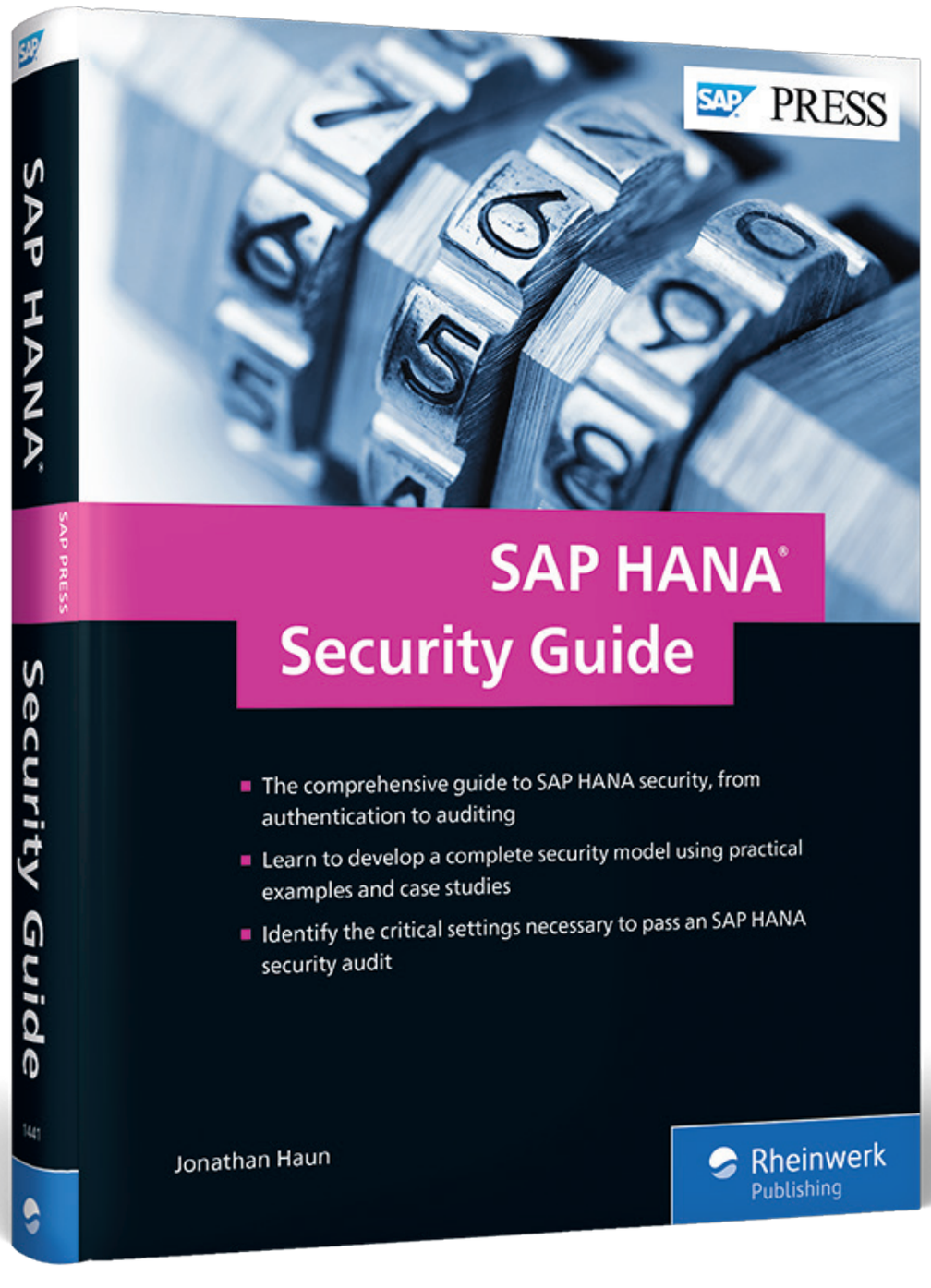


Reading Sample

This sample chapter describes the different types of catalog objects available in SAP HANA. It covers creating, managing, and deploying both native and repository catalog object, and ends with a practical case study.

Jonathan Haun
SAP HANA Security Guide
541 Pages, 2017, $79.95
ISBN 978-1-4932-1441-9

www.sap-press.com/4227

Chapter 3
Catalog Objects

In this chapter, we'll explore the different types of catalog objects and walk through the two major options available for creating them. How you choose to create catalog objects has a profound effect on the security model structure and management processes.

Object privileges play an important role in the development of any SAP HANA security model. They define the types of SQL script actions a user can perform on a catalog object (also sometimes referred to as a *database object*). SAP HANA hosts multiple types of catalog objects. In this chapter, we'll explore the different types of catalog objects that SAP HANA hosts. We'll also explain that there are different ways to create most catalog objects: Catalog objects can be created directly using SQL `CREATE` statements, in which case they're called *native catalog objects*. The user issuing the SQL statement is the owner of the native catalog object. SAP HANA also supports the creation of some catalog objects as repository development artifacts, in which case they're called *repository catalog objects*. Repository catalog objects are owned by the _SYS_REPO system account.

Because the principal of object ownership is a critical concept in all security models, the acute distinction of ownership—between native and repository catalog objects—is very important. Therefore, in this chapter we'll take a detailed look at the different options for creating catalog objects. The options an organization chooses to use when create catalog objects is of critical importance to all security administrators. We'll define the different types of catalog objects, demonstrate how to create both native and repository objects, review the process for deploying repository objects, and conclude with a case study to demonstrate a real-world example.

3.1 What Are SAP HANA Catalog Objects?

Catalog objects consist of various relational database management system (RDBMS) artifacts, such as tables, views, stored procedures, triggers, sequences, and a few other

items. Items such as tables both store and define the structure of the data. Other items, such as views, are used to express query logic. The following list describes the most common catalog objects available in SAP HANA:

- **Schema**
  A *schema object* is a logical organizational unit that is a parent to all other catalog objects. When other catalog objects are created, a schema must be specified. From the perspective of a security model, it's the most important catalog object. Whether to secure the entire schema or to secure the individual catalog objects within the schema is a critical choice to make.
- **Tables**
  Tables define the structure of data and store data within an RDBMS. Their structure is defined by columns, rows, and data types, and rows of data are logically stored in this table structure. Because the data contained in tables might be sensitive, it's important that organizations limit the types of SQL statements that can be executed against tables.
- **Triggers**
  Triggers are catalog objects that contain code that executes a procedure depending on a variety of events that occur within the RDBMS. They are used for a variety of purposes, ranging from database management to data integrity. Because their code can manipulate data and other catalog objects, it's important for organizations to restrict where and how they are used.
- **Views**
  Views act as logical tables. However, in SAP HANA they are not used to store data. Instead, they are defined using SQL query statements. Like tables, they offer columns and data types and can be queried. Within their definitions, their SQL query statements can perform actions such as combining tables, restricting data, and aggregating results. From a security standpoint, most catalog views need to be secured for the same reasons that we secure access to tables.
- **Synonyms**
  Synonyms are used as logical reference points to other catalog objects. They act aliases to obscure the identity of other catalog objects. Synonyms exclusively use the granted privileges of the objects they reference.
- **Sequences**
  Sequences are catalog objects used to keep track of an incremented number value by means of its definition. They can be queried in procedural code to determine the current or next value within a defined sequence. They often are used to create

unique identity columns within a table. For example, we can use a sequence to generate a new value in a primary key column, with the requirement that the value be unique for the row.

- **Procedures**
  A *stored procedure* is a catalog object that contains complex developed SQL code. It can return a dataset, much like a catalog view. However, the code can also perform tasks within the RDBMS system. These procedures support both input and output parameters, which makes them dynamic.
- **Indexes**
  An *index* is an object used to optimize the retrieval of data from a table. Although it increases the performance of some queries, it also comes at the cost of increased storage. Security administrators typically limit privileges to create and drop indexes because of the impact they have on both performance and storage cost.
- **Functions**
  Functions are custom snippets of SQL code used to perform read-only operations on data. They can be defined as single-valued input and output or table-valued input and output. Once created, they can be referenced within SQL `SELECT` statements and SQL stored procedures. Depending on the purpose of the catalog function, regulating access to these objects can be important. For example, if a function is used to apply core business logic within a SQL query, security administrators likely will need to limit the number of individuals that can alter the function.

Now that we've listed the common catalog objects used in an SAP HANA system, let's look at the options for creating them. For this discussion, we'll demonstrate the processes necessary to create a schema and the processes necessary to create a table. We'll highlight the differences based on instances in which we plan to use SQL statements and instances in which we plan to use repository objects. How you create objects is important to your security model, so pay close attention to these options while focusing on how they affect an object's ownership.

