

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.

-  "Catalog Objects"
-  Contents
-  Index
-  The Author

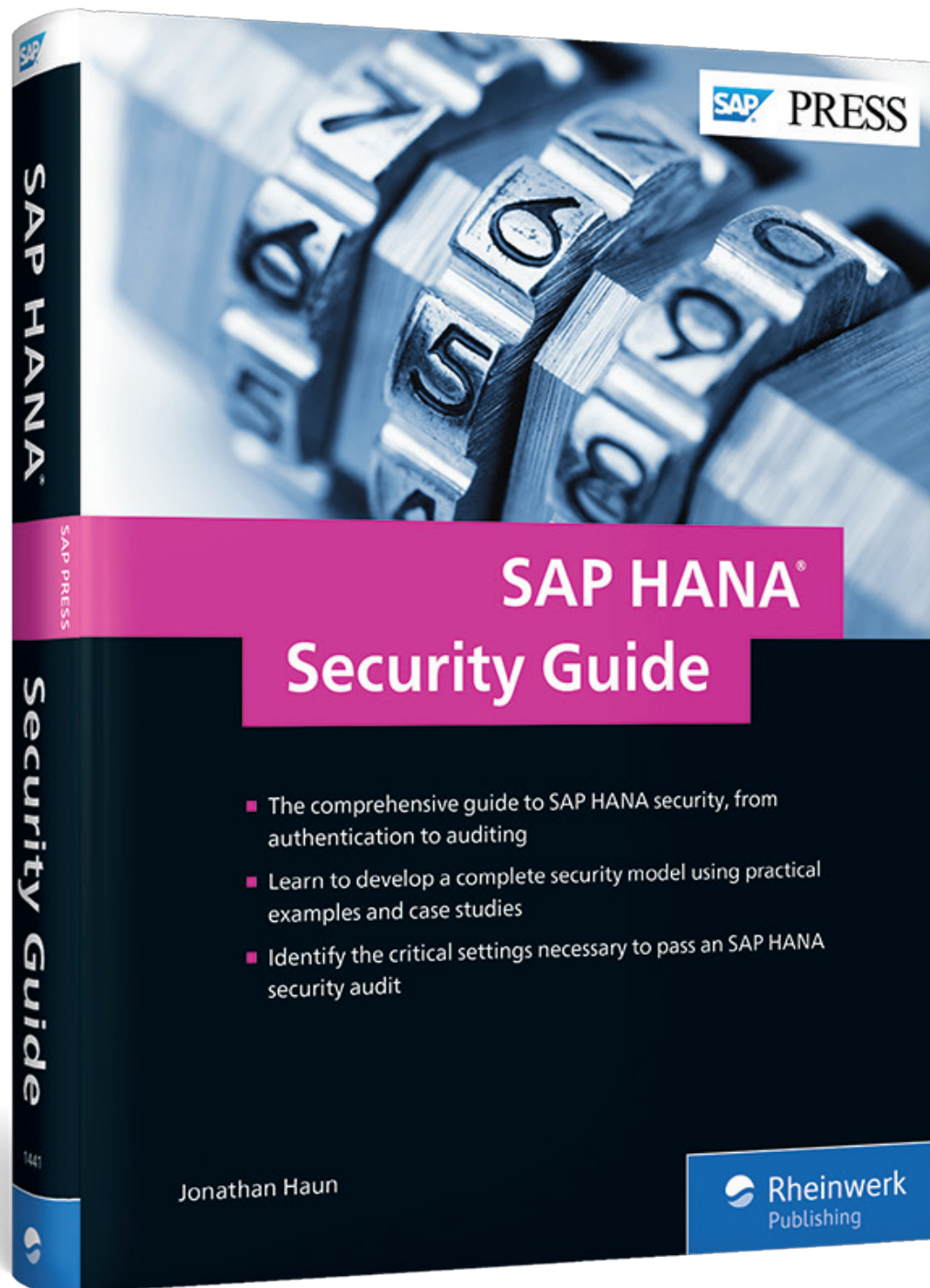
Jonathan Haun

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 as 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.

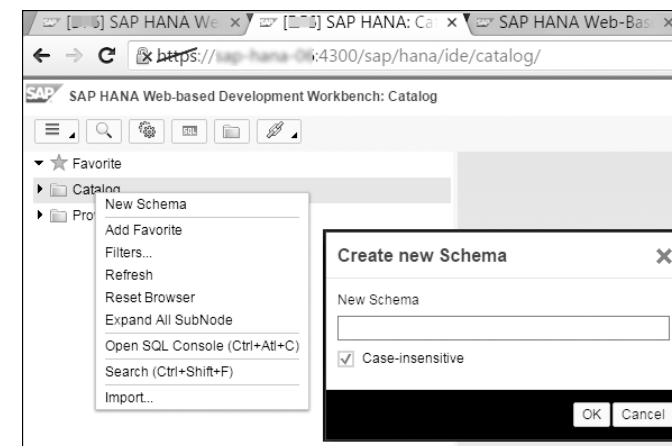


Figure 3.1 Creating Schema with SAP HANA Web-Based Development Workbench

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.

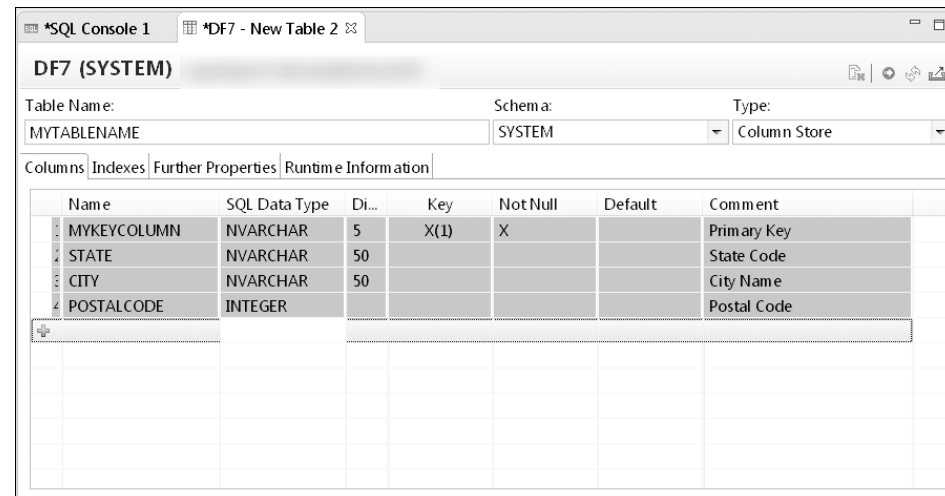


Figure 3.2 Create Table GUI in SAP HANA Studio

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 copied 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.

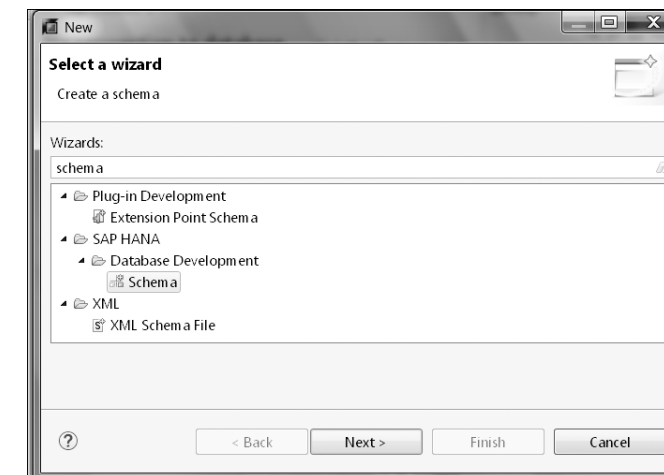


Figure 3.3 Searching for Schema Repository Objects

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.

