## Content

1. **Getting Started with the HDBTable Syntax.** .......................................................... 3  
   1.1 Developing Native SAP HANA Applications. ................................................. 4  
   1.2 Roles and Permissions for XS Development. .................................................. 6  
   1.3 Setting up the Data Persistence Model in SAP HANA. ................................... 8  
   1.4 Developer Information Map. ................................................................. 10  
      SAP HANA Developer Information by Guide. ............................................. 13  
      SAP HANA Developer Information by Task. ........................................... 17  
      SAP HANA Developer Information by Scenario. ...................................... 19  

2. **Creating the Persistence Model with HDBTable.** .......................................... 25  
   2.1 Create a Schema. ................................................................................. 26  
   2.2 Create a Table. ..................................................................................... 30  
      Tables. .................................................................................................. 31  
      Table Configuration Syntax. ...................................................................... 33  
   2.3 Create a Reusable Table Structure. ............................................................. 39  
      Reusable Table Structures. ............................................................... 41  
   2.4 Create a Sequence. ............................................................................... 42  
      Sequences. ............................................................................................ 44  
      Sequence Configuration Syntax. ............................................................. 46  
   2.5 Create an SQL View. ............................................................................... 51  
      SQL Views. ............................................................................................ 53  
      SQL View Configuration Syntax. ............................................................. 54  
   2.6 Create a Synonym. ............................................................................... 57  
      Synonyms. ............................................................................................... 59  
      Synonym Configuration Syntax. ............................................................. 60  
   2.7 Import Data with hdbtable Table-Import. ...................................................... 62  
      Data Provisioning Using Table Import. .................................................... 65  
      Table-Import Configuration. ................................................................. 66  
      Table-Import Configuration-File Syntax. .................................................. 69  
      Table-Import Configuration Error Messages. ........................................... 74
1 Getting Started with the HDBTable Syntax

HDBTable is a language syntax that can be used by database developers to create the underlying (persistent) data model which the application services expose to UI clients.

The database developer defines the data-persistence and analytic models that are used to expose data in response to client requests via HTTP. With HDBTable, you can define a persistence model that includes objects such as tables, views, schemas, and sequences; the database objects specify what data to make accessible for consumption by applications and how. This guide takes you through the tasks required to use the HDBTable syntax to define the objects that are most often used in a data persistence model, for example:

- Create a schema
- Create a table (entity)
- Create a table type (reusable table structure)
- Create an SQL view
- Create a sequence

The SAP HANA HDBTable Syntax Reference also provides code examples that illustrate how to specify the various object types. This reference guide also includes the complete specification of the HDBTable syntax required for each object type.

Building the data model is the first step in the overall process of developing applications that provide access to the SAP HANA database. When you have created the underlying data persistence model, application developers can build the application services that expose selected elements of the data model to client application by means of so-called “data end-points”. The client applications bind UI controls such as buttons or charts and graphs to the application services which in turn retrieve and display the requested data.

Prerequisites

Before you can start using HDBTable to define the objects that comprise your persistence model, you need to ensure that the following prerequisites are met:

- You must have access to an SAP HANA system.
- You must have already created a development workspace and a project.
- You must have shared a project for the HDBTable artifacts so that the newly created files can be committed to (and synchronized with) the repository.
- The owner of the schema must have SELECT privileges in the schema to be able to see the generated catalog objects.

Related Information

Setting up the Data Persistence Model in SAP HANA [page 8]
1.1 Developing Native SAP HANA Applications

In SAP HANA, native applications use the technology and services provided by the integrated SAP HANA XS platform.

The term “native application” refers to a scenario where applications are developed in the design-time environment provided by SAP HANA extended application services (SAP HANA XS) and use the integrated SAP HANA XS platform illustrated in the following graphic.

Note
A program that consists purely of SQLScript is also considered a native SAP HANA application.

The server-centric approach to native application development envisaged for SAP HANA assumes the following high-level scenario:

- All application artifacts are stored in the SAP HANA repository
- Server-side procedural logic is defined in server-side (XS) JavaScript or SQLScript
- UI rendering occurs completely in the client (browser, mobile applications)

Each of the levels illustrated in the graphic is manifested in a particular technology and dedicated languages:
• Calculation Logic - data-processing technology:
  ○ Data:
    SQL / SQLScript, Core Data Services (CDS), DDL, HDBtable
  ○ SQL / SQLScript
  ○ Calculation Engine Functions (CE_*)

**Note**
SAP recommends you use SQL rather than the Calculation Engine functions.

○ Application Function Library (AFL)

• Control-flow logic with SAP HANA XS:
  ○ OData
    Validation models for OData services can be written in XS JavaScript or SQLScript
  ○ Server-Side JavaScript (XSJS)
    HTTP requests are implemented directly in XS JavaScript
  ○ XMLA

• Client UI/Front-end technology:
  ○ HTML5 / SAPUI5
  ○ Client-side JavaScript

The development scenarios for native application development are aimed at the following broadly defined audiences:

<table>
<thead>
<tr>
<th>Audience</th>
<th>Language</th>
<th>Tools</th>
<th>Development Artifacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database developers</td>
<td>SQLScript, CDS, hdb* SAP</td>
<td>• SAP HANA studio</td>
<td>Database tables, views, procedures; user-defined functions (UDF) and triggers; analytic objects; data authorization...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SAP HANA Web-based Workbench</td>
<td></td>
</tr>
<tr>
<td>Application developers:</td>
<td>XS JavaScript, OData, SQLScript, ...</td>
<td>• SAP HANA studio</td>
<td>Control-flow logic, data services, calculation logic...</td>
</tr>
<tr>
<td>Professional (XS JS)</td>
<td></td>
<td>• SAP HANA Web-based Workbench</td>
<td></td>
</tr>
<tr>
<td>Casual/business</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UI/client developers</td>
<td>SAPUI5, JavaScript, ...</td>
<td>• SAP HANA studio</td>
<td>UI shell, navigation, themes (look/feel), controls, events, ...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SAP HANA Web-based Workbench</td>
<td></td>
</tr>
</tbody>
</table>

**Related Information**

Database Development Scenarios [page 22]
1.2 Roles and Permissions for XS Development

An overview of the authorizations required to develop database artifacts for SAP HANA using the CDS syntax.

To enable application-developers to start building native applications that take advantage of the SAP HANA Extended Application Services (SAP HANA XS), the SAP HANA administrator must ensure that developers have access to the tools and objects that they need to perform the tasks required during the application- and database-development process.

Before you start developing applications using the features and tools provided by the SAP HANA XS, bear in mind the following prerequisites. Developers who want to build applications to run on SAP HANA XS need the following tools, accounts, and privileges:

- SAP HANA XS Classic Model [page 6]
- SAP HANA XS Advanced Model [page 7]

**Note**
The required privileges can only be granted by someone who has the necessary authorizations in SAP HANA, for example, an SAP HANA administrator.

**SAP HANA XS Classic Model**

To develop database artifacts for use by applications running in the SAP HANA XS classic environment, bear in mind the following prerequisites:

- Access to a running SAP HANA development system (with SAP HANA XS classic)
- A valid user account in the SAP HANA database on that system
- Access to development tools, for example, provided in:
  - SAP HANA studio
  - SAP HANA Web-based Development Workbench
- Access to the SAP HANA repository
- Access to selected run-time catalog objects

**Note**
To provide access to the repository for application developers, you can use a predefined role or create your own custom role to which you assign the privileges that the application developers need to perform the everyday tasks associated with the application-development process.

To provide access to the repository from the SAP HANA studio, the EXECUTE privilege is required for SYS.REPOSITORY_REST, the database procedure through with the REST API is tunneled. To enable the activation and data preview of information views, the technical user _SYS_REPO also requires SELECT privilege on all schemas where source tables reside.

In SAP HANA, you can use roles to assign one or more privileges to a user according to the area in which the user works; the role defines the privileges the user is granted. For example, a role enables you to assign SQL...
privileges, analytic privileges, system privileges, package privileges, and so on. To create and maintain artifacts in the SAP HANA repository, you can assign application-development users the following roles:

- One of the following:
  - MODELING
    The predefined MODELING role assigns wide-ranging SQL privileges, for example, on _SYS_BI and _SYS_BIC. It also assigns the analytic privilege _SYS_BI_CP_ALL, and some system privileges. If these permissions are more than your development team requires, you can create your own role with a set of privileges designed to meet the needs of the application-development team.
  - Custom DEVELOPMENT role
    A user with the appropriate authorization can create a custom DEVELOPMENT role specially for application developers. The new role would specify only those privileges an application-developer needs to perform the everyday tasks associated with application development, for example: maintaining packages in the repository, executing SQL statements, displaying data previews for views, and so on.

- PUBLIC
  This is a role that is assigned to all users by default.

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

**Tip**
The role sap.hana.xs.ide.roles::Developer grants the privileges required to use all the tools included in the SAP HANA Web-based Development Workbench. However, to enable a developer to use the debugging features of the browser-based IDE, your administrator must also assign the role sap.hana.xs.debugger::Debugger. In addition, the section debugger with the parameter enabled and the value true must be added to the file xsengine.ini file, for example, in the SAP HANA studio Administration perspective.

**SAP HANA XS Advanced Model**

To develop database artifacts for use by applications running in the SAP HANA XS advanced environment, bear in mind the following prerequisites:

- Access to a running SAP HANA development system (with SAP HANA XS advanced)
- A valid user account in the SAP HANA database on that system
- Access to development tools, for example, provided in:
  - SAP Web IDE for SAP HANA
  - SAP HANA Run-time Tools (included in the SAP Web IDE for SAP HANA)

**Note**
To provide access to tools and for application developers in XS advanced, you define a custom role to which you add the privileges required to perform the everyday tasks associated with the application- and database-development process. The role is then assigned to a role collection which is, in turn, assigned to the developer.
Access to the SAP HANA XS advanced design-time workspace and repository
Access to selected run-time catalog objects
Access to the XS command-line interface (CLI); the XS CLI client needs to be downloaded and installed

1.3 Setting up the Data Persistence Model in SAP HANA

The persistence model defines the schema, tables, sequences, and views that specify what data to make accessible for consumption by XS applications and how.

In SAP HANA Extended Application Services (SAP HANA XS), the persistence model is mapped to the consumption model that is exposed to client applications and users so that data can be analyzed and displayed in the appropriate form in the client application interface. The way you design and develop the database objects required for your data model depends on whether you are developing applications that run in the SAP HANA XS classic or XS advanced run-time environment.

- SAP HANA XS Classic Model [page 8]
- SAP HANA XS Advanced Model [page 9]

SAP HANA XS Classic Model

SAP HANA XS classic model enables you to create database schema, tables, views, and sequences as design-time files in the SAP HANA repository. Repository files can be read by applications that you develop. When implementing the data persistence model in XS classic, you can use either the Core Data Services (CDS) syntax or HDBtable syntax (or both). “HDBtable syntax” is a collective term; it includes the different configuration schema for each of the various design-time data artifacts, for example: schema (.hdbschema), sequence (.hdbsequence), table (.hdbtable), and view (.hdbview).

All repository files including your view definition can be transported (along with tables, schema, and sequences) to other SAP HANA systems, for example, in a delivery unit. A delivery unit is the medium SAP HANA provides to enable you to assemble all your application-related repository artifacts together into an archive that can be easily exported to other systems.

Note

You can also set up data-provisioning rules and save them as design-time objects so that they can be included in the delivery unit that you transport between systems.

The rules you define for a data-provisioning scenario enable you to import data from comma-separated values (CSV) files directly into SAP HANA tables using the SAP HANA XS table-import feature. The complete data-import configuration can be included in a delivery unit and transported between SAP HANA systems for reuse.
As part of the process of setting up the basic persistence model for SAP HANA XS, you create the following artifacts in the XS classic repository:

<table>
<thead>
<tr>
<th>XS Classic Artifact Type</th>
<th>CDS</th>
<th>HDBTable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema</td>
<td>.hdbschema</td>
<td>.hdbschema</td>
</tr>
<tr>
<td>Synonym</td>
<td>.hdbsynonym</td>
<td>.hdbsynonym</td>
</tr>
<tr>
<td>Table</td>
<td>.hdbdd</td>
<td>.hdbtable</td>
</tr>
<tr>
<td>Table Type</td>
<td>.hdbdd</td>
<td>.hdbstructure</td>
</tr>
<tr>
<td>View</td>
<td>.hdbdd</td>
<td>.hdbview</td>
</tr>
<tr>
<td>Association</td>
<td>.hdbdd</td>
<td></td>
</tr>
<tr>
<td>Sequence</td>
<td>.hdbsequence</td>
<td>.hdbsequence</td>
</tr>
<tr>
<td>Structured Types</td>
<td>.hdbdd</td>
<td></td>
</tr>
<tr>
<td>Data import</td>
<td>.hdbti</td>
<td>.hdbti</td>
</tr>
</tbody>
</table>

*i Note*

(*) To create a schema, a synonym, or a sequence, you must use the appropriate HDBTable syntax, for example, .hdbschema, .hdbsynonym, or .hdbsequence. In a CDS document, you can include references to both CDS and HDBTable artifacts.

On activation of a repository artifact, the file suffix (for example, .hdbdd or .hdb[table|view]) is used to determine which run-time plug-in to call during the activation process. When you activate a design-time artifact in the SAP HANA Repository, the plug-in corresponding to the artifact’s file suffix reads the contents of repository artifact selected for activation (for example, a table, a view, or a complete CDS document that contains multiple artifact definitions), interprets the artifact definitions in the file, and creates the appropriate corresponding run-time objects in the catalog.

**SAP HANA XS Advanced Model**

For the XS advanced run time, you develop multi-target applications (MTA), which contain modules, for example: a database module, a module for your business logic (Node.js), and a UI module for your client interface (HTML5). The modules enable you to group together in logical subpackages the artifacts that you need for the various elements of your multi-target application. You can deploy the whole package or the individual subpackages.

As part of the process of defining the database persistence model for your XS advanced application, you use the database module to store database design-time artifacts such as tables and views, which you define using Core Data Services (CDS). However, you can also create procedures and functions, for example, using SQLScript, which can be used to insert data into (and remove data from) tables or views.

*i Note*

In general, CDS works in XS advanced (HDI) in the same way that it does in the SAP HANA XS classic Repository. For XS advanced, however, there are some incompatible changes and additions, for example, in
the definition and use of name spaces, the use of annotations, the definition of entities (tables) and structure types. For more information, see CDS Documents in XS Advanced in the list of Related Links below.

In XS advanced, application development takes place in the context of a project. The project brings together individual applications in a so-called Multi-Target Application (MTA), which includes a module in which you define and store the database objects required by your data model.

