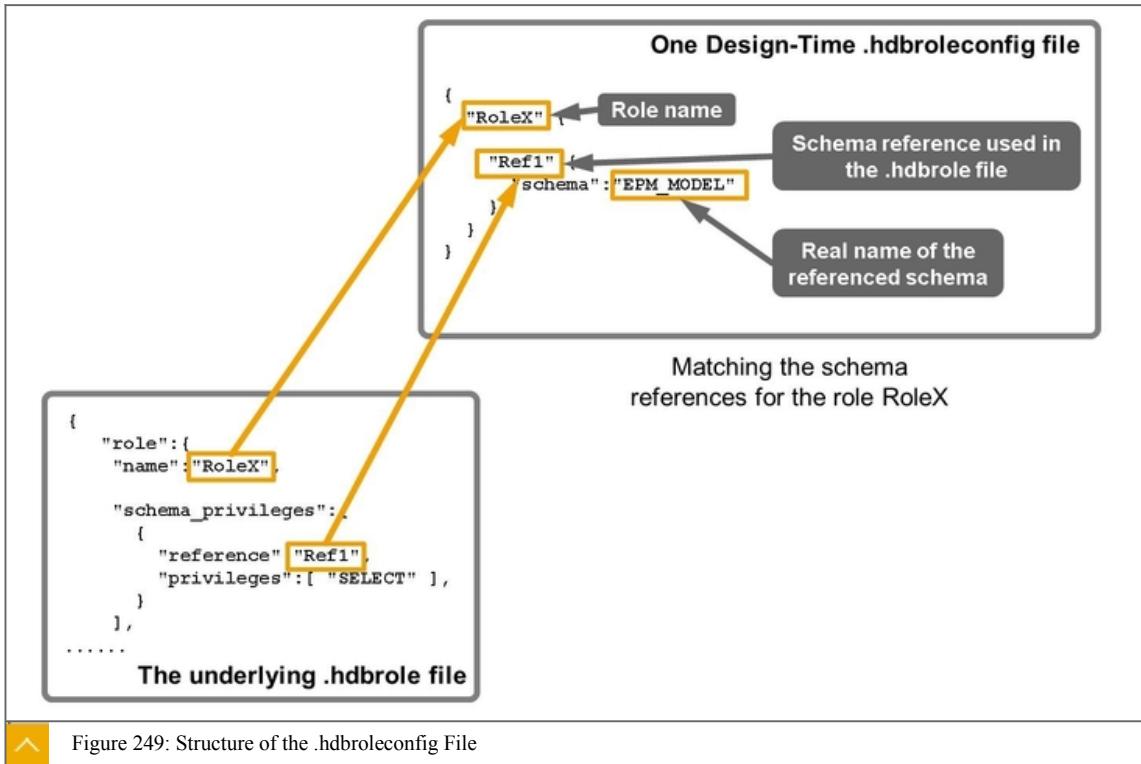


Figure 249: Structure of the .hdbroleconfig File

```

        "type": "TABLE",
        "privileges": [ "SELECT" ],
        "privileges_with_grant_option": [ "UPDATE" ]
    }
],
"schema_analytic_privileges": [
{
    "schema_reference": "APSchemaRef1",
    "privileges": [ "AP1", "AP2" ],
    "privileges_with_grant_option": [ "AP4" ]
},
]
}
}

```

**Note:**

The complete syntax of the .hdbrole design-time file can be found in the SAP HANA Developer Guide for SAP HANA XS Advanced Model, on the SAP Help Portal. Go to <http://help.sap.com/hana>.

Granting a Privilege With Grant Option

When a privilege is granted (to a user or to a role), there are two possible options:

- “Simple” Grant

In turn, the grantee is NOT allowed to grant the privilege to another user or role.

- Grant With Grant Option

In this case, the grantee is allowed to grant the privilege to another user or role. So this approach gives more “power” to the grantee.

Privilege With Grant Option — Example

```

{
  "role": {
    "name": "RoleABC#",
    "schema_analytic_privileges": [
      {
        "privileges": [ "AP1", "AP2" ],
        "privileges_with_grant_option": [ "AP4" ]
      },
    ]
  }
}

```

In the example, RoleABC# includes three Analytic Privileges: AP1, AP2 and AP4.

A user who is granted RoleABC# will be allowed to grant AP4 to another user, but not AP1 or AP2.

**Caution:**

A role name MUST always end with the # sign (for example, **RoleABC#**) if the role includes privileges WITH GRANT OPTION.

Unit 8: Security in SAP HANA Modeling

If this naming rule is not applied, the build of the role fails and the error is reported in the build log.

**Note:**

As you can see from the example, the three analytic privileges AP1, AP2 and AP4 must exist in the local container, because there is no schema reference.

Roles and Privileges Relevant for the Modeler Role

Among the different types of privileges we discussed before, the most relevant for the Modeler role are the following ones:

Privilege or Role	Use Case for the Modeler Role
Schema Analytic Privilege	Assign the analytic privileges defined in your project to roles.
Object Privilege	Control accurately the access to the database objects defined in your local container (schema), such as tables, views, procedures, and so on.
Privilege on a Schema	Grant to a role some schema privileges that were given by the technical user configured in a user-provided service (for cross schema access). For example, SELECT on the external schema.
Schema Role	Group the roles defined in your project together in order to have a relevant nesting of roles.

Default Roles Created in a Container

Whenever you build a HDB module, two roles are created by default in the corresponding container schema. These roles are named as follows:

- <CONTAINER_SCHEMA_NAME>::access_role
- <CONTAINER_SCHEMA_NAME>::external_privileges_role

For example, when you built the HDB module of your HA300## project, during Exercise 1, it created the two default roles mentioned above. You also granted the role HA300##_HDI_CONTAINER::access_role to the TRAINING_ROLE##.

The ::external_privileges_role does not contain any privileges by default.

The ::access_role contains the following privileges:

Privileges Included in the <CONTAINER_SCHEMA_NAME>::access_role

- The following schema privileges on the local container schema: CREATE TEMPORARY TABLE, DELETE, EXECUTE, INSERT, SELECT, SELECT CDS METADATA, UPDATE
- The privileges granted (with grant option) to the technical user(s) that run the user-provided services enabling access to external schemas.

**Caution:**

The <CONTAINER_SCHEMA_NAME>::access_role can be assigned to some developers during the development phase of your project, but it provides a number of authorizations that might not be needed/relevant for the application users. For example, select/update/insert/delete/execute on the entire container schema, which means, all the tables, views, procedures, and so on.

For this reason, it is recommended to define your own roles and adjust the granted privileges to the exact access requirements of each category of users.

Enabling Access to an External Schema

When your application requires access to an external schema, an administrator must define a dedicated user-provided service. A technical user is assigned to this service, and brings its own authorizations to the database objects.

As part of the security implementation in your project, it is necessary to define which of these authorizations will be granted to the different roles. For example, some users might need insert/update/delete privileges on a particular set of tables, while other users only need select privileges.

**Note:**

When creating calculation views, the main authorization you need is only a SELECT privilege on the data sources.

But other types of applications must write data to an external persistence layer, and in that case privileges such as INSERT/UPDATE/DELETE on the corresponding tables must be granted to some users.

The .hdbgrants File

A dedicated file, with extension .hdbgrants, is used to define the set of authorizations that will be given to two specific users, the Object Owner and the Application User.

Unit 8: Security in SAP HANA Modeling

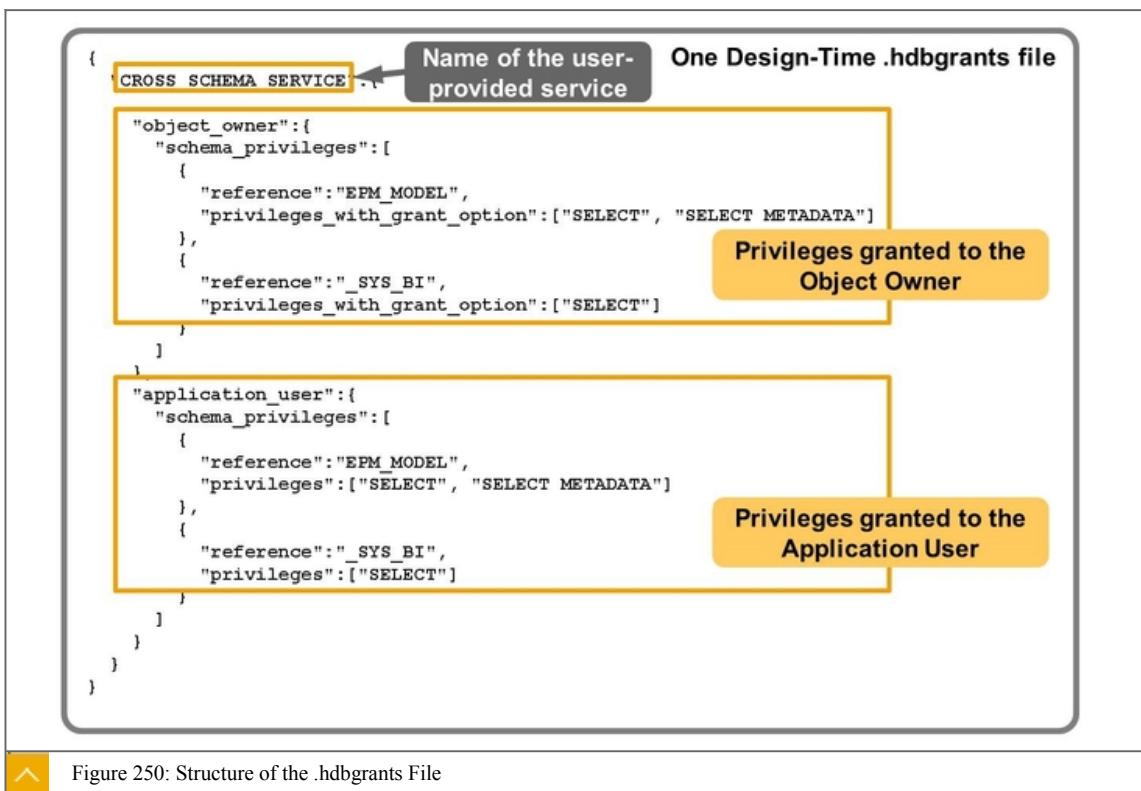


Figure 250: Structure of the .hdbgrants File

The <filename>.hdbgrants file is structured into three levels:

- The name of the user-provided service
- The users to whom the privileges are granted

There are two possible “values”:

- object_owner is the technical user that owns all the objects of the container schema.
- application_user represent the users who are bound to the application modules.

When you create models in the SAP Web IDE, for example, the application user is the technical user SBSS.... that works “on your behalf” inside the container and accesses the external schemas, if any.

- The set of privileges granted

The syntax of this third level is very similar to the syntax of what you find in a .hbrole file.



Note:

A single .hdbgrants file can list authorizations from more than one user-provided services.

It is essential to give the application_user a correct set of authorizations. For example, if a SELECT privilege on an external table is granted to the object_owner , this will allow the creation of a synonym for this table. But if the same SELECT privilege is not granted to the application_user , you won’t be able to display the content of the target table.



LESSON SUMMARY

You should now be able to:

- Create a design-time role

Unit 8

Learning Assessment

1. Which of the following are valid types of privileges in SAP HANA?

Choose the correct answers.

- A** System privileges
- B** Attribute privileges
- C** Package privileges
- D** Application privileges
- E** Data privileges

2. Which scenario is supported to base an analytic privilege on a hierarchy defined on an attribute column?

Choose the correct answer.

- A** Create a classical analytic privilege based on a level hierarchy
- B** Create a classical analytic privilege based on a parent-child hierarchy
- C** Create an SQL analytic privilege based on a level hierarchy
- D** Create an SQL analytic privilege based on a parent-child hierarchy

3. What are the features of dynamic analytic privileges?

Choose the correct answers.

- A** Reuse the same analytic privilege for several users who need to access different data
- B** Filter the data based on a variable entered by the user when they query the view
- C** Change the filter condition in the underlying tables and views

Unit 8

Learning Assessment - Answers

1. Which of the following are valid types of privileges in SAP HANA?

Choose the correct answers.

- A** System privileges
- B** Attribute privileges
- C** Package privileges
- D** Application privileges
- E** Data privileges

2. Which scenario is supported to base an analytic privilege on a hierarchy defined on an attribute column?

Choose the correct answer.

- A** Create a classical analytic privilege based on a level hierarchy
- B** Create a classical analytic privilege based on a parent-child hierarchy
- C** Create an SQL analytic privilege based on a level hierarchy
- D** Create an SQL analytic privilege based on a parent-child hierarchy

3. What are the features of dynamic analytic privileges?

Choose the correct answers.