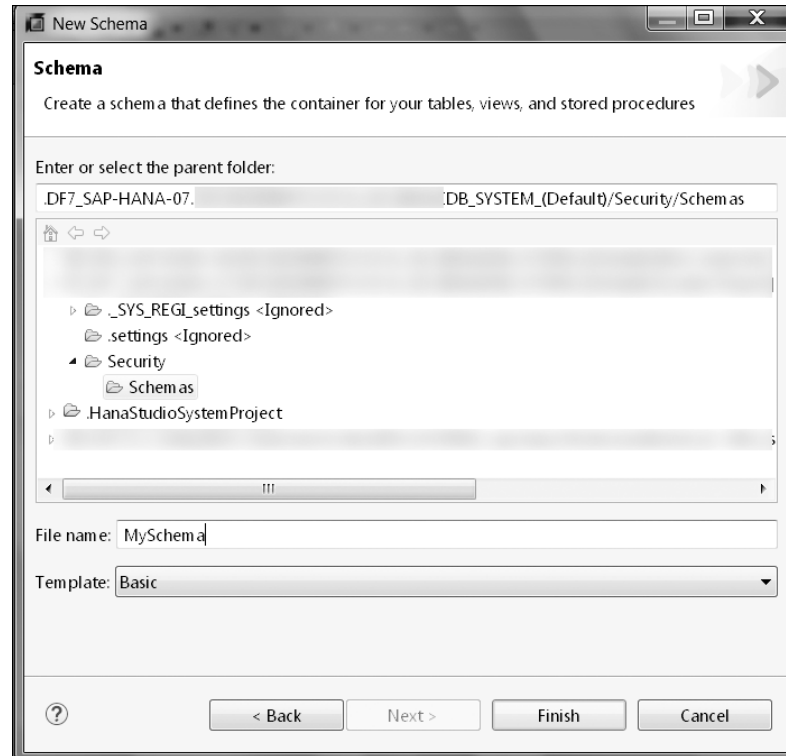


Figure 3.4 Defining Repository Schema Name and Choosing Template

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.

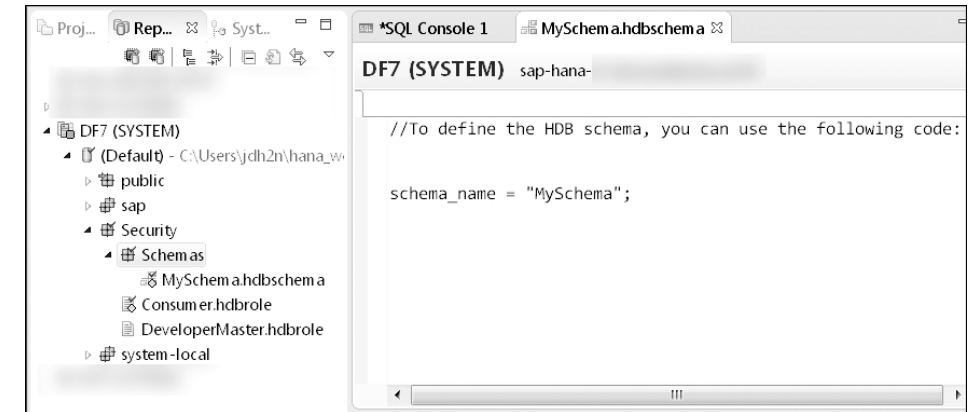


Figure 3.5 Syntax to Define Repository Schema

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 recommend 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.

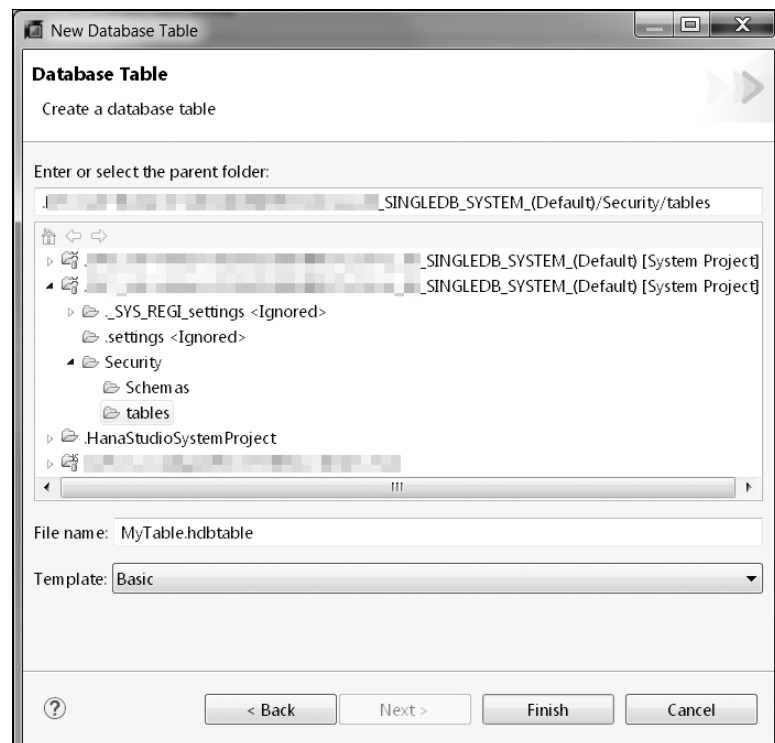


Figure 3.6 Defining Repository Table Name and Choosing Template

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.

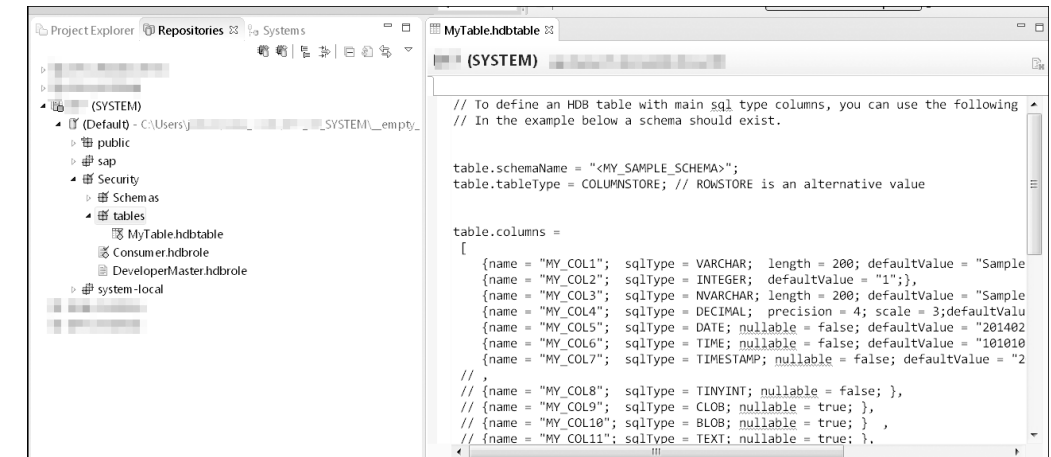


Figure 3.7 Defining Repository Table

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 RECONFIGURE` 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.

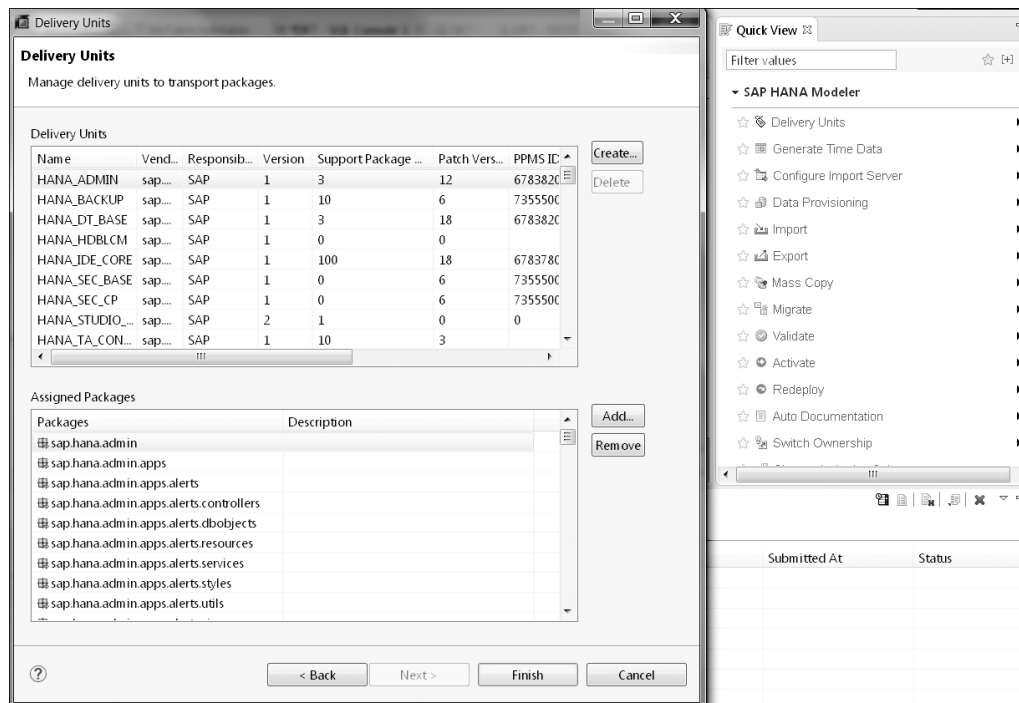


Figure 3.8 Quick View Window and Delivery Units Management Window in SAP HANA Studio

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.