1. Define the data model.
   Set up the folder structure for the design-time representations of your database objects; this could include CDS documents that define tables, data types, views, and so on. But it could also include other database artifacts, too, for example: your stored procedures, synonyms, sequences, scalar (or table) functions, and any other artifacts your application requires.

   **Tip**
   You can also define the analytic model, for example, the calculation views and analytic privileges that are to be used to analyze the underlying data model and specify who (or what) is allowed access.

2. Set up the SAP HANA HDI deployment infrastructure.
   This includes the following components:
   - The HDI configuration
     Map the design-time database artifact type (determined by the file extension, for example, .hdbprocedure, or .hdbcds in XS advanced) to the corresponding HDI build plug-in in the HDI configuration file (.hdiconfig).
   - Run-time name space configuration (optional)
     Define rules that determine how the run-time name space of the deployed database object is formed. For example, you can specify a base prefix for the run-time name space and, if desired, specify if the name of the folder containing the design-time artifact is reflected in the run-time name space that the deployed object uses. Alternatively, you can specify the use of freestyle names, for example, names that do not adhere to any name-space rules.

3. Deploy the data model.
   Use the design-time representations of your database artifacts to generate the corresponding active objects in the database catalog.

4. Consume the data model.
   Reference the deployed database objects from your application, for example, using OData services bound to UI elements.

### 1.4 Developer Information Map

The developer information road map is designed to help developers find the information they need in the library of user and reference documentation currently available for SAP HANA development projects.

The development environment for SAP HANA supports a wide variety of application-development scenarios. For example, database developers need to be able to build a persistence model or design an analytic model; professional developers want to build enterprise-ready applications; business experts with a development background might like to build a simple server-side, line-of-business application; and application developers...
need to be able to design and build a client user interface (UI) that displays the data exposed by the data model and business logic. It is also essential to set up the development environment correctly and securely and ensure the efficient management of the various phases of the development lifecycle.

The following image displays essential information sources for people planning to develop applications in SAP HANA Extended Application Services classic model.

Figure 2: Application Development in SAP HANA XS Classic Model

The following image displays the essential information sources for developing applications in SAP HANA Extended Application Services advanced model.
With such a wide variety of people needing to find such a broad range of information about so many different tasks, it is sometimes not easy to know what information is available or where to look to find it. This section is designed to help you navigate the documentation landscape by looking at the information available for the SAP HANA developer from the following perspectives:

- Information by developer guide
- Information by developer task
- Information by developer scenario

**Related Information**

- SAP HANA Developer Information by Guide [page 13]
- SAP HANA Developer Information by Task [page 17]
- SAP HANA Developer Information by Scenario [page 19]
1.4.1 SAP HANA Developer Information by Guide

The design and organization of the SAP HANA developer documentation library makes it easy to use the name of a guide to find the relevant information. For example, the SAP HANA SQLScript Reference describes how to use the SQL extension SAP HANA SQLScript to define a data model.

The SAP HANA developer information set includes a selection of guides that describe the complete application-development process, from defining user roles, privileges, and data models through application setup to UI design and testing; the information available covers background and concepts, task-based tutorials, and detailed reference material. The following tables illustrate which guides are available for the developer who wants to build applications for SAP HANA, what information the guide contains, and which typical high-level tasks the selected guide covers:

**Table 3: Core SAP HANA Developer Guides**

<table>
<thead>
<tr>
<th>SAP HANA Guide</th>
<th>Description</th>
<th>Typical Tasks</th>
</tr>
</thead>
</table>
| Developer Quick Start Guide (for XS classic) | Contains a selection of tutorials which teach the basic steps required to build data models and the native applications that run on SAP HANA XS classic model. | Build data-persistence models  
Write procedures and user-defined functions (UDF)  
Build applications with SAP HANA XS JavaScript or OData.  
Build a simple UI with SAPUI5 |
| Developer Guide (for XS classic and SAP HANA studio) | Describes the complete application-development process for SAP HANA Extended Application Services Classic Model using the tools included in SAP HANA studio. | Build a data model  
Build XS classic applications (XS JavaScript)  
Build SAPUI5 clients  
Manage the application lifecycle |
| Developer Guide (for XS classic and Web Workbench) | Describes the complete application-development process for SAP HANA Extended Application Services Classic Model using the tools included in SAP HANA Web-based Development Workbench. | Build a data model  
Build XS classic applications (XS JavaScript)  
Build SAPUI5 clients  
Manage the application lifecycle |
| Developer Guide (for XS advanced) | Describes the complete application-development process for SAP HANA Extended Application Services Advanced Model. | Build a data model  
Build XS advanced applications (Node.js, JavaScript, Java, ...)  
Build SAPUI5 clients  
Deploy applications to desired run-time environment  
Manage the application lifecycle |
<table>
<thead>
<tr>
<th>SAP HANA Guide</th>
<th>Description</th>
<th>Typical Tasks</th>
</tr>
</thead>
</table>
| Modeling Guide (for XS classic)      | Explains how to use the SAP HANA modeler in to create information models for use by XS classic applications based on data that can be used for analytical purposes. | Create attribute, analytic, and calculation views  
Create decision tables  
Import/Export data |
| Modeling Guide (for XS advanced)     | Explains how to use the SAP HANA modeler to create information models for XS advanced applications based on data that can be used for analytical purposes. | Create attribute, analytic, and calculation views  
Create decision tables  
Import/Export data |
| SAPUI5 Demo Kit and Documentation    | Describes how to develop SAPUI5 applications based on SAP HANA, a user interface technology for building and adapting client applications | Build client UI models  
Design UI view (buttons/boxes)  
Bind UI views to data  
Bind UI controls to services |

The SAP HANA developer information set also includes a selection of reference guides that describe the various languages that you use to define the underlying data model (for example, SQL, CDS, or HDBTable) as well as the application business logic (for example, XS JavaScript with XS classic). The following tables illustrate which reference guides are available for the more experienced developer who needs more detailed information about how to build specific components.

Table 4: SAP HANA Language Reference Guides

<table>
<thead>
<tr>
<th>SAP HANA Guide</th>
<th>Description</th>
<th>Typical Tasks</th>
</tr>
</thead>
</table>
| SQL System Views        | Describes all SQL data types, predicates, operators, expressions, functions, statements, and error codes. | Query state of SAP HANA using SQL commands  
Alter system configuration/initialization services *  
Manage extended storage *  
Manage remote sources, subscriptions, adapters, tasks *  
Perform data analysis/mining *  
Manage data streams * |
| SQLScript Reference     | Describes how to use the SQL extension SAP HANA SQLScript to embed data-intensive application logic into SAP HANA. | Build SQL scripts  
Create UDFs  
Build SQL procedures |
<table>
<thead>
<tr>
<th>SAP HANA Guide</th>
<th>Description</th>
<th>Typical Tasks</th>
</tr>
</thead>
</table>
| **SAP HANA Analytics Catalog (BIMC Views) Reference** | Describes how to use the SAP HANA analytics catalog for tables and views with the BIMC prefix. The catalog contains metadata used by analytics clients such as the Business Objects Cloud and for access to SAP HANA via MDX. | Pass and map variables and parameters  
Use parameters in hierarchies  
Read values from BIMC tables  
Build SQL queries  
Build MDX queries |
| **SQLScript Command Network Protocol** | Describes the SQL Command Network Protocol that is used by SAP HANA clients to communicate with SAP HANA.                                                                                                 | Define routes for SQL statements  
Set up authentication (SAML...)  
Handle large data objects  
Enable distributed transactions |
| **Spatial Reference(*)**               | Describes how to store, manipulate, and manage spatial data, for example, geographic locations, routing information, and shape data.                                                                              | Store and manage spatial data  
Access and manipulate spatial data  
Calculate the distance between geometries  
Determine the union/intersection of multiple objects |
| **XS JavaScript Reference**            | Describes how to use XS JavaScript to build native SAP HANA applications for XS classic, explains the underlying concepts, and lists the various APIs that are available.                                              | Create XSJS services (for XS classic)  
Create XSJS libraries  
Build application logic |
| **XS JavaScript API Reference**        | Describes the API functions, methods, and classes provided for use with server-side JavaScript code running inside (SAP HANA XS).                                                                            | Use the XS JavaScript API  
Search for XSJS API classes  
Locate XSJS methods |
| **XSUnit JavaScript API Reference**    | Describes the API functions, methods, and classes provided with the XSUnit test framework to automate the tests that you want to run for SAP HANA XS applications.                                                  | Test server-side (XS) JavaScript code  
Test SQLScript code (stored procedures, views)  
Test modeled calculation view |
| **XS DB Utilities JavaScript API Reference** | Describes the API that provides access to a library of JavaScript utilities, which can be used to enable server-side JavaScript applications to consume data models that are defined using Core Data Services or call stored procedures as if they were JavaScript objects. | Call a stored procedure  
Query a CDS entity  
Update a CDS entity |
<table>
<thead>
<tr>
<th>SAP HANA Guide</th>
<th>Description</th>
<th>Typical Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>SINA Search JavaScript Reference *</td>
<td>Describes the SAP HANA simple information access (SINA) API, a client-side JavaScript API for developing browser-based search UIs</td>
<td>Create a search query</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Create a suggestion query</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Create a bar- or line-chart query</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Create a SAPUI5 bar chart</td>
</tr>
<tr>
<td>Core Data Services (CDS) Reference (for XS classic)</td>
<td>Explains how to use Core Data Services (CDS) to build design-time data-persistence models in SAP HANA Extended Application Services (for XS classic). The data-persistence model defines the data to expose in response to client requests via HTTP, for example, from an SAPUI5-based application.</td>
<td>Create CDS Documents (for XS classic)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Define tables, table types, and SQL views</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Define associations between data objects</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Import data into a table</td>
</tr>
<tr>
<td>HDBTable Syntax Reference</td>
<td>Explains how to use the hdbtable syntax to build design-time data-persistence models in SAP HANA XS (for XS classic). The data-persistence model is used to define the data to expose in response to client requests via HTTP, for example, from an SAPUI5-based application.</td>
<td>Define schemas and sequences (for XS classic)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Define tables, SQL views, and table types</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Import data into a table</td>
</tr>
<tr>
<td>SAP HANA REST API (for XS classic)</td>
<td>Describes the REST API for SAP HANA (for XS classic), which enables development tools to access SAP HANA platform components such as the for XS classic repository using REST-based calls.</td>
<td>Maintain repository workspaces</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Maintain projects and files</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Interact with the SAP HANA repository</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Access catalog objects</td>
</tr>
<tr>
<td>BFL Reference</td>
<td>Describes the SAP HANA Business Function Library (BFL), which contains pre-built financial functions.</td>
<td>Assign AFL user roles/privileges</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Create the AFL wrapper procedure</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Generate and call a PAL procedure</td>
</tr>
<tr>
<td>PAL Reference</td>
<td>Describes the SAP HANA Predictive Analysis Library (PAL), which contains functions that can be called from within SAP HANA SQLScript procedures to perform analytic algorithms.</td>
<td>Assign AFL user roles/privileges</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Create the AFL wrapper procedure</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Generate and call a BFL procedure</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Create input/output tables</td>
</tr>
</tbody>
</table>

**Caution**

(*) SAP HANA server software and tools can be used for several SAP HANA platform and options scenarios as well as the respective capabilities used in these scenarios. The availability of these is based on the available SAP HANA licenses and the SAP HANA landscape, including the type and version of the back-end systems the SAP HANA administration and development tools are connected to. There are several types of licenses available for SAP HANA. Depending on your SAP HANA installation license type, some of the features and tools described in the SAP HANA platform documentation may only be available in the SAP HANA options and capabilities, which may be released independently of an SAP HANA Platform Support.
1.4.2 SAP HANA Developer Information by Task

The design and organization of the SAP HANA developer documentation library enables easy access to information according to the particular development task to be performed, for example, creating a view or procedure, or setting up an application project.

The SAP HANA developer can make use of a large number of guides that include information describing the complete application-development process. The following figure shows the information that is available from the perspective of the development tasks that must be performed in a particular development area, for example, setting up the persistence model; creating an XSJS or OData service, or managing the development lifecycle. Each of the tasks described is supported by information covering the underlying concepts and detailed reference material. The figure also indicates where to find information based on the development task you want to perform. The tasks are split according to development area, for example, database development, application development, or UI design.
<table>
<thead>
<tr>
<th>SAP HANA Development Area</th>
<th>Typical Tasks</th>
<th>SAP HANA Guide</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Developer</td>
</tr>
<tr>
<td>Database</td>
<td>Set up the persistence model</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Set up the analytic model</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Create SQLScript procedures</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Create user-defined functions</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Create decision tables</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Set up lifecycle management</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Create full-text search</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Build SQL search queries</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Model spatial data</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Manage extended storage</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Perform data analysis/mining</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Manage data streams</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Manage remote sources</td>
<td>X</td>
</tr>
<tr>
<td>Applications</td>
<td>Set up an application</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Set up a project</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Create an OData/XMLA service</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Create an XSJS service</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Bind XS service to UI</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Set up lifecycle management</td>
<td>X</td>
</tr>
<tr>
<td>User Interface &amp; Clients</td>
<td>Set up SAPUI5 tools</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Create SAPUI5 apps</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Create UI views</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Define UI event handlers</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Bind data to a view</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Create UI widgets</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Define search UIs</td>
<td>X</td>
</tr>
<tr>
<td>Repository Access</td>
<td>Logon credentials</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Roles and privileges</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Application artifacts</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Life-cycle management</td>
<td>X</td>
</tr>
</tbody>
</table>

**Figure 4: SAP HANA Information by Developer Task**

⚠️ **Caution**

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

1.4.3 SAP HANA Developer Information by Scenario

The design and organization of the SAP HANA developer documentation library enables easy access to information according to the underlying development scenario, for example, lifecycle management, or application development.

The SAP HANA developer can make use of a large number of guides that include information describing the complete application-development process from the perspective of the development scenario, for example, database development, application development, or client UI design and testing; the information available covers background and concepts, task-based tutorials, and detailed reference material. The following table indicates where to find information based on the development scenario you choose, for example:

- Database Development [page 19]
- Application Development [page 21]
- UI Client Design [page 22]

The particular scenario you select can be based on the underlying development area you are assigned to, the choice of programing language, the required development objects, or the tools you want to use:

Database Development Scenarios

A database developer uses a variety of languages to develop a data model that can be exposed to a UI client, for example by HTTP (with an application service) or SQL (with a client interface such as ADBC or JDBC). In a database-development scenario, developers typically use languages such as SQLScript, the .hdbtable syntax family (hdbview, hdbsequence,...), or Core Data Services (CDS) to define the data-persistence model to which you add the corresponding analytic model. If you want to develop a data model that can be exposed to client requests, use the following table to help find the information you need to complete the most common development tasks.