- A** Reuse the same analytic privilege for several users who need to access different data
- B** Filter the data based on a variable entered by the user when they query the view
- C** Change the filter condition in the underlying tables and views

UNIT 9

Advanced Data Processing

Lesson 1

Introducing Advanced Data Modeling

347

Lesson 2

Optional: Implementing Text Processing

351

Lesson 3

Optional : Working with Geospatial Data

363

Lesson 4

Optional: Developing Predictive Models

371

Lesson 5

Optional: Working with SAP HANA Graph

376

UNIT OBJECTIVES

- Develop awareness of advanced modeling possibilities
- Explain the full-text search capabilities of SAP HANA
- Invoke the text search processes
- Create a Fuzzy Search
- Explain Text Analysis
- Work with geospatial data
- Query geographical data using SQL
- Describe SAP HANA Predictive Analysis capabilities
- Describe graph processing with SAP HANA

Unit 9

Lesson 1

Introducing Advanced Data Modeling



LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Develop awareness of advanced modeling possibilities

— IMPORTANT NOTE —

This lesson provides a short introduction to Advanced Modeling and is part of the **standard delivery of HA300** and summarizes the key information covered in the four remaining lessons in this unit, which are optional. The four lessons in this unit that follow this one are provided for self-study in case you would like to go a little deeper. If you plan to study the four lessons that follow, you can skip this lesson.

Scope of advanced modeling

When we describe modeling in SAP HANA we typically refer to **core modeling** and **advanced modeling**. Core modeling is based on calculation view development that focuses on traditional attributes and measures. Advanced modeling covers four main areas:

- Text
- Spatial
- Predictive
- Graph

Although it is possible to build models in these areas separate from calculation view modeling, quite often modeling in these areas is blended with modeling with calculation views and they work together. For example, for predictive modeling we could develop a data model where the sources are database tables. But in reality, it is more likely that the data sources to the predictive data model will come from a calculation view, which is used to present the data in a suitable manner.

So a solid foundation in core modeling using calculation views is highly recommended before we move to the advanced modeling areas.

Text

Text modeling covers three key areas:

- Text Search
- Text Analysis
- Text Mining

Unit 9: Advanced Data Processing



What types of text processing capabilities are supported?

Search

In addition to string matching, HANA features **full-text search** which works on content stored in tables or exposed via views. Just like searching on the Internet, full-text search finds terms irrespective of the sequence of characters and words.

Text Analysis

Capabilities range from basic tokenization and stemming to more complex semantic analysis in the form of **entity and fact extraction**. Text analysis applies within individual documents and is the foundation for both full-text search and text mining.

Text Mining

Text mining makes semantic determinations about the overall content of documents relative to other documents. Capabilities include **key term identification and document categorization**. Text mining is complementary to text analysis.



Figure 251: Text Processing with SAP HANA

Key terms to become familiar with when describing

Text Search :

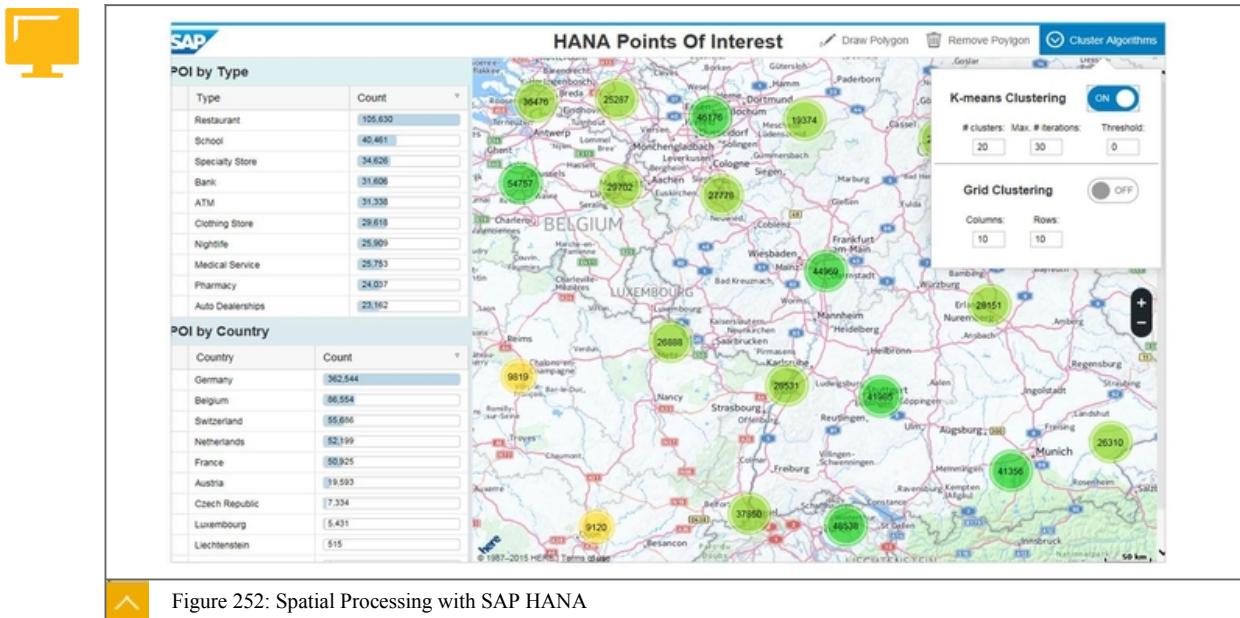
- String search — search through structured data stored in well-defined columns
- Full-text search — search complete unstructured documents
- Fuzzy search — fault tolerant search

Text Analysis is used to extract interesting words or expressions from text. For example countries, companies, dates, and even words that convey how customers feel about our services and products. SAP supply comprehensive dictionaries out of the box but we can also create our own custom dictionaries. Dictionaries alone are not enough, and we need rules that determine interesting combination of words. For example, when we find the word 'plc' we expect the preceding word to be the company name. Dictionaries and rules are combined to form **Text Analysis Configurations**. SAP provide configurations out of the box but we can also develop as many configurations as we need and customize existing ones.

Text Mining enables complex evaluation of documents, for example, to cluster documents that cover similar topics. We can use weighting and scoring methods to determine closeness of fit. We use sophisticated algorithms to determine the closeness of fit. Text mining is a document level analysis and not an individual string analysis.

Spatial

Spatial modeling can enrich traditional analysis by adding information related to the three dimensions of space. Typical uses cases are to add geographic information to business reports so we can easily visualize business performance. But we can go much further and apply analysis, for example by evaluating distance between entities, overlap, within, and more.



SAP HANA supports spatial modeling by providing special data types to allow us to store spatial information in the database as geometries. For example, line, polygon, circle, and so on. But storing data is not enough, we also need to analyze the spatial data. So SAP HANA provides a spatial engine that allows us to perform super-fast in-memory evaluations of complex spatial data. Combined with traditional analysis, adding spatial analysis can provide incredible, additional insight.

Predictive

Predictive modeling in SAP HANA extends traditional analysis with additional information to identify likely outcomes.



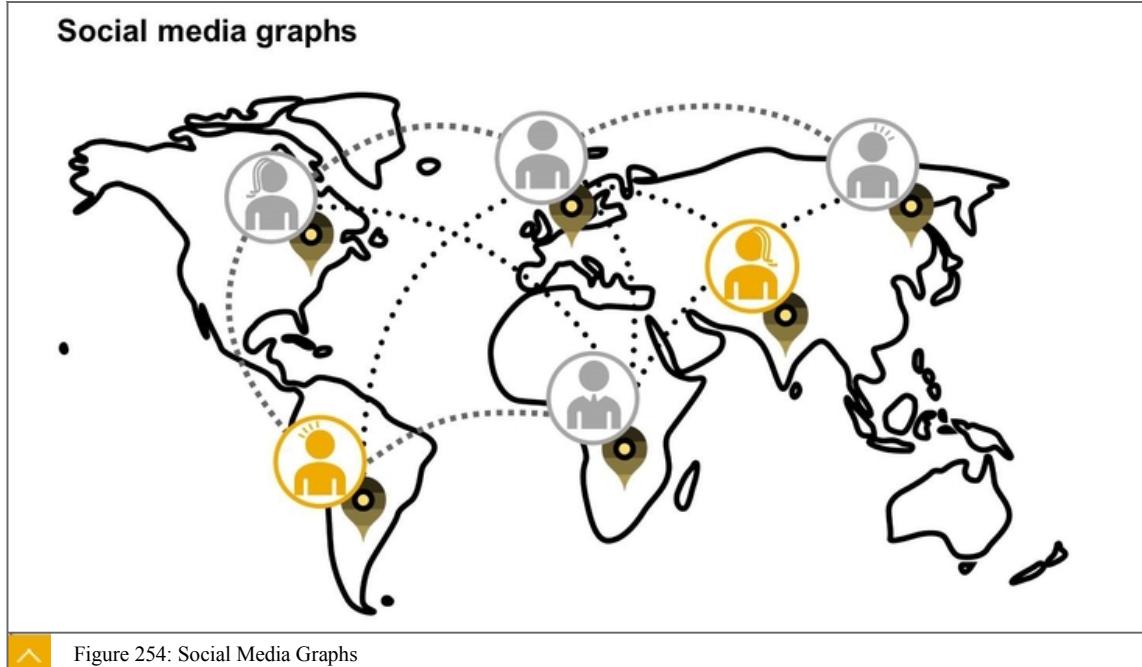
SAP provides us with a library of algorithms and also data preparation functions. This library is known as PAL. We can graphically develop predictive models using flowgraphs which depict

Unit 9: Advanced Data Processing

the sources, the data processing including predictive analysis, and the outputs. SAP HANA provides everything you need to get started with predictive modeling.

Graph

Graph modeling is used when data is best described as a network of highly-connected entities, rather than a traditional set of columns and rows. A simple example is a social network where friends are highly-connected. Storing this information in a traditional table structure would be inefficient.



As with most modeling scenarios, there are two aspects to consider: the data storage or model creation, and the analysis. For graph model creation, SAP HANA provides new SQL syntax to develop the key object known as a **graph workspace**. For analysis, SAP HANA provides enhancement to the calculation view graphical editor to allow us to integrate and query graphs with the calculation view.

Adding graph modeling to core modeling means we can enrich our traditional analysis data with deep information on relationships between entities in order to gain insight.



LESSON SUMMARY

You should now be able to:

- Develop awareness of advanced modeling possibilities

Unit 9

Lesson 2

Optional: Implementing Text Processing

LESSON OVERVIEW

This lesson introduces you to the text search and text analysis capabilities of SAP HANA.

Business Example

Your corporate memory contains a lot of unstructured data in text format. Data quality is also an issue within your business.

To explore this large amount of data more effectively than with traditional text processing approaches, you use the SAP HANA full text search and text analysis features.



LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Explain the full-text search capabilities of SAP HANA
- Invoke the text search processes
- Create a Fuzzy Search
- Explain Text Analysis

— IMPORTANT NOTE —

This optional lesson will be replaced by more comprehensive coverage found in the new SAP HANA course : HA301 — Advanced Modeling. Once this course is available, you should study that content instead. We plan to remove this lesson from the next release of this course. So for this release we continue to include this lesson whilst we build the new HA301 course. We also continue to include this lesson because the questions in the C_HANAIMP_13 certification are based on this content. From C_HANAIMP_14 the questions on this topic will be based on the new HA301 course content.

Please note that this lesson is not expected to be covered in the standard HA300 classroom delivery and is provided for self-study.

Types of Text Processing Available with SAP HANA

Organizations collect unstructured data, such as text and multimedia content, at a phenomenal rate. In fact, approximately 90 percent of world's digital data is unstructured, including social media posts, text messages, e-mails, PDFs, videos, photos, audio files, presentations, web pages and many other kinds of documents. Analyzing such data can be difficult and very costly. Let's consider e-mails: for each e-mail we have some very basic metadata, such as date sent or received, time, sender, recipient, and subject. However, the body of an e-mail is where the really valuable data lies, and this remains unstructured and difficult to examine without someone reading each one.

SAP HANA Text Processing is used to process structured and also unstructured data. The figure, Text Processing with SAP HANA, lists features of the following main types of text process in SAP HANA:

Unit 9: Advanced Data Processing

- Search
- Text Analysis
- Text Mining



What types of text processing capabilities are supported?

Search

In addition to string matching, HANA features **full-text search** which works on content stored in tables or exposed via views. Just like searching on the Internet, full-text search finds terms irrespective of the sequence of characters and words.