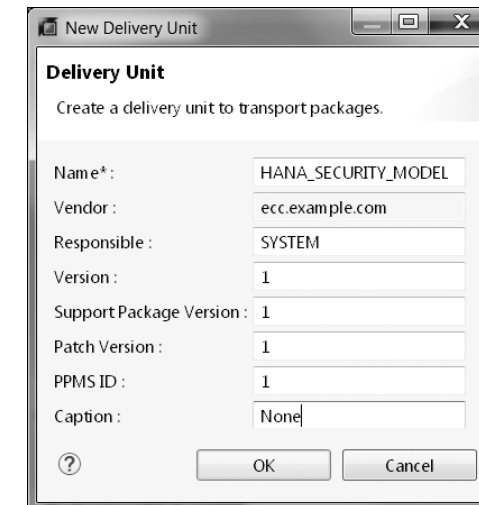


Figure 3.9 New Delivery Unit Configuration Window

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.

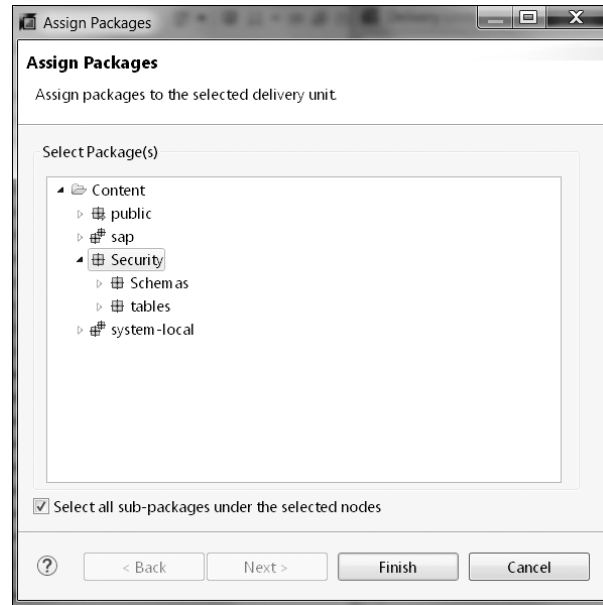


Figure 3.10 Assign Packages Window

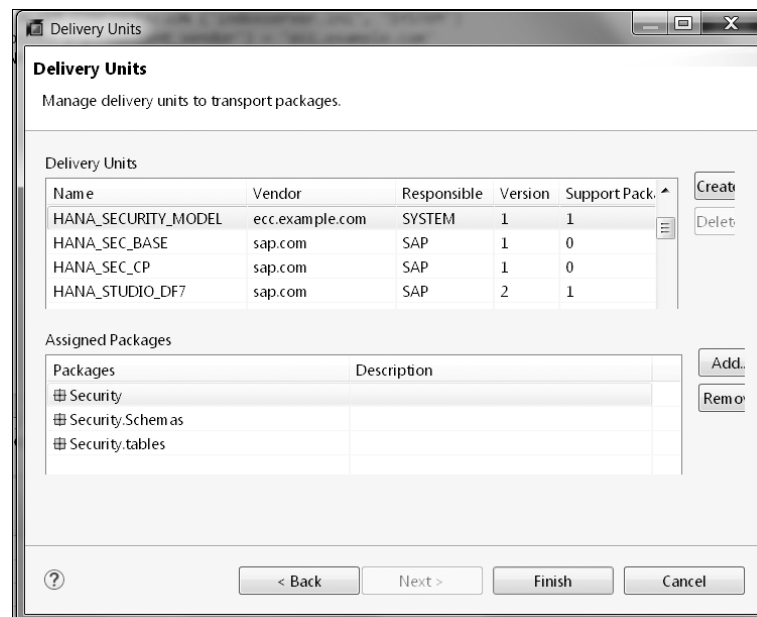


Figure 3.11 Packages Assigned to HANA_SECURITY_MODEL Delivery Unit

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 unit, 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.

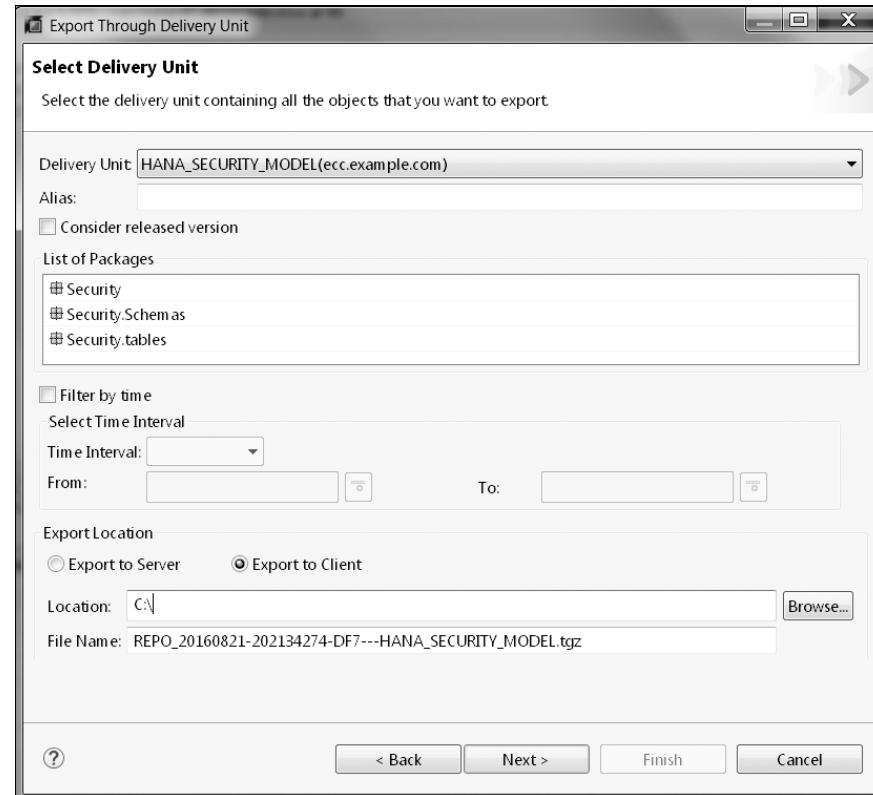


Figure 3.12 Delivery Unit Export Options Window

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.