Using features available in additional optional components (*), you can develop applications that perform the following tasks:

- Manage data streams, for example, to filter, aggregate or enrich data before it is committed to the database
- Perform real-time operational analytics on data
- Enhance, cleanse, and transform data from local or remote sources to make it more accurate and useful
- Analyze and process geospatial information
- Analyze and mine both structured and unstructured textual data and interlinked structured data

Table 5: Information by Database-Development Scenario

<table>
<thead>
<tr>
<th>Programming Language</th>
<th>Development Artifacts</th>
<th>Tools</th>
<th>SAP HANA Information Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL</td>
<td>Database elements, services, policies, extended storage/index, adapters and agents, remote sources and subscriptions, search, text mining and analysis, data streams (filters aggregators, and enrichment), information management, geo-spatial data *</td>
<td>Eclipse (SAP HANA studio) SAP HANA Web Workbench SAP Web IDE for SAP HANA</td>
<td>SQL and System Views Reference Text Analysis Developer Guide * Text Mining Developer Guide * Smart Data Streaming Developer Guide *</td>
</tr>
<tr>
<td>SQLScript</td>
<td>Tables, SQL Views, Procedures, UDFs, application &amp; business functions...</td>
<td>Eclipse (SAP HANA studio) SAP HANA Web Workbench</td>
<td>SQLScript Reference BFL Reference PAL Reference</td>
</tr>
<tr>
<td>SAP HANA DB (.hdb syntax; hdbtable, hdbview...)</td>
<td>Tables, SQL Views, Procedures, Search index (InA) ...</td>
<td>Eclipse (SAP HANA studio) SAP HANA Web Workbench</td>
<td>Developer Guide (XS classic) SQLScript Reference</td>
</tr>
<tr>
<td>SAP HANA DB (CDS syntax)</td>
<td>Entities, data types, contexts, SQL views, geo-spatial *, ...</td>
<td>Eclipse (SAP HANA studio) SAP HANA Web Workbench</td>
<td>Developer Guide (XS classic) Developer Guide (for XS advanced) Spatial Reference</td>
</tr>
<tr>
<td>SAP HANA HDI</td>
<td>Tables, indexes, data types, procedures, SQL views, triggers, calculation views, analytic privileges, ...</td>
<td>SAP Web IDE for SAP HANA</td>
<td>Developer Guide (for XS advanced)</td>
</tr>
<tr>
<td>MDX</td>
<td>Analytics, BIMC tables and views</td>
<td>Eclipse (SAP HANA studio) SAP HANA Web Workbench SAP Web IDE for SAP HANA</td>
<td>SAP HANA Analytics Catalog (BIMC Views) Reference</td>
</tr>
</tbody>
</table>
Caution

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

Application Development Scenarios

As an application developer, you use a variety of languages to develop applications that expose a SAP HANA data model to requests from UI clients. In an application-development scenario, developers typically use languages such as server-side JavaScript (XSJS) or an OData service to define the application business model that exposes the data model built by the database developer. You can call the application service from a client interface, for example, a browser or UI client. If you want to develop an application service that exposes an SAP HANA data model to client requests, use the following table to help find the information you need to complete the most common development tasks.

Table 6: Information by Application-Development Scenario

<table>
<thead>
<tr>
<th>Programming Language</th>
<th>Development Artifacts</th>
<th>Tools</th>
<th>Information Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLScript</td>
<td>SQLScript</td>
<td>Eclipse (SAP HANA studio)</td>
<td>Developer Guide</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SAP HANA Web Workbench</td>
<td>SQLScript Reference</td>
</tr>
<tr>
<td>XSJS (server-side Java-</td>
<td>Server-side JavaScript services, libraries, API</td>
<td>Eclipse (SAP HANA studio)</td>
<td>Developer Guide (for XS classic)</td>
</tr>
<tr>
<td>Script)</td>
<td></td>
<td>SAP HANA Web Workbench</td>
<td>XS JavaScript API Reference</td>
</tr>
<tr>
<td>Node.js</td>
<td>Server-side services, modules, libraries, API</td>
<td>SAP Web IDE for SAP HANA</td>
<td>Developer Guide (for XS advanced)</td>
</tr>
<tr>
<td>Java</td>
<td>Server-side services, libraries, API</td>
<td>SAP Web IDE for SAP HANA</td>
<td>Developer Guide (for XS advanced)</td>
</tr>
<tr>
<td>OData</td>
<td>OData services, query options, parameters</td>
<td>Eclipse (SAP HANA studio)</td>
<td>Developer Guide</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SAP HANA Web IDE</td>
<td>OData Reference</td>
</tr>
</tbody>
</table>
UI Client Development Scenarios

As a developer of client applications, you use a variety of languages to develop a user interface (UI) client that displays permitted elements of an SAP HANA data model. In a UI-client development scenario, developers typically use languages such as SAPUI5 (HTML5) or JavaScript to define the UI client application. The UI client binds interface controls to actions that request data and display it in the required format. If you want to develop a UI client application that can be used to display an SAP HANA data model, use the following table to help find the information you need to complete the most common development tasks.

Table 7: Information by UI-Client Development Scenario

<table>
<thead>
<tr>
<th>Programming Language</th>
<th>Development Artifacts</th>
<th>Tools</th>
<th>Information Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAPUI5</td>
<td>JS, UI5 Library, View, Control, ...</td>
<td>Eclipse (SAP HANA studio)</td>
<td>Developer Guide</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SAPUI5 Demo Kit and Documentation</td>
</tr>
<tr>
<td>JavaScript</td>
<td>Search queries, results, suggestions</td>
<td>Eclipse (SAP HANA studio)</td>
<td>Developer Guide</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SINA Search JavaScript Reference *</td>
</tr>
</tbody>
</table>

Caution

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

1.4.3.1 Database Development Scenarios

The focus of the database developer is primarily on the underlying data model which the application services expose to UI clients.

The database developer defines the data-persistence and analytic models that are used to expose data in response to client requests via HTTP. The following table lists some of the tasks typically performed by the database developer and indicates where to find the information that is required to perform the task.
<table>
<thead>
<tr>
<th>Task</th>
<th>Details</th>
<th>Information Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create tables, SQL views, sequences...</td>
<td>Code, syntax, ...</td>
<td>SQLScript Reference</td>
</tr>
<tr>
<td></td>
<td></td>
<td><a href="http://help.sap.com/hana/">http://help.sap.com/hana/</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SAP_HANA_SQL_and_System_Views_Reference_en.pdf</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Developer Guide</td>
</tr>
<tr>
<td>Packaging, activation, implementation, ...</td>
<td></td>
<td>Developer Guide</td>
</tr>
<tr>
<td>Create attribute, analytic, calculation views</td>
<td>Code, syntax, ...</td>
<td>SQLScript Reference</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modeling Guide</td>
</tr>
<tr>
<td>Packaging, activation, implementation, ...</td>
<td></td>
<td>Developer Guide</td>
</tr>
<tr>
<td>Examples, background</td>
<td></td>
<td>Modeling Guide</td>
</tr>
<tr>
<td>Create/Write SQLScript procedures, UDFs, triggers...</td>
<td>Code, syntax, ...</td>
<td>SQLScript Reference</td>
</tr>
<tr>
<td></td>
<td></td>
<td><a href="http://help.sap.com/hana/">http://help.sap.com/hana/</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SAP_HANA_SQL_and_System_Views_Reference_en.pdf</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Developer Guide</td>
</tr>
<tr>
<td>Packaging, activation, implementation, ...</td>
<td></td>
<td>Developer Guide</td>
</tr>
<tr>
<td>Create/Use application functions</td>
<td>Code, syntax, ...</td>
<td>SQLScript Reference</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BFL Reference (*)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PAL Reference (*)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Developer Guide</td>
</tr>
<tr>
<td>Packaging, activation, implementation, ...</td>
<td></td>
<td>Developer Guide</td>
</tr>
</tbody>
</table>

**Caution**

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

HDBTable is a language syntax that can be used to define a design-time representation of the artifacts that comprise the persistent data models in SAP HANA.

In SAP HANA Extended Application Services (SAP HANA XS), the persistence model defines the schema, tables, and views that specify what data to make accessible and how. The persistence model is mapped to the consumption model that is exposed to client applications and users, so that data can be analyzed and displayed.

SAP HANA XS enables you to create database schema, tables, views, and sequences as design-time files in the repository. Repository files can be read by applications that you develop.

**Note**

All repository files including your view definition can be transported (along with tables, schema, and sequences) to other SAP HANA systems, for example, in a delivery unit. A delivery unit is the medium SAP HANA provides to enable you to assemble all your application-related repository artifacts together into an archive that can be easily exported to other systems.

You can also set up data-provisioning rules and save them as design-time objects so that they can be included in the delivery unit that you transport between systems.

As part of the process of setting up the basic persistence model for SAP HANA XS, you perform the following tasks:

<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create a schema</td>
<td>Define a design-time schema and maintain the schema definition in the repository; the transportable schema has the file extension .hdbschema, for example, MYSCHEMA.hdbschema.</td>
</tr>
<tr>
<td>Create a synonym</td>
<td>Define a design-time synonym and maintain the synonym definition in the repository; the transportable synonym has the file extension .hdbsynonym, for example, MySynonym.hdbsynonym.</td>
</tr>
<tr>
<td>Create a table</td>
<td>Define a design-time table and maintain the table definition in the repository; the transportable table has the file extension .hdbtable, for example, MYTABLE.hdbtable</td>
</tr>
<tr>
<td>Create a reusable table structure</td>
<td>Define the structure of a database table in a design-time file in the repository; you can reuse the table-structure definition to specify the table type when creating a new table.</td>
</tr>
<tr>
<td>Create a view</td>
<td>Define a design-time view and maintain the view definition in the repository; the transportable view has the file extension .hdbview, for example, MYVIEW.hdbview</td>
</tr>
<tr>
<td>Create a sequence</td>
<td>Define a design-time sequence and maintain the sequence definition in the repository; the transportable sequence has the file extension .hdbsequence, for example, MYSEQUENCE.hdbsequence</td>
</tr>
<tr>
<td>Import table content</td>
<td>Define data-provisioning rules that enable you to import data from comma-separated values (CSV) files into SAP HANA tables using the SAP HANA XS table-import feature; the complete configuration can be included in a delivery unit and transported between SAP HANA systems.</td>
</tr>
</tbody>
</table>
Note

On activation of a repository file, the file suffix, for example, .hdbview, .hdbschema, or .hdbtable, is used to determine which runtime plug-in to call during the activation process. The plug-in reads the repository file selected for activation, for example, a table, sees the object descriptions in the file, and creates the appropriate runtime object.

Related Information

Create a Schema [page 26]
Create a Table [page 30]
Create an SQL View [page 51]
Create a Synonym [page 57]

2.1 Create a Schema

A schema defines the container that holds database objects such as tables, views, and stored procedures.

Prerequisites

To complete this task successfully, note the following prerequisites:

- You must have access to an SAP HANA system.
- You must have already created a development workspace and a project.
- You must have shared the project so that the newly created files can be committed to (and synchronized with) the repository.

Context

This task describes how to create a file containing a schema definition using the hdbschema syntax. Schema definition files are stored in the SAP HANA repository.

Note

A schema generated from an .hdbschema artifact can also be used in the context of Core Data Services (CDS).
To create a schema definition file in the repository, perform the following steps:

**Procedure**

1. Start the SAP HANA studio.
2. Open the **SAP HANA Development** perspective.
3. Open the **Project Explorer** view.
4. Create the schema definition file.
   - Browse to the folder in your project workspace where you want to create the new schema-definition file and perform the following tasks:
     a. Right-click the folder where you want to save the schema-definition file and choose **New > Schema** in the context-sensitive popup menu.
     b. Enter or select the parent folder.
     c. Enter the name of the schema in the **File Name** field.

   **Tip**
   File extensions are important. If you are using SAP HANA Studio to create artifacts in the SAP HANA Repository, the file-creation wizard adds the required file extension automatically and, if appropriate, enables direct editing of the new file in the corresponding editor.

   - d. Select a template to use. Templates contain sample source code to help you.
   - e. Choose **Finish** to save the new schema in the repository.
5. Define the schema name.
   - To edit the schema file, in the **Project Explorer** view double-click the schema file you created in the previous step, for example, **MYSHEMA.hdschema**, and add the schema-definition code to the file:

     ```
     schema_name="MYSHEMA";
     ```

   **Note**
   The following code example is provided for illustration purposes only.


   **Note**
   Saving a file in a shared project automatically commits the saved version of the file to the repository. To explicitly commit a file to the repository, right-click the file (or the project containing the file) and choose **Team ➔ Commit** from the context-sensitive popup menu.

7. Activate the schema.
   - a. Locate and right-click the new schema file in the **Project Explorer** view.
   - b. In the context-sensitive pop-up menu, choose **Team ➔ Activate**.
8. Grant SELECT privileges to the owner of the new schema.
After activation in the repository, the schema object is only visible in the catalog to the _SYS_REPO user. To enable other users, for example the schema owner, to view the newly created schema in the SAP HANA studio's Modeler perspective, you must grant the user the required SELECT privilege.

a. In the SAP HANA studio Systems view, right-click the SAP HANA system hosting the repository where the schema was activated and choose SQL Console in the context-sensitive popup menu.
b. In the SQL console, execute the statement illustrated in the following example, where <SCHEMANAME> is the name of the newly activated schema, and <username> is the database user ID of the schema owner:

```sql
CALL _SYS_REPO.GRANT_SCHEMA_PRIVILEGE_ON_ACTIVATED_CONTENT('select', '<SCHEMANAME>', '<username>');
```

**Related Information**

Schema [page 28]

**2.1.1 Schema**

Relational databases contain a catalog that describes the various elements in the system. The catalog divides the database into sub-databases known as schema. A database schema enables you to logically group together objects such as tables, views, and stored procedures. Without a defined schema, you cannot write to the catalog.

SAP HANA Extended Application Services (SAP HANA XS) enables you to create a database schema as a transportable design-time file in the repository. Repository files can be read by applications that you develop.

If your application refers to the repository (design-time) version of a schema rather than the runtime version in the catalog, for example, by using the explicit path to the repository file (with suffix), any changes to the repository version of the file are visible as soon as they are committed to the repository. There is no need to wait for the repository to activate a runtime version of the schema.

If you want to define a transportable schema using the design-time hdbschema specifications, use the configuration schema illustrated in the following example:

```sql
string schema_name
```

The following example shows the contents of a valid transportable schema-definition file for a schema called MYSHEMA:

```sql
schema_name="MYSHEMA";
```

The schema is stored in the repository with the schema name MYSHEMA as the file name and the suffix .hdbschema, for example, MYSHEMA.hdbschema.
**Note**
A schema generated from an `.hdbschema` artifact can also be used in the context of Core Data Services (CDS).