3.2 Creating and Managing Native Catalog Objects

The process used to create standard catalog objects typically involves executing a SQL statement containing a `CREATE` statement and the supporting syntax specific to the objects being created. Once a `CREATE` statement is successfully executed, a runtime version of the object will exist in the SAP HANA system. Active objects within

the RDBMS catalog are called *runtime objects*. However, the design-time script will remain within the SQL console. The design-time state or *design-time objects* represent the script or code used to define an object. With SQL, if you need to save the design-time script, you'll need to save to a file system independent from the SAP HANA system.

Those familiar with SAP HANA Studio will likely be aware that some objects can be created using the GUI. Although the workflow appears graphical in nature, these GUI windows are issuing SQL statements to the system based on the options selected within the GUI. Like using SQL statements, the design-time definition of the objects isn't stored within the SAP HANA system. In fact, it's more difficult to save the script when using the GUI, because the script needs to be extracted from the GUI execution status windows. However, SAP HANA Studio allows you to copy the text of the SQL statement from the status window; if you extract the SQL statement, you can then save it to a file for safekeeping.

Regardless of the method you use to create a standard catalog object, SQL statements inevitably will be issued to the SAP HANA system. In this section, we'll introduce the basic workflows and SQL syntax needed to create a standard schema and table and discuss how they influence a security model. When applicable, we'll demonstrate how objects can be generated using SAP HANA Studio.

3.2.1 Creating Schemas

Schemas are core relational database objects used to logically store all other catalog objects. Standard catalog schemas are generated in three ways:

1. When a standard user account is created, the system automatically generates a schema for that user. The user is the sole owner of that schema, and the account owner must grant privileges to other users or roles before they can interact with the user's schema.
2. Schemas can be created using a SQL `CREATE` statement. The following SQL statements can be used to create a schema. In the first example, we can specify the owner of the schema. In the second example, the system assumes that the user executing the statement will be the owner:
   - `CREATE SCHEMA "MySchema" OWNED BY USER_NAME`
   - `CREATE SCHEMA "MySchema"`
3. The third option for creating a schema is via the SAP HANA Web-Based Development Workbench catalog editor. The SAP HANA system offers several web-based

GUIs to aid in the creation and modification of both catalog and repository-based objects. Not all objects can be created utilizing a GUI within this interface, but schemas can be. The user account used to log on to the SAP HANA Web-Based Development Workbench catalog editor will be the owner of the schema.

To access this web-based GUI, enter the correct URL into a supported web browser's address bar using the following template:

http://<sap_hana_host>:80<instance_number>/sap/hana/ide/catalog

Replace *<sap_hana_host>* with the hostname of the SAP HANA system in your environment and *<instance_number>* with the two-digit instance number corresponding to your SAP HANA system.

For secure access, use the following template:

https://<sap_hana_host>:43<instance_number>/sap/hana/ide/catalog

If you have the appropriate privileges to access the SAP HANA Web-Based Development Workbench catalog editor, you can log on to the web interface using these URLs. Once logged in, you can create a catalog schema, assuming you have the necessary system privileges.

To create a catalog schema within this interface, right-click the **Catalog** node on the left side of the window. From the context menu, choose **New Schema**. A window titled **Create new Schema** will open. Enter the name of the schema in the text box labeled **New Schema** (see Figure 3.1). If you want the exact case of the text (uppercase/lowercase) to be used in the schema name, uncheck **Case-insensitive**. Click **OK** to complete the workflow and generate the schema.

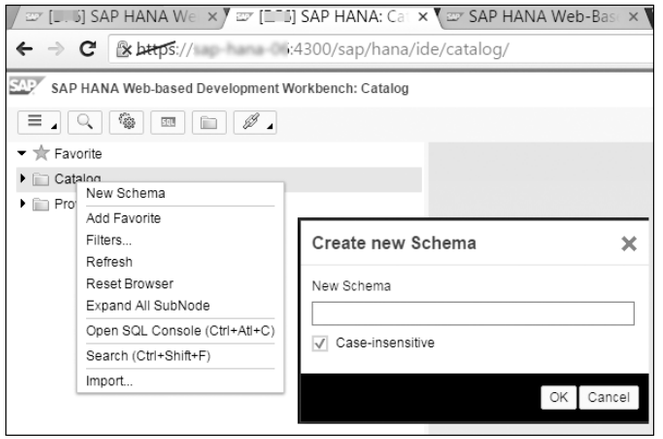


Figure 3.1 Creating Schema with SAP HANA Web-Based Development Workbench

SAP HANA Web-Based Development Workbench

For more information on accessing and using the SAP HANA Web-Based Development Workbench, please download the SAP HANA Developer Guide for the SAP HANA Web-Based Development Workbench from *https://help.sap.com/viewer/b3d0daf2a98e49ada00bf31b7ca7a42e/2.0.00/en-US*.

For a direct link to the PDF version of the guide, use the following URL: *https://help.sap.com/doc/7d957b1b6c1a4791a1212b45e9a797df/2.0.00/en-US/SAP_HANA_Developer_Guide_for_SAP_HANA_Web_Workbench_en.pdf*.

All three options will result in a schema that will be owned by the individual executing the script or completing the action within the GUI. Because standard catalog schemas are owned by standard user accounts, maintaining access to these accounts is critical. This is because only the owner of the schema can initially grant privileges on the schema's catalog objects to other users and roles. This is an important concept that needs to be understood by all security administrators.

Depending on your SAP HANA solution, a standard schema and its associated service account will likely be generated programmatically by other SAP software. For example, when we're installing an SAP NetWeaver-based solution on SAP HANA, the software provisioning manager will create a service account user and use its user schema to store the system tables, views, and other objects used by the SAP NetWeaver application. If your organization plans to provide direct access to these SAP HANA objects, maintaining access to the service account that owns this schema will be critical. At some point, the security administrator or Basis administrator will likely need to log on to SAP HANA Studio with the service account to grant privileges to other users or roles.

As mentioned before, granting privileges to a schema will inherently grant any applicable privileges to tables, views, functions, sequences, column views, and other objects associated with that schema. Some organizations might choose to grant schema privileges only to users and roles. Doing so will greatly reduce the maintenance of the security model, because all new objects will inherit the granted privileges from the schema. However, some organizations might find this too broad of a methodology and choose to secure objects individually.

Let's now look at how to create individual objects by exploring the options available when creating tables.

3.2.2 Creating Catalog Tables

There are two different methods we can use to create standard tables within SAP HANA. We can execute SQL statements using the `CREATE` table syntax to produce a catalog table. Such objects are owned by the user account issuing the statement. We can also use the GUI within SAP HANA Studio to define a table. Both methods issue SQL statements to the system, and the user account executing the workflow owns the object.

Let's look at how to use the `CREATE` SQL statement to generate a table. First, we can execute SQL statements within the SQL console. Refer to Chapter 1 if you need a refresher on how to access the SQL console within SAP HANA Studio. To create a column store table with two columns, you would execute the following SQL statement within the SQL console:

```
CREATE COLUMN TABLE "MySchema"."MyTable"
("MyColumnA" NVARCHAR(5) PRIMARY KEY, "MyColumnB" DECIMAL(26,6) );
```

To create a table using another table's definition as a template, execute the following SQL statement:

```
CREATE TABLE "MySchema"."MyCopyTable" like "MySchema"."MyTable" WITH NO DATA;
```

Creating Tables or Schemas

The complete syntax for creating tables, schemas, and other catalog objects is beyond the scope of this book. To review the full syntax and all available options, please access the SAP HANA SQL and System Views Reference Guide found at *https://help.sap.com/hana_platform* or use the following direct link to the guide in PDF format: *https://help.sap.com/doc/9b40bf74f8644b898fb07dabdd2a36ad/2.0.00/en-US/SAP_HANA_SQL_and_System_Views_Reference_en.pdf*.

The second option for creating catalog tables relies on a GUI within SAP HANA Studio. SAP HANA Studio provides an interface containing many of the standard options necessary to create a table. To access this GUI and create a table, use the following steps:

1. Open SAP HANA Studio and connect to the desired system. This workflow will work in any perspective containing the **Systems** window.
2. Within the **Systems** window, expand the system to reveal the **Catalog** node.
3. Within the **Catalog** node, expand the tree under the schema in which you want to create the table.

4. Within the schema node, right-click the **Tables** folder and choose **New Table**. A window will appear like the one shown in Figure 3.2. To properly define the table, enter a name in the **Table Name** field and define one or more columns within the **Columns** tab. When defining a column, you must provide the **SQL Data Type** and **Dimension** for applicable column types.