Text Analysis

Capabilities range from basic tokenization and stemming to more complex semantic analysis in the form of **entity and fact extraction**. Text analysis applies within individual documents and is the foundation for both full-text search and text mining.

Text Mining

Text mining makes semantic determinations about the overall content of documents relative to other documents. Capabilities include **key term identification and document categorization**. Text mining is complementary to text analysis.



Figure 255: Text Processing with SAP HANA

Full-Text Search

In SAP HANA, you can search on single or multiple columns of almost any visible data type. In addition to standard string search, SAP HANA also supports full-text search.

During a full-text search, the SAP HANA search engine examines both structured text and unstructured text. Structured text is where data values are stored in their own individual column, such as book, author, and publishing date. Unstructured text includes such text as the book itself, where words are not contained in their own separate columns but are stored as a very large and continuous sequence of words in one column of a suitable data type, such as TEXT.

Unlike a string search, the sequence of words and characters used for a full-text search is not critical for finding matches. You start by creating a full-text index on the columns that you wish to process. A full-text index is created by running a pre-process that analyzes the column of text and generates and stores important information that can be used later for search and analysis. This information includes the following:

- Normalization, where the same words are adjusted so that they appear identical, such as removing punctuation, or converting to the same case.
- Tokenization, also known as segmenting, where words are broken into core elements. For example, don't is broken into "do" and "not".
- Stemming, where dictionary base forms are identified. For example, "talking" and "talked" have the same stem: "talk".

- Part-of-speech tagging, where each word is labeled according to its grammatical role in a sentence, such as noun or adjective.

Full-Text Search SQL Syntax

A full-text index is automatically generated on columns that use the data types SHORTTEXT and TEXT. For all other data types you must create the full-text index yourself if you plan to process the column with full-text search. The full-text index can be created using either Core Data Services (CDS) or SQL, but we recommend CDS.

When creating a full-text index, you can decide when the index is updated. When a record in the underlying table is inserted or updated you may choose to have the full-text index updated synchronously or asynchronously.

To perform a full-text search within a SQL SELECT statement, you can use the CONTAINS () function within your WHERE clause. If you specify a column name and a string as parameters to the CONTAINS (), an exact search of the string is performed. You can see an example of the syntax in the figure Full-Text Search SQL Syntax.



- You call the Full Text Search by using the CONTAINS() function in the WHERE-clause of a SELECT statement.
Without the FUZZY option the search will only return results that contain the exact phrase searched for.

```
SELECT
    SCORE() AS score, *
FROM
    documents
WHERE
    CONTAINS(doc_content, 'Driethanolamyn')
ORDER BY
    score DESC;
```



Figure 256: Full-Text Search SQL Syntax

Fuzzy Search

A powerful and popular feature of full-text searching is the fault tolerant search, or fuzzy search. Fuzzy search looks for approximate matches of words and expressions, similar to the way Google works. Instead of returning no results on a slightly misspelled search, as happened with traditional searching tools, fuzzy search almost always returns values even with misspelled words or partial words entered. You can set the sensitivity of the fuzzy search so that you do not return long lists of very weakly matched items and also avoid returning no values at all.

Unit 9: Advanced Data Processing



- Fuzzy Search is a fast and fault-tolerant search feature for SAP HANA. The term "fault-tolerant search" means that a database query returns records even if the search term (the user input) contains additional or missing characters or other types of spelling error.
- Examples of applications include:
 - **Fault-tolerant search in text columns** (for example, html or pdf): Search for documents on 'Driethanolamyn' and find all documents that contain the term 'Triethanolamine'.
 - **Fault-tolerant search in structured database content:** Search for a product called 'coffe krisp biscuit' and find 'Toffee Crisp Biscuits'.
 - **Fault-tolerant check for duplicate records:** Before creating a new customer record in a CRM system, search for similar customer records and verify that there are no duplicates already stored in the system.



Figure 257: What is Fuzzy Search?

The figure, What is Fuzzy Search, lists some typical fuzzy searches. You might also need fuzzy search when looking for duplicate master data where mistyped names are common. You may need to perform linguistic searches. For example, you might want a search for the string **computed** to also return **compute**, **computing**, and **computes**.

You may also have additional search requirements, such as ranking your search results so that the best results are at the top of the list, or providing the whole sentence of a document where the matched word is found so that it is easy to find out how the word is used. SAP HANA offers native search capabilities to support these and other business needs with the full-text search.

Fuzzy searching works out of the box on the following column store data types :

- TEXT
- SHORTTEXT
- VARCHAR
- NVARCHAR
- DATE

Fuzzy search also works right away on any column where a full-text index has been created.

Fuzzy searching can be speeded up by creating an additional fuzzy search index on a column that you wish to search. When manually creating a full-text index, you simply state that you also require the fuzzy search index to be created. For columns which already have a full text generated (TEXT,SHORTTEXT), you simply enable the fuzzy search index for each column where you require fuzzy searching. This is done during table creation. The downside to creating the additional fuzzy search index is the need to allow extra memory.

Fuzzy Search Enablement

To perform a fault-tolerant search, you can specify a fuzziness threshold, which is a number between **0.0** and **1.0**. A threshold of 1.0 corresponds to the requirement of an exact match: the lower the threshold, the more tolerant the search. In your search results, you can also return a SCORE for each result item, where again 1.0 stands for an exact match, and 0.0 indicates it has nothing in common with the search string. Be aware that the fuzziness

threshold and the SCORE are distinct, although related, concepts. It is possible to see a result with a SCORE lower than the fuzziness threshold.



- The fuzzy search algorithm calculates a fuzzy score for each string comparison. The higher the score, the more similar the strings are. A score of 1.0 means the strings are identical. A score of 0.0 means the strings have nothing in common.
- You can request the score in the SELECT statement by using the SCORE() function. You can sort the results of a query by score in descending order to get the best records first (the best record is the record that is most similar to the user input). When a fuzzy search of multiple columns is used in a SELECT statement, the score is returned as an average of the scores of all columns used.

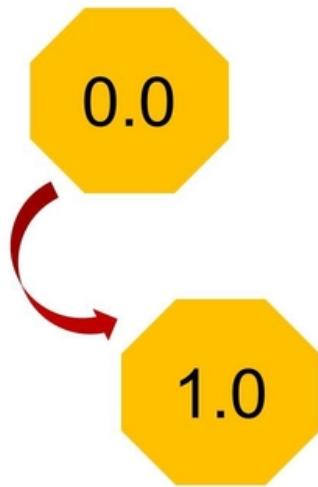


Figure 258: Fuzzy Search Relevance Score

String Type Fuzzy Search

The SCORE computation differs depending on the data type, as outlined in the figures, String Type Fuzzy Search, and Text-Type Fuzzy Search.



String Type Search

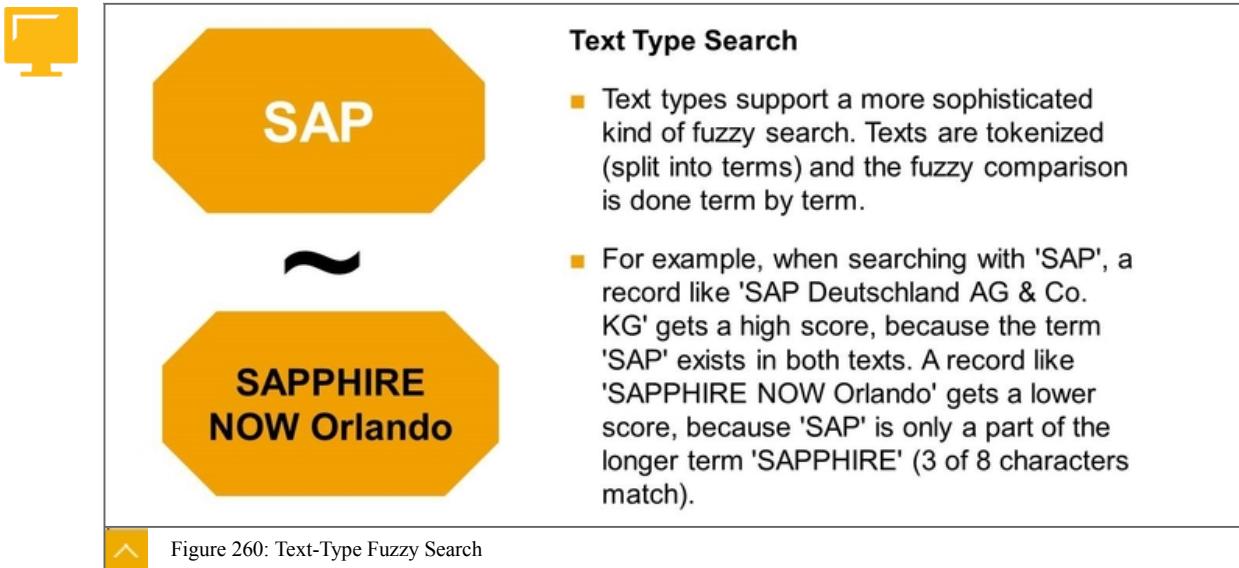
- String types support a basic fuzzy string search. The values of a column are compared with the user input, using the fault-tolerant fuzzy string comparison.
- When working with string types, the fuzzy string comparison always compares the full strings. If searching with 'SAP', for example, a record like 'SAP Deutschland AG & Co. KG' gets a very low score, because only a very small part of the string is equal (3 of 27 characters match).

Figure 259: String Type Fuzzy Search

A string search considers all words in a sequence and not just one word.

Unit 9: Advanced Data Processing

Text-Type Fuzzy Search



A text-type search considers each word, or term, in a sequence as a separate entity.

In the example in the figure, the search term “SAP” is an exact match for one of the terms in one of the records (SAP Deutschland AG and Co. KG) so it gets a high score when we work on each term separately.

Date Type Searches

Fuzzy search on date values checks for date-specific errors, such as the following:

- Date-specific typos. For example, days or dates that have month and day exchanged such as US versus British date format.
- Dates lying within a user-defined maximum distance.

Fuzzy search for dates works with valid dates only. A search with an invalid date does not return any results. The following example does not return any results because **31** is not a valid month: `... WHERE CONTAINS(mydate, '2012-31-01', FUZZY(0.8)) ...`

The maximum allowed distance between dates can be defined using the search option `maxDateDistance`, which defines a number of days.

Fuzzy Search SQL Syntax

As shown in the figure Fuzzy Search SQL Syntax, you can perform a fuzzy search using a `SELECT` statement, by explicitly specifying the option `FUZZY` in the `CONTAINS` function. The fuzziness threshold can be passed as an optional parameter; otherwise, a default value **0.8** is used.



- You can call the fuzzy search by using the CONTAINS() function with the FUZZY() option in the WHERE-clause of a SELECT statement.

SELECT

SCORE() AS score, *

FROM

documents

WHERE

CONTAINS(doc_content, 'Driethanolamyn', FUZZY(0.6))

ORDER BY

score DESC;

- Optionally, the **fuzziness threshold** can manually be set when making the FUZZY() call.
- If set to for example 0.6, no matches lower than 0.6 will be returned. **Default is 0.8.**



Figure 261: Fuzzy Search SQL Syntax

Full-Text Search Syntax: Highlighting and Snippets

The figure Full-Text Search Syntax: Highlighting and Snippets illustrates how to return search results with specific terms or snippets highlighted.