**Schema Activation**

If you want to create a schema definition as a design-time object, you must create the schema as a flat file. You save the file containing the schema definition with the suffix `.hdbschema` in the appropriate package for your application in the SAP HANA repository. You can activate the design-time objects at any point in time.

**Note**
On activation of a repository file, the file suffix, for example, `.hdbschema`, is used to determine which runtime plugin to call during the activation process. The plug-in reads the repository file selected for activation, parses the object descriptions in the file, and creates the appropriate runtime objects.

If you activate a schema-definition object in SAP HANA, the activation process checks if a schema with the same name already exists in the SAP HANA repository. If a schema with the specified name does not exist, the repository creates a schema with the specified name and makes `_SYS_REPO` the owner of the new schema.

**Note**
The schema cannot be dropped even if the deletion of a schema object is activated.

If you define a schema in SAP HANA XS, note the following important points regarding the schema name:

- **Name mapping**
  The schema name must be identical to the name of the corresponding repository object.

- **Naming conventions**
  The schema name must adhere to the SAP HANA rules for database identifiers. In addition, a schema name must not start with the letters `SAP*`; the `SAP*` namespace is reserved for schemas used by SAP products and applications.

- **Name usage**
  The Data Definition Language (DDL) rendered by the repository contains the schema name as a delimited identifier.

**Related Information**

Create a Schema [page 26]
2.2 Create a Table

SAP HANA Extended Application Services (SAP HANA XS) enables you to create a database table as a design-time file in the repository.

Prerequisites

To complete this task successfully, note the following prerequisites:

- You must have access to an SAP HANA system.
- You must have already created a development workspace and a project.
- You must have shared the project so that the newly created files can be committed to (and synchronized with) the repository.
- You must have created a schema definition `MYSCHEMA.hdbschema`.

Context

This task describes how to create a file containing a table definition using the `hdbtable` syntax. Table definition files are stored in the SAP HANA repository. To create a table file in the repository, perform the following steps:

Procedure

1. Start the SAP HANA studio.
2. Open the SAP HANA Development perspective.
3. Open the Project Explorer view.
4. Create the table definition file.
   Browse to the folder in your project workspace where you want to create the new table file and perform the following steps:
   a. Right-click the folder where you want to save the table file and choose New > Database Table in the context-sensitive popup menu.
   b. Enter or select the parent folder.
   c. Enter the name of the table in the File Name box.

Tip

File extensions are important. If you are using SAP HANA Studio to create artifacts in the SAP HANA Repository, the file-creation wizard adds the required file extension automatically and, if appropriate, enables direct editing of the new file in the corresponding editor.
d. Select a template to use. Templates contain sample source code to help you.
e. Choose Finish to save the new table definition file.

5. Define the table.

To edit the table definition, in the Project Explorer view double-click the table-definition file you created in the previous step, for example, MYTABLE.hdbtable, and add the table-definition code to the file:

```java
Note
The following code example is provided for illustration purposes only.

```java
table.schemaName = "MYSCHEMA";
table.tableType = COLUMNSTORE;
table.columns = [
   {name = "Col1"; sqlType = VARCHAR; nullable = false; length = 20; comment = "dummy comment"},
   {name = "Col2"; sqlType = INTEGER; nullable = false},
   {name = "Col3"; sqlType = NVARCHAR; nullable = true; length = 20; defaultValue = "Defaultvalue"},
   {name = "Col4"; sqlType = DECIMAL; nullable = false; precision = 2; scale = 3}];
table.indexes = [
   {name = "MYINDEX1"; unique = true; indexColumns = ["Col2"]},
   {name = "MYINDEX2"; unique = true; indexColumns = ["Col1", "Col4"]};

table.primaryKey.pkcolumns = ["Col1", "Col2"];
```


```java
Note
Saving a file in a shared project automatically commits the saved version of the file to the repository. To explicitly commit a file to the repository, right-click the file (or the project containing the file) and choose Team Commit from the context-sensitive popup menu.
```

7. Activate the changes in the repository.

a. Locate and right-click the new table file in the Project Explorer view.

b. In the context-sensitive pop-up menu, choose Team Activate.

Related Information

Tables [page 31]
Table Configuration Syntax [page 33]
Create a Schema [page 26]

2.2.1 Tables

In the SAP HANA database, as in other relational databases, a table is a set of data elements that are organized using columns and rows. A database table has a specified number of columns, defined at the time of
table creation, but can have any number of rows. Database tables also typically have meta-data associated with them; the meta-data might include constraints on the table or on the values within particular columns.

SAP HANA Extended Application Services (SAP HANA XS) enables you to create a database table as a design-time file in the repository. All repository files including your table definition can be transported to other SAP HANA systems, for example, in a delivery unit.

**Note**
A delivery unit is the medium SAP HANA provides to enable you to assemble all your application-related repository artifacts together into an archive that can be easily exported to other systems.

If your application is configured to use the design-time version of a database table in the repository rather than the runtime version in the catalog, any changes to the repository version of the table are visible as soon as they are committed to the repository. There is no need to wait for the repository to activate a runtime version of the table.

If you want to define a transportable table using the design-time `.hdbtable` specifications, use the configuration schema illustrated in the following example:

```plaintext
struct TableDefinition {
    string SchemaName;
    optional bool temporary;
    optional TableType tableType;
    optional bool public;
    optional TableLoggingType loggingType;
    list<ColumnDefinition> columns;
    optional list<IndexDefinition> indexes;
    optional PrimaryKeyDefinition primaryKey;
    optional string description
};
```

The following code illustrates a simple example of a design-time table definition:

```plaintext
table.schemaName = "MYSCHEMA";
table.tableType = COLUMNSTORE;
table.columns = [
    {name = "Col1"; sqlType = VARCHAR; nullable = false; length = 20; comment = "dummy comment"},
    {name = "Col2"; sqlType = INTEGER; nullable = false;},
    {name = "Col3"; sqlType = NVARCHAR; nullable = true; length = 20; defaultValue = "Defaultvalue"},
    {name = "Col4"; sqlType = DECIMAL; nullable = false; precision = 2; scale = 3}];
table.indexes = [
    {name = "MYINDEX1"; unique = true; order = DSC; indexColumns = ["Col2"]},
    {name = "MYINDEX2"; unique = true; order = DSC; indexColumns = ["Col1", "Col4"]}];
table.primaryKey.pkcolumns = ["Col1", "Col2"];
```

If you want to create a database table as a repository file, you must create the table as a flat file and save the file containing the table dimensions with the suffix `.hdbtable`, for example, `MYTABLE.hdbtable`. The new file is located in the package hierarchy you establish in the SAP HANA repository. You can activate the repository files at any point in time.

**Note**
On activation of a repository file, the file suffix, for example, `.hdbtable`, is used to determine which runtime plug-in to call during the activation process. The plug-in reads the repository file selected for
Security Considerations

It is important to bear in mind that an incorrectly defined table can lead to security-related problems. If the content of the table you create is used to determine the behavior of the application, for example, whether data is displayed depends on the content of a certain cell, any modification of the table content could help an attacker to obtain elevated privileges. Although you can use authorization settings to restrict the disclosure of information, data-modification issues need to be handled as follows:

- Make sure you specify the field type and define a maximum length for the field
- Avoid using generic types such as VARCHAR or BLOB.
- Keep the field length as short as possible; it is much more difficult to inject shell-code into a string that is 5 characters long than one that can contain up to 255 characters.

Related Information

Table Configuration Syntax [page 33]
Create a Table [page 30]

2.2.2 Table Configuration Syntax

SAP HANA Extended Application Services (SAP HANA XS) enables you to use the hdbtable syntax to create a database table as a design-time file in the repository. The design-time artifact that contains the table definition must adhere to the .hdbtable syntax specified below.

Table Definition

The following code illustrates a simple example of a design-time table definition using the .hdbtable syntax.

```
// Note
// Keywords are case-sensitive, for example, tableType and loggingType, and the schema referenced in the table definition, for example, MYSHEMA, must already exist.

table.schemaName = "MYSHEMA";
table.temporary = true;
table.tableType = COLUMNSTORE;
table.loggingType = NOLOGGING;
table.columns = [
```

Table-Definition Configuration Schema

The following example shows the configuration schema for tables defined using the .hdbtable syntax. Each of the entries in the table-definition configuration schema is explained in more detail in a dedicated section below:

```typescript
struct TableDefinition {
    string SchemaName;
    optional bool temporary;
    optional TableType tableType;
    optional bool public;
    optional TableLoggingType loggingType;
    list<ColumnDefinition> columns;
    optional list<IndexDefinition> indexes;
    optional PrimaryKeyDefinition primaryKey;
    optional string description
};
```

### Schema Name

To use the .hdbtable syntax to specify the name of the schema that contains the table you are defining, use the `schemaName` keyword. In the table definition, the `schemaName` keyword must adhere to the syntax shown in the following example.

```typescript
table.schemaName = "MYSCHEMA";
```

### Temporary

To use the .hdbtable syntax to specify that the table you define is temporary, use the boolean `temporary` keyword. Since data in a temporary table is session-specific, only the owner session of the temporary table is allowed to INSERT/READ/TRUNCATE the data. Temporary tables exist for the duration of the session, and data from the local temporary table is automatically dropped when the session is terminated. In the table definition, the `temporary` keyword must adhere to the syntax shown in the following example.

```typescript
table.temporary = true;
```
Table Type

To specify the table type using the `.hdbtable` syntax, use the `tableType` keyword. In the table definition, the `TableType` keyword must adhere to the syntax shown in the following example.

```
  table.tableType = [COLUMNSTORE | ROWSTORE];
```

The following configuration schema illustrates the parameters you can specify with the `tableType` keyword:

- **COLUMNSTORE**
  Column-oriented storage, where entries of a column are stored in contiguous memory locations. SAP HANA is particularly optimized for column-order storage.

- **ROWSTORE**
  Row-oriented storage, where data is stored in a table as a sequence of records.

Table Logging Type

To enable logging in a table definition using the `.hdbtable` syntax, use the `tableLoggingType` keyword. In the table definition, the `tableLoggingType` keyword must adhere to the syntax shown in the following example.

```
  table.tableLoggingType = [LOGGING | NOLOGGING];
```

Table Column Definition

To define the column structure and type in a table definition using the `.hdbtable` syntax, use the `columns` keyword. In the table definition, the `columns` keyword must adhere to the syntax shown in the following example.

```
  table.columns = [
    {name = "Col1"; sqlType = VARCHAR; nullable = false; length = 20; comment = "dummy comment"},
    {name = "Col2"; sqlType = INTEGER; nullable = false;},
    {name = "Col3"; sqlType = NVARCHAR; nullable = true; length = 20; defaultValue = "Defaultvalue"},
    {name = "Col4"; sqlType = DECIMAL; nullable = false; precision = 2; scale = 3;}
  ];
```

The following configuration schema illustrates the parameters you can specify with the `columns` keyword:

```
struct ColumnDefinition {
  string name;
  SqlDataType sqlType;
  optional bool nullable;
  optional bool unique;
  optional int32 length;
  optional int32 scale;
  optional int32 precision;
  optional string defaultValue;
  optional string comment;
};
```
SQL Data Type

To define the SQL data type for a column in a table using the .hdbtable syntax, use the sqlType keyword. In the table definition, the sqlType keyword must adhere to the syntax shown in the following example.

```javascript
table.columns = [
    {name = "Col1"; sqlType = VARCHAR; nullable = false; length = 20; comment = "dummy comment"},
    ...];
```

The following configuration schema illustrates the data types you can specify with the sqlType keyword:

```javascript
enum SqlDataType {
    DATE; TIME; TIMESTAMP; SECONDDATE; INTEGER; TINYINT;
    SMALLINT; BIGINT; REAL; DOUBLE; FLOAT; SMALLDECIMAL;
    DECIMAL; VARCHAR; NVARCHAR; CLOB; NCLOB;
    ALPHANUM; TEXT; SHORTTEXT; BLOB; VARBINARY;
};
```

Primary Key Definition

To define the primary key for the specified table using the .hdbtable syntax, use the primaryKey and pkcolumns keywords. In the table definition, the primaryKey and pkcolumns keywords must adhere to the syntax shown in the following example.

```javascript
table.primaryKey.pkcolumns = ["Col1", "Col2"];
```

The following configuration schema illustrates the parameters you can specify with the primaryKey keyword:

```javascript
struct PrimaryKeyDefinition {
    list<string> pkcolumns;
    optional IndexType indexType;
};
```

Table Index Definition

To define the index for the specified table using the .hdbtable syntax, use the indexes keyword. In the table definition, the indexes keyword must adhere to the syntax shown in the following example.

```javascript
table.indexes = [
    {name = "MYINDEX1"; unique = true; order = DSC; indexColumns = ["Col2"]},
    {name = "MYINDEX2"; unique = true; order = DSC; indexColumns = ["Col1", "Col4"]};
];
```

You can also use the optional parameter indexType to define the type of index, for example, B_TREE or CPB_TREE, as described in Table Index Type [page 37].
Table Index Type

To define the index type for the specified table using the .hdbtable syntax, use the indexType keyword. In the table definition, the indexType keyword must adhere to the syntax shown in the following example.

```
indexType = [B_TREE | CPB_TREE];
```

B_TREE specifies an index tree of type B+, which maintains sorted data that performs the insertion, deletion, and search of records. CPB_TREE stands for “Compressed Prefix B_TREE” and specifies an index tree of type CPB+, which is based on pkB-tree. CPB_TREE is a very small index that uses a “partial key”, that is; a key that is only part of a full key in index nodes.

**Note**

If neither the B_TREE nor the CPB_TREE type is specified in the table-definition file, SAP HANA chooses the appropriate index type based on the column data type, as follows:

- CPB_TREE
  - Character string types, binary string types, decimal types, when the constraint is a composite key or a non-unique constraint
- B_TREE
  - All column data types other than those specified for CPB_TREE

Table Index Order

To define the order of the table index using the .hdbtable syntax, use the order keyword. Insert the order with the desired value (for example, ascending or descending) in the index type definition; the order keyword must adhere to the syntax shown in the following example.

```
order = [ASC | DSC];
```

You can choose to filter the contents of the table index either in ascending (ASC) or descending (DSC) order.

Complete Table-Definition Configuration Schema

The following example shows the complete configuration schema for tables defined using the .hdbtable syntax.