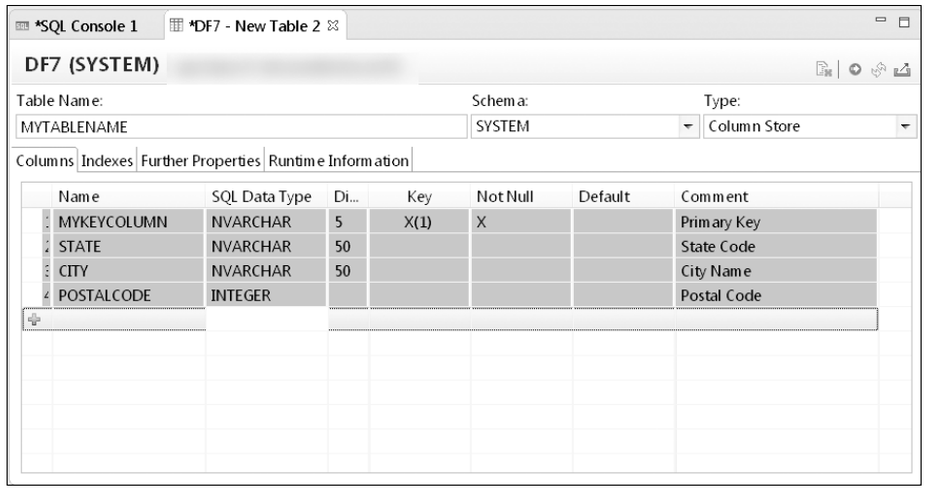


Figure 3.2 Create Table GUI in SAP HANA Studio

5. To add additional columns, click **+** within the **Columns** tab to add an additional row. Within each new row, you can define an additional column.
6. Once you've defined all the desired properties of the table, execute the object to activate it within the system. To do so, press F8 to execute the SQL code generated by the GUI. A small window will appear at the top containing the status of the execution and the SQL code that was executed by the system. If you choose to do so, the SQL statements within this status window can be coped and saved to a local file.

It's important to note that this graphical workflow technically issues SQL statements to the system. Therefore, the table will be owned by the individual executing the workflow. From a security perspective, the owner of the table or anyone with full control of the table's schema will need to grant privileges to other users or roles to allow them access to the table. The table will also be removed from the system if the executing user is removed from the system at any time in the future.

3.2.3 Creating Other Catalog Objects

All catalog objects can be created using the SQL `CREATE` statement. The syntax in most cases follows a pattern like the following:

```
CREATE <Catalog Object Type>  <Catalog Object Name> <Options>
```

The catalog object type can be an item such as `TABLE`, `VIEW`, `PROCEDURE`, `FUNCTION`, `SCHEMA`, `TYPE`, or another catalog object. The catalog object name can be any name you choose, so long as it contains supported characters. The options will vary based on the catalog object you're creating.

For example, to create a catalog view, you can execute the following SQL code:

```
CREATE VIEW "MyView" AS
SELECT * FROM "MyTable" WHERE "State" = 'Active';
```

To create a stored procedure named `MyProcedureName` that outputs a dataset from a table, define it using the code in Listing 3.1.

```
CREATE PROCEDURE MyProcedureName(IN id INT, OUT data CUSTOMER) LANGUAGE
SQLSCRIPT
READS SQL DATA WITH RESULT VIEW ProcView AS
BEGIN
data = SELECT * FROM DMART.DIMCUSTOMER WHERE CUSTOMER_ID = :id;
END;
```

Listing 3.1 Code to Define MyProcedureName Example Stored Procedure

These are just a few examples to help demonstrate how catalog objects are created using SQL scripts. When creating objects using SQL, remember that the user executing the statement will be the owner of the object.

Creating Catalog Objects

The complete syntax for creating tables, schemas, and other catalog objects is beyond the scope of this book. To review the full syntax and all available options, please access the SAP HANA SQL and System Views Reference Guide found at *https://help.sap.com/hana_platform* or by using the following direct link to the guide in PDF format: *https://help.sap.com/doc/9b40bf74f8644b898fb07dabdd2a36ad/2.0.00/en-US/SAP_HANA_SQL_and_System_Views_Reference_en.pdf*.

For information pertaining to creating objects using the GUI within SAP HANA Studio, please access the SAP HANA Administration Guide at *https://help.sap.com/hana_platform*. Look for the sections labeled *Create a Table in Runtime or Create a View in Runtime*.

3.3 Creating and Managing Repository Catalog Objects

For many RDBMS administrators, the process of issuing `CREATE` statements is a well-known methodology used to generate database catalog objects. However, SAP HANA offers an alternative to this traditional approach. It can offer this alternative because SAP HANA is also a development environment that allows organizations to define and store both design-time and runtime versions of objects. Design-time objects are stored in the SAP HANA repository, whereas runtime objects exist within the SAP HANA RDBMS catalog.

For example, we can define a development artifact called a *repository table* within the development repository found in SAP HANA studio. Once defined and activated, a catalog table is created in a specified schema. Repository tables function exactly like tables created with a SQL `CREATE` statement. However, it's important to note that a repository table is owned by the system account _SYS_REPO and not by the developer that activates it.

_SYS_REPO is the main system account responsible for managing, generating, owning, and activating repository-based objects. This includes repository schemas, tables, roles, views, and procedures. Leveraging the use of repository objects greatly reduces maintenance within a security model, because _SYS_REPO is already the owner of these objects and many objects within the system. There will be no need to log on as the traditional object owner and grant rights to other users and roles for repository objects. Most developers have a hard time remembering that they need to grant privileges to security administrators each time they create a schema or sometimes objects within a schema. By creating repository objects, developers are decoupled from this responsibility, because their repository objects are activated as catalog objects owned by _SYS_REPO. This simplifies the process and reduces the communication that needs to be established between SAP HANA developers and security administrators

_SYS_REPO is also responsible for managing the activation of SAP HANA information views. These multidimensional views reference catalog tables and can't be activated unless the _SYS_REPO account has been granted `SELECT`, `EXECUTE`, and `WITH GRANT OPTION`/**Grantable to others** for the object. Again, if we choose to create repository tables or schemas, the _SYS_REPO account will already have the required privileges.

In Chapter 6, we'll discuss this concept further. For now, note that most of the security model simplification is only achieved when we use both repository catalog objects and repository roles within our environment. Both objects are owned and executed by the _SYS_REPO account.

As mentioned, the designation of ownership is vital within the security model. Repository objects are owned by the _SYS_REPO system account. Therefore, we must pay close attention to the methods developers use when creating catalog objects. To further your understanding, let's look at a few examples of the process used to create both a repository schema and a repository table.

3.3.1 Creating Repository Schemas

To create a repository schema, we must first log on to an SAP HANA system and switch to the development perspective within SAP HANA Studio. As discussed in Chapter 1, we then need to access the **Repositories** tab and ensure that we have a local repository workspace defined. Once the workspace is defined, expand the package hierarchy to the desired package, then right-click and choose **New • Other.**

A window labeled **Select a wizard** will open. Within the **Search** dialog, type "schema". Choose the **Schema** object under **SAP HANA • Database Development**, then click **Next**, as shown in Figure 3.3.

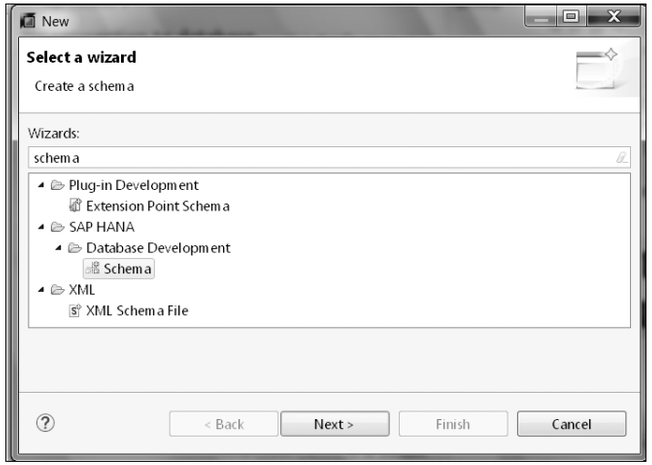


Figure 3.3 Searching for Schema Repository Objects

In the next window, labeled **New Schema**, enter a file name in the **File name** field (see Figure 3.4). The file name will become the name of the schema, so choose a name that matches the desired schema name. From the **Templates** dropdown, choose **Basic**; this will provide the basic syntax necessary to define a repository schema in the next window. Click **Next** to continue.