Highlighting – Mark the search terms in the text

```
SELECT HIGHLIGHTED(DEFECT), CAMP_NO FROM
"_SYS_BIC"."nhtsa/J_RECALS_S" WHERE CONTAINS(DEFECT, 'throttle');
```

HIGHLIGHTED(DEFECT)	CAMP_NO
inner and outer carburetor throttle return springs may have been misinstalled possibly resulting in spring failure this can prevent the throttle from retu...	79V074000
under a combination of circumstances in cold weather icing may prevent return of the throttle butterfly to the idle position	79V020000
due to an improperly seated bushing in the carburetor secondary barrel body a portion of the spiral return spring of the throttle plate opener may restrict the fre...	78V262000
on the involved vehicles the vacuum line connecting the carburetor and the thermo valve may have been improperly installed this condition may prevent the thrott...	77V214000
the possibility exists that the aluminum engine rear support brackets may fail due to metal fatigue this type of failure would allow the engine and transmission assembly t...	78V170000

Snippets – Returns a short snippet of the text with the search term highlighted

```
SELECT SNIPPETS(DEFECT), CAMP_NO FROM
"_SYS_BIC"."nhtsa/J_RECALS_S" WHERE CONTAINS(DEFECT, 'throttle');
```

SNIPPETS(DEFECT)	CAMP_NO
inner and outer carburetor throttle return springs may have been misinstalled possibly ...	79V074000
... icing may prevent return of the throttle butterfly to the idle position	79V020000
... the spiral return spring of the throttle plate opener may restrict the ...	78V262000
... installed this condition may prevent the throttle linkage plate from returning to ...	77V214000
... dislocate downward this will result in throttle control loss with the engine ...	78V170000



Figure 262: Full-Text Search Syntax: Highlighting and Snippets

Unit 9: Advanced Data Processing

Further Full Text Search Syntax

**Why found – Information about where a search term was found**

```
SELECT WHY_FOUND(), CAMP_NO FROM "_SYS_BIC"."nhtsa/J_RECALS_S"
WHERE CONTAINS(*, 'throttle engine')
```

WHY_FOUND	CAMP_NO
<TERM>engine</TERM><FOUND><DEFECT </FOUND><TERM>throttle</TERM><FOUND><DEFECT </FOUND><TERM>CONSEQUENCE</TERM><FOUND><DEFECT </FOUND>	05V172000
<TERM>engines</TERM><FOUND><DEFECT </FOUND><TERM>throttle</TERM><FOUND><DEFECT </FOUND>	10V015000
<TERM>engines</TERM><FOUND><DEFECT </FOUND><TERM>CONSEQUENCE</TERM><FOUND><DEFECT </FOUND><TERM>throttle</TERM><FOUND><DEFECT </FOUND>	09V471000
<TERM>engines</TERM><FOUND><DEFECT </FOUND><TERM>throttle</TERM><FOUND><DEFECT </FOUND><TERM>CONSEQUENCE</TERM><FOUND><DEFECT </FOUND>	10V327000
<TERM>engines</TERM><FOUND><DEFECT </FOUND><TERM>throttle</TERM><FOUND><DEFECT </FOUND><TERM>CONSEQUENCE</TERM><FOUND><DEFECT </FOUND>	09V360000
<TERM>engines</TERM><FOUND><DEFECT </FOUND><TERM>throttle</TERM><FOUND><DEFECT </FOUND><TERM>CONSEQUENCE</TERM><FOUND><DEFECT </FOUND>	07E053000
<TERM>engines</TERM><FOUND><DEFECT </FOUND><TERM>throttle</TERM><FOUND><DEFECT </FOUND><TERM>CONSEQUENCE</TERM><FOUND><DEFECT </FOUND>	03E038000

Faceted Search – Aggregated count for a column

```
SELECT MAKE_FACET, COUNT(*) AS C FROM
"_SYS_BIC"."nhtsa/J_RECALS_S" WHERE CONTAINS(DEFECT, 'throttle
engine') GROUP BY MAKE_FACET ORDER BY C DESC;
```

MAKE_FACET	C
1 NISSAN	424
2 CHEVROLET	424
3 FORD	300
4 MACK	276
5 AIR BYPASS VALVE	216

Figure 263: Why Found and Faceted Search

Overview of Text Analysis

Text analysis is the extraction of interesting entities and facts from within text.

An entity can be any word that describes a distinct item, but is often a proper noun or name, such as a person's name, country name, or product name. Entities can also be well-known words such as currency or dates.

Facts can be divided into three areas:

- Public-sector facts — facts from the general world that are known to all people such as travel events. For example, Tina Sparkle traveled by train from London to Birmingham on 4th Feb 2016.
- Enterprise facts — facts from within companies such as people leaving or joining, or company mergers, acquisitions, or product launches.
- Sentiment facts — facts that relate to how people feel about products and services.

SAP HANA Text Analysis is a feature enabled with the full-text index to discover and classify entities in your documents. Technically, it is a set of Python-based scripts.

Text Analysis provides a large number of predefined common entity and fact types such as names, countries, and companies. Text Analysis also provides rules for how these entities are identified in strings. For example, if we detect the word 'Inc.' then the preceding word must be a company name. These entities and rules are available for many industries, and many languages, and can easily be customized.

The language modules included with the software contain system dictionaries and provide an extensive set of predefined entity types, such as a list of all country names.

Extraction classifies each entity extracted by entity type and presents this metadata in a standardized format. You can also customize the text analysis process and define your own entity types, such as course codes in a training company. Imagine being able to search all the tweets that contained the word ‘HA300’. That would provide a great marketing lead.

Text Analysis Configurations provide :

- **Dictionaries** that provide the entity values and entity types (for example, Rupert: Male First Name)
- **Extraction Rules** that provide rules (patterns) for extracting complex entities and relationships using text analysis (for example, look for person’s title in front of name)

Example of Text Analysis Configurations

SAP HANA Text Analysis includes a number of predefined configurations. You can use any of these, or create your own custom text analysis configurations to apply your own text analysis extraction dictionaries and extraction rules.



■ **Named Entity Recognition**

Identification of persons, companies, dates etc. in “text”

```
<PERSON>John</PERSON> sold
<CURRENCY>$300</CURRENCY> shares of
<ORGANIZATION>Acme Corp.</ORGANIZATION> in
<DATE>May 2013</DATE>
```

■ **Sentiment Analysis**

Extract subjective information like positive or negative sentiments, problems etc.



Figure 264: Example of Text Analysis Configurations

As the figure Example of Text Analysis Configurations shows, standard predefined configurations include the following:

- Common entities, such as organizations, persons, countries, dates, or measures.
- Fact extraction, such as a new product launch by Company ABC Inc. in December 2016. This is a product launch fact.
- Specialized extraction content, such as Marketing (“Voice of the Customer”) where expected customer sentiments and feedback can be identified using the language that the customer uses.

Text Analysis Fact Types

To set up a sentiment analysis, you create a FULLTEXT index on a column table referring specifically to the column that stores the text to be analyzed (such as a tweet) with a predefined configuration of sentiments, such as ‘Voice of Customer’ that defines the possible dictionary of sentiments and whether they are positive or negative sentiments and how strong.

Of particular interest to most organizations is the extraction of entities and facts relating to the enterprise.

The following major fact types are classified:

Unit 9: Advanced Data Processing

- Membership Information — information about a person's affiliations
- Management Changes — information about joiners, leavers, and movers
- Product Releases — information about product launches, retirements, and so on
- Mergers and Acquisitions — information about company mergers and acquisitions
- Organizational Information — Information about the company location, structure, contacts, and so on

The results of a text analysis are stored in a table and can be consumed through all supported HANA scenarios. For example, using calculation views, in R language scripts with its functions from the Predictive Analytic Library, or by using the SAP HANA Information Access toolkit for HTML5, after building a search user interface.

Text Analysis is currently available in 33 languages.

Grammatical Role Analysis (GRA)

SAP HANA SPS11 introduced a text analysis feature called Grammatical Role Analysis (GRA). GRA can deeply analyze key elements of a sentence to tag the subject, the verb, and an object. This is known as a triple. A triple contains a dependent (the argument), a governor (the verb) and a dependency type (for example, the subject). You use GRA when you need to apply linguistic analysis that goes beyond single-word analysis, and understand the functional relationship between two words in a sentence. Sometimes you need to analyze how words in a sentence actually work together in order to truly identify the context and meaning and therefore extract key information.

GRA is able to identify the role that each word plays and how each word relates to the other words in a sentence, as shown in the figure Grammatical Role Analysis.



An Example of Grammatical Role Output

An example of the grammatical roles that are output for a sentence.

Input sentence:

Oracle was rumoured to buy marketing-software maker Responsys Inc. for \$1.5 billion.

The resulting six token output is shown in the table:

Token	Dependent	Role	Governor
1	Oracle	Subject	3
2	Oracle	Subject	4
3	rumored	Root/MainVerb/Passive	(none)
4	buy	MainVerb/Active	(none)
5	marketing-software maker Responsys Inc	DirectObject	4
6	\$1.5 billion	OtherObject/for	4



Figure 265: Grammatical Role Analysis



Note:
Currently GRA is only supported in English language.

Text Mining

In contrast to text analysis, text mining operates at the document level. The key use case for text mining is to classify streams of documents, such as newspaper articles or blogs, or support incidents, and classify them by adding metadata. Text mining allows you to rank documents according to importance or cluster documents that discuss closely related topics.

Imagine an oncologist wants to quickly locate all research papers of the last 12 months that discuss the use of the drug Avastin in cancer treatments, specifically where the side effects are described when the drug is mixed with the Cisplatin chemotherapy drug. They want to cluster the resulting documents by type of side-effect. With SAP HANA Text Mining you can trawl through the documents in real time to identify the documents where there is a heavy weighting towards the required subject. Text Mining uses sophisticated algorithms to produce a matrix of documents and terms.



Note:
Since SPS12, it is possible to interrogate the results matrix so a data scientist can carry out post-processing analysis, answering such questions as why these particular documents were found. The Term Document Matrix is a system table where the results are stored and are exposed to developers. It can be queried using SQL. The table is \$TM_MATRIX.

Syntax Example



```
CREATE FULLTEXT INDEX myIndexName ON
myTable(myColumn)
TEXT MINING ON;
```

Figure 266: Syntax Example

Enabling the Text Mining functionality can be done column by column. It is an additional clause to the syntax that creates the full-text index, as seen in the figure Syntax Example.

SQL Extensions for Text Mining

Since SPS10, in addition to the XS-based Text Mining API, SAP HANA provides SQL functions to retrieve text mining statistics and enable data categorization. These extensions are listed in the table Example of Text Mining SQL Functions.

Table 24: Example of Text Mining SQL Functions

Function	Retrieved Data
TM_GET RELATED TERMS	top-ranked terms related to a term
TM_GET RELEVANT TERMS	top-ranked relevant terms (key phrases) that describe a document
TM_GET SUGGESTED TERMS	top-ranked terms matching an initial substring

Unit 9: Advanced Data Processing

Function	Retrieved Data
TM_GET_RELATED_DOCUMENTS	top-ranked documents related to a document
TM_GET_RELEVANT_DOCUMENTS	top-ranked documents relevant to a term
TM_CATEGORIZE_KNN	classification of an input document according to taxonomies (sets of categories), using the KNN (k nearest neighbor) method

Text Mining SQL Functions Example



```

select T.RANK, T.TERM, T.DOCUMENT_FREQUENCY, T.SCORE
  FROM TM_GET_SUGGESTED_TERMS (
    TERM 'john'
    SEARCH DISTINCT "content"
      FROM "myschema"."news"
    RETURN
      TOP 5
  ) AS T
 WHERE T.SCORE > 0.1 and T.TERM_FREQUENCY > 5

```

Rank	Term	Normalized Term	Term Type	Term Frequency	Document Frequency	Score
1	John	john	Proper	25	11	0.86
2	John Doe	john doe	Proper	12	8	0.24
3	John Miller	john miller	Proper	5	3	0.21
4	Johnny	johnny	Proper	7	7	0.15
5	Johnson and Sons	johson and sons	Organization	5	2	0.09



Figure 267: Text Mining SQL Functions Example

In the figure, Text Mining SQL Functions Example, the SQL code returns the top-ranked terms “matching” the text **john** from a news table.



LESSON SUMMARY

You should now be able to:

- Explain the full-text search capabilities of SAP HANA
- Invoke the text search processes
- Create a Fuzzy Search
- Explain Text Analysis

Unit 9

Lesson 3

Optional : Working with Geospatial Data

LESSON OVERVIEW

This lesson gives you an introduction about how to store, process, and publish geospatial data in SAP HANA.

Business Example

You want to calculate the distances between your customers and your stores.

Typically, locations or addresses in a database are stored in text fields, making it difficult to pinpoint the exact geographical locations.

To calculate distances accurately, you decide to store the geographical locations of your customers and stores, using the SAP HANA geospatial model and its related functions.



LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Work with geospatial data
- Query geographical data using SQL

— IMPORTANT NOTE —

This optional lesson will be replaced by more comprehensive coverage found in the new SAP HANA course: HA301 — Advanced Modeling. Once this course is available, you should study that content instead. We plan to remove this lesson from the next release of this course. So for this release we continue to include this lesson whilst we build the new HA301 course. We also continue to include this lesson because the questions in the C_HANAIMP_13 certification are based on this content. From C_HANAIMP_14 the questions on this topic will be based on the new HA301 course content.

Please note that this lesson is not expected to be covered in the standard HA300 classroom delivery and is provided for self-study.

Spatial Data in SAP HANA

Column-oriented data structures and in-memory computing have developed into powerful components of today's enterprise applications. While the focus of these developments has primarily been on analyzing sales data, the potential for using these technologies to analyze geographic information is significant. Support for the processing of spatial data represents a key evolution in SAP HANA.

Using spatial features enables you to create database queries that answer questions such as the following:

- Which of my customers are close to a certain store?
- What is the distance between a port and the furthest customer?

Unit 9: Advanced Data Processing

- How many customers are within my sales region?