```
enum TableType {
  COLUMNSTORE; ROWSTORE;
};
enum TableLoggingType {
  LOGGING; NOLOGGING;
};
enum IndexType {
  B_TREE; CPB_TREE;
};
enum Order {
```
ASC; DSC;

};
enum SqlDataType {
    DATE; TIME; TIMESTAMP; SECONDDATE;
    INTEGER; TINYINT; SMALLINT; BIGINT;
    REAL; DOUBLE; FLOAT; SMALLDECIMAL; DECIMAL;
    VARCHAR; NVARCHAR; CLOB; NCLOB;
    ALPHANUM; TEXT; SHORTTEXT; BLOB; VARBINARY;
};
struct PrimaryKeyDefinition {
    list<string> pkcolumns;
    optional IndexType indexType;
};
struct IndexDefinition {
    string name;
    bool unique;
    optional Order order;
    optional IndexType indexType;
    list<string> indexColumns;
};
struct ColumnDefinition {
    string name;
    SqlDataType sqlType;
    optional bool nullable;
    optional bool unique;
    optional int32 length;
    optional int32 scale;
    optional int32 precision;
    optional string defaultValue;
    optional string comment;
};
struct TableDefinition {
    string schemaName;
    optional bool temporary;
    optional TableType tableType;
    optional bool public;
    optional TableLoggingType loggingType;
    list<ColumnDefinition> columns;
    optional list<IndexDefinition> indexes;
    optional PrimaryKeyDefinition primaryKey;
    optional string description;
};
TableDefinition table;

Related Information

Tables [page 31]
Create a Table [page 30]
2.3 Create a Reusable Table Structure

SAP HANA Extended Application Services (SAP HANA XS) enables you to define the structure of a database table in a design-time file in the repository. You can reuse the table-structure definition to specify the table type when creating a new table.

Prerequisites

To complete this task successfully, note the following prerequisites:

- You must have access to an SAP HANA system.
- You must have already created a development workspace and a project.
- You must have shared the project so that the newly created files can be committed to (and synchronized with) the repository.
- You must have created a schema definition MYSCHEMA.hdbschema

Context

This task describes how to create a file containing a table-structure definition using the hdbstructure syntax. Table-structure definition files are stored in the SAP HANA repository with the .hdbstructure file extension, for example, TableStructure.hdbstructure. The primary use case for a design-time representation of a table structure is creating reusable type definitions for procedure interfaces. To create a table-structure file in the repository, perform the following steps:

Procedure

1. Start the SAP HANA studio.
2. Open the SAP HANA Development perspective.
3. Open the Project Explorer view.
4. Create a folder (package) to hold the table-structure definition files.
   - Browse to the folder (package) in your project workspace where you want to create the new folder (package), and perform the following steps:
     a. In the Project Explorer view, right-click the folder where you want to create a new folder called Structures, and choose New Folder in the context-sensitive popup menu.
     b. Enter a name for the new folder in the Folder Name box, for example, Structures.
     c. Choose Finish to create the new Structures folder.
5. Create the table-structure definition file.
   - Browse to the Structures folder (package) in your project workspace and perform the following steps:
a. In the Project Explorer view, right-click the Structures folder you created in the previous step and choose New File in the context-sensitive popup menu.

b. Enter a name for the new table-structure in the File Name box and add the .hdbstructure file extension, for example, TableStructure.hdbstructure.

Tip
File extensions are important. If you are using SAP HANA Studio to create artifacts in the SAP HANA Repository, the file-creation wizard adds the required file extension automatically and, if appropriate, enables direct editing of the new file in the corresponding editor.

c. Choose Finish to save the new table-structure definition file.

6. Define the table structure.
To edit the table-structure definition file, in the Project Explorer view double-click the table file you created in the previous step, for example, TableStructure.hdbstructure, and add the table-structure code to the file:

```javascript
Note
The following code example is provided for illustration purposes only.

table.schemaName = "MYSCHEMA";
table.columns = [
    {name = "Col1"; sqlType = VARCHAR; nullable = false; length = 20; comment = "dummy comment"},
    {name = "Col2"; sqlType = INTEGER; nullable = false;},
    {name = "Col3"; sqlType = NVARCHAR; nullable = true; length = 20; defaultValue = "Defaultvalue"},
    {name = "Col4"; sqlType = DECIMAL; nullable = false; precision = 12; scale = 3;}];
table primaryKey.pkcolumns = ["Col1", "Col2"];
```

7. Save the table-structure definition file.

i Note
Saving a file in a shared project automatically commits the saved version of the file to the repository. To explicitly commit a file to the repository, right-click the file (or the project containing the file) and choose Team Commit from the context-sensitive popup menu.

8. Activate the changes in the repository.
You can activate the changes to the folder structure and the folder contents in one step.

a. In the Project Explorer view, locate and right-click the new folder (Structures) that contains the new table-structure definition file TableStructure.hdbstructure.

b. In the context-sensitive pop-up menu, choose Team Activate.

Activating a table-definition called TableStructure.hdbstructure in the package Structures creates a new table type in SAP HANA, in the same way as the following SQL statement:

```sql
CREATE TABLE "MySchema"."MyTypeTable" like "MySchema"."Structures::TableStructure"
```

9. Check that the new table-type object Structures::TableStructure is added to the catalog.
You can find the new table type in the Systems view under Catalog MYSCHEMA Procedures Table Types.

a. In the SAP HANA Development perspective, open the Systems view.

b. Select the SAP HANA System where the new is located and navigate to the following node: Catalog MYSCHEMA Procedures Table Types.

c. Right-click the new table-structure object and choose Open Definition to display the specifications for the reusable table-structure in the details panel.

d. Check that the entry in the Type box is Table Type.

Related Information

Reusable Table Structures [page 41]
Create a Table [page 30]

2.3.1 Reusable Table Structures

A table-structure definition is a template that you can reuse as a basis for creating new tables of the same type and structure. You can reference the table structure in an SQL statement (CREATE TABLE [...] like [...] ) or an SQLScript procedure.

SAP HANA Extended Application Services (SAP HANA XS) enables you to create a database table structure (or type) as a design-time file in the repository. All repository files including your table-structure definition can be transported to other SAP HANA systems, for example, in a delivery unit. The primary use case for a design-time representation of a table structure is creating reusable table-type definitions for procedure interfaces. However, you can also use table-type definitions in table user-defined functions (UDF).

If you want to define a design-time representation of a table structure with the .hdbstructure specifications, use the configuration schema illustrated in the following example:

```plaintext
struct TableDefinition {
    string SchemaName;
    optional bool public;
    list<ColumnDefinition> columns;
    optional PrimaryKeyDefinition primaryKey;
};
```

Note

The .hdbstructure syntax is a subset of the syntax used in .hdbtable. In a table structure definition, you cannot specify the table type (for example, COLUMN/ROW), define the index, or enable logging.

The following code illustrates a simple example of a design-time table-structure definition:

```plaintext
table.schemaName = "MYSCHEMA";
table.columns = [
    {name = "Col1"; sqlType = VARCHAR; nullable = false; length = 20; comment = "dummy comment"},
];
```
If you want to create a database table structure as a repository file, you must create the table structure as a flat file and save the file containing the structure definition with the .hdbstructure file extension, for example, `TableStructure.hdbstructure`. The new file is located in the package hierarchy you establish in the SAP HANA repository. You can activate the repository files at any point in time.

**i Note**

On activation of a repository file, the file suffix is used to determine which runtime plug-in to call during the activation process. The plug-in reads the repository file selected for activation, in this case a table structure element with the file extension `.hdbstructure`, parses the object descriptions in the file, and creates the appropriate runtime objects.

You can use the SQL command `CREATE TABLE` to create a new table based on the table structure, for example, with the `like` operator, as illustrated in the following example:

```
CREATE TABLE "MySchema"."MyTypeTable" like "MySchema"."Structures::TableStructure"
```

**Related Information**

- Create a Table Structure [page 39]
- Table Configuration Syntax [page 33]

### 2.4 Create a Sequence

A database sequence generates a serial list of unique numbers that you can use while transforming and moving data to between systems.

**Prerequisites**

To complete this task successfully, note the following prerequisites:

- You must have access to an SAP HANA system.
- You must have already created a development workspace and a project.
- You must have shared the project so that the newly created files can be committed to (and synchronized with) the repository.
You must have created a schema definition, for example, `MYSCHEMA.hdbschema`

**Context**

SAP HANA Extended Application Services (SAP HANA XS) enables you to create a database sequence as a design-time file in the repository. This task describes how to create a file containing a sequence definition using the `hdbsequence` syntax.

**Note**

A schema generated from an `.hdbsequence` artifact can also be used in the context of Core Data Services (CDS).

To create a sequence-definition file in the repository, perform the following steps:

**Procedure**

1. Start the SAP HANA studio.
2. Open the SAP HANA Development perspective.
3. Open the Project Explorer view.
4. Create the sequence definition file.

   Browse to the folder in your project workspace where you want to create the new sequence definition file and perform the following tasks:
   a. Right-click the folder where you want to save the sequence definition file and choose **New > Sequence Definition** in the context-sensitive popup menu.
   b. Enter or select the parent folder.
   c. Enter the name of the sequence in the **File Name** box.

   In SAP HANA, sequence-definition files require the file extension `.hdbsequence`, for example, `MySequence.hdbsequence`.

   **Tip**

   File extensions are important. If you are using SAP HANA Studio to create artifacts in the SAP HANA Repository, the file-creation wizard adds the required file extension automatically and, if appropriate, enables direct editing of the new file in the corresponding editor.

   d. Select a template to use. Templates contain sample source code to help you.
   e. Choose **Finish** to save the new sequence in the repository.

5. Define the sequence properties.

   To edit the sequence file, in the Project Explorer view double-click the sequence file you created in the previous step, for example, `MYSEQUENCE.hdbsequence`, and add the sequence code to the file:

   ```
   schema= "MYSCHEMA";
   start_with= 10;
   maxvalue= 30;
   ```
nomaxvalue=false;
minvalue=1;
nominnvalue=true;
cycles=false;
reset_by=
"SELECT T1."Column2" FROM "MYSCHEMA"."com.acme.test.tables::MY_TABLE1" AS T1
LEFT JOIN "MYSCHEMA"."com.acme.test.tables::MY_TABLE2" AS T2 ON T1."Column1" = T2."Column1";
depends_on=["com.acme.test.tables::MY_TABLE1",
"com.acme.test.tables::MY_TABLE2"];

6. Save the sequence-definition file.

i Note
Saving a file in a shared project automatically commits the saved version of the file to the repository. To
explicitly commit a file to the repository, right-click the file (or the project containing the file) and
choose Team Commit from the context-sensitive popup menu.

7. Activate the changes in the repository.
   a. Locate and right-click the new sequence file in the Project Explorer view.
   b. In the context-sensitive pop-up menu, choose Team Activate.

Related Information

Sequences [page 44]
Sequence Configuration Syntax [page 46]

2.4.1 Sequences

A sequence is a database object that generates an automatically incremented list of numeric values according
to the rules defined in the sequence specification. The sequence of numeric values is generated in an
ascending or descending order at a defined increment interval, and the numbers generated by a sequence can
be used by applications, for example, to identify the rows and columns of a table.

Sequences are not associated with tables; they are used by applications, which can use CURRVAL in a SQL
statement to get the current value generated by a sequence and NEXTVAL to generate the next value in the
defined sequence. Sequences provide an easy way to generate the unique values that applications use, for
example, to identify a table row or a field. In the sequence specification, you can set options that control the
start and end point of the sequence, the size of the increment size, or the minimum and maximum allowed
value. You can also specify if the sequence should recycle when it reaches the maximum value specified. The
relationship between sequences and tables is controlled by the application. Applications can reference a
sequence object and coordinate the values across multiple rows and tables.

SAP HANA Extended Application Services (SAP HANA XS) enables you to create a database sequence as a
transportable design-time file in the repository. Repository files can be read by applications that you develop.

You can use database sequences to perform the following operations:

- Generate unique, primary key values, for example, to identify the rows and columns of a table
• Coordinate keys across multiple rows or tables

The following example shows the contents of a valid sequence-definition file for a sequence called MYSEQUENCE. Note that, in this example, no increment value is defined, so the default value of 1 (ascend by 1) is assumed. To set a descending sequence of 1, set the increment_by value to -1.

```plaintext
schema= "TEST_DUMMY";
start_with= 10;
maxvalue= 30;
nomaxvalue=false;
minvalue= 1;
nominvalue=true;
cycles= false;
reset_by= "SELECT T1."Column2" FROM \"MYSCHEMA\".
\"com.acme.test.tables::MY_TABLE1\" AS T1 LEFT JOIN \"MYSCHEMA\".
\"com.acme.test.tables::MY_TABLE2\" AS T2 ON T1."Column1" = T2."Column1"";
depends_on=["com.acme.test.tables::MY_TABLE1",
"com.acme.test.tables::MY_TABLE2"];
```

The sequence definition is stored in the repository with the suffix hdbsequence, for example, MYSEQUENCE.hdbsequence.

**Note**

A schema generated from an .hdbsequence artifact can also be used in the context of Core Data Services (CDS).

If you activate a sequence-definition object in SAP HANA XS, the activation process checks if a sequence with the same name already exists in the SAP HANA repository. If a sequence with the specified name does not exist, the repository creates a sequence with the specified name and makes _SYS_REPO the owner of the new sequence.

In a sequence defined using the .hdbsequence syntax, the reset_by keyword enables you to reset the sequence using a query on any view, table or even table function. However, any dependency must be declared explicitly, for example, with the depends_on keyword. The target table or view specified in the depends_on keyword must be mentioned in the SELECT query that defines the reset condition. If the table or view specified in the dependency does not exist, the activation of the object in the repository fails.

**Note**

On initial activation of the sequence definition, no check is performed to establish the existence of the target view (or table) in the dependency; such a check is only made on reactivation of the sequence definition.

---

**Security Considerations**

It is important to bear in mind that an incorrectly defined sequences can lead to security-related problems. For example, if the sequencing process becomes corrupted, it can result in data overwrite. This can happen if the index has a maximum value which rolls-over, or if a defined reset condition is triggered unexpectedly. A roll-over can be achieved by an attacker forcing data to be inserted by flooding the system with requests.
Overwriting log tables is a known practice for deleting traces. To prevent unexpected data overwrite, use the following settings:

- cycles= false
- Avoid using the reset_by feature

**Related Information**

Create a Sequence [page 42]
Sequence Configuration Syntax [page 46]

### 2.4.2 Sequence Configuration Syntax

SAP HANA Extended Application Services (SAP HANA XS) enables you to use the `hdbsequence` syntax to create a database sequence as a design-time file in the repository. The design-time artifact that contains the sequence definition must adhere to the `.hdbsequence` syntax specified below.

**Sequence Definition**

The following code illustrates a simple example of a design-time sequence definition using the `.hdbsequence` syntax.