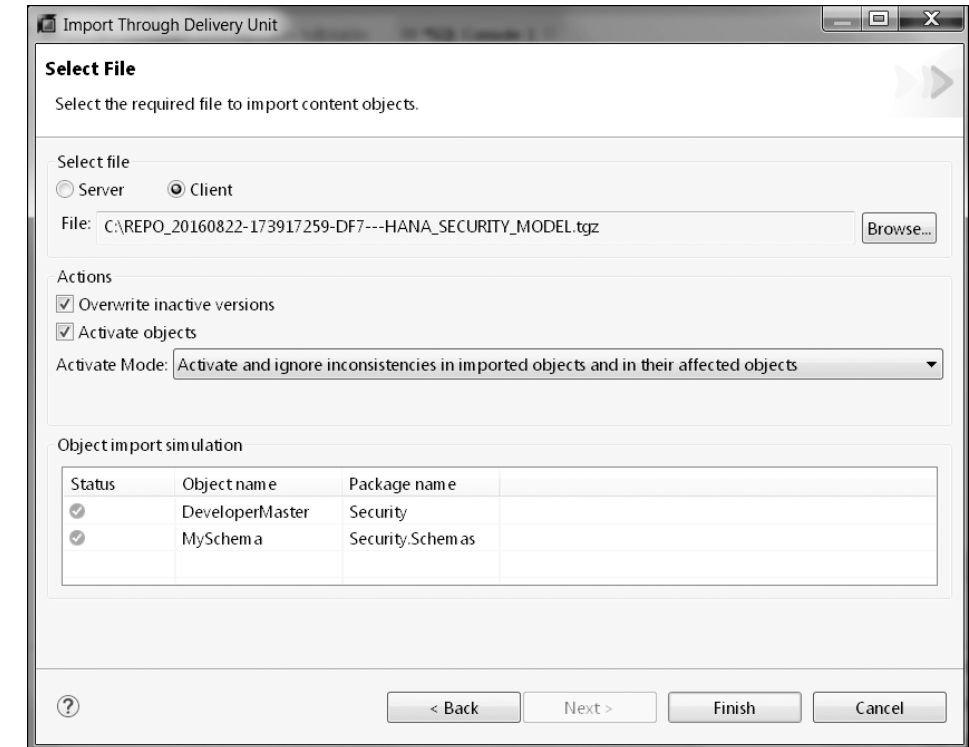


Figure 3.13 SAP HANA Delivery Unit Import Wizard Options

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 activate 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.

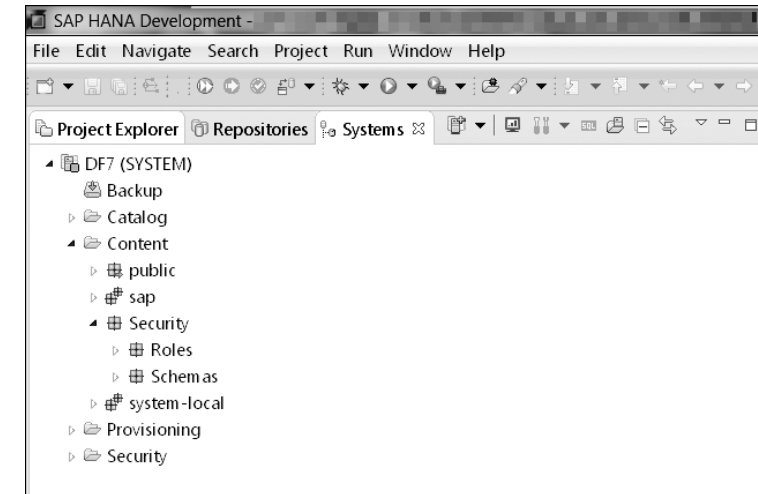


Figure 3.14 E-Corporation's Initial Package Hierarchy

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
<code>StagingMart</code>	<code>schema_name = "StagingMart";</code>
<code>SalesMart</code>	<code>schema_name = "SalesMart";</code>

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.

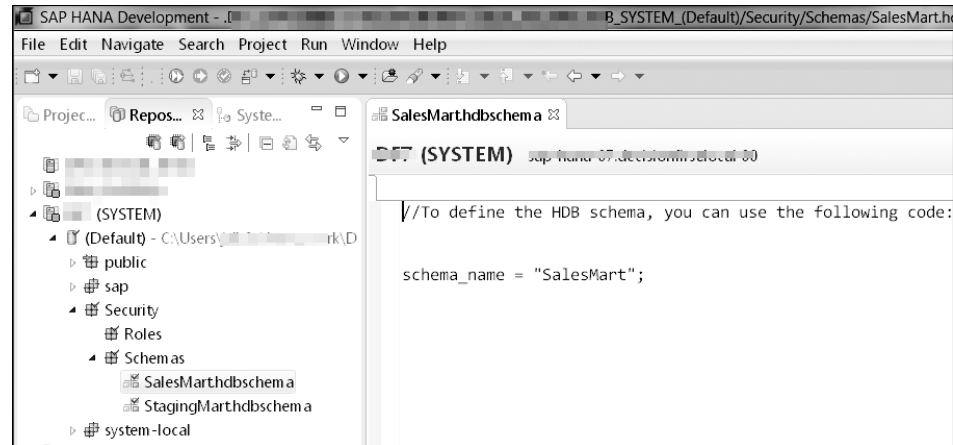


Figure 3.15 Development Repository Schema Artifacts and Corresponding Script Syntax

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.

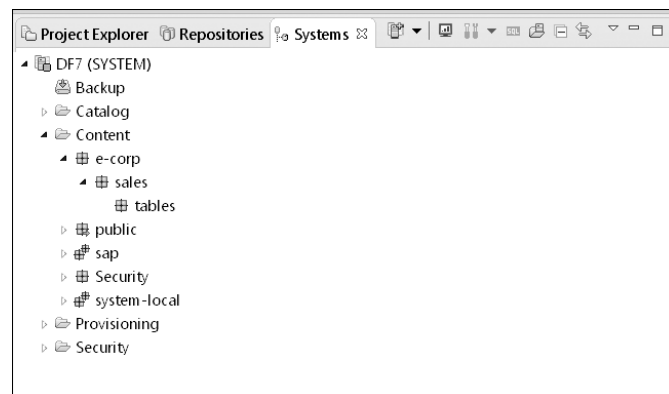


Figure 3.16 E-Corporation's Package Hierarchy

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.

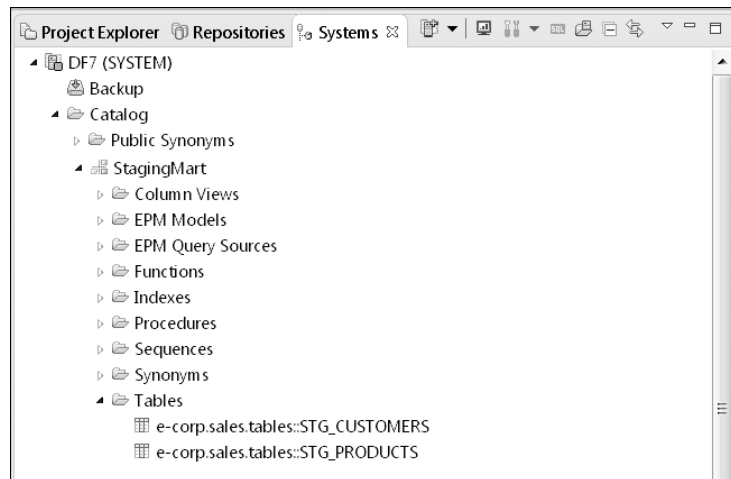


Figure 3.17 Repository Tables Created in StagingMart Schema

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

Preface	19
Introduction	27

1 Managing Security with SAP HANA Studio 39

1.1 SAP HANA Studio Overview	40
1.1.1 Getting Started with SAP HANA Studio	41
1.1.2 Navigating SAP HANA Studio	47
1.2 The Administration Console	57
1.3 Managing Perspectives in SAP HANA Studio	58
1.3.1 Administration Console Perspective	59
1.3.2 Development Perspective	60
1.3.3 Modeler Perspective	66
1.4 SQL Console	69
1.5 Security Settings in SAP HANA Studio	70
1.5.1 User Management	70
1.5.2 Role Management	71
1.5.3 Security Console	71
1.5.4 Development Perspective	71
1.5.5 SQL Console	71
1.5.6 Configuration Tab	72
1.6 Summary	72

2 Introduction to SAP HANA Privileges 73

2.1 Privileges within SAP HANA	74
2.1.1 System Privileges	74
2.1.2 Object Privileges	75
2.1.3 Analytic Privileges	77
2.1.4 Package Privileges	78
2.1.5 Application Privileges	79