Spatial Processing with SAP HANA

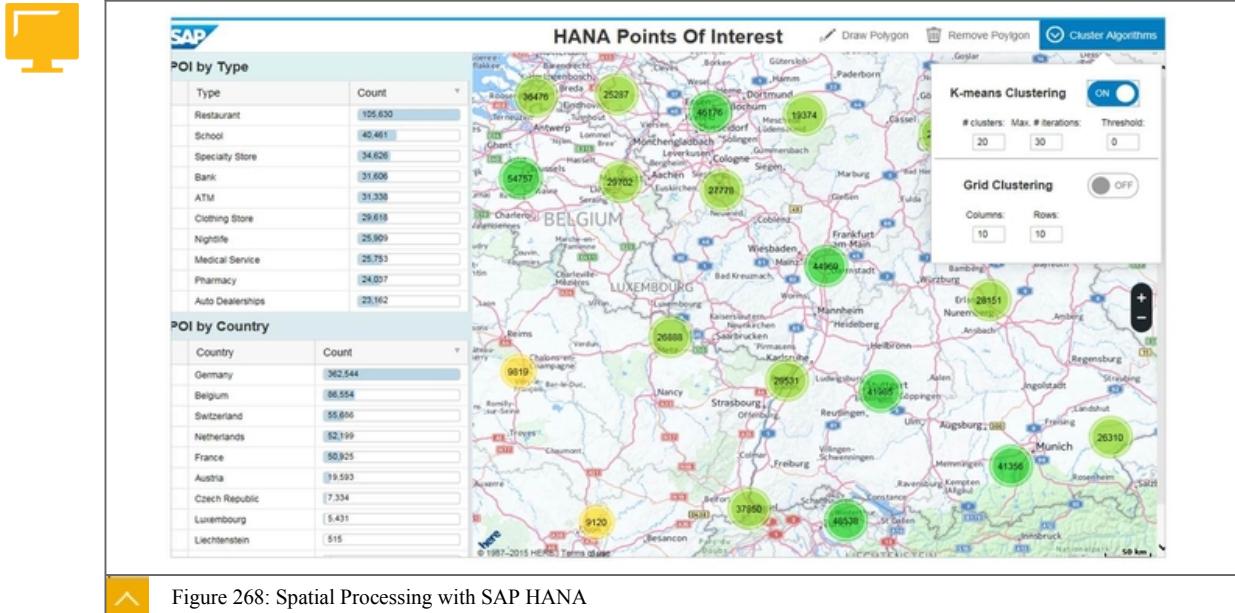


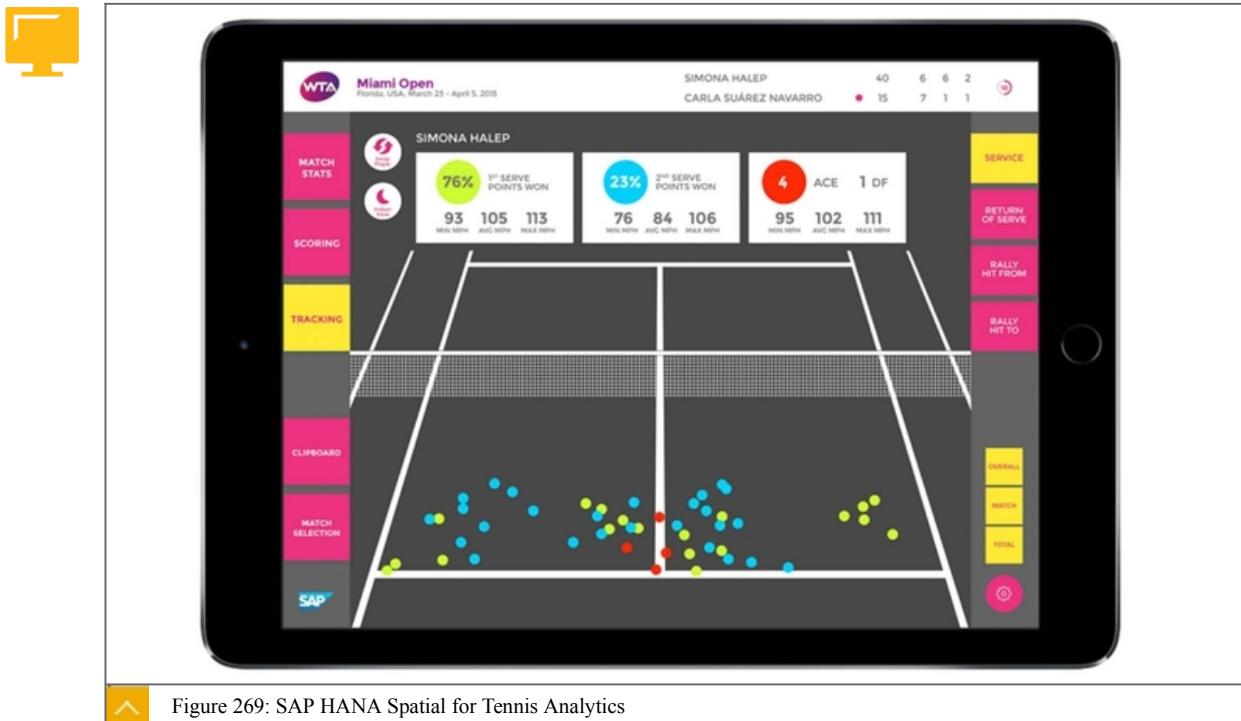
Figure 268: Spatial Processing with SAP HANA

As the figure, Spatial Processing with SAP HANA, illustrates, with SAP HANA Spatial you visualize entities and their relationships to gain insight. When you add geographic information to your traditional business data you can really start to gain deeper insight compared to traditional reports. For example, imagine a dashboard where you can observe a cluster of stores where performance is poor. You then overlay a rail network and can immediately see that the stores are not well connected to customers and there are potential bottlenecks in the transport network. You can zoom in and out of the visualization to gain more detail insight as required. Adding spatial analysis adds the dimension that is missing from many business reports.

Although we focus in this lesson on geographic (map) analysis, SAP HANA Spatial can add huge value wherever any aspect of space is relevant, such as medical applications, where size, density, distance, or movement comparisons in analysis of MRI or PET scans can provide potentially life saving information to consultants.

Another exciting area for the application of SAP HANA Spatial is in sports analytics. With the real-time capabilities of SAP HANA, we can deliver performance related data to the coaches to support the development of players.

SAP HANA Spatial for Tennis Analytics



As the figure, SAP HANA for Tennis Analytics, illustrates, combined with wearable and mobile sensor technology, SAP HANA Spatial can track the player and the ball in relation to the playing field. As well as providing information to coaches, the same data can be refined and relayed to sports fans in real time to provide a truly connected experience.

Spatial Features as Part of the SAP HANA Platform

SAP HANA natively supports many types of spatial data, and SAP has introduced many new dedicated data types for this purpose. SAP HANA also includes a dedicated Spatial Engine to process all spatial queries for high performance.

Spatial Data Definition and Storage

As the figure Spatial Data Types illustrates, spatial data can be represented as 2D or 3D geometries in the form of points, lines, and polygons.

Unit 9: Advanced Data Processing



Spatial data types are stored in tables in the same way as other datatypes.

You can add geographical datatypes to a table using SQL or using the graphical interface.

Table Name: COUNTRY			
Columns Indexes Further Properties Runtime Information			
Name	SQL Data Type	Dim	Column Store Data Type
1 SIMPLIFIED	CHAR	1	FIXEDSTRING
2 ZOOMLEVEL	TINYINT	3	INT
3 ADMINLEVEL	TINYINT	3	INT
4 RDF_CARTO_ID	INTEGER		
5 DESCRIPTION	VARCHAR	255	STRING
6 ISO_COUNTRY_CODE	CHAR	3	FIXEDSTRING
7 SHAPE	ST_GEOOMETRY	16	GEOMETRY

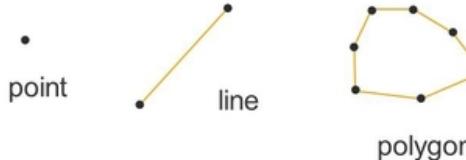


Table Name: POI			
Columns Indexes Further Properties Runtime Information			
Name	SQL Data Type	Dim	Column Store Data Type
1 TYPE	VARCHAR	255	STRING
2 POINT	ST_POINT	16	POINT
3 NAME	VARCHAR	255	STRING
4 STREET	VARCHAR	255	STRING
5 CITY	VARCHAR	255	STRING
6 COUNTRY	VARCHAR	255	STRING
7 RATING	VARCHAR	255	STRING
8 FACTOR	INTEGER		INT



Figure 270: Spatial Data Types

SAP HANA Spatial Data Types

The table Spatial Data Types describes the data types used for spatial data.



Table 25: Spatial Data Types

Data Type	Use Case
ST_GEOMETRY	The geometry type is the supertype for all supported spatial data types.
ST_POINT	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_POLYGON	A polygon defines a region of space. In GIS data, polygons are typically used to represent territories (countries, towns, states).
ST_LINESTRING	A linestring is geometry with a length, but without any area. In GIS data, linestrings are typically used to represent rivers, roads, or delivery routes.
ST_MULTILINE	A multilinestring is a collection of linestrings. In GIS data, multilinestrings are often used to represent geographic features like a highway network.
ST_MULTIPOINT	A multipoint is a collection of individual points.
ST_MULTIPOYGON	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).

Data Type	Use Case
ST_CIRCULARSTRING	A circularstring uses circular line segments between control points. Three points (start, intermediary, and end) determine a circularstring. You can then add another circular line segment with an additional intermediary and end point.

SAP HANA Spatial is compatible with all the standardized formats for spatial data:

- Well-Known Binary (WKB): an ASCII format maintained by Open Geospatial Consortium (OGC)
- Well-Known Text (WKT): a binary format maintained by Open Geospatial Consortium (OGC)
- GeoJSON: a subset of the popular JSON language
- ESRI shapefiles: a very popular geospatial vector data format developed by Environmental System Research Institute, Inc and used already by many organizations
- Scalable Vector Graphic (SVG) files — an XML based format maintained by the World Wide Web Consortium (W3C).

Spatial Methods

To use geographical locations in a meaningful way, you often have to compare spatial geometries.

SAP HANA provides spatial functions that can query spatial columns using SQL.

There are a large number of spatial methods available to handle all types of spatial queries. The table Spatial Data Comparison Function lists examples of some spatial methods, their syntax, and when they can be used.

The table Spatial Data Comparison Functions describes the functions and their usage.



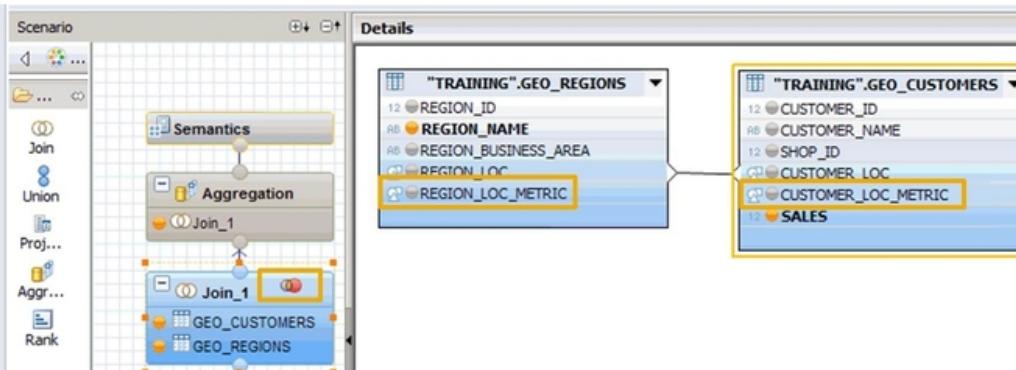
Table 26: Spatial Data Comparison Functions

Spatial Function and Syntax	Usage
[geo1].ST_EQUALS([geo2])	Tests if a ST_GEOMETRY value is spatially equal to another ST_GEOMETRY value.
[geo1].ST_CONTAINS([geo2]) and [geo1].ST_COVERS([geo2])	Tests if a geometry value spatially contains another geometry value.*
[geo1].ST_WITHIN([geo2]) and [geo1].ST_COVERED-BY([geo2])	Tests if a geometry value is spatially contained within another geometry value.*
[geo1].ST_WITHIN_DISTANCE([geo2], [distance])	Test if two geometries are within a specified distance of each other.
[geo1].ST_DISTANCE([geo2])	Returns the smallest distance between specified geometry values.