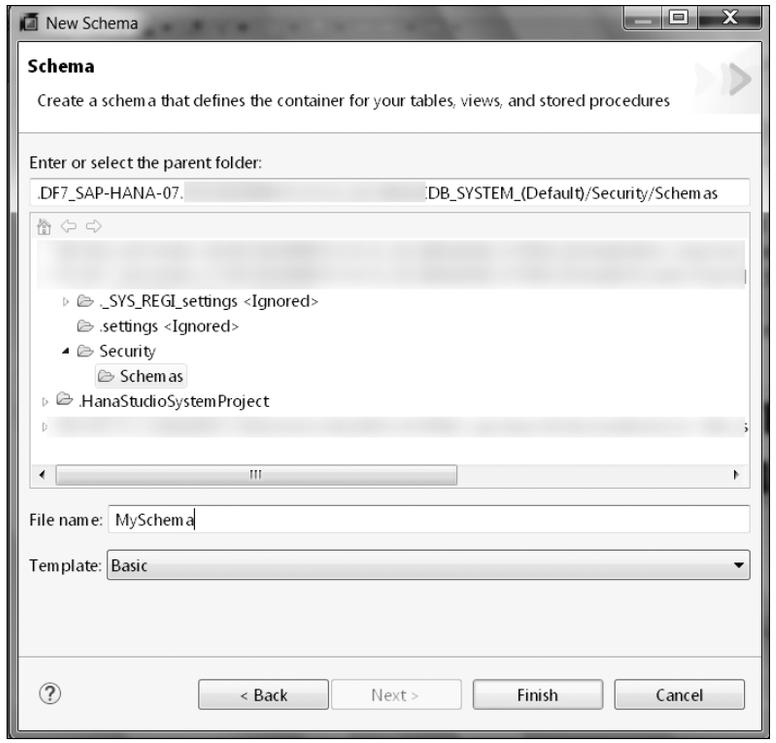


Figure 3.4 Defining Repository Schema Name and Choosing Template

A window containing the template's example text used to define a repository schema will now open, as shown in Figure 3.5. Because you chose the basic template, most of the necessary syntax has already been defined for you. Press Ctrl+F3 to save and activate the repository schema. The design-time object will be saved in the SAP HANA repository, and the runtime object will be created in the SAP HANA RDBMS catalog.

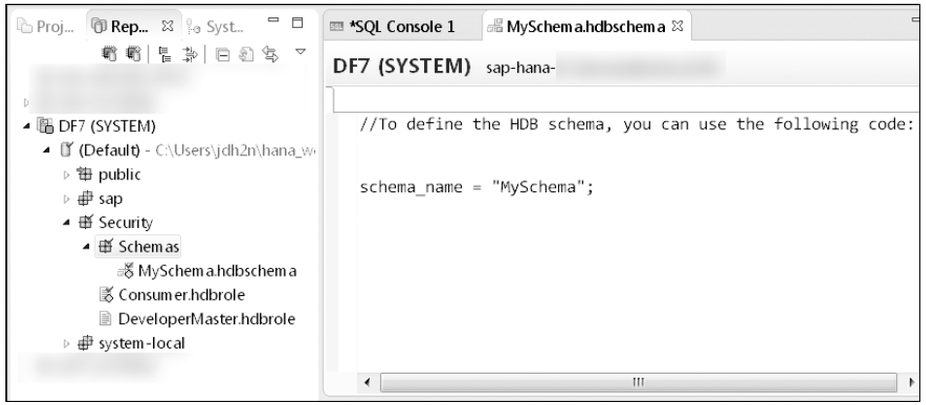


Figure 3.5 Syntax to Define Repository Schema

The syntax for creating a repository schema is very basic: The text preceded by two slash marks (//) is a comment only visible to the developer. The two slash marks tell the SAP HANA code compiler to ignore the information on that line. The core of the script is the following text:

```
Schema_name = "MySchema";
```

The text within the double quotes should match the name of the desired schema and the name given to the .hdbschema file in the repository.

Once activated, users with the CREATE ANY privilege for the schema will be able to create standard catalog objects within the schema using CREATE statements. However, we recommended that developers use the repository to create these objects. The repository schema will be owned by the _SYS_REPO user, with no need for the developer to be granted CREATE ANY on the schema. Developers only need to have access to activate objects within the SAP HANA repository; the _SYS_REPO user account creates the objects on behalf of the developer. Once again, we can simplify the security model, because developers do not need to be granted CREATE ANY privileges for a schema to create a catalog object.

Another important security aspect of a repository schema is that the _SYS_REPO account will own the schema. Thus, _SYS_REPO will have full privileges for all objects created in the schema, regardless of the method used to create them. Again, we recommend that developers use the repository to create objects. However, if for some reason they do not, then the _SYS_REPO account will already have the privileges necessary to establish repository roles that reference objects within the repository schema.

3.3.2 Creating Repository Tables

To create a repository table, first log on to an SAP HANA system and switch to the development perspective within SAP HANA Studio. As discussed in Chapter 1, you then access the **Repositories** tab and ensure that you have a local repository workspace defined. If the workspace is defined, continue to expand the package hierarchy to the desired package, then right-click it and choose **New • Other**.

A window will appear, labeled **Select a Wizard**. Within the **Search** dialog, type "database table". Choose the **Database Table** object under **SAP HANA • Database Development**, then click **Next**.

In the next window, labeled **New Database Table**, enter a file name in the **File name** field (see Figure 3.6). The file name will become the name of the table, so define a name that matches the desired table name. From the **Template** dropdown, choose **Basic**. As before, this selection will provide you with the basic syntax necessary to define a repository table in the next window. Click **Next** to continue.

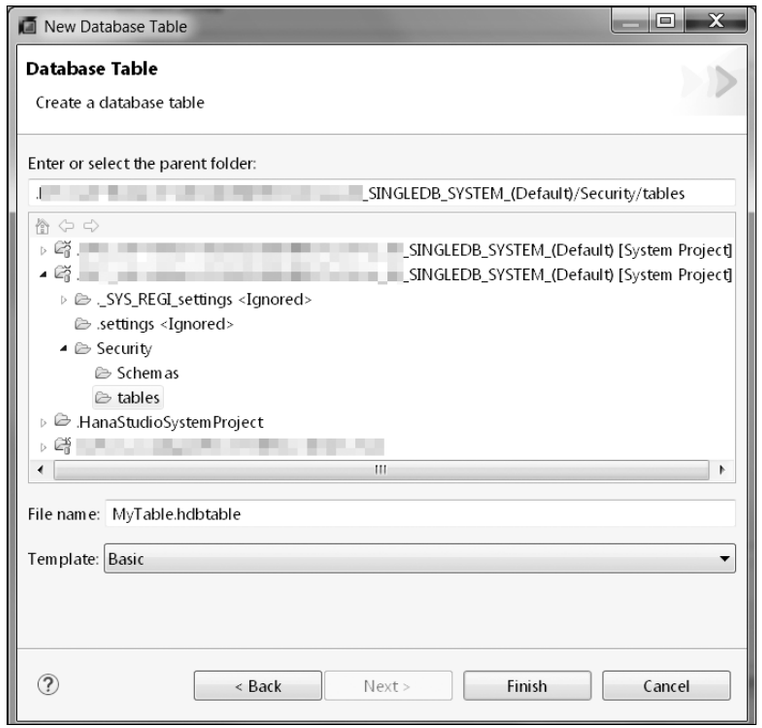


Figure 3.6 Defining Repository Table Name and Choosing Template

A window will now appear, as shown in Figure 3.7, which contains the template's example text used to define a repository table. Because you chose the basic template, several examples and comments on the syntax are included here. To define a table, simply adjust the template syntax to match your requirements.

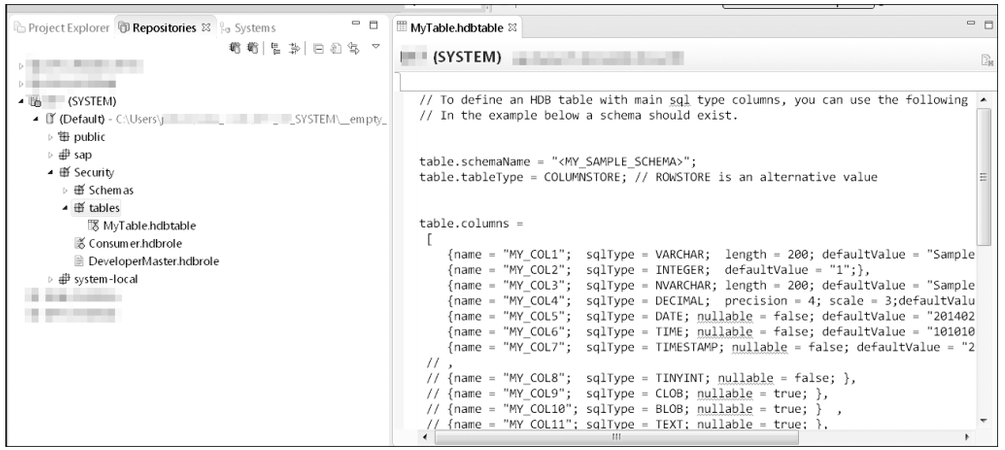


Figure 3.7 Defining Repository Table

If you've entered the correct syntax, press Ctrl+F3 to save and activate the repository table. The design-time object will be saved in the SAP HANA repository, and the runtime object will be created in the SAP HANA RDBMS catalog. Within the script, you specify the schema that will house the repository table. You will now be able to find the table within the SAP HANA catalog or within the schema node when browsing from SAP HANA Studio.