2.2	Privilege Validation and Assignment	79
2.2.1	Assigning Privileges	80
2.2.2	Validating Privileges	81
2.3	Summary	83
3	Catalog Objects	85
3.1	What Are SAP HANA Catalog Objects?	85
3.2	Creating and Managing Native Catalog Objects	87
3.2.1	Creating Schemas	88
3.2.2	Creating Catalog Tables	91
3.2.3	Creating Other Catalog Objects	93
3.3	Creating and Managing Repository Catalog Objects	94
3.3.1	Creating Repository Schemas	95
3.3.2	Creating Repository Tables	98
3.4	Deploying Repository Objects	100
3.5	Case Study	108
3.6	Summary	113
4	User Accounts	115
4.1	What Are User Accounts?	115
4.1.1	Standard User Accounts	116
4.1.2	Technical User Accounts	117
4.1.3	Restricted User Accounts	118
4.2	Creating and Managing User Accounts	119
4.2.1	Creating and Managing Users with SQL Statements	120
4.2.2	Creating and Managing Users in SAP HANA Studio	121
4.2.3	Creating and Managing Users with SAP HANA Web-Based Development Workbench	124
4.2.4	User Account System Views	126
4.2.5	Deleting User Accounts	129

4.3	Granting and Revoking Privileges	133
4.3.1	Granting and Revoking Privileges with SQL	133
4.3.2	Granting and Revoking Privileges with SAP HANA Studio	141
4.3.3	Granting and Revoking Privileges with SAP HANA Web-Based Development Workbench	146
4.3.4	Effective Privileges System View	148
4.4	Managing User Role Assignments	149
4.4.1	Granting and Revoking Roles with SQL	150
4.4.2	Granting and Revoking Roles with SAP HANA Studio	152
4.4.3	Granting and Revoking Roles with SAP HANA Web-Based Development Workbench	154
4.4.4	Effective Roles System View	154
4.5	Case Study: Provisioning Users with SQL Scripts and Stored Procedures	155
4.5.1	Creating a Repository Schema	156
4.5.2	Creating a Repository Table	157
4.5.3	Importing a CSV File into a Table	158
4.5.4	Creating Repository Stored Procedures	161
4.5.5	Executing the Repository Stored Procedure	165
4.6	Summary	166
5	Database Roles	167
5.1	What Are Roles?	167
5.2	Creating and Managing Roles	171
5.2.1	Creating and Deleting Roles with SQL Statements	171
5.2.2	Creating and Deleting Roles with SAP HANA Studio	172
5.2.3	Creating and Deleting Roles with SAP HANA Web-Based Development Workbench	174
5.3	Granting and Revoking Privileges	176
5.3.1	Methodologies for Granting Privileges to Roles	176
5.3.2	Granting and Revoking Privileges with SQL	178
5.3.3	Granting and Revoking Privileges with SAP HANA Studio	187

5.3.4	Granting and Revoking Privileges with SAP HANA Web-Based Development Workbench	192
5.4	Managing Nested Roles	194
5.4.1	Granting and Revoking Roles with SQL	194
5.4.2	Granting and Revoking Roles with SAP HANA Studio	195
5.4.3	Granting and Revoking Roles with SAP HANA Web-Based Development Workbench	196
5.5	Summary	197
6	Repository Roles	199
6.1	What Are Repository Roles?	199
6.1.1	User Account _SYS_REPO and Repository Roles	200
6.1.2	Grantor and Privileges	202
6.1.3	Grantor and Roles	203
6.1.4	Why Use Repository Roles?	203
6.2	Managing Repository Roles with Design-Time Scripts	205
6.2.1	Creating Repository Roles within a Package	206
6.2.2	Defining the Role Name Tag	207
6.2.3	Extending Roles	208
6.2.4	Assigning Privileges	208
6.2.5	Save and Activate	209
6.2.6	Runtime Repository Roles	210
6.3	Granting and Revoking Privileges in Design-Time Scripts	211
6.3.1	System Privileges	211
6.3.2	Schema Privileges	212
6.3.3	Object Privileges	214
6.3.4	Structured Privileges	215
6.3.5	Remote Sources	216
6.3.6	Analytic Privileges	217
6.3.7	Application Privileges	217
6.3.8	Package Privileges	218
6.4	Managing Repository Roles with SAP HANA Web-Based Development Workbench	218

6.4.1	Accessing and Navigating the SAP HANA Web-Based Development Workbench Editor	219
6.4.2	System Privileges	222
6.4.3	Object Privileges	223
6.4.4	Analytic Privileges	226
6.4.5	Package Privileges	227
6.4.6	Application Privileges	229
6.5	Granting Repository Roles to Users	231
6.5.1	Granting and Revoking Repository Roles with Stored Procedures	231
6.5.2	Granting and Revoking Repository Roles with SAP HANA Studio	232
6.5.3	Granting and Revoking Repository Roles with SAP HANA Web-Based Development Workbench	234
6.6	Case Study: Creating Basic Repository Roles	234
6.6.1	Consumer Repository Role	235
6.6.2	Power User Repository Role	236
6.6.3	Developer Repository Role	236
6.6.4	Security Administrator Repository Role	238
6.7	Summary	239
7	System Privileges	241
7.1	What Are System Privileges?	241
7.2	Default System Privileges	242
7.2.1	Developer-Related System Privileges	242
7.2.2	Security Admin-Related System Privileges	243
7.2.3	System Admin-Related System Privileges	246
7.2.4	Environment Monitoring-Related System Privileges	252
7.2.5	Environment Performance-Related System Privileges	252
7.3	Granting System Privileges	253
7.3.1	Granting System Privileges with SQL	253
7.3.2	Granting System Privileges with SAP HANA Studio	254
7.3.3	Granting System Privileges with SAP HANA Web-Based Development Workbench	256
7.3.4	Granting System Privileges with Repository Roles	257

7.4	Case Study: Security Administrator System Privileges	262
7.4.1	User Management Role	262
7.4.2	Role Management Role	264
7.4.3	Data and Communication Encryption Role	265
7.4.4	System Auditing Role	266
7.5	Summary	267
8	Object Privileges	269
<hr/>		
8.1	Overview of Object Privileges	269
8.1.1	Catalog Object Privileges	270
8.1.2	Security Considerations for Catalog Objects	275
8.2	Granting Object Privileges with SQL	279
8.2.1	Securing Schemas with SQL	280
8.2.2	Securing Individual Catalog Objects with SQL	282
8.3	Granting Object Privileges with SAP HANA Studio	284
8.4	Granting Object Privileges with Repository Roles	286
8.4.1	Script-Based Repository Roles	287
8.4.2	SAP HANA Web-Based Development Workbench	289
8.5	Case Study: Updating Repository Roles to Access Information Views	292
8.5.1	Consumer	292
8.5.2	Power User	293
8.5.3	Developer	295
8.6	Summary	296
9	Package Privileges	297
<hr/>		
9.1	The SAP HANA Development Repository	297
9.1.1	Structure of the Development Repository	297
9.1.2	Creating Packages and Subpackages	298
9.1.3	Overview of Delivery Units	300
9.2	Overview of Package Privileges	301

9.3	Granting Package Privileges	303
9.3.1	Granting Package Privileges with SQL	303
9.3.2	Granting Package Privileges with SAP HANA Studio	304
9.3.3	Granting Package Privileges with SAP HANA Web-Based Development Workbench	305
9.3.4	Granting Package Privileges within Repository-Based Roles	307
9.4	Case Study: Preventing Content Developers from Elevating Their Privileges	311
9.4.1	Assessing the Current Configuration	311
9.4.2	Recommendations	312
9.5	Summary	315
10	Analytic Privileges	317
<hr/>		
10.1	Overview of SAP HANA Information Views	317
10.1.1	Attribute Views	318
10.1.2	Analytic Views	318
10.1.3	Calculation Views	319
10.2	Overview of Analytic Privileges	320
10.2.1	XML-Based Analytic Privileges	320
10.2.2	SQL-Based Analytic Privileges	323
10.3	_SYS_BI_CP_ALL: A System-Generated Analytic Privilege	325
10.4	Managing Static Analytic Privileges	326
10.4.1	Creating Static XML-Based Analytic Privileges	326
10.4.2	Creating Static SQL-Based Analytic Privileges	331
10.5	Managing Dynamic Analytic Privileges	334
10.5.1	Dynamic XML-Based Analytic Privileges	334
10.5.2	Dynamic SQL-Based Analytic Privileges	336
10.6	Managing Dynamic Expression-Based SQL Analytic Privileges	344
10.6.1	Creating a Repository-Based Security Table	346
10.6.2	Defining Dynamic Expression-Based SQL Analytic Privileges	348
10.7	Troubleshooting Effective Analytic Privileges and Filter Conditions	351