```plaintext
Note

Keywords are case-sensitive, for example, maxvalue and start_with, and the schema referenced in the table definition, for example, MYSCHEMA, must already exist.

```
Sequence-Definition Configuration Schema

The following example shows the configuration schema for sequences defined using the `.hdbsequence` syntax. Each of the entries in the sequence-definition configuration schema is explained in more detail in a dedicated section below:

```java
string schema;
int32 increment_by(default=1);
int32 start_with(default=-1);
optional int32 maxvalue;
bool nomaxvalue(default=false);
optional int32 minvalue;
bool nominvalue(default=false);
optional bool cycles;
optional string reset_by;
bool public(default=false);
optional string depends_on_table;
optional string depends_on_view;
optional list<string> depends_on;
```

Schema Name

To use the `.hdbsequence` syntax to specify the name of the schema that contains the sequence you are defining, use the `schema` keyword. In the sequence definition, the `schema` keyword must adhere to the syntax shown in the following example.

```java
schema = "MYSCHEMA";
```

Increment Value

To use the `.hdbsequence` syntax to specify that the sequence increments by a defined value, use the `increment_by` keyword. `increment_by` specifies the amount by which the next sequence value is incremented from the last value assigned. The default increment is 1. In the sequence definition, the `increment_by` keyword must adhere to the syntax shown in the following example.

```java
increment_by = 2;
```

To generate a descending sequence, specify a negative value.

**Note**

An error is returned if the `increment_by` value is 0.
Start Value

To use the `.hdbsequence` syntax to specify that the sequence starts with a specific value, use the `start_with` keyword. If you do not specify a value for the `start_with` keyword, the value defined in `minvalue` is used for ascending sequences, and value defined in `maxvalue` is used for descending sequences. In the sequence definition, the `start_with` keyword must adhere to the syntax shown in the following example.

```
start_with= 10;
```

Maximum Value

To use the `.hdbsequence` syntax to specify that the sequence stops at a specific `maximum` value, for example, 30, use the optional keyword `maxvalue`. In the sequence definition, the `maxvalue` keyword must adhere to the syntax shown in the following example.

```
maxvalue= 30;
```

**Note**
The maximum value (`maxvalue`) a sequence can generate must be between -4611686018427387903 and 4611686018427387902.

No Maximum Value

To use the `.hdbsequence` syntax to specify that the sequence does not stop at any specific `maximum` value, use the boolean keyword `nomaxvalue`. When the `nomaxvalue` keyword is used, the maximum value for an `ascending` sequence is 4611686018427387903 and the maximum value for a `descending` sequence is -1. In the sequence definition, the `nomaxvalue` keyword must adhere to the syntax shown in the following example.

```
nomaxvalue= true;
```

**Note**
Note that the default setting for `nomaxvalue` is `false`.


**Minimum Value**

To use the `.hdbsequence` syntax to specify that the sequence stops at a specific minimum value, for example, 1, use the `minvalue` keyword. In the sequence definition, the `minvalue` keyword must adhere to the syntax shown in the following example.

```
minvalue= 1;
```

**Note**
The minimum value (minvalue) a sequence can generate must be between -4611686018427387903 and 4611686018427387902.

---

**No Minimum Value**

To use the `.hdbsequence` syntax to specify that the sequence does not stop at any specific minimum value, use the boolean keyword `nominvalue`. When the `nominvalue` keyword is used, the minimum value for an ascending sequence is 1 and the minimum value for a descending sequence is -4611686018427387903. In the sequence definition, the `nominvalue` keyword must adhere to the syntax shown in the following example.

```
nominvalue= true;
```

**Note**
Note that the default setting `nominvalue is false`.

---

**Cycles**

In a sequence defined using the `.hdbsequence` syntax, the optional boolean keyword `cycles` enables you to specify whether the sequence number will be restarted after it reaches its maximum or minimum value. For example, the sequence restarts with `minvalue after having reached maxvalue` (where `increment_by` is greater than zero (0)) or restarts with `maxvalue after having reached minvalue` (where `increment_by` is less than zero (0)). In the `.hdbsequence` definition, the `cycles` keyword must adhere to the syntax shown in the following example.

```
cycles= false;
```

---

**Reset by Query**

In a sequence defined using the `.hdbsequence` syntax, the `reset_by` keyword enables you to reset the sequence using a query on any view, table or even table function. However, any dependency must be declared...
explicitly, for example, with the depends_on_view or depends_on_table keyword. If the table or view specified in the dependency does not exist, the activation of the sequence object in the repository fails.

In the .hdbsequence definition, the reset_by keyword must adhere to the syntax shown in the following example.

```reset_by= "SELECT "Col2" FROM "MYSCHEMA"."acme.com.test.tables::MY_TABLE" WHERE "Col2"='12'';```

During a restart of the database, the system automatically executes the reset_by statement and the sequence value is restarted with the value determined from the reset_by subquery.

**Note**

If reset_by is not specified, the sequence value is stored persistently in the database. During the restart of the database, the next value of the sequence is generated from the saved sequence value.

### Depends on

In a sequence defined using the .hdbsequence syntax, the optional keyword depends_on enables you to define a dependency to one or more specific tables or views, for example when using the reset_by option to specify the query to use when resetting the sequence. In the .hdbsequence definition, the depends_on keyword must adhere to the syntax shown in the following example.

```depends_on= ["<repository.package.path>::<MY_TABLE_NAME1>","<repository.package.path>::<MY_VIEW_NAME1>"];```

**Note**

The depends_on keyword replaces and extends the keywords depends_on_table and depends_on_view.

For example, to specify multiple tables and views with the depends_on keyword, use a comma-separated list enclosed in square brackets []

```depends_on= ["com.acme.test.tables::MY_TABLE1", "com.acme.test.tables::MY_TABLE2", "com.acme.test.views::MY_VIEW1"];```

The target table or view specified in the depends_on keyword must be mentioned in the SELECT query that defines the reset condition. On initial activation of the sequence definition, no check is performed to establish the existence of the target table or view specified in the dependency; such a check is only made during reactivation of the sequence definition. If one or more of the target tables or views specified in the dependency does not exist, the re-activation of the sequence object in the repository fails.

### Related Information

Create a Sequence [page 42]
**2.5 Create an SQL View**

A view is a virtual table based on the dynamic results returned in response to an SQL statement. SAP HANA Extended Application Services (SAP HANA XS) enables you to create a database view as a design-time file in the repository.

**Prerequisites**

To complete this task successfully, note the following prerequisites:

- You must have access to an SAP HANA system.
- You must have already created a development workspace and a project.
- You must have shared the project so that the newly created files can be committed to (and synchronized with) the repository.
- You must have created a schema definition, for example, MYSCHEMA.hdbschema

**Context**

This task describes how to create a file containing an SQL view definition using the hdbview syntax. SQL view-definition files are stored in the SAP HANA repository. To create an SQL view-definition file in the repository, perform the following steps:

**Procedure**

1. Start the SAP HANA studio.
2. Open the SAP HANA Development perspective.
3. Open the Project Explorer view.
4. Create the view definition file.
   a. Right-click the folder where you want to save the view-definition file and choose New in the context-sensitive popup menu.
   b. Enter the name of the view-definition file in the File Name box, for example, MYVIEW.hdbview.
Tip

File extensions are important. If you are using SAP HANA Studio to create artifacts in the SAP HANA Repository, the file-creation wizard adds the required file extension automatically and, if appropriate, enables direct editing of the new file in the corresponding editor.

c. Select a template to use. Templates contain sample source code to help you.
d. Choose Finish to save the new view-definition file in the repository.

5. Define the view.

If the new view-definition file is not automatically displayed by the file-creation wizard, in the Project Explorer view double-click the view-definition file you created in the previous step, for example, MYVIEW.hdbview, and add the view definition code to the file replacing object names and paths to suit your requirements:

Note

The following code example is provided for illustration purposes only.

```sql
schema="MYSCHEMA";
query="SELECT T1."Column2" FROM "MYSCHEMA".
"acme.com.test.views::MY_VIEW1" AS T1 LEFT JOIN "MYSCHEMA".
"acme.com.test.views::MY_VIEW2" AS T2 ON T1."Column1" = T2."Column1";
depends_on=["acme.com.test.views::MY_VIEW1", "acme.com.test.views::MY_VIEW2"];```

6. Save the SQL view-definition file.

Note

Saving a file in a shared project automatically commits the saved version of the file to the repository. To explicitly commit a file to the repository, right-click the file (or the project containing the file) and choose Team Commit from the context-sensitive popup menu.

7. Activate the changes in the repository.

   a. Locate and right-click the new view-definition file in the Project Explorer view.
   b. In the context-sensitive pop-up menu, choose Team Activate

Related Information

SQL Views [page 53]
SQL View Configuration Syntax [page 54]
2.5.1 SQL Views

In SQL, a view is a virtual table based on the dynamic results returned in response to an SQL statement. Every time a user queries an SQL view, the database uses the view’s SQL statement to recreate the data specified in the SQL view. The data displayed in an SQL view can be extracted from one or more database tables.

An SQL view contains rows and columns, just like a real database table; the fields in an SQL view are fields from one or more real tables in the database. You can add SQL functions, for example, WHERE or JOIN statements, to a view and present the resulting data as if it were coming from one, single table.

SAP HANA Extended Application Services (SAP HANA XS) enables you to create a database view as a design-time file in the repository. Repository files can be read by applications that you develop. In addition, all repository files including your view definition can be transported to other SAP HANA systems, for example, in a delivery unit.

If your application refers to the design-time version of a view from the repository rather than the runtime version in the catalog, for example, by using the explicit path to the repository file (with suffix), any changes to the repository version of the file are visible as soon as they are committed to the repository. There is no need to wait for the repository to activate a runtime version of the view.

The following example shows the contents of a valid transportable view-definition file for a view called MYVIEW:

```
schema="MYSCHEMA";
query="SELECT T1."Column2" FROM "MYSCHEMA"."acme.com.test.views::MY_VIEW1" AS T1 LEFT JOIN "MYSCHEMA"."acme.com.test.views::MY_VIEW2" AS T2 ON T1."Column1" = T2."Column1";
depends_on=["acme.com.test.views::MY_VIEW1", "acme.com.test.views::MY_VIEW2"];```

If you want to create a view definition as a design-time object, you must create the view as a flat file and save the file containing the view definition with the suffix .hdbview, for example, MYVIEW.hdbview in the appropriate package in the package hierarchy established for your application in the SAP HANA repository. You can activate the design-time object at any point in time.

**Tip**

On activation of a repository file, the file suffix (for example, .hdbview) is used to determine which runtime plugin to call during the activation process. The plug-in reads the repository file selected for activation, parses the object descriptions in the file, and creates the appropriate runtime objects.

In an SQL view defined using the .hdbview syntax, any dependency to another table or view must be declared explicitly, for example, with the depends_on keyword. The target view or table specified in the depends_on keyword must also be mentioned in the SELECT query that defines the SQL view. If one of more of the tables or views specified in the dependency does not exist, the activation of the object in the repository fails.

**i Note**

On initial activation of the SQL view, no check is performed to establish the existence of the target view (or table) in the depends_on dependency: such a check is only made on reactivation of the SQL view.
Column Names in a View

If you want to assign names to the columns in a view, use the SQL query in the .hdbview file. In this example of design-time view definition, the following names are specified for columns defined in the view:

- idea_id
- identity_id
- role_id

```sql
schema = "MYSCHEMA";
query = "SELECT role_join.idea_id AS idea_id, ident.member_id AS identity_id, 
role_join.role_id AS role_id 
FROM "acme.com.odin.db.iam::t_identity_group_member_transitive" AS ident 
INNER JOIN "acme.com.odin.db.idea::t_idea_identity_role" AS role_join 
ON role_join.identity_id = ident.group_id UNION DISTINCT 
SELECT idea_id, identity_id, role_id 
FROM "acme.com.odin.db.idea::t_idea_identity_role" 
WITH read only";
```

Related Information

Create an SQL View [page 51]

2.5.2 SQL View Configuration Syntax

SAP HANA Extended Application Services (SAP HANA XS) enables you to use the hdbview syntax to create an SQL view as a design-time file in the repository. The design-time artifact that contains the SQL view definition must adhere to the .hdbview syntax specified below.

SQL View Definition

The following code illustrates a simple example of a design-time definition of an SQL view using the .hdbview syntax.

```sql
Note

Keywords are case-sensitive, for example, schema and query, and the schema referenced in the table definition, for example, MYSCHEMA, must already exist.

```

```sql
schema="MYSCHEMA";
public=false
query="SELECT T1."Column2" FROM "MYSCHEMA".
"acme.com.test.tables::MY_TABLE1" AS T1 LEFT JOIN "MYSCHEMA".
"acme.com.test.views::MY_VIEW1" AS T2 ON T1."Column1" = T2."Column1"";
```
depends_on= "acme.com.test.tables::MY_TABLE1","acme.com.test.views::MY_VIEW1";

SQL View Configuration Schema

The following example shows the configuration schema for an SQL view that you define using the .hdbview syntax. Each of the entries in the view-definition configuration schema is explained in more detail in a dedicated section below:

```cpp
string schema;
string query;
bool public(default=true);
optional list<string> depends_on_table;
optional list<string> depends_on_view;
```

Schema Name

To use the .hdbview syntax to specify the name of the schema that contains the SQL view you are defining, use the schema keyword. In the SQL view definition, the schema keyword must adhere to the syntax shown in the following example.

```python
schema= "MYSCHEMA";
```

query

To use the .hdbview syntax to specify the query that creates the SQL view you are defining, use the query keyword. In the SQL view definition, the query keyword must adhere to the syntax shown in the following example.

```sql
query="SELECT * FROM "<MY_SCHEMA_NAME>"."
"<repository.package.path>::<MY_TABLE_NAME>"";
```

For example:

```sql
query="SELECT * FROM "MY_SCHEMA".\"com.test.tables::02_HDB_DEPARTMENT_VIEW\"";
```

public

To use the .hdbview syntax to specify whether or not the SQL view you are defining is publicly available, use the boolean keyword public. In the SQL view definition, the public keyword must adhere to the syntax shown in the following example.

```python
public=[false|true];
```
For example:

| public=false |

**Note**

The default value for the `public` keyword is `true`.

---

### Depends on

In an SQL view defined using the `.hdbview` syntax, the optional keyword `depends_on` enables you to define a dependency to one or more tables or views. In the `.hdbview` definition, the `depends_on` keyword must adhere to the syntax shown in the following example.

```
depends_on= ["<repository.package.path>::MY_TABLE_NAME1","<repository.package.path>::MY_VIEW_NAME1"];
```

**Note**

The `depends_on` keyword replaces and extends the keywords `depends_on_table` and `depends_on_view`.