It's worth mentioning that the design-time object name and catalog object name will be different. The catalog object name will contain the name you specified for the repository object and will be prefixed with the package hierarchy name. For example, if we created the design-time repository table, named MyTable, in a package hierarchy of `myco.app.tables`, the name of the table would be `myco.app.tables::MyTable`. To query this table using SQL, you would execute the following SQL statement:

```
SELECT * FROM "MySchema"."myco.app.tables::MyTable"
```

Notice that the table name is prefixed with the package hierarchy name and two colons (`myco.app.tables::`). Because we used lowercase and camel case for the package name and table name, respectively, we must set the schema and catalog table name between double quotes in the SQL statement. SAP HANA is case-sensitive by

default. Without the double quotes, the system would look for the object name while assuming you intended to use uppercase on all objects, and thus would fail to find the objects, because they were created with lowercase or camel case.

SAP HANA Repository Catalog Objects

For more information on the process used to create and manage repository tables and other repository-based objects, access the SAP HANA Developer Guide found at *http://help.sap.com/hana_platform*.

For a direct link to the PDF version of the guide, use the following URL: *https://help.sap.com/doc/fbb802faa34440b39a5b6e3814c6d3b5/2.0.00/en-US/SAP_HANA_Developer_Guide_for_SAP_HANA_Studio_en.pdf*.

Again, repository tables are owned by the _SYS_REPO system account, which will greatly simplify the setup of your security model. It will also aid in the promotion of a security model and its referenced repository objects when we need to move them into another SAP HANA environment. For example, we can define a security model in an SAP HANA development environment, within which we can create a delivery unit containing a design-time repository table, schema, and role. The delivery unit, containing a security model, can be exported to a file and later imported into a production environment. This process is only possible because we chose to define catalog objects as repository objects. In the next section, we'll take a closer look at how to deploy repository objects using a delivery unit.

3.4 Deploying Repository Objects

Chapter 14 will provide information about the processes, procedures, and steps necessary to manage a repository-based security model's lifecycle using SAP HANA application lifecycle management. In this section, we simply will introduce you to lifecycle management terms and concepts specific to catalog objects. Specifically, we'll define the terms *content vendor* and *delivery unit*. We'll also demonstrate how to package repository-based content into a delivery unit and export it to a file using SAP HANA Studio.

With most security models, there will come a point at which we need to move, share, or migrate to a new SAP HANA environment. Before we can properly do that, we need to make sure that all the objects that the model references are also available in each environment. We refer to these objects as *security model dependencies*. Security model

dependencies are typically composed of various catalog objects. The makeup of a properly designed security model consists of a series of repository roles. When these repository roles reference repository-based catalog objects, we can move all parts of the model to another environment via the lifecycle management tools built into SAP HANA and SAP HANA Studio.

We've already introduced the processes necessary for creating repository schemas and tables. Assuming we develop all other objects as repository-based artifacts, moving these objects between environments will be possible. However, before we can export these objects and then import them into another environment, we must ensure that the source SAP HANA system is set up to support such activities. Two key items must be configured before we can proceed: First, we need to define the *content vendor* within our source SAP HANA system, and second, we must define a *delivery unit*.

Each SAP HANA instance must be configured with a *content vendor* before we can conduct lifecycle management activities. The *content vendor* is a value used to uniquely identify the organization or system that generated a delivery unit or exported content. The typical format of its value looks similar to that of a domain name used in an email address, but it can take on other supported formats. For example, the value can be something like `mycompany.com` or `MyCompanyName`. We don't recommend using spaces between characters, because some SAP HANA interfaces don't support them when specifying the content vendor name.

The simplest way to establish a content vendor within an SAP HANA system is to ask your SAP HANA administrator to execute an `ALTER SYSTEM` SQL statement. Our first example will set the content vendor to `'mycompany.com'`, as follows:

```
ALTER SYSTEM ALTER CONFIGURATION ('indexserver.ini', 'SYSTEM')
   SET ('repository', 'content_vendor') = 'mycompany.com'
WITH RECONFIGURE;
```

The second example will set the content vendor to `'MyCompanyName'`:

```
ALTER SYSTEM ALTER CONFIGURATION ('indexserver.ini', 'SYSTEM')
   SET ('repository', 'content_vendor') = 'MyCompanyName'
WITH RECONFIGURE;
```

For this statement to work in your environment, simply change `mycompany.com` to a value appropriate for your organization. Make sure you include the `WITH CONFIGURE` statement to activate the change without needing to restart the system. There are a few other ways to change the content vendor using a GUI; however, we find that

running this SQL statement is the simplest and most direct way to implement the setting.

Once a content vendor is defined, we can then define a delivery unit. A *delivery unit* is a logical object that can be configured with one or more repository packages. As mentioned in Chapter 2, a package is a logical organization unit used to store development artifacts. The main purpose of a delivery unit is to create bundles of packages containing repository objects so that we can transport them to another SAP HANA system.

To create a delivery unit, access SAP HANA Studio and switch to the SAP HANA modeler perspective. In the default layout of this perspective, there will be a **Quick View** tab or window on the right side of the interface. Within this area, you'll see an option labeled **Delivery Units**; click it to open the **Delivery Units** management window. Figure 3.8 shows both the **Quick View** window on the right and the **Delivery Units** management window on the left.

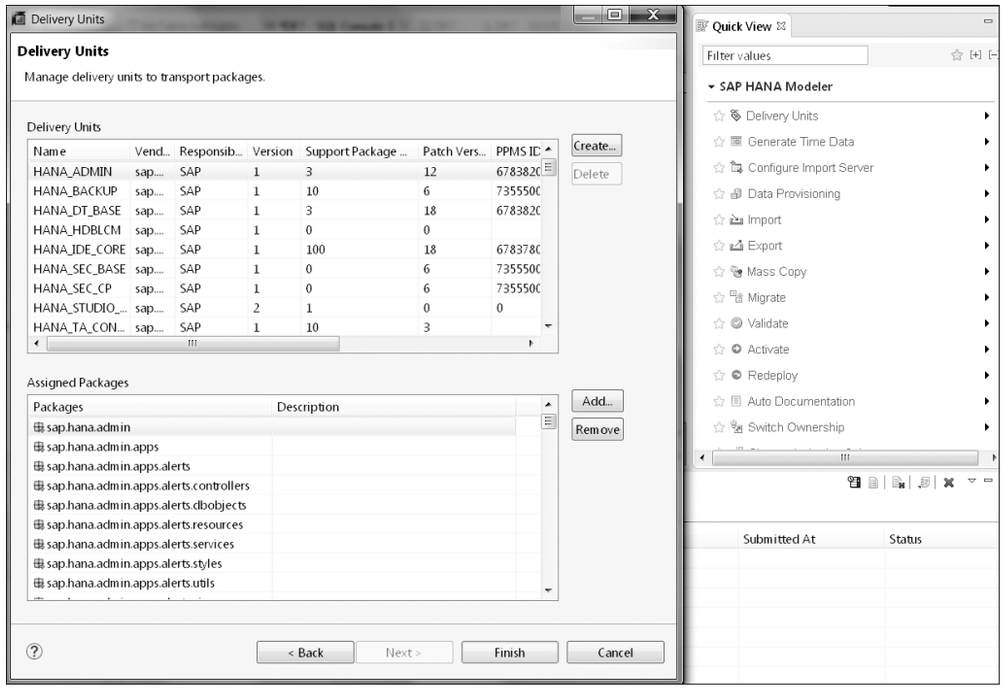


Figure 3.8 Quick View Window and Delivery Units Management Window in SAP HANA Studio

In the **Delivery Units** management window, click the **Create** button located to the right of the **Delivery Units** section. A window titled **New Delivery Unit** will open. Enter a name in the required **Name** field. All other information is optional, but Figure 3.9 shows an example of the settings to use to initialize the delivery unit. Click **OK** to create the delivery unit.

Figure 3.9 New Delivery Unit Configuration Window

We now need to assign packages to this delivery unit. After clicking **OK** in the **New Delivery Unit** configuration window, you should have been returned to the **Delivery Units** management window. To assign packages to our newly created delivery unit, first select the delivery unit within the **Delivery Units** section. To the right of the **Assigned Packages** section, click the **Add** button. A new window will appear, titled **Assign Packages** (see Figure 3.10).