10.8 Granting Analytic Privileges	352
10.8.1 Granting Analytic Privileges with SQL	352
10.8.2 Granting Analytic Privileges with SAP HANA Studio	353
10.8.3 Granting Analytic Privileges with SAP HANA Web-Based Development Workbench	354
10.8.4 Granting Analytic Privileges within Repository Roles	355
10.9 Summary	359
11 Application Privileges	361
<hr/>	
11.1 Application Privileges in SAP HANA	361
11.2 Creating Application Privileges	362
11.3 Granting Application Privileges	364
11.3.1 Granting Application Privileges with SQL	364
11.3.2 Granting Application Privileges with SAP HANA Studio	365
11.3.3 Granting Application Privileges with SAP HANA Web-Based Development Workbench	366
11.3.4 Granting Application Privileges within Repository Roles	367
11.4 Privileges on Users	372
11.4.1 Granting Privileges on Users with SAP HANA Studio	372
11.4.2 Granting Privileges on Users with SQL	373
11.5 Summary	373
12 Authentication	375
<hr/>	
12.1 SAP HANA Internal Authentication Mechanism	376
12.1.1 Protecting SAP HANA Passwords with Encryption	376
12.1.2 Configuring the Internal Authentication Password Policy	377
12.1.3 Managing Password Policy Settings with SQL	383
12.1.4 Managing Password Policy Settings in GUIs	384
12.2 Supported Third-Party Authentication Providers	389
12.2.1 Kerberos Authentication	390
12.2.2 SAML Authentication	393

12.2.3 X509 Authentication	396
12.2.4 SAP Logon Ticket	397
12.2.5 SAP Assertion Ticket	399
12.3 Case Study: Adding SAML Identity User Accounts	400
12.4 Summary	402
13 Certificate Management and Encryption	403
<hr/>	
13.1 SSL Certificates	403
13.1.1 In-Database Certificate Management	404
13.1.2 External SAP HANA PSE File and Certificate Management	408
13.2 Client Encryption Settings	412
13.2.1 SAP HANA Studio	412
13.2.2 XS Engine Web-Based Applications	414
13.2.3 JDBC and ODBC Drivers	416
13.3 Encrypting Data	419
13.3.1 Server-Side Data Encryption	420
13.3.2 Changing New Root Keys within the SSFS	421
13.3.3 Encrypting the Data Volume	424
13.3.4 Encrypting the Log Volume	425
13.4 Summary	426
14 Security Lifecycle Management	427
<hr/>	
14.1 Maintaining a Consistent Security Model	427
14.1.1 Best Practices	428
14.1.2 Testing Security Model Changes	430
14.1.3 Keeping Repository Roles in Sync	432
14.2 Create Delivery Units for Security-Related Packages	434
14.2.1 Creating a Delivery Unit with SAP HANA Studio	435
14.2.2 Creating a DU with SAP HANA Application Lifecycle Management	438

14.3 Transport Security Packages to Other SAP HANA Systems	442
14.3.1 Transport a DU with SAP HANA Application Lifecycle Management	443
14.3.2 Export a DU to a File	448
14.3.3 Import a DU from a File	449
14.4 Additional Options in SAP HANA Application Lifecycle Management	452
14.4.1 Change Recording	452
14.4.2 Using SAP CTS	453
14.5 Summary	453
15 Auditing	455
15.1 Why Do We Need Auditing?	455
15.2 Configuring Auditing	457
15.2.1 Enable Auditing with SAP HANA Studio	457
15.2.2 Enable Auditing with SAP HANA Web-Based Development Workbench	461
15.2.3 Enable Auditing with SQL	463
15.3 Creating Audit Policies	465
15.3.1 Components of the Audit Policy	466
15.3.2 Managing Policies with SAP HANA Web-Based Development Workbench	471
15.3.3 Managing Audit Policies with SQL	473
15.3.4 Creating Policies with SAP HANA Studio	476
15.4 Querying Audit Data	477
15.4.1 AUDIT_ACTIONS	478
15.4.2 AUDIT_LOG	478
15.4.3 AUDIT_POLICIES	478
15.5 Case Study: Defining Audit Policies	479
15.5.1 Proactive Event Monitoring	479
15.5.2 Audit Reporting	480
15.5.3 Authentication Auditing	480
15.5.4 Unauthorized Action Auditing	481
15.5.5 System Change Auditing	482
15.5.6 Security Management Task Auditing	483

15.5.7 Super User Event Auditing	485
15.6 Summary	486
16 Security Tracing and Troubleshooting	487
16.1 Authorization Tracing	487
16.1.1 Enable Tracing with SAP HANA Studio	488
16.1.2 Enable Tracing with SQL	491
16.1.3 Viewing the Trace File in SAP HANA Studio	493
16.2 Query the System to Review Effective Privileges	495
16.2.1 Granted Privileges	495
16.2.2 Granted Roles	496
16.2.3 Accessible Views	497
16.2.4 Effective Privilege Grantees	498
16.2.5 Effective Structured Privileges	499
16.2.6 Effective Privileges	501
16.2.7 Effective Role Grantees	502
16.2.8 Effective Roles	503
16.3 Case Study: Identifying Deficiencies in Information View Access	504
16.3.1 Troubleshooting the Problem	504
16.3.2 Reviewing the Results	505
16.3.3 Reviewing the Solution	506
16.4 Summary	506
17 Security Recommendations	507
17.1 Password Authentication Settings	507
17.1.1 Standard User Password Policies	507
17.1.2 Service Accounts	510
17.2 Encryption Settings	511
17.3 Identifying Users with Elevated Privileges	511
17.3.1 System Privileges	512
17.3.2 Root Package Privileges	514

17.3.3	Bypass Analytic Privileges	515
17.3.4	Default Standard Roles	518
17.3.5	WITH GRANT or WITH ADMIN	519
17.4	Disabling the SYSTEM Account	520
17.5	Identify Privilege Escalation Vulnerabilities	521
17.6	Handover from Hardware Vendors	522
17.7	Create Audit Policies	523
17.8	Summary	523
18	SAP HANA 2.0 Security	525
<hr/>		
18.1	Authorizations	525
18.1.1	Granting or Revoking the PUBLIC Role	525
18.1.2	Granting or Revoking Access to a User's Own Schema	526
18.1.3	Map LDAP Groups to SAP HANA Roles	527
18.2	Encryption	527
18.2.1	Log Volume Encryption	527
18.2.2	Root Key Backup and Password	527
18.2.3	Using SQL to Update All Encryption Keys	528
18.3	XS Engine Applications and Roles	528
18.4	SAP HANA 2.0 Cockpit	529
18.5	Summary	529
	The Author	531
	Index	533