For example, to specify multiple tables and views with the `depends_on` keyword, use a comma-separated list enclosed in square brackets {}.

```
depends_on= ["acme.com.test.tables::MY_TABLE1","acme.com.test.views::MY_VIEW1"];  
```

The target table or view specified in the `depends_on` keyword must be mentioned in the `SELECT` query that defines the SQL view. On initial activation of the SQL view, no check is performed to establish the existence of the target tables or views specified in the dependency; such a check is only made during reactivation of the SQL view. If one or more of the target tables or views specified in the dependency does not exist, the reactivation of the SQL view object in the repository fails.

---

### Related Information

Create an SQL View [page 51]
2.6 Create a Synonym

Extended Application Services (SAP HANA XS) enables you to create a local database synonym as a design-time file in the repository.

Prerequisites

To complete this task successfully, note the following prerequisites:

- You must have access to an SAP HANA system.
- You must have already created a development workspace and a project.
- (SAP HANA studio only) You must have shared the project so that the newly created files can be committed to (and synchronized with) the repository.

Context

In SAP HANA, a design-time synonym artifact has the suffix .hdbsynonym and defines the target object by specifying an authoring schema and an object name; its activation evaluates a system’s schema mapping to determine the physical schema in which the target table is expected, and creates a local synonym that points to this object.

Restriction

A design-time synonym cannot refer to another synonym, and you cannot define multiple synonyms in a single design-time synonym artifact. In addition, the target object specified in a design-time synonym must only exist in the catalog; it is not possible to use .hdbsynonym to define a synonym for a catalog object that originates from a design-time artifact.

Procedure

1. Start the SAP HANA studio.
   a. Open the SAP HANA Development perspective.
   b. Open the Project Explorer view.
2. Create the synonym definition file.
   Browse to the folder in your project workspace where you want to create the new synonym-definition file and perform the following steps:
   
   To generate a synonym called "acme.com.appl1::MySynonym1", you must create a design-time synonym artifact called MySynonym1.hdbsynonym in the repository package acme.com.appl1; the first line of the design-time synonym artifact must be specified as illustrated in the following example.
a. Right-click the folder where you want to create the synonym-definition file and choose ➤ **New** ➤ **General ➤ File** in the context-sensitive popup menu.

b. Enter the name of the new synonym-definition file in the **File Name** box and add the appropriate extension, for example, *MySynonym1.hdbsynonym*.

c. Choose **Finish** to save the new synonym definition file.

3. Define the synonym.

To edit the synonym definition, in the **Project Explorer** view double-click the synonym-definition file you created in the previous step, for example, *MySynonym.hdbsynonym*, and add the synonym-definition code to the new file, as illustrated in the following example.

```json
{ "acme.com.app1::MySynonym1" : { ...
```

4. Save and activate the changes in the repository.

a. Locate and right-click the new synonym-definition file in the **Project Explorer** view.

b. In the context-sensitive pop-up menu, choose ➤ **Team ➤ Activate**.

**Related Information**

- Synonyms [page 59]
- Synonym Configuration Syntax [page 60]
- Schema [page 28]
2.6.1 Synonyms

SAP HANA Extended Application Services (SAP HANA XS) enables you to create a design-time representation of a local database synonym. The synonym enables you to refer to a table (for example, from a view) that only exists as a catalog object.

In SAP HANA XS, a design-time representation of a local synonym has the suffix .hdbsynonym that you can store in the SAP HANA repository. The syntax of the design-time synonym artifact requires you to define the target object by specifying an authoring schema and an object name. You also need to specify the schema in which to create the new synonym. On activation of a design-time synonym artifact, SAP HANA XS evaluates a system's schema mapping to determine the physical schema in which the target table is expected, and creates a local synonym in the specified schema which points to this object. You can use this type of synonym if you need to define a CDS view that refers to a table which only exists in the catalog; that is, the catalog table has no design-time representation.

⚠️ Restriction

A synonym cannot refer to another synonym, and you cannot define multiple synonyms in a single design-time synonym artifact. In addition, the target object specified in a design-time synonym must only exist in the catalog; it is not possible to define a define-time synonym for a catalog object that originates from a design-time artifact.

In the following example of a design-time synonym artifact, the table MY_ERP_TABLE_1 resides in the schema DEFAULT_SCHEMA. The activation of the design-time synonym artifact illustrated in the example would generate a local synonym ("acme.com.appl1::MySynonym1") in the schema SCHEMA_2. Assuming that a schema-mapping table exists that maps DEFAULT_SCHEMA to the schema SAP_SCHEMA, the newly generated synonym "SCHEMA_2"."acme.com.appl1::MySynonym1" points to the run-time object "SAP_SCHEMA"."MY_ERP_TABLE_1".

 millennial Synonym1.hdbsynonym

```json
{ "acme.com.appl1::MySynonym1" : {
    "target" : {
        "schema": "DEFAULT_SCHEMA",
        "object": "MY_ERP_TABLE_1"
    },
    "schema": "SCHEMA_2"
}}
```

Tip

To generate a synonym called "acme.com.appl1::MySynonym1", a design-time artifact called MySynonym1.hdbsynonym must exist in the repository package acme.com.appl1; the first line of the design-time synonym artifact must be specified as illustrated in the example above.
2.6.2 Synonym Configuration Syntax

A specific syntax is required to create a design-time representation of a local database synonym in SAP HANA Extended Application Services.

Synonym Definition

SAP HANA Extended Application Services (SAP HANA XS) enables you to use the hdsynonym syntax to create a database synonym as a design-time file in the repository. On activation, a local synonym is generated in the catalog in the specified schema. The design-time artifact that contains the synonym definition must adhere to the .hdsynonym syntax specified below.

Note

The activation of the design-time synonym artifact illustrated in the following example generates a local synonym ("acme.com.app1::MySynonym1") in the schema SCHEMA_2.

Sample Code

MySynonym1.hdsynonym

```json
{ "acme.com.app1::MySynonym1" : {
    "target" : {
        "schema": "DEFAULT_SCHEMA",
        "object": "MY_ERP_TABLE_1"
    },
    schema [page 61]": "SCHEMA_2"
}
}
```

Synonym Location

In the first line of the synonym-definition file, you must specify the absolute repository path to the package containing the synonym artifact (and the name of the synonym artifact) itself using the syntax illustrated in the following example.
For example, to generate a synonym called "acme.com.app1::MySynonym1", you must create a design-time artifact called MySynonym1.hdbsynonym in the repository package acme.com.app1; the first line of the design-time synonym artifact must be specified as illustrated in the following example.

```json
{ "acme.com.app1::MySynonym1" : {...}}
```

**target**

To specify the name and location of the object for which you are defining a synonym, use the `target` keyword together with the keywords `schema` and `object`. In the synonym definition, the `target` keyword must adhere to the syntax shown in the following example.

```json
"target" : {
    "schema": "<Name_of_schema_containing_<"object">",
    "object": "<Name_of_target_object>"
},
```

In the context of the `target` keyword, the following additional keywords are required:

- `schema` defines the name of the schema where the target object (defined in `object`) is located.
- `object` specifies the name of the catalog object to which the synonym applies.

**Restriction**

The target object specified in a design-time synonym must only exist in the catalog; it is not possible to define a design-time synonym for a catalog object that originates from a design-time artifact.

**schema**

To specify the catalog location of the generated synonym, use the `schema` keyword. In the synonym definition, the `schema` keyword must adhere to the syntax shown in the following example.

```json
"schema": "<Schema_location_of_generated_synonym>"
```
2.7 Import Data with hdbtable Table-Import

The table-import function is a data-provisioning tool that enables you to import data from comma-separated values (CSV) files into SAP HANA database tables.

Prerequisites

Before you start this task, make sure that the following prerequisites are met:

- An SAP HANA database instance is available.
- The SAP HANA database client is installed and configured.
- You have a database user account set up with the roles containing sufficient privileges to perform actions in the repository, for example, add packages, add objects, and so on.
- The SAP HANA studio is installed and connected to the SAP HANA repository.
- You have a development environment including a repository workspace, a package structure for your application, and a shared project to enable you to synchronize changes to the project files in the local file system with the repository.

Context

In this tutorial, you import data from a CSV file into a table generated from a design-time definition that uses the .hdbtable syntax. The names used in the following task are for illustration purposes only; where necessary, replace the names of schema, tables, files, and so on shown in the following examples with your own names.

Tip

File extensions are important. If you are using SAP HANA Studio to create artifacts in the SAP HANA Repository, the file-creation wizard adds the required file extension automatically and, if appropriate, enables direct editing of the new file in the corresponding editor.
Procedure

1. Create a root package for your table-import application.
   In SAP HANA studio, open the SAP HANA Development perspective and perform the following steps:
   a. In the package hierarchy displayed in the Systems view, right-click the package where you want to create the new package for your table-import configuration and choose New > Package...
   b. Enter a name for your package, for example TiTest. You must create the new TiTest package in your own namespace, for example mycompany.tests.TiTest

   i Note
   Naming conventions exist for package names, for example, a package name must not start with either a dot (.) or a hyphen (-) and cannot contain two or more consecutive dots (..). In addition, the name must not exceed 190 characters.

   a. Choose OK to create the new package.

2. Create a set of table-import files.

   The following files are required for a table import scenario.

   i Note
   For the purposes of this tutorial, the following files must all be created in the same package, for example, a package called TiTest. However, the table-import feature also allows you to use files distributed in different packages.

   ○ The table-import configuration file, for example, TiConfiguration.hdbti
     Specifies the source file containing the data values to import and the target table in SAP HANA into which the data must be inserted
   ○ A CSV file, for example, myTiData.csv
     Contains the data to be imported into the SAP HANA table during the table-import operation; values in the .csv file can be separated either by a comma (,) or a semi-colon (;).
   ○ A target table.
     The target table can be either a runtime table in the catalog or a table definition, for example, a table defined using the .hdbtable syntax (TiTable.hdbtable) or the CDS-compliant .hdbdd syntax (TiTable.hdbdd).

   i Note
   In this tutorial, the target table for the table-import operation is TiTable.hdbtable, a design-time table defined using the .hdbtable syntax.

   ○ The schema definition, for example, TISCEMA.hdbschema
     Specifies the name of the schema in which the target import table is created

   When all the necessary files are available, you can import data from a source file, such as a CSV file, into the desired target table.

3. Using any code editor, create or open the schema definition (AMT.hdbschema) file and enter the name of the schema you want to use to contain the target table.

   schema_name="AMT";
4. Create or open the table definition file for the target import table (inhabitants.hdbtable) and enter the following lines of text; this example uses the .hdbtable syntax.

```plaintext
table.schemaName = "AMT";
table.tableType = COLUMNSTORE;
table.columns =
[  
{name = "ID"; sqlType = VARCHAR; nullable = false; length = 20; comment = ""},
{name = "surname"; sqlType = VARCHAR; nullable = true; length = 30; comment = ""},
{name = "name"; sqlType = VARCHAR; nullable = true; length = 30; comment = ""},
{name = "city"; sqlType = VARCHAR; nullable = true; length = 30; comment = ""}],
table.primaryKey.pkcolumns = ["ID"];
```

5. Open the CSV file containing the data to import, for example, inhabitants.csv in a text editor and enter the values shown in the following example.

```
0,Annan,Kwesi,Accra
1,Essuman,Wiredu,Tema
2,Tetteh,Kwame,Kumasi
3,Nterful,Akye,Tarkwa
4,Acheampong,Kojo,Tamale
5,Assamoah,Adjoa,Takoradi
6,Mensah,Afua,Cape Coast
```

**i** Note

You can import data from multiple .csv files in a single, table-import operation. However, each .csv file must be specified in a separate code block (table= ...) in the table-import configuration file.

6. Create a table import configuration file.

To create a table import configuration file, perform the following steps:

**i** Note

You can also open and use an existing table-import configuration file (for example, inhabitants.hdbti).

a. Right-click the folder where you want to save the table file and choose New > Table Import Configuration in the context-sensitive popup menu.

b. Enter or select the parent folder, where the table-import configuration file will reside.

c. Using the wizard, enter the name of the table-import configuration in the File Name field, for example, MyTableConfiguration.

This creates the file MyTableConfiguration.hdbti.

**Tip**

File extensions are important. If you are using SAP HANA Studio to create artifacts in the SAP HANA Repository, the file-creation wizard adds the required file extension automatically and, if appropriate, enables direct editing of the new file in the corresponding editor.
d. Edit the details of the new table-import configuration in the new (or existing) table-import configuration file.

Enter the following lines of text in the table-import configuration file.

```json
import = [
  {
    table = "mycompany.tests.TiTest::inhabitants";
    schema = "AMT";
    file = "mycompany.tests.TiTest:inhabitants.csv";
    header = false;
  }
];
```

e. Choose Finish to save the table-import configuration.

7. Deploy the table import.

a. Select the package that you created in the first step, for example, mycompany.tests.TiTest.
b. Click the alternate mouse button and choose Commit.
c. Click the alternate mouse button and choose Activate.

This activates all the repository objects. The result is that the data specified in the CSV file inhabitants.csv is imported into the SAP HANA table inhabitants using the data-import configuration defined in the inhabitants.hdbti table-import configuration file.

8. Check the contents of the runtime table inhabitants in the catalog.

To ensure that the import operation completed as expected, use the SAP HANA studio to view the contents of the runtime table inhabitants in the catalog. You need to confirm that the correct data was imported into the correct columns.

a. In the SAP HANA Development perspective, open the Systems view.
b. Navigate to the catalog location where the inhabitants object resides, for example:

   ```<SID> Catalog AMT Tables```
c. Open a data preview for the updated object.
   Right-click the updated object and choose Open Data Preview in the context-sensitive menu.

### 2.7.1 Data Provisioning Using Table Import

You can import data from comma-separated values (CSV) into the SAP HANA tables using the SAP HANA Extended Application Services (SAP HANA XS) table-import feature.

In SAP HANA XS, you create a table-import scenario by setting up an table-import configuration file and one or more comma-separated value (CSV) files containing the content you want to import into the specified SAP HANA table. The import-configuration file links the import operation to one or more target tables. The table definition (for example, in the form of a .hdbdd or .hdbtable file) can either be created separately or be included in the table-import scenario itself.

To use the SAP HANA XS table-import feature to import data into an SAP HANA table, you need to understand the following table-import concepts:

- **Table-import configuration**
  You define the table-import model in a configuration file that specifies the data fields to import and the target tables for each data field.
Note
The table-import file must have the .hdbti extension, for example, myTableImport.hdbti.

CSV Data File Constraints

The following constraints apply to the CSV file used as a source for the table-import feature in SAP HANA XS:

- The number of table columns must match the number of CSV columns.
- There must not be any incompatibilities between the data types of the table columns and the data types of the CSV columns.
- Overlapping data in data files is not supported.
- The target table of the import must not be modified (or appended to) outside of the data-import operation. If the table is used for storage of application data, this data may be lost during any operation to re-import or update the data.

Related Information

Table-Import Configuration [page 66]
Table-Import Configuration-File Syntax [page 69]

2.7.2 Table-Import Configuration

You can define the elements of a table-import operation in a design-time file; the configuration includes information about source data and the target table in SAP HANA.

SAP HANA Extended Application Services (SAP HANA XS) enables you to perform data-provisioning operations that you define in a design-time configuration file. The configuration file is transportable, which means you can transfer the data-provisioning between SAP HANA systems quickly and easily.

The table-import configuration enables you to specify how data from a comma-separated-value (.csv) file is imported into a target table in SAP HANA. The configuration specifies the source file containing the data values to import and the target table in SAP HANA into which the data must be inserted. As further options, you can specify which field delimiter to use when interpreting data in the source .csv file and if keys must be used to determine which columns in the target table to insert the imported data into.

Note
If you use multiple table import configurations to import data into a single target table, the keys keyword is mandatory. This is to avoid problems relating to the overwriting or accidental deletion of existing data.
The following example of a table-import configuration shows how to define a simple import operation which inserts data from the source files `myData.csv` and `myData2.csv` into the table `myTable` in the schema `mySchema`.