To assign packages to the delivery unit, select the highest node level that contains all the appropriate packages, subpackages, and development artifacts. Make sure that the **Select all sub-packages under the selected nodes** checkbox is selected to ensure that all assigned subpackages are included within the delivery unit. Click **Finish** to complete the process. In Figure 3.11, the delivery unit named **HANA_SECURITY_MODEL** has been assigned the **Security**, **Security.Schemas**, and **Security.tables** packages. Click **Finish** to close the **Delivery Units** management window.

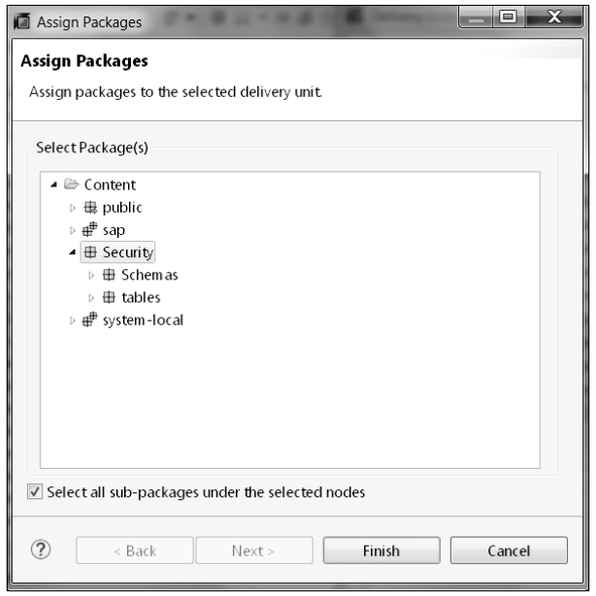


Figure 3.10 Assign Packages Window

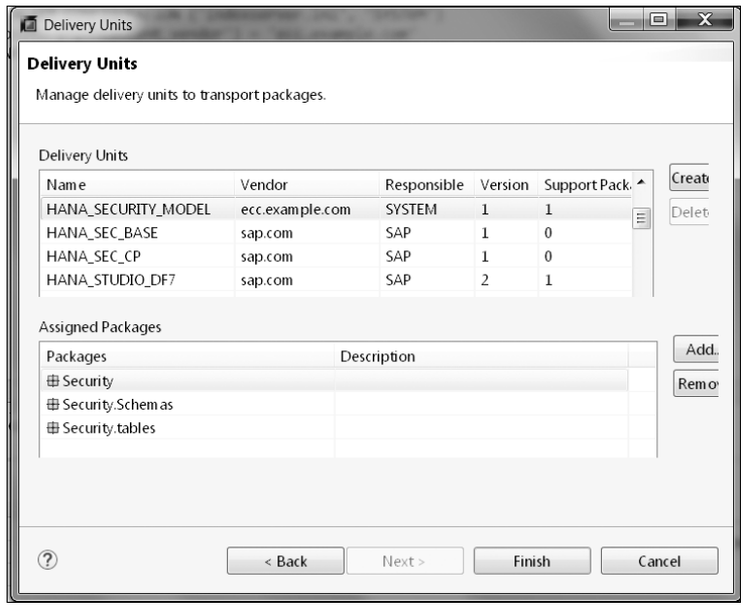


Figure 3.11 Packages Assigned to HANA_SECURITY_MODEL Delivery Unit

Managing Delivery Units and Establishing a Content Vendor Value

For more information and for alternative options available to manage a delivery unit or for alternative options for setting the content vendor, please refer to the SAP HANA Application Lifecycle Management Guide located at the following URL: *http://help.sap.com/hana_platform#section5*.

To access the guide directly in PDF format, visit *http://help.sap.com/hana/SAP_HANA_Application_Lifecycle_Management_en.pdf*.

Now that we've defined a delivery unity, we can use SAP HANA Studio to export our security model to a file. We can use this file as an import source, or we can share the file with others. To export the delivery unit, open SAP HANA Studio, then select **File • Export** from the menu bar.

A window titled **Export** will appear. Within the search box titled **Select and Export Destination**, enter "delivery unit". This will filter the available selections down to the desired option. Under the **SAP HANA Content** node, choose **Delivery Unit**. Click **Next**.

Select the SAP HANA system that will be the source of the delivery unit and click **Next**. Within the **Select Delivery Unit** window, use the dropdown menu titled **Delivery Unit** to choose the **HANA_SECURITY_MODEL** delivery unit. In the **Export Location** area, choose **Export to Client** and enter the folder location path and desired file name. Typically, the default file name is an acceptable name. Figure 3.12 shows the **Select Delivery Unit** window. Click **Next** to export the delivery unit to the specified file path.

The delivery unit file should now exist under the specified file path. Delivery unit files use a file extension of *TGZ*, commonly used on UNIX and Linux systems for files containing a compressed archive. Therefore, the delivery unit file in fact is a compressed archived containing all the SAP HANA development artifacts associated with the exported delivery unit. This file can be used as a source and imported into another SAP HANA system or even the same SAP HANA system. In most instances, these files provide an excellent backup mechanism for the content we create in the SAP HANA repository.

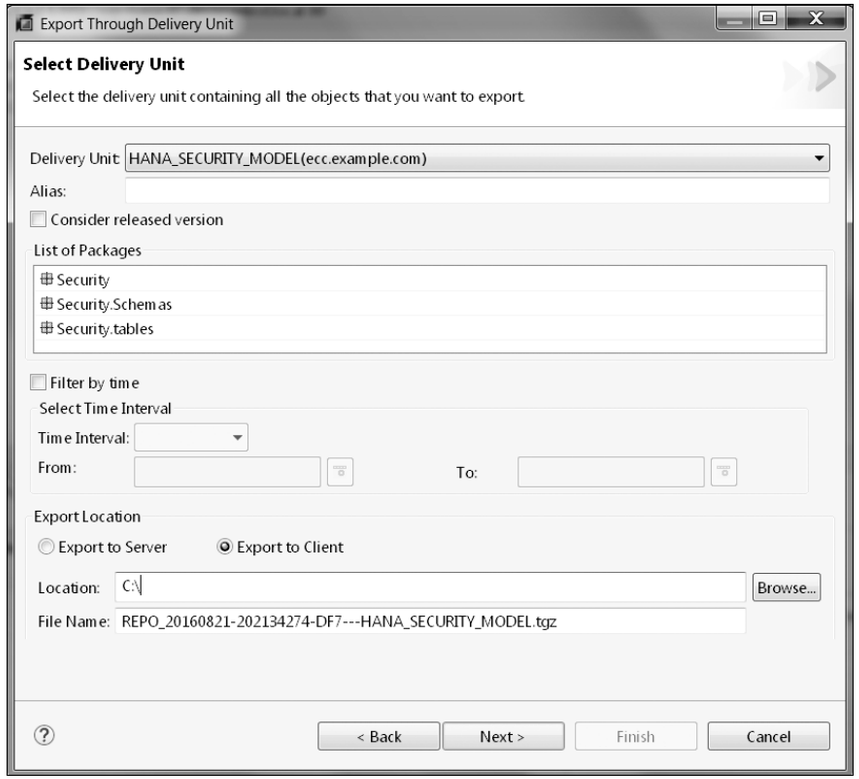


Figure 3.12 Delivery Unit Export Options Window

To import a delivery unit file, open SAP HANA Studio and log on to the desired SAP HANA system. From the menu bar, select **File • Import**. A window titled **Import** will appear. In the **Select an Import Source** search bar, type "delivery unit". Select **Delivery Unit** below the **SAP HANA Content** node, and click **Next**. A window will appear, containing the text **Target System** below the title **Import Through Delivery Unit**. In this window, select the system into which you want to import the delivery unit, and click **Next**. As shown in Figure 3.13, a **Select File** window will appear.