Unit 9: Advanced Data Processing

Implementing Spatial Joins in Calculation Views

 Spatial joins can be utilized in graphical Calculation Views



It is also possible to call Spatial functions using SQL Script.
Additionally, Core Data Services (CDS) supports spatial data types and functions

 Figure 271: Spatial Functions in Calculation Views and CDS

As shown in the figure, Spatial Functions in Calculation Views and CDS, SAP HANA enables you to use spatial joins directly in Calculation Views. This allows you to define easily a semantic comparison of two geometries. For example, whether a point contained within, or covered by, a polygon. The CDS also support spatial data types and functions.

Spatial Joins in Calculation Views

 A spatial data join behaves as a spatial function that compares two geographies.
In this example, the Calculation View will match customer and region only if the customer's location is included in the region (Within).

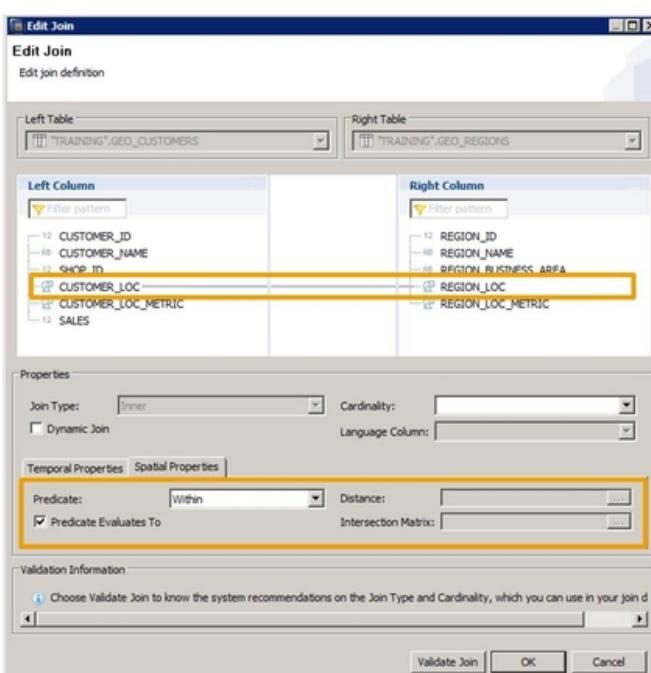
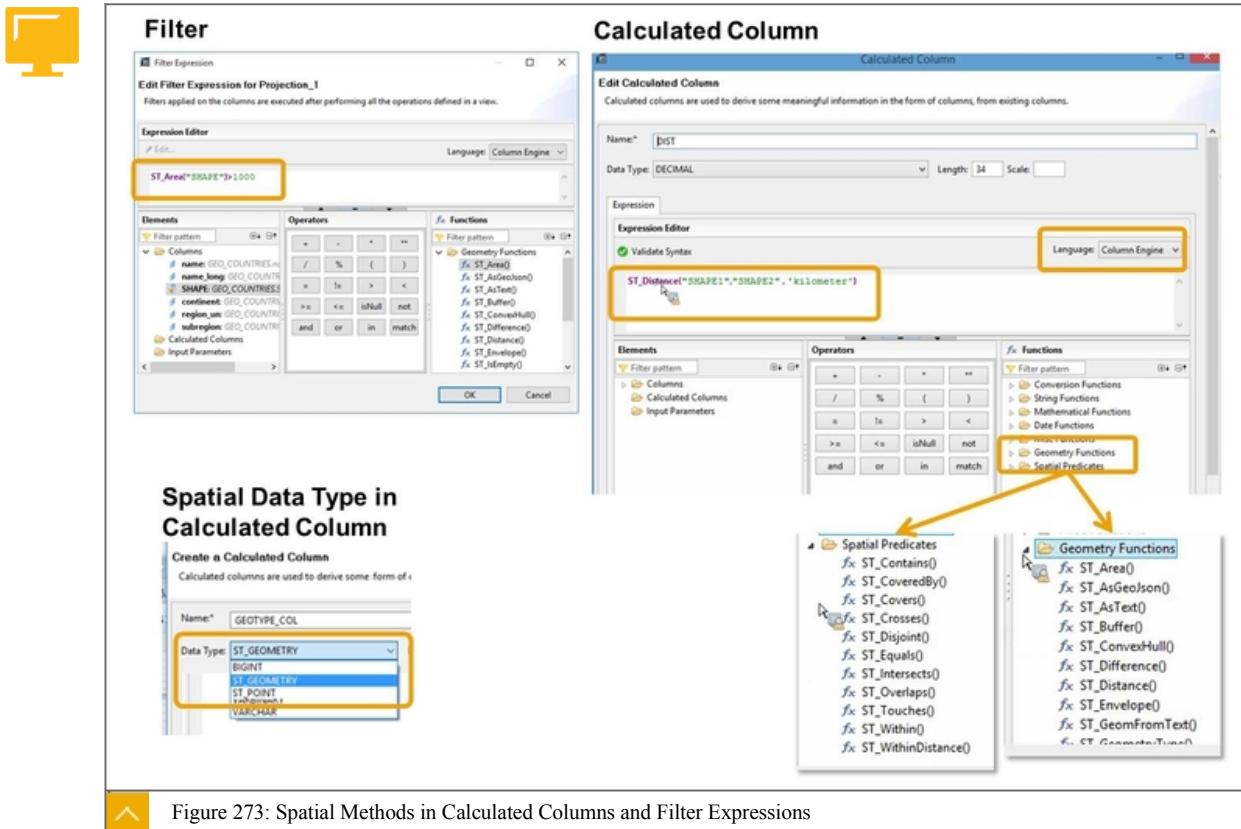


 Figure 272: Spatial Joins in Calculation Views

The figure, Spatial Joins in Calculation Views, illustrates an example of a spatial join in a calculation view.

Creating Calculated Columns and Applying Filters Using Spatial Expressions



The figure Spatial Methods in Calculated Columns and Filter Expressions illustrates an example of creating calculated columns of spatial data types **ST_GeOMETRY** and **ST_POINT**.

In addition to spatial predicates such as **ST_Contains()**, **ST_Intersects()**, calculated columns also support spatial methods, for example, **ST_Area()**, **ST_Intersection()**.

You can also use spatial expressions in filter expressions. For example, you can filter geometries based on their area.

Spatial Clustering

SAP HANA supports algorithms to group spatial points into set of clusters. The figure Spatial Clustering Overview gives two use cases for spatial clustering.

Unit 9: Advanced Data Processing



Spatial Clustering groups a set of points, that meets certain criteria, into clusters

- A cluster is a partition of the original set of points
- Spatial clustering essentially allows geographical information to be used to group data

Example use cases:

Retail Store Location

- Retailer wants to establish two chains of stores
- One luxury brand, one discount
- Has location & income data of 120M US households

Goal:

- Locate areas of high income and low income households to strategically locate the two different chains

Vending Outlets

- Retailer has vending machines located across the US
- Has location and revenue generated of each machine

Goal:

- Analyze lucrative and non-lucrative locations
- Easily visualize these locations
- Do ad-hoc analysis – e.g. zoom in and get specific machine details



Figure 274: Spatial Clustering Overview



LESSON SUMMARY

You should now be able to:

- Work with geospatial data
- Query geographical data using SQL

Unit 9

Lesson 4

Optional: Developing Predictive Models

LESSON OVERVIEW

In addition to reviewing historical information, successful organizations look forward. They do this to gain insight that helps them to take advantage of opportunities and also avoid risk. In this lesson, we will get you started on predictive modeling with SAP HANA using the Predictive Analysis Library (PAL) functions.



LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Describe SAP HANA Predictive Analysis capabilities

— IMPORTANT NOTE —

This optional lesson will be replaced by more comprehensive coverage found in the new SAP HANA course: HA301 — Advanced Modeling. Once this course is available, you should study that content instead. We plan to remove this lesson from the next release of this course. So for this release we continue to include this lesson whilst we build the new HA301 course. We also continue to include this lesson because the questions in the C_HANAIMP_13 certification are based on this content. From C_HANAIMP_14 the questions on this topic will be based on the new HA301 course content.

Please note that this lesson is not expected to be covered in the standard HA300 classroom delivery and is provided for self-study.

Predictive Models in SAP HANA PAL

Predictive analysis is an area of data mining that deals with extracting information from data, deeply analyzing it, and using the findings to predict trends and behavior patterns and likely outcomes. Often the unknown event of interest is in the future, but predictive analysis can be applied to any type of unknown, whether in the past, present, or future.

Predictive analysis is gaining momentum in business applications as organizations shift from rearward-facing analysis to forward-facing analysis. The figure Predictive Analysis Examples lists some of the many applications of predictive analysis in any organization or industry to help identify opportunities or avoid risk.

Unit 9: Advanced Data Processing



Healthcare Predict likelihood of disease to begin early treatment; identify clinical trial outcomes.	CRM Sales Identify revenue forecast based on customer opportunities and pipelines execution.
Banking Identify key behaviours of customers likely to leave the bank; improve credit risk analysis.	Utilities Forecast demand and usage for seasonal operations; provide anticipated resources.
CRM Marketing Identify potential leads among existing customers and intelligently market to them based on individual preferences and histories	Government Predict community movement and trends that affect taxing districts; anticipate revenue.
Retail Intelligent selection of store locations based on demographics; inventory planning.	Telco Forecast demand on system load for capacity planning and customer scale.

Figure 275: Predictive Analysis Examples

Predictive analysis used to be regarded as a specialist topic that was the domain of experts, such as data scientists. Today, predictive analytics is a core part of the business intelligence (BI) picture. When integrated with historical analysis and combined with text and spatial processing, predictive analytics can support the development of sophisticated applications.

Predictive Analysis Library (PAL)

SAP has developed a library of predefined predictive algorithms and functions and organized them into a library called the Predictive Analysis Library (PAL). The algorithms and functions are parameter driven and just need inputs from your data flow to produce results.

Although many of the PAL functions call complex algorithms, you do not need a complete understanding of these algorithms and the math they contain. Many of the algorithms are well known, industry standard, and publicly available and so those people who are experienced in this field will recognize many of them. It is important to understand the purpose of each algorithm and function, how they can be used, the input parameters needed, and how the parameters affect the outcome.

One of the biggest challenges when building predictive models is to prepare the input data to produce high value results. Preparing input data often involves the following activities:

- Removing unhelpful information, such as unusual or invalid values.
- Finding a data set that provides a good sample of the data but is small enough to yield fast results.
- Breaking up the data set.
- Filling in missing values.

SAP has provided many data preparation functions to support this task.

Predictive Analysis Library (PAL) Functions

PAL functions are written in C++ and can be called from SAP HANA Procedures. You can use tables or views or output from other procedures as the data source to the predictive analysis. The results can be passed to tables or as inputs to other procedures.



■ Association Analysis	<ul style="list-style-type: none"> ■ Apriori ■ FP-Growth ■ K-Optimal Rule Discovery (KORD) 	■ Cluster Analysis	<ul style="list-style-type: none"> ■ ContentNaive Bayes ■ Parameter Selection and Model Evaluation (PSME) ■ Predict with Tree Model ■ Random Forests ■ Support Vector Machine ■ Incremental Classification on SAP HANA Smart Data Streaming
■ Classification Analysis	<ul style="list-style-type: none"> ■ Area Under Curve (AUC) ■ Back Propagation Neural Network ■ C4.5 Decision Tree ■ CART Decision Tree ■ CHAID Decision Tree ■ Confusion Matrix ■ KNN ■ Logistic Regression (with Elastic Net Regularization) ■ Multi-Class Logistic Regression 	■ Regression	<ul style="list-style-type: none"> ■ Bi-Variate Geometric Regression ■ Bi-Variate Natural Logarithmic Regression ■ Exponential Regression ■ Multiple Linear Regression ■ Polynomial Regression

Figure 276: PAL Function Categories: Association to Cluster Analysis

PAL Function Categories: Time Series to Other



■ Time Series Analysis	■ Social Network Analysis	■ Statistics Functions
<ul style="list-style-type: none"> ■ ARIMA ■ Brown Exponential Smoothing ■ Croston's Method ■ Forecast Accuracy Measures ■ Forecast Smoothing ■ Linear Regression with Damped Trend and Seasonal Adjust ■ Single Exponential Smoothing ■ Double Exponential Smoothing ■ Triple Exponential Smoothing ■ Seasonality Test ■ Trend Test ■ White Noise Test 	<ul style="list-style-type: none"> ■ Link Prediction 	<ul style="list-style-type: none"> ■ Chi-Squared Test for Fitness ■ Chi-Squared Test for Independent ■ Cumulative Distribution Function ■ Distribution Fitting ■ Grubbs' Test ■ Kaplan-Meier Survival Analysis ■ Multivariate Statistics ■ Quantile Function ■ Univariate Statistics ■ Variance Equal Test
	■ Data Preparation	■ Other
	<ul style="list-style-type: none"> ■ Binning ■ Binning Assignment ■ Convert Category Type to Binary Vector ■ Inter-Quartile Range Test ■ Partition ■ Posterior Scaling ■ Principal Component Analysis (PCA) ■ Random Distribution Sampling ■ Sampling ■ Scaling Range ■ Substitute Missing Values 	<ul style="list-style-type: none"> ■ ABC Analysis ■ Weighted Score Table