```json
import = [
  {
    table = "myTable";
    schema = "mySchema";
    file = "sap.ti2.demo:myData.csv";
    header = false;
    delimField = ":";
    keys = [ "GROUP_TYPE" : "BW_CUBE" ];
  },
  {
    table = "sap.ti2.demo::myTable";
    file = "sap.ti2.demo:myData2.csv";
    header = false;
    delimField = ":";
    keys = [ "GROUP_TYPE" : "BW_CUBE" ];
  }
];
```

In the table import configuration, you can specify the target table using either of the following methods:

- **Public synonym ("sap.ti2.demo::myTable")**
  If you use the public synonym to reference a target table for the import operation, you must use either the `hdbtable` or `cdstable` keyword, for example, `hdbtable = "sap.ti2.demo::myTable"`;

- **Schema-qualified catalog name ("mySchema", "MyTable")**
  If you use the schema-qualified catalog name to reference a target table for the import operation, you must use the `table` keyword in combination with the `schema` keyword, for example, `table = "myTable"; schema = "mySchema";`

**Note**

Both the schema and the target table specified in the table-import operation must already exist. If either the specified table or the schema does not exist, SAP HANA XS displays an error message during the activation of the configuration file, for example: `Table import target table cannot be found. or Schema could not be resolved.`

You can also use one table-import configuration file to import data from multiple `.csv` source files. However, you must specify each import operation in a new code block introduced by the `[hdb | cds]table` keyword, as illustrated in the example above.

By default, the table-import operation assumes that data values in the `.csv` source file are separated by a comma (,). However, the table-import operation can also interpret files containing data values separated by a semi-colon (;).

- **Comma (,) separated values**
  ```csv
  ,,,BW_CUBE,,40000000,2,40000000,all
  ```

- **Semi-colon (;) separated values**
  ```csv
  ;;;BW_CUBE;;40000000;3;40000000;all
  ```
Note
If the activated .hdbti configuration used to import data is subsequently deleted, only the data that was imported by the deleted .hdbti configuration is dropped from the target table. All other data including any data imported by other .hdbti configurations remains in the table. If the target CDS entity has no key (annotated with @nokey) all data that is not part of the CSV file is dropped from the table during each table-import activation.

You can use the optional keyword keys to specify the key range taken from the source .csv file for import into the target table. If keys are specified for an import in a table import configuration, multiple imports into same target table are checked for potential data collisions.

Note
The configuration-file syntax does not support wildcards in the key definition; the full value of a selectable column value has to be specified.

Security Considerations

In SAP HANA XS, design-time artifacts such as tables (.hdbtable or .hdbdd) and table-import configurations (.hdbti) are not normally exposed to clients via HTTP. However, design-time artifacts containing comma-separated values (.csv) could be considered as potential artifacts to expose to users through HTTP. For this reason, it is essential to protect these exposed .csv artifacts by setting the appropriate application privileges; the application privileges prevents data leakage, for example, by denying access to data by users, who are not normally allowed to see all the records in such tables.

Tip
Place all the .csv files used to import content to into tables together in a single package and set the appropriate (restrictive) application-access permissions for that package, for example, with a dedicated .xsaccess file.

Related Information

Table-Import Configuration-File Syntax [page 69]
2.7.3 Table-Import Configuration-File Syntax

The design-time configuration file used to define a table-import operation requires the use of a specific syntax. The syntax comprises a series of keyword=value pairs.

If you use the table-import configuration syntax to define the details of the table-import operation, you can use the keywords illustrated in the following code example. The resulting design-time file must have the .hdbti file extension, for example, myTableImportCfg.hdbti.

```plaintext
import = [  
  {  
    table = "myTable";
    schema = "mySchema";
    file = "sap.ti2.demo:myData.csv";
    header = false;
    useHeaderNames = false;
    delimField = ";";
    delimEnclosing="\"";
    distinguishEmptyFromNull = true;
    keys = [ "GROUP_TYPE" : "BW_CUBE", "GROUP_TYPE" : "BW_DSO", "GROUP_TYPE" : "BW_PSA" ];
  }
];
```

table

In the table-import configuration, the table, cdstable, and hdbtable keywords enable you to specify the name of the target table into which the table-import operation must insert data. The target table you specify in the table-import configuration can be a runtime table in the catalog or a design-time table definition, for example, a table defined using either the .hdbtable or the .hdbdd (Core Data Services) syntax.

**Note**
The target table specified in the table-import configuration must already exist. If the specified table does not exist, SAP HANA XS displays an error message during the activation of the configuration file, for example: Table import target table cannot be found.

Use the table keyword in the table-import configuration to specify the name of the target table using the qualified name for a catalog table.

```plaintext
table = "target_table";
schema = "mySchema";
```

**Note**
You must also specify the name of the schema in which the target catalog table resides, for example, using the schema keyword.

The hdbtable keyword in the table-import configuration enables you to specify the name of a target table using the public synonym for a design-time table defined with the .hdbtable syntax.

```plaintext
hdbtable = "sap.ti2.demo::target_table";
```
The `cdstable` keyword in the table-import configuration enables you to specify the name of a target table using the public synonym for a design-time table defined with the CDS-compliant `.hdbdd` syntax.

```plaintext
cdstable = "sap.ti2.demo::target_table";
```

**Caution**

There is no explicit check if the addressed table is created using the `.hdbtable` or CDS-compliant `.hdbdd` syntax.

If the table specified with the `cdstable` or `hdbtable` keyword is not defined with the corresponding syntax, SAP HANA displays an error when you try to activate the artifact, for example, `Invalid combination of table declarations found, you may only use [cdstable | hdbtable | table].`

### schema

The following code example shows the syntax required to specify a schema in a table-import configuration.

```plaintext
schema = "TI2_TESTS";
```

**i Note**

The schema specified in the table-import configuration file must already exist.

If the schema specified in a table-import configuration file does not exist, SAP HANA XS displays an error message during the activation of the configuration file, for example:

- Schema could not be resolved.
- If you import into a catalog table, please provide schema.

The `schema` is only required if you use a table's schema-qualified catalog name to reference the target table for an import operation, for example, `table = "myTable"; schema = "mySchema";`. The schema is **not** required if you use a public synonym to reference a table in a table-import configuration, for example, `hdbtable = "sap.ti2.demo::target_table";`.

### file

Use the `file` keyword in the table-import configuration to specify the source file containing the data that the table-import operation imports into the target table. The source file must be a `.csv` file with the data values separated either by a comma (,) or a semi-colon (;). The file definition must also include the full package path in the SAP HANA repository.

```plaintext
file = "sap.ti2.demo:myData.csv";
```
**header**

Use the `header` keyword in the table-import configuration to indicate if the data contained in the specified `.csv` file includes a header line. The `header` keyword is optional, and the possible values are `true` or `false`.

```java
header = false;
```

**useHeaderNames**

Use the `useHeaderNames` keyword in the table-import configuration to indicate if the data contained in the first line of the specified `.csv` file must be interpreted. The `useHeaderNames` keyword is optional; it is used in combination with the `header` keyword. The `useHeaderNames` keyword is boolean; possible values are `true` or `false`.

```java
useHeaderNames = false;
```

**Note**

The `useHeaderNames` keyword only works if `header` is also set to "true".

The table-import process considers the order of the columns; if the column order specified in the `.csv` file does not match the order used for the columns in the target table, an error occurs on activation.

**delimField**

Use the `delimField` keyword in the table-import configuration to specify which character is used to separate the values in the data to be imported. Currently, the table-import operation supports either the comma (,) or the semi-colon (;). The following example shows how to specify that values in the `.csv` source file are separated by a semi-colon (;).

```java
delimField = "";
```

**Note**

By default, the table-import operation assumes that data values in the `.csv` source file are separated by a comma (,). If no delimiter field is specified in the `.hdbti` table-import configuration file, the default setting is assumed.
delimEnclosing

Use the `delimEnclosing` keyword in the table-import configuration to specify a single character that indicates both the start and end of a set of characters to be interpreted as a single value in the `.csv` file, for example “This is all one, single value”. This feature enables you to include in data values in a `.CSV` file even the character defined as the field delimiter (in `delimField`), for example, a comma (,) or a semi-colon (;).

**Tip**

If the value used to separate the data fields in your `.csv` file (for example, the comma (,)) is also used inside the data values themselves (“This, is, a, value”), you must declare and use a delimiter enclosing character and use it to enclose all data values to be imported.

The following example shows how to use the `delimEnclosing` keyword to specify the quote (”) as the delimiting character that indicates both the start and the end of a value in the `.csv` file. Everything enclosed between the `delimEnclosing` characters (in this example, ””) is interpreted by the import process as one, single value.

```sql
DELIMENCLOSING="";
```

**Note**

Since the `hdbti` syntax requires us to use the quotes (""`) to specify the delimiting character, and the delimiting character in this example is, itself, also a quote (""), we need to use the backslash character (\) to escape the second quote ("`).

In the following example of values in a `.csv` file, we assume that `delimEnclosing=""`, and `delimField=","`. This means that imported values in the `.csv` file are enclosed in the quote character (“value”) and multiple values are separated by the comma ("value1","value 2"). Any commas inside the quotes are interpreted as a comma and not as a field delimiter.

```
"Value 1, has a comma","Value 2 has, two, commas","Value3"
```

You can use other characters as the enclosing delimiter, too, for example, the hash (#). In the following example, we assume that `delimEnclosing="#"` and `delimField="#"`. Any semi-colons included inside the hash characters are interpreted as a semi-colon and not as a field delimiter.

```
#Value 1; has a semi-colon;#Value 2 has; two; semi-colons;#Value3#
```

distinguishEmptyFromNull

Use the `distinguishEmptyFromNull` keyword in combination with `delimEnclosing` to ensure that the table-import process correctly interprets any empty value in the `.csv` file, which is enclosed with the value defined in the `delimEnclosing` keyword, for example, as an empty space. This ensures that an empty space is imported “as is” into the target table. If the empty space in incorrectly interpreted, it is imported as NULL.

```sql
distinguishEmptyFromNull = true;
```
i Note

The default setting for distinguishEmptyFromNull is false.

If distinguishEmptyFromNull=false is used in combination with delimEnclosing, then an empty value in the .CSV (with or without quotes "") is interpreted as NULL.

"Value1","",Value2

The table-import process would add the values shown in the example .csv above into the target table as follows:

Value1 | NULL | NULL | Value2

keys

Use the keys keyword in the table-import configuration to specify the key range to be considered when importing the data from the .csv source file into the target table.

keys = [ "GROUP_TYPE" : "BW_CUBE", "GROUP_TYPE" : "BW_DSO", "GROUP_TYPE" : "BW_PSA" ];

In the example above, all the lines in the .csv source file where the GROUP_TYPE column value matches one of the given values (BW_CUBE, BW_DSO, or BW_PSA) are imported into the target table specified in the table-import configuration.

;;;BW_CUBE;;;40000000;3;40000000;slave
;;;BW_DSO;;;40000000;3;40000000;slave
;;;BW_PSA;;;2000000000;1;2000000000;slave

In the following example, the GROUP_TYPE column is specified as empty("").

keys = [ "GROUP_TYPE" : "" ];

All the lines in the .csv source file where the GROUP_TYPE column is empty are imported into the target table specified in the table-import configuration.

;;;;;40000000;2;40000000;all
2.7.4 Table-Import Configuration Error Messages

During the course of the activation of the table-import configuration and the table-import operation itself, SAP HANA checks for errors and displays the following information in a brief message.

Table 10: Table-Import Error Messages

<table>
<thead>
<tr>
<th>Message Number</th>
<th>Message Text</th>
<th>Message Reason</th>
</tr>
</thead>
</table>
| 40200          | Invalid combination of table declarations found, you may only use [cdstable | hdbtable | table] | 1. The *table* keyword is specified in a table-import configuration that references a table defined using the `.hdbtable` (or `.hdbdd`) syntax.  
2. The *hdbtable* keyword is specified in a table-import configuration that references a table defined using another table-definition syntax, for example, the `.hdbdd` syntax.  
3. The *cdstable* keyword is specified in a table-import configuration that references a table defined using another table-definition syntax, for example, the `.hdbtable` syntax. |
| 40201          | If you import into a catalog table, please provide schema | 1. You specified a target table with the *table* keyword but did not specify a schema with the *schema* keyword. |
| 40202          | Schema could not be resolved | 1. The schema specified with the *schema* keyword does not exist or could not be found (wrong name).  
2. The public synonym for an `.hdbtable` or `.hdbdd` (CDS) table definition cannot be resolved to a catalog table. |
| 40203          | Schema resolution error | 1. The schema specified with the *schema* keyword does not exist or could not be found (wrong name).  
2. The database could not complete the schema-resolution process for some reason - perhaps unrelated to the table-import configuration (.hdbti), for example, an inconsistent database status. |
<p>| 40204          | Table import target table cannot be found | 1. The table specified with the <em>table</em> keyword does not exist or could not be found (wrong name or wrong schema name). |
| 40210          | Table import syntax error | 1. The table-import configuration file (.hdbti) contains one or more syntax errors. |</p>
<table>
<thead>
<tr>
<th>Message Number</th>
<th>Message Text</th>
<th>Message Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>40211</td>
<td>Table import constraint checks failed</td>
<td>1. The same key is specified in multiple table-import configurations (.hdbti files), which leads to overlaps in the range of data to import.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. If keys are specified for an import in a table-import configuration, multiple imports into the same target table are checked for potential data collisions</td>
</tr>
</tbody>
</table>
Important Disclaimer for Features in SAP HANA Platform, Options and Capabilities

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

Coding Samples

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

Accessibility

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

Gender-Neutral Language

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

Internet Hyperlinks

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