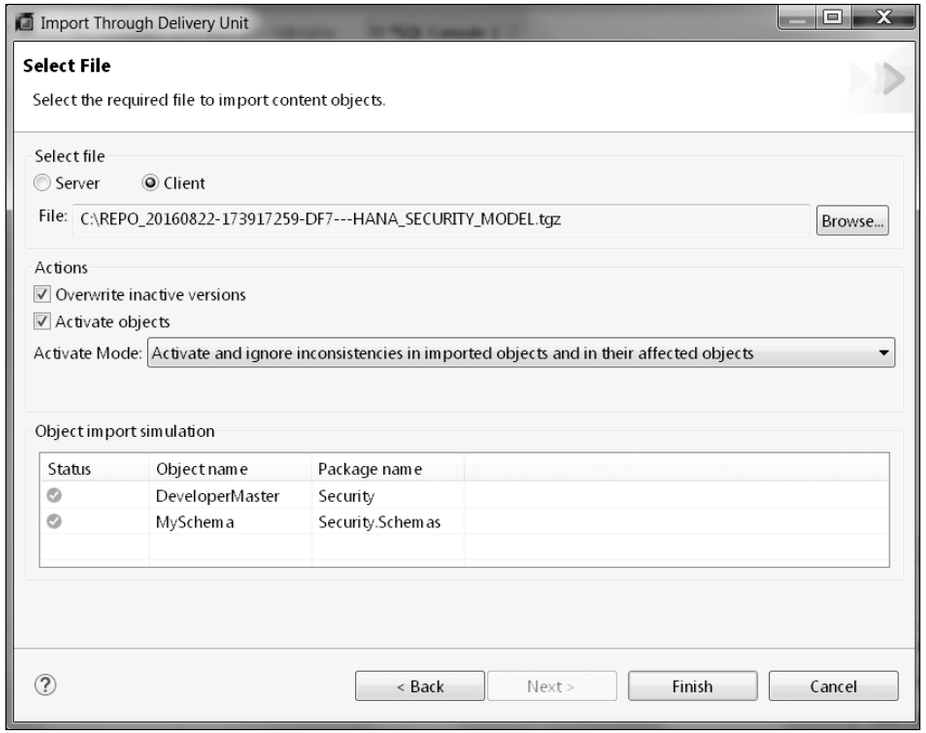


Figure 3.13 SAP HANA Delivery Unit Import Wizard Options

To import a file saved to your local file system, select the radio button labeled **Client** in the **Select file** area. Choose **Browse** to locate the file on your file system. In the **Actions** area, we typically recommend that both of the options **Overwrite inactive versions** and **Activate objects** are checked. This will both overwrite inactive versions in the target and active the newly imported content. The default **Activation Mode** of **Activate and ignore inconsistencies in imported object and in their affected objects** should be selected. The **Object import simulation** area will provide the expected import **Status** for the listed **Object name** and **Package name**. Click **Finish** to complete the import process. The status of the import process will be listed in the **Job Log** window. For more information about the **Job Log** window, see Chapter 1.

3.5 Case Study

E-Corporation is the world's largest robotics manufacturer and a large multinational conglomerate. The company has a large SAP ERP implementation and has recently purchased SAP HANA to facilitate a high-performance sales analytics data mart that combines data from multiple core SAP Business Suite systems and data from a few subsidiaries that are using non-SAP applications to manage their sales operations.

The new E-Corporation analytics project is still in its early phases. Developers are just starting to analyze and profile data from the various source systems. However, the director of analytics for E-Corporation is aware that extremely sensitive data will be hosted in the SAP HANA system. Because the data is sensitive in nature, implementing a sound security model is of the utmost importance. To help the E-Corporation team implement a sound security model, the company has contracted an SAP HANA security expert from a trusted consulting partner.

During initial project meetings, the security consultant recommends that E-Corporation start planning for the security model during the development phase. The security consultant knows that the development team needed to create several catalog tables and a few schemas to house the data from the various sources, and he highly recommends that the company begin the development process by creating two repository-based schemas to host the staging and data mart tables. The consultant also recommends that E-Corporation create repository-based tables to help simplify the development and management of a security model.

Before activating any repository-based objects, the security consultant recommends defining a package hierarchy to store the core repository objects that will be used to define the security model. (Chapter 1 provides the necessary information for creating packages and a package hierarchy.) Knowing that package privileges are critical to the security of the security model itself, the consultant recommends defining a top-level package named **Security**, followed by a subpackage named **Schemas** and another named **Roles**. Figure 3.14 shows the complete package hierarchy as viewed in the **Systems** tab of SAP HANA Studio.

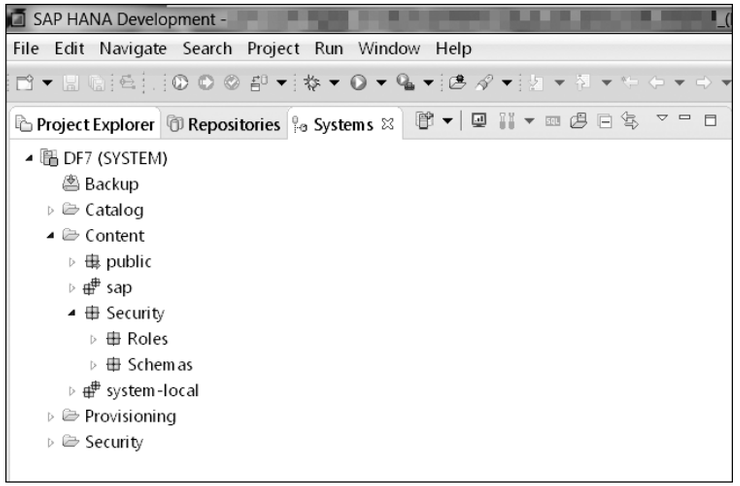


Figure 3.14 E-Corporation's Initial Package Hierarchy

E-Corporation decides to create two repository schemas (using the steps outlined in Section 3.3.1). The first schema will be named `StagingMart` and the second schema named `SalesMart`. The repository schemas will be activated within the SAP HANA package named `Security.Schemas`. The syntax for creating each repository schema is listed in Table 3.1.

Schema Name	Repository Schema Syntax
`StagingMart`	`schema_name = "StagingMart";`
`SalesMart`	`schema_name = "SalesMart";`

Table 3.1 Repository Schema Names and Syntax

Figure 3.15 shows what will be present in the package hierarchy. Under the subpackage **Schemas** will be two objects, both with the *.hdbschema* file extension. Each object or file is preceded by the name of the file and then the name of the schema.

Now that the schemas are in place, the developers can create tables to store the data. The security consultant recommends that the developers use repository-based tables, for two reasons. First, the developers won't need the `CREATE ANY` object privilege for the two schemas, which simplifies the security model. Second, repository tables can be promoted easily with the security model to other SAP HANA environments in order to support sound lifecycle management.

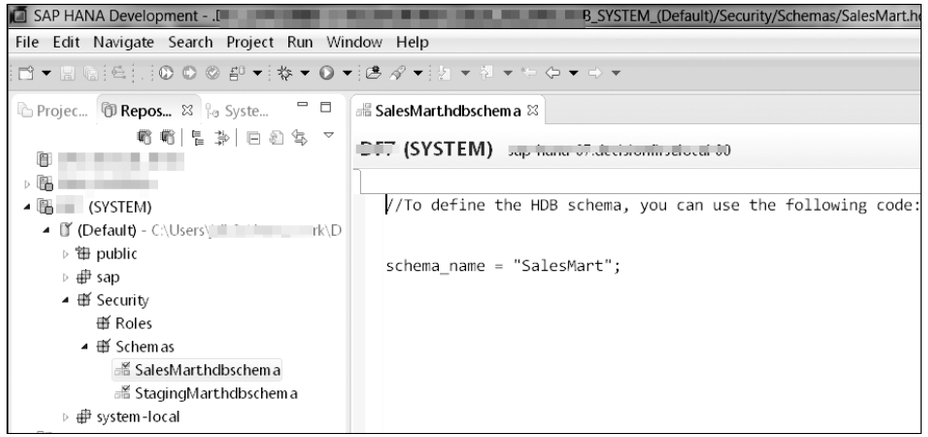


Figure 3.15 Development Repository Schema Artifacts and Corresponding Script Syntax

Because the tables created by the developers are expected to be secured in such a way that developers can frequently make changes, the security consultant recommends that E-Corporation create a new package hierarchy to store development artifacts that will be used by the development team for the sales analytics project. It's best to set up a separate package hierarchy in these instances, because it can be secured independently from the top-level security package. The consultant recommends that a top-level package named `e-corp` be created, and within it, the subpackage named `sales`. Within the `sales` package, he recommends that a package named `tables` be created. Figure 3.16 shows how the package hierarchy will appear within the **System** tab in SAP HANA Studio.