Index

A

ABAP Workbench	453	Attribute views	77, 318
Accessible views	497	Audit action status	469
ACID compliance	30	Audit actions view	478
Active Directory	376, 391	Audit data	477
Administration console	57	Audit level trail targets	459
Administration console perspective	59	Audit levels	459, 470
Administration editor	488	Audit log data	456
Analytic privileges ... 68, 77, 148, 190, 193, 215, 217, 226, 244, 317, 344, 469		Audit log database table	460
<i>bypassing</i>	515	Audit log targets	464
<i>default</i>	325	Audit log view	478
<i>filters</i>	321	Audit operator	244
<i>granting</i>	352	Audit operator system privilege	513
<i>management window</i>	328	Audit policies	465, 472, 479, 523
<i>overview</i>	320	<i>audit trail target</i>	471
<i>repository roles</i>	355	<i>components</i>	466
SAP HANA Studio	353	<i>creation</i>	474, 476
SAP HANA Web-Based Development		<i>management</i>	471, 473
Workbench	354	SAP HANA Studio	476
<i>search</i>	359	SQL	475
SQL	352	<i>target objects</i>	470, 473
SYS_BI_CP_ALL	516	<i>target user</i>	470
<i>troubleshooting</i>	351	Audit policies view	478
Analytic views	77, 318	Audit reporting	480
Application function library (AFL)	34	Audit rule	459
Application privilege objects	363	Audit trail	464
Application privileges ... 79, 140, 185, 217, 229, 361, 366, 371		Auditing	51, 434, 455, 488
<i>creation</i>	362	<i>all actions</i>	466
<i>creation guide</i>	364	<i>backup deletion</i>	466
<i>granting</i>	364	<i>benefits</i>	455
SAP HANA Studio	365	<i>case study</i>	479
SAP HANA Web-Based Development		<i>certificate and PSE store</i>	466
Workbench	366	<i>configuration</i>	457
SQL	364	<i>data definition</i>	466
<i>within repository roles</i>	367	<i>data provisioning</i>	467
Application-access file	363	<i>data query and manipulation</i>	467
Application-privileges file	362	<i>data volume encryption</i>	467
Atomicity, consistency, isolation, and		<i>granting and revoking authorizations</i> ...	467
durability (ACID)	30	LDAP provider manager	468
Attribute data	317	<i>license deletion</i>	468
		<i>license installation</i>	468
		<i>monitoring</i>	456
		<i>regulatory compliance</i>	456
		<i>repository content operations</i>	468

- Auditing (Cont.)
 - SAP HANA dynamic tiering* 468
 - SAP HANA Studio* 457
 - SAP HANA Web-Based Development Workbench* 461
 - session management and system configuration* 468
 - SQL* 463
 - structured privileges management* 469
 - user and role management* 469
 - Authentication 115, 375, 404
 - auditing* 480
 - case study* 400
 - Authorization component 487
 - Authorization trace 487, 490
 - configuration* 490
 - Authorization validation 82
 - Authorizations 115, 375
 - catalog object-level* 277
 - grantable to others* 276
 - schema-level* 277
- ## B
- Backup admin system privilege 247
 - Basis administrator 90
 - Business Function Library (BFL) 28
- ## C
- Calculation views 77, 319
 - CASCADE option 130
 - Case studies 108, 155, 234, 262, 292, 311, 400, 479, 504
 - Catalog object access 177
 - Catalog object privileges 76, 136, 181, 270, 291–292
 - Catalog objects 75, 85, 99, 269, 277, 285, 287, 499
 - case study* 108
 - creatopm* 93
 - securing* 282
 - security* 275
 - Catalog read system privilege 252
 - Catalog schema privileges 134, 180
 - Catalog schemas 89–90, 270, 288
 - Catalog sequence 275
 - Catalog stored procedures 273, 278
 - Catalog tables 271, 283
 - creation* 91
 - Catalog views 93, 272, 278
 - Central processing unit 29
 - Certificate admin 244
 - Certificate collection 404, 406
 - Certificate keys 409
 - Certificate management 403, 407
 - external* 408
 - Certificate store 405
 - Certificates 404
 - configuration* 412
 - Change recording 452
 - Change tracking 452
 - Child objects 76
 - referenced* 81
 - Client certificate 397
 - Compile server 34
 - Consumer role 292
 - Content vendor 100–101
 - CONTENT_ADMIN role 518
 - CPU 29
 - Create scenario system privilege 247
 - Create schema 74
 - CREATE statement 200
 - Create users 164
 - CSV file 158, 459, 464
- ## D
- Data admin system privilege 248, 512
 - Data and communication encryption
 - role* 265
 - Data Definition Language (DDL) 512
 - Data encryption 423
 - Data source name (DSN) 417
 - Data volume encryption 51, 424, 467
 - SAP HANA Studio* 424
 - SQL* 424
 - Database accounts 522
 - Database forensics 457
 - Database objects 75, 85, 204
 - Database roles 167
 - consistency* 170
 - standard* 168
 - Database store 405
 - Database views 77
 - DBA_COCKPIT role 169

- Default roles 518
 - Definer mode 82, 278
 - Delivery units 100, 102, 204, 242, 297, 429
 - assign packages* 103, 441
 - creation* 434
 - export options* 448
 - export to file* 448
 - import* 106
 - import from file* 449
 - management window* 103, 436
 - overview* 300
 - packages* 436
 - SAP HANA application lifecycle management* 438
 - SAP HANA Studio* 435
 - Design-time objects 88, 94, 99, 214, 223
 - Design-time repository objects 215
 - Design-time schemas 213
 - Design-time scripts 170, 205, 307, 356, 368
 - Design-time stored procedure 343
 - Developer repository role 295
 - Development (DEV) 428
 - Development artifacts 82, 204, 302
 - Development objects 65
 - Development perspective 60, 71
 - Development repository 297
 - structure* 297
 - Dimensional calculation views 319
 - DU
 - transport* 443
 - Dynamic analytic privileges 320, 322
 - management* 334
 - Dynamic expression-based SQL analytic
 - privileges* 325
 - definition* 348
 - management* 344
 - mapping table* 345
 - Dynamic random-access memory (DRAM) 29
 - Dynamic SQL-based analytic privileges 324, 336
 - Dynamic XML-based analytic privileges 334
- ## E
- Effective privileges 148
 - Effective roles 154
- Encrypting data 419
 - Encryption 403, 511
 - settings* 412
 - Encryption root key admin 245
 - Encryption root keys 523
 - Event auditing 485
 - Event monitoring 479
 - Events 459–460
 - Export system privilege 249
 - Expression editor interface 350
 - External authentication 376
- ## F
- Federated identity system 393
 - Filter conditions 329, 340, 498
 - troubleshooting* 351
 - Filter criteria 322
 - Filter expressions 325, 350
 - Filters 321
 - Functions 87
- ## G
- Geospatial engines 28
 - Grant procedure 232
 - GRANT statement 199
 - Granted privileges 495
 - Granted roles 187, 496
 - Grantee management 304
 - Grantor 73, 171, 202
 - Graphical analytic privilege objects 324
- ## H
- HTTP communication encryption 409
 - HTTP/HTTPS 377
- ## I
- Identity provider (IdP) 393
 - Import job 451
 - Import or export privilege 74
 - Import system privilege 249
 - Imported package privileges 79, 301
 - Inactive user accounts 382

- Including repository roles 312
 - In-database certificate management 404
 - Index 87
 - Index server 33
 - Information models 329
 - Information views 66, 317, 327, 332, 343, 350, 504, 516–517
 - analytic privilege checks* 517
 - overview* 317
 - Inheritance 302
 - INI settings 377
 - INIFILE admin system privilege 514
 - In-memory database 28
 - Input parameter 151
 - Instance SSFS 420
 - Internal database table 458
 - Invoker mode 82, 278
- J**
- Java Database Connectivity (JDBC) 35, 46
 - JDBC 169, 377, 416
 - driver* 418
- K**
- Kerberos 375, 390, 392
 - Kerberos authentication 44
 - Kerberos identity 510
 - Key distribution center (KDC) 390
 - Knowledge Base Articles (KBA) 391
- L**
- LDAP 376
 - LDAP system 158
 - Licenses 468
 - Lifecycle management 170
 - Linux syslog 458, 464
 - Lockout policy 509
 - Log volume encryption 423, 425
- M**
- Mapping data 338
 - Master data 318
 - Modeler perspective 66, 102
 - Modeling objects 67
 - MODELING role 168, 519
 - Multidimensional modeling 319
- N**
- Name server 34
 - Native catalog objects 85
 - Native package privileges 79, 301–302
 - Nested roles 167, 194
 - Network hops 412
- O**
- OASIS 393
 - Object ownership 76, 277
 - Object privileges 75, 143, 148, 189, 193, 214, 223, 269
 - case study* 292
 - overview* 269
 - repository roles* 286
 - SAP HANA Studio* 284
 - SQL* 279
 - Objects 310, 318
 - ODBC 169, 377, 416
 - data source administrator* 417
 - ODBC/JDBC access 118
 - ODBC/JDBC connectivity 526
 - Online transaction processes (OLTP) 28
 - Open Database Connectivity (ODBC) 35
- P**
- Package hierarchy 99, 109, 156, 291, 298, 312–313, 327, 356, 416
 - Package privileges 78, 140, 146, 148, 186, 191, 193, 218, 227, 297, 314
 - case study* 311
 - granting* 303
 - overview* 301
 - SAP HANA Studio* 304
 - SAP HANA Web-Based Development Workbench* 305
 - SQL* 303
 - Packages 49, 206, 297, 310
 - creation* 298
 - Parent objects 81