Figure 277: PAL Function Categories: Time Series to Other

As listed in the figures PAL Function Categories: Association to Cluster Analysis and Time Series to Other, PAL includes well-known predictive analysis algorithms organized into data-mining categories:

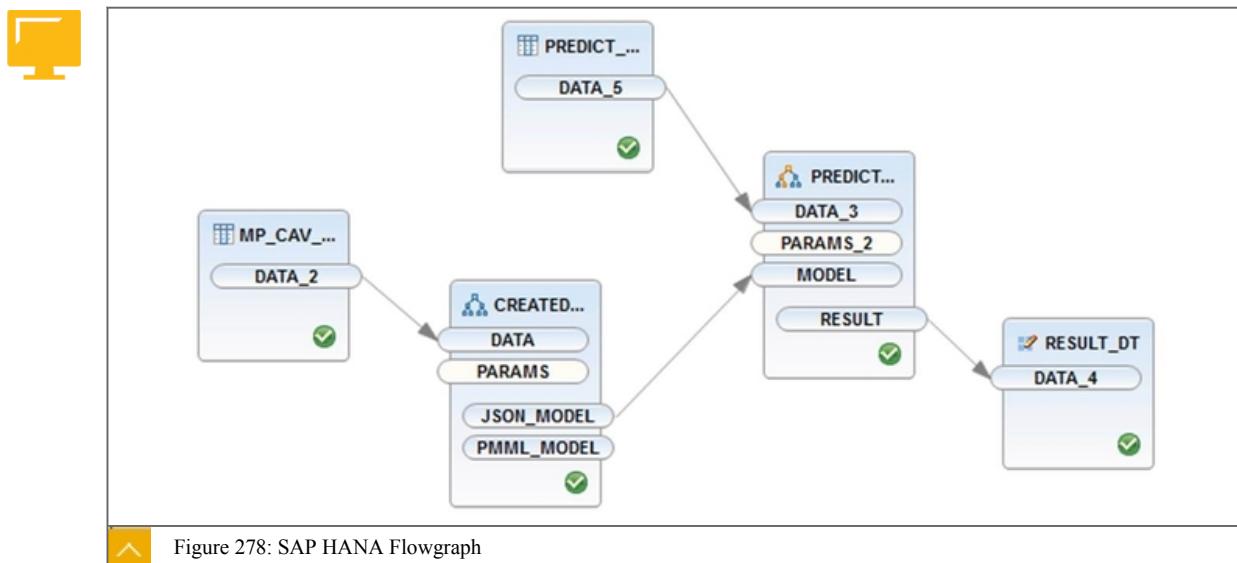
- Clustering

Unit 9: Advanced Data Processing

- Classification
- Regression
- Association
- Preprocessing
- Statistics
- Social Network Analysis
- Time Series

PAL also provides several incremental machine learning algorithms that continually learn and update a model on the fly, so that predictions are based on a moving model that improves over time.

Building Predictive Models Using an SAP HANA Flowgraph



You can build predictive models using code by creating procedures or you can use a graphical approach by developing flowgraphs. Flowgraphs can be built using either the development perspective of SAP HANA Studio (for XS based projects) or Web IDE for HANA (for XSA projects). As the figure SAP HANA Flowgraph shows, you simply add PAL algorithm and function nodes to your flowgraph and connect the nodes. For each node, you specify the modeling parameters, define input sources and output targets, and when you activate the flowgraph, a procedure is generated.

You can find the flowgraph design-time objects easily in SAP HANA Studio or SAP Web IDE because they have the .hdbflowgraph extension.

As an example of a powerful predictive model, let's consider a decision tree. A decision tree produces an outcome based on one or more input variables. The more input variables you can provide to the decision tree, the higher the quality of prediction. A decision tree is a hierarchy of decisions. For outcomes that are not strong influencers, the decision tree grows more lower branches where more questions can be asked. You can supply parameters that limit the extent to which the decision tree is allowed to grow.

A decision tree is one of a class of predictive models that need to be first trained before they can be deployed. Training a model involves passing known outcomes into the model so that the model is able to figure out what were the most influential variables that caused the various

historical outcomes. The outcomes are often binary, in other words, a simple ‘delayed’ or ‘on-time’ decision. But decision trees can also provide multiple outcomes, for example, ‘high’, ‘medium’ and ‘low’.



LESSON SUMMARY

You should now be able to:

- Describe SAP HANA Predictive Analysis capabilities

Unit 9

Lesson 5

Optional: Working with SAP HANA Graph

LESSON OVERVIEW

Graph processing enables you to make better use of your data by organizing it into nodes and relationships between nodes. This facilitates the visualization and in-depth analysis of your data, in particular with the help of specific algorithms. In this lesson, you will learn how SAP HANA Graph, a feature of SAP HANA SPS12, enables seamless graph processing.



LESSON OBJECTIVES

After completing this lesson, you will be able to:

- Describe graph processing with SAP HANA

— IMPORTANT NOTE —

This optional lesson will be replaced by more comprehensive coverage found in the new SAP HANA course : HA301 — Advanced Modeling. Once this course is available, you should study that content instead. We plan to remove this lesson from the next release of this course. So for this release we continue to include this lesson whilst we build the new HA301 course. We also continue to include this lesson because the questions in the C_HANAIMP_13 certification are based on this content. From C_HANAIMP_14 the questions on this topic will be based on the new HA301 course content.

Please note that this lesson is not expected to be covered in the standard HA300 classroom delivery and is provided for self-study.

Graph Modeling

Graphs are used to model data where entities are highly connected, such as complex multi-point supply chains, utility networks, and social media networks. The basic idea behind graph modeling is that it allows a modeler to easily define a series of entities, or nodes, and link them in a network that represents how they relate to each other. Graph models can indicate flow direction between entities so that additional meaning can be added to the network and traces can be made. Imagine a complex supply chain mapped using a graph, where all manufacturers, suppliers, distributors, customers, and consumers are all represented with information stored at each node and also along the connections. But why would you want to define such models? The benefit is that it is easy to develop applications that can traverse huge graphs at high speed so you can ask questions such as the following:

- How many hours has the product traveled between two specified points in the network?
- Where are all the possible points of origin of this product?
- What is the shortest path between a supplier and a consumer, bypassing all usual distribution centres?

Graph processing allows you to discover hidden patterns and relationships in huge amounts of data and all in real time.

Example Business Graph Model

 CREATE SCHEMA "GREEK_MYTHOLOGY";

```
CREATE COLUMN TABLE "GREEK_MYTHOLOGY"."MEMBERS" (
    "NAME" VARCHAR(100) PRIMARY KEY,
    "TYPE" VARCHAR(100),
    "RESIDENCE" VARCHAR(100)
```

Vertices


```
CREATE COLUMN TABLE "GREEK_MYTHOLOGY"."RELATIONSHIPS" (
    "KEY" INT UNIQUE NOT NULL,
    "SOURCE" VARCHAR(100) NOT NULL
        REFERENCES "GREEK_MYTHOLOGY"."MEMBERS" ("NAME")
        ON UPDATE CASCADE ON DELETE CASCADE,
    "TARGET" VARCHAR(100) NOT NULL
        REFERENCES "GREEK_MYTHOLOGY"."MEMBERS" ("NAME")
        ON UPDATE CASCADE ON DELETE CASCADE,
    "TYPE" VARCHAR(100)
```

Edges

 Figure 279: Example Business Graph Model

The scenario used in the figure Example Business Graph Model is familiar to most people, but you could also use graph models in medicine, to create a network of patients, conditions, treatments, and outcomes for reuse in diagnosis and planning treatments of other patients.

As the figure Social Media Graphs illustrates, you can also use social media portals to find your customers and their friends, friends of friends, and their likes and dislikes, to create marketing opportunities.

Social Media Graphs

 **Social media graphs**



 Figure 280: Social Media Graphs

Unit 9: Advanced Data Processing

Why do we need graph processing when we have SQL? Although you can use standard SQL data definitions and query syntax to create and process a similar model, both the definition of the model and the SQL to query the graph would be extremely complex. Processing times for such a model could also be challenging. SAP HANA Graph provides tools for graph definition and language for graph processing to ensure that model development is more natural and simplified and the processing is flexible and, of course, optimized for SAP HANA column in-memory processing.

Graph Model Creation in SAP HANA

SAP HANA Graph was introduced with SAP HANA 1.0 SPS12 and received some significant updates for SAP HANA 2.0.

The best way to understand how a graph model works in SAP HANA is to build one. Let's use the example of Greek mythology, as shown in the figure SAP HANA Graph Greek Model.

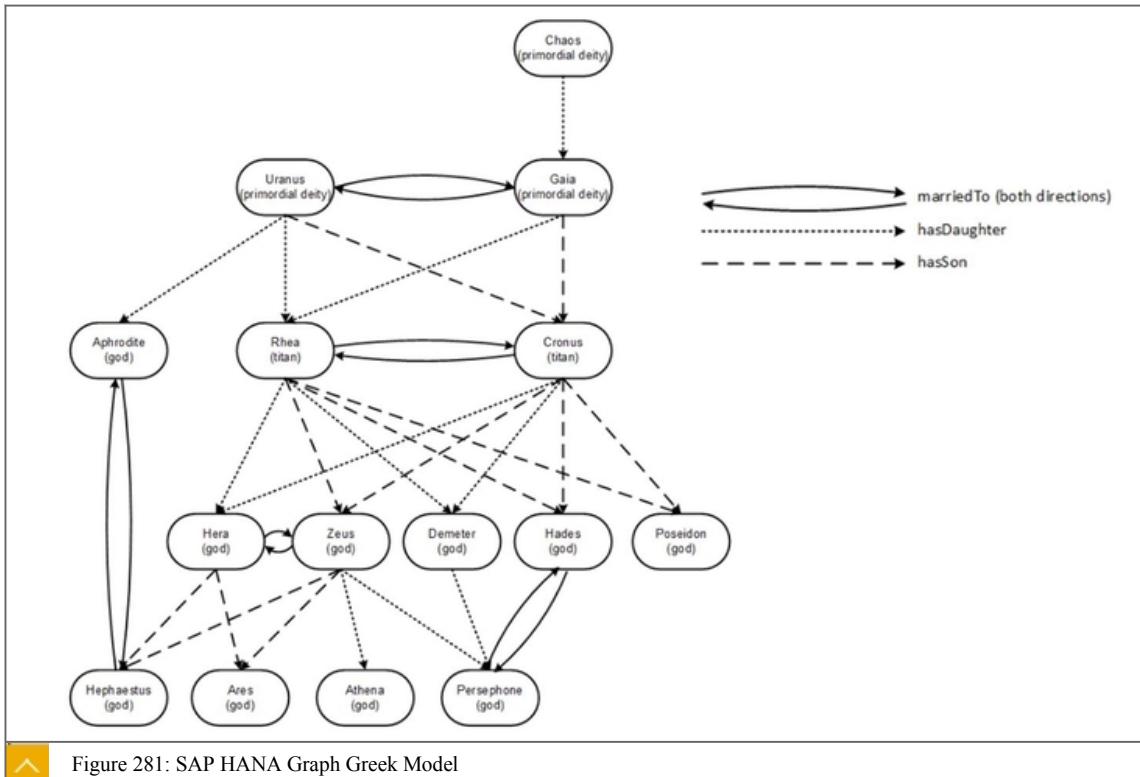


Figure 281: SAP HANA Graph Greek Model



Note:

This example is taken from the standard SAP documentation where all the SQL code is available to copy and paste to create your very own Greek mythology graph model, just like this one.

The key objects in a graph model are the vertices and edges. Vertices are stored in column tables and represent the nodes of the graph. Each vertex can store multiple attributes. The edges are also stored in column tables and describe the lines between the members. Each edge allows us to store relationship attributes such as distance, preference, strength, and so on. Edges describe the relationships between the vertices. As the figure shows, you can have multiple edges between two vertices. Edges can go in both directions between the vertices to represent the direction of the relationship, such as a YouTube vlogger and a follower, or a

father and daughter. Attributes assigned to the edges or vertices can be used in conditional expressions, such as filters, in queries.