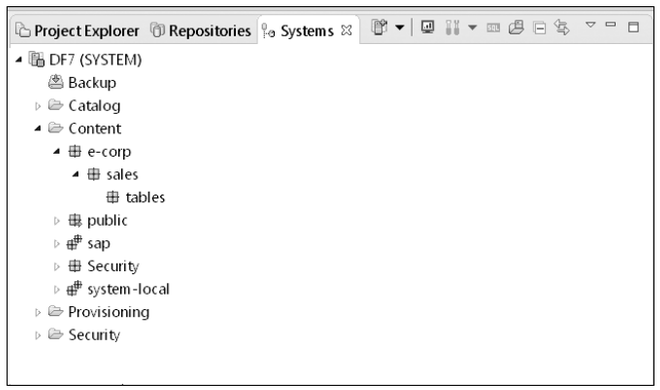


Figure 3.16 E-Corporation's Package Hierarchy

E-Corporation creates the `STG_CUSTOMERS` table (Listing 3.2) and the `STG_PRODUCTS` table (Listing 3.3) in the `StagingMart` schema (following the steps in Section 3.3.2). The repository tables are created within the `e-corp.sales.tables` package. The repository object type will be a database table or an HDBTABLE file.

```
// To define an HDB table with main SQL type columns, use the following code.
// In the following example, a schema should exist.
table.schemaName = "StagingMart";
table.tableType = COLUMNSTORE;
table.columns =
 [
    {name = "CustomerID";  sqlType = BIGINT;  comment = "Customer ID"; },
    {name = "CustomerName";  sqlType = NVARCHAR; length = 255; comment =
 "Customer Name";},
    {name = "CustomerAddressLine1";  sqlType = NVARCHAR; length =
 255; comment = "Customer Street Number and Name";},
    {name = "CustomerCity";  sqlType = NVARCHAR;  length = 255; comment =
 "Customer City";},
    {name = "CustomerLatt"; sqlType = DECIMAL; precision = 26; scale =
 6; comment = "Customer Location Latitude";},
    {name = "CustomerLong"; sqlType = DECIMAL; precision = 26; scale =
 6; comment = "Customer Location Longitude";}
];
table.primaryKey.pkcolumns = ["CustomerID"];
```

Listing 3.2 STG_CUSTOMERS Syntax

```
table.schemaName = "StagingMart";
table.tableType = COLUMNSTORE;
table.columns =
 [
    {name = "CustomerID";  sqlType = BIGINT;  comment = "Customer ID"; },
    {name = "CustomerName";  sqlType = NVARCHAR; length = 255;
comment = "Customer Name";},
    {name = "CustomerAddressLine1";  sqlType = NVARCHAR;
length = 255; comment = "Customer Street Number and Name";},
    {name = "CustomerCity";  sqlType = NVARCHAR;  length = 255;
comment = "Customer City";},
    {name = "CustomerLatt"; sqlType = DECIMAL; precision = 26;
scale = 6; comment = "Customer Location Latitude";},
```

```
    {name = "CustomerLong"; sqlType = DECIMAL; precision = 26;
scale = 6; comment = "Customer Location Longitude";}
];
table.primaryKey.pkcolumns = ["CustomerID"];
```

Listing 3.3 STG_PRODUCTS Syntax

Once the two repository table scripts are activated, two new tables will be visible within the `StagingMart` schema. The names of the tables are shown in Figure 3.17, in the **Systems** tab, under **Catalog • StagingMart • Tables**. Notice how the name of the runtime version of each table includes the repository package hierarchy name and two colons.

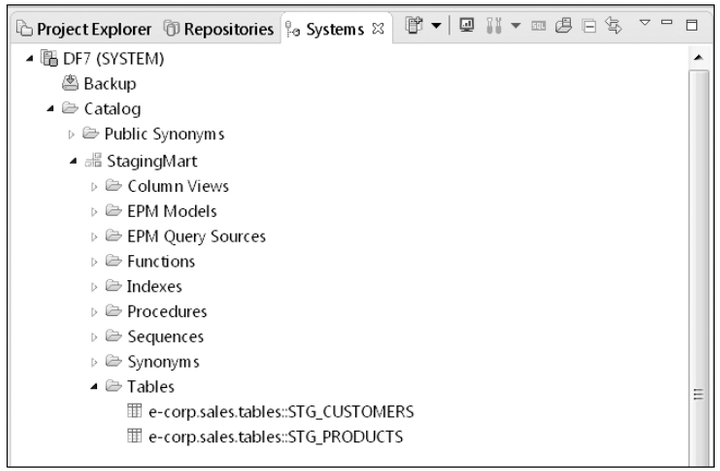


Figure 3.17 Repository Tables Created in StagingMart Schema

SAP HANA requires unique object names for all runtime versions of objects created within the repository. To achieve this requirement, the package hierarchy name is automatically included as a prefix for every catalog object created within the repository.

E-Corporation now has two design-time and runtime versions of each schema and table. The foundation for a sound security model has been implemented. Developers that have been granted privileges for these objects can now begin loading data into these SAP HANA tables. Security administrators can relax, knowing that their security model can be transported and backed up.

3.6 Summary

In this chapter, you learned about catalog objects. How you choose to create catalog objects is one of the most important factors that effects the type of security model and security process that will be implemented. If you create standard catalog objects, then developers and security administrators constantly will need to follow tedious processes for the model to be maintained. However, if you choose to create repository-based catalog objects, then developers mostly can be decoupled from the security model maintenance process. In addition, the security model and its dependent catalog objects can be transported easily to another system or into a backup file.

In the next chapter, we'll dive deeper into the security model by exploring the processes necessary for provisioning users. We'll also discuss the processes necessary to assign privileges and roles to a user account.

Contents

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

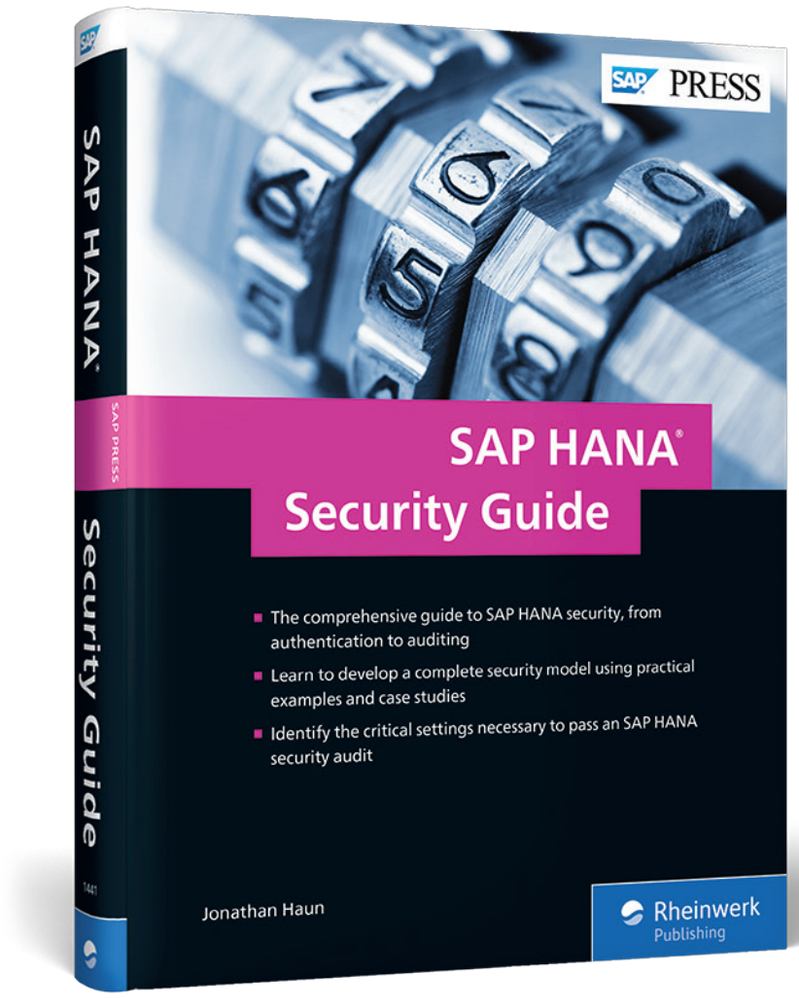
T

U

V

W

X



Jonathan Haun

SAP HANA Security Guide

541 Pages, 2017, $79.95
ISBN 978-1-4932-1441-9

www.sap-press.com/4227

Jonathan Haun currently serves as a director within Protiviti's Data and Analytic Group. Over the past several years, he has had the opportunity to help several clients implement solutions using SAP HANA. In addition to being certified in multiple SAP BusinessObjects BI tools, he also holds the certifications of SAP Certified Application Associate - SAP HANA 1.0, SAP Certified Technology Associate - SAP HANA 1.0 and SAP Certified Technology Specialist - SAP HANA Installations.

Jonathan has worked in the field of business intelligence and database administration for more than 14 years. During this time, he has gained invaluable experience helping customers implement solutions using the tools from the SAP BusinessObjects BI and SAP HANA product lines. Before working as a full-time SAP consultant, he worked in a variety of information technology management and administrative roles. His combination of experience and wealth of technical knowledge make him an ideal source of information pertaining to SAP BusinessObjects BI and SAP HANA. You can follow Jonathan on Twitter at @jdh2n or visit his blog at http://sapbi.blog.