SAP HANA Graph Create Vertices and Edge Tables

To get started with SAP HANA Graph you need to do the following:

1. Create and fill the vertices table and edge table with graph data.
 2. Create a graph workspace that refers to the vertices and edge tables.

The figure SAP HANA Graph Create Vertices and Edge Tables gives an example of the syntax for this step. If the data is already available in existing tables, you can also create views over the tables so that the data is in the correct format for graph modeling. As the figure shows, the vertices table and the edge table can reside in separate schemas.

SAP HANA Graph Create Workspace



Create a graph workspace

```
CREATE GRAPH WORKSPACE "GREEK_MYTHOLOGY"."GRAPH"
  EDGE TABLE "GREEK_MYTHOLOGY"."RELATIONSHIPS"
    SOURCE COLUMN "SOURCE"
    TARGET COLUMN "TARGET"
    KEY COLUMN "KEY"
  VERTEX TABLE "GREEK_MYTHOLOGY"."MEMBERS"
    KEY COLUMN "NAME";
```

The syntax for the graph workspace is shown in the figure SAP HANA Graph Create Workspace. The graph workspace simply creates the metadata that defines the graph

Unit 9: Advanced Data Processing

model and how the vertices and edge tables relate to each other. No actual business data is stored in a graph workspace. When activated, the resulting workspace will display in the system table SYS.GRAPH_WORKSPACES, where all graph workspaces are listed.

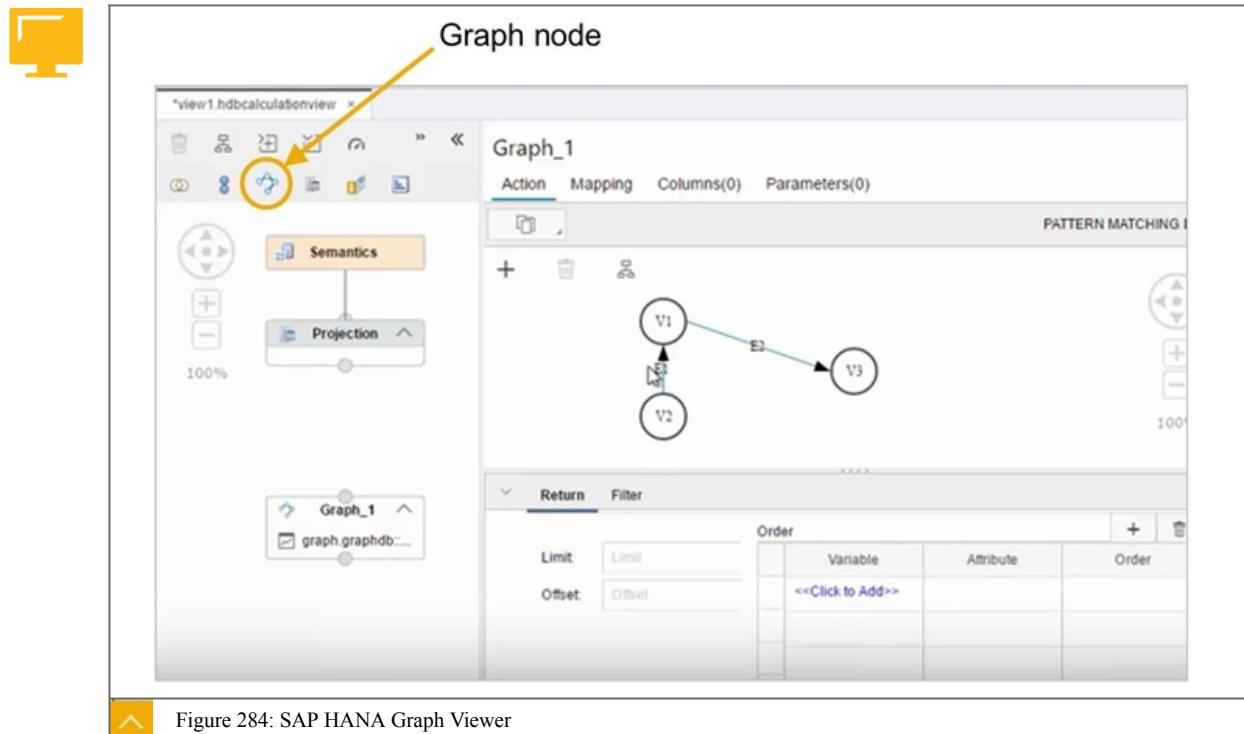
SAP HANA Graph provides a native processing language for graph query and manipulation called Weakly Structured Information Processing and Exploration (WIPE). WIPE is a declarative language very similar to SQL and contains special keywords to allow easy graph-based questions to be formulated such as: How far? How deep? What's the source? Where is the end point? Where is the strongest connection? What is the shortest path?

SAP HANA Graph utilizes a dedicated graph engine that works with the other data processing engines in SAP HANA. The graph engine is part of the standard SAP HANA installation.

SAP HANA Graph processing can be combined with all other types of SAP HANA data processing such as textual, spatial and predictive so that sophisticated and innovative applications can be developed on any data types.

The Graph Viewer

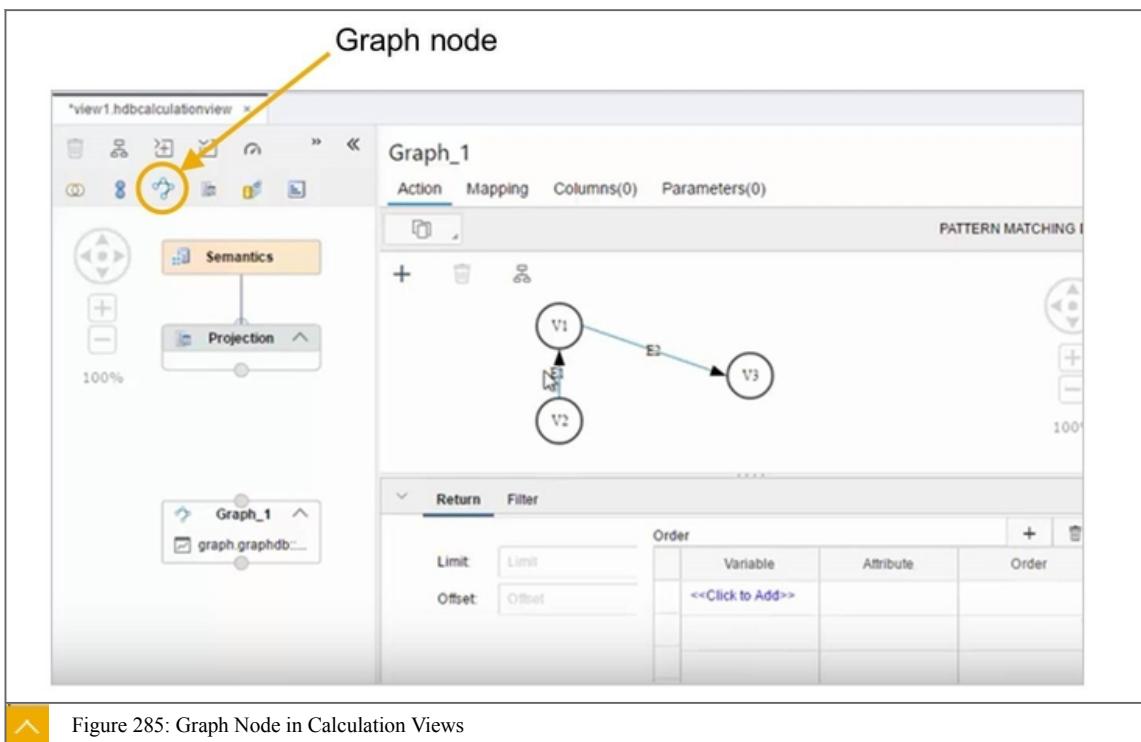
SAP supplies a HTML5 tool called the SAP HANA Graph Viewer that is very easy to use. As the figure SAP HANA Graph Viewer shows, this tool helps to quickly visualize the graph and formulate queries over the workspace using a simple form. The tool even generates the SQLScript for your query, which you can then copy and paste to your own applications.



Graph Nodes in Calculation Views

As the figure Graph Node in Calculation Views shows, graph nodes are available in SAP HANA 2.0, in the calculation views created with SAP Web IDE for SAP HANA. With the graph node you can define a query using one of the following algorithms that were delivered with SPS12:

- Shortest path
- Neighborhood
- Strongly connected components



Pattern matching is also available in SAP HANA 2.0. Pattern matching allows you to define a query inside a calculation view node by developing a pattern made up of nodes and edges. You define filters and conditions on the pattern, either on the edges or the nodes of the pattern, so that you return only the parts of the graph that match your pattern.

Additional Languages for Graph Processing

SAP HANA 2.0 offers Cypher Query Language (Cypher) to develop queries over SAP HANA Graphs. Cypher is the industry standard for graph processing. SAP HANA Graph now supports a subset of Cypher, focusing on the querying aspects, which allows you to migrate existing applications to SAP HANA more easily or simply take advantage of existing skills.

Also available in SAP HANA 2.0 is the new language GraphScript. For cases where graph processing stretches beyond the standard supplied algorithms of neighborhood search, shortest path, strongly connected components, and pattern matching, you can use GraphScript. When you build a procedure you simply specify the language as Graphscript.



LESSON SUMMARY

You should now be able to:

- Describe graph processing with SAP HANA

Unit 9

Learning Assessment

1. What are the three areas of SAP HANA Text Processing?

Choose the correct answers.

- A** Text Mining
- B** Text Profiling
- C** Text Analysis
- D** Text Search

2. In a text-type fuzzy search I enter ‘SAP’ as the search word. Against term ‘SAPPHIRE’ will I get a high or low score?

Choose the correct answer.

- A** Low
- B** High

3. What is an example of a use of Text Analysis?

Choose the correct answer.

- A** Cluster documents that discuss similar topics.
- B** Developing a user input field where misspellings are tolerated.
- C** Extraction of key facts from a business article.

4. What are the basic spatial shapes you can store in SAP HANA Spatial?

Choose the correct answers.

- A** Lines
- B** Polygons
- C** Graphs
- D** Points

5. Spatial expressions can be defined in SQLScript?

Determine whether this statement is true or false.

- A** True
 B False

6. Which of the following would you create to build a predictive model?

Choose the correct answer.

- A** A decision tree
 B A flowgraph
 C A function

7. Where do you build a flowgraph?

Choose the correct answers.

- A** SAP Web IDE for SAP HANA
 B SAP HANA Studio modeler perspective
 C Web-based development workbench
 D SAP HANA Studio development perspective

8. In which of the following circumstances do you use graph processing?

Choose the correct answers.

- A** To identify customer sentiments from social media content.
 B To improve performance on complex charts and dashboards.
 C To identify the strongly connected members of a social media group.
 D To find the shortest path between supplier and consumer.

9. Which of the following languages supported by SAP HANA Graph are optimized for graph processing?

Choose the correct answers.

- A** Cypher Query Language
 B GraphScript
 C SQL

Unit 9

Learning Assessment - Answers

1. What are the three areas of SAP HANA Text Processing?

Choose the correct answers.

- A** Text Mining
- B** Text Profiling
- C** Text Analysis
- D** Text Search

2. In a text-type fuzzy search I enter ‘SAP’ as the search word. Against term ‘SAPPHIRE’ will I get a high or low score?

Choose the correct answer.

- A** Low
- B** High

3. What is an example of a use of Text Analysis?

Choose the correct answer.

- A** Cluster documents that discuss similar topics.
- B** Developing a user input field where misspellings are tolerated.
- C** Extraction of key facts from a business article.

4. What are the basic spatial shapes you can store in SAP HANA Spatial?

Choose the correct answers.

- A** Lines
- B** Polygons
- C** Graphs
- D** Points

5. Spatial expressions can be defined in SQLScript?

Determine whether this statement is true or false.

T True

F False

6. Which of the following would you create to build a predictive model?

Choose the correct answer.

A A decision tree

B A flowgraph

C A function

7. Where do you build a flowgraph?

Choose the correct answers.

A SAP Web IDE for SAP HANA

B SAP HANA Studio modeler perspective

C Web-based development workbench

D SAP HANA Studio development perspective

8. In which of the following circumstances do you use graph processing?

Choose the correct answers.

A To identify customer sentiments from social media content.

B To improve performance on complex charts and dashboards.

C To identify the strongly connected members of a social media group.

D To find the shortest path between supplier and consumer.

Unit 9: Learning Assessment - Answers

9. Which are of the following languages supported by SAP HANA Graph are optimized for graph processing?

Choose the correct answers.

A Cypher Query Language

B GraphScript

C SQL