- Parent schemas 76
 - Part of roles 188
 - Password manager 510
 - Password policy 51
 - configuration* 377
 - manage settings* 383
 - SAP HANA Studio configuration* 384
 - SAP HANA Studio security manager* 386
 - SAP HANA Web-Based Development Workbench* 388
 - security settings* 377
 - Passwords 115, 122
 - authentication* 375, 507
 - blacklist* 387
 - change initial* 379
 - complexity* 378, 508
 - encryption* 376
 - expiration* 383
 - failed logon attempts* 380
 - initial* 509
 - length* 378
 - lifetime* 381, 508
 - lock time* 381
 - policies* 507
 - reuse* 508
 - reuse settings* 380
 - special characters* 379
 - update* 165
 - Personal security environment (PSE) 404, 406
 - Personal security environment (PSE)
 - files* 409
 - in-database* 407
 - manage certificates* 406
 - management options* 411
 - Perspectives 58
 - Planning engines 28
 - Power user role 293
 - Predictive Analytics Library (PAL) 28
 - Preprocessor server 34
 - Privilege escalation vulnerabilities 521
 - Privilege grantees 498
 - Privilege types 209
 - Privileges 35, 73, 124, 177, 202, 311, 501
 - assignment* 80, 208
 - granting and revoking* 133, 176
 - Privileges (Cont.)
 - granting to roles* 176
 - modification* 284
 - SAP HANA Studio* 187
 - SAP HANA Web-Based Development Workbench* 192
 - SQL* 178
 - validation* 81
 - Privileges on users 361, 372
 - SAP HANA Studio* 372
 - SQL* 373
 - Product availability matrix (PAM) 31
 - Production (PROD) 428
 - Profitability Analysis (CO-PA) 318
 - Provisioning users case study 155
 - Public key infrastructure (PKI) 396, 403
 - PUBLIC role 169
- Q**
- Quality assurance (QA) 428
- R**
- Relational database management system (RDBMS) 28, 85, 118, 199, 361
 - Remote source catalog object privileges 216
 - Remote source privileges 141
 - Remote sources 139, 184, 274
 - REPO.READ privileges 314
 - Repository analytic privileges 139, 185
 - Repository catalog objects 85, 94, 136
 - Repository object privileges 136, 182
 - Repository objects 117, 203, 279
 - deployment* 100
 - Repository role scripts 221
 - Repository roles 150–151, 194, 199, 203, 205, 257, 286, 308, 358, 367, 433
 - benefits* 203
 - case study* 234
 - consumer* 235
 - creation* 206, 234
 - dependencies* 438
 - developer* 236
 - granting* 231
 - granting and revoking* 211

- Repository roles (Cont.)
 - management* 205, 218
 - power user* 236
 - revoking* 232
 - security administrator* 238
 - sync* 432
 - Repository schema privileges 135, 180, 213
 - Repository schemas 109, 135, 156–157
 - creation* 95
 - creation syntax* 97
 - Repository tables 94, 112, 157
 - creation* 98
 - Repository workspace 62
 - navigating packages* 64
 - Repository-based model 201
 - Repository-based objects 178
 - Repository-based roles 233, 307, 361
 - Repository-based stored procedure 162, 339, 400
 - Repository-based table 338
 - creation* 346
 - Restricted user accounts 118
 - Revoke privileges 283
 - REVOKE statement 199
 - ROLE ADMIN privilege 203
 - Role admin system privilege 513
 - Role grantees 502
 - Role management 71
 - Role management role 264
 - Role name tag 207
 - Roles 52, 80, 149, 167, 199, 221, 370, 503
 - creation and management* 171
 - design-time version* 200
 - extensions* 208
 - granting* 153
 - SAP AHAN Studio* 152
 - SAP HANA Studio* 172
 - SAP HANA Web-Based Development Workbench* 154, 174
 - SQL* 150
 - SQL statements* 171
 - Root key backup (RKB) 528
 - Root keys 421, 423
 - generation* 421
 - Root package privileges 307, 514
 - Row-level security 78, 320
 - rules engines 28
 - Runtime 199
 - Runtime objects 214, 223
 - Runtime repository roles 210
 - Runtime schemas 212
- ## S
- SAML 375, 393, 395, 402
 - additional resources* 394
 - SAML identity providers 51
 - SAP assertion tickets 399
 - SAP Business Warehouse (SAP BW) 73
 - SAP BusinessObjects 393
 - SAP BusinessObjects BI 393
 - SAP BusinessObjects Lumira 33
 - SAP Change and Transport System (SAP CTS) 452–453
 - SAP HANA 27, 199, 335, 393, 404
 - audit policies* 510
 - bottom-up approach* 82
 - code development* 61
 - connecting* 43
 - development repository* 201, 297
 - hardware* 31
 - hardware layers* 29
 - in-memory database* 28
 - instance* 44, 63
 - instance number* 43
 - internal authentication* 376
 - multinode* 30
 - nodes* 30
 - overview* 27
 - package repository* 204, 362
 - privileges* 73
 - RDBMS catalog* 99
 - repository* 96
 - security model* 27
 - security overview* 35
 - security recommendations* 507
 - software appliance* 28
 - software layers* 28
 - transport system* 442
 - SAP HANA 1.0 318
 - SAP HANA 1.0 SPS 11 171, 319, 362
 - SAP HANA 1.0 SPS 12 32, 172
 - SAP HANA 2.0 318, 362, 421, 525
 - authorizations* 525
 - encryption* 527

- SAP HANA 2.0 (Cont.)
 - LDAP group membership* 527
 - log volume encryption* 527
 - PUBLIC role* 525
 - roles* 528
 - root key backup* 527
 - security* 525
- SAP HANA 2.0 Cockpit 34, 457, 463, 529
- SAP HANA 2.0 SPS 00 32, 215, 319, 376, 425
- SAP HANA application lifecycle
 - management* 243, 301, 427–428, 439, 443
 - additional options* 452
 - additional resources* 442
 - change recording* 452
 - pull method* 443
 - transport management window* 444
- SAP HANA Cockpit 362
- SAP HANA dynamic tiering 468
- SAP HANA modeler 169
- SAP HANA repository 301
- SAP HANA smart data access (SDA) 216, 274
- SAP HANA smart data integration 216
- SAP HANA Studio 39, 88, 118, 131, 190, 195, 205, 232, 276, 286, 304, 320, 326, 331, 346, 395, 448–449
 - backup folder* 48
 - catalog folder* 48
 - certificate trust store* 47
 - client encryption* 412
 - configuration tab* 72
 - content folder* 49
 - enable auditing* 457
 - enable tracing* 488
 - getting started* 41
 - GUI* 91
 - initial launch* 42
 - keystore* 413
 - management interface* 365
 - managing tabs* 55
 - minimum privileges* 436
 - navigation* 47
 - overview* 40
 - perspectives* 58
 - privileges* 141
 - provisioning folder* 50
 - role deletion* 173
 - role management interface* 353
 - security folder* 51
- SAP HANA Studio (Cont.)
 - security settings* 70
 - supported operating system* 40
 - switching perspectives* 68
 - user management area* 141
 - viewing trace files* 493
- SAP HANA Web-Based Development Workbench 88, 119, 124, 132, 146, 196, 218, 234, 289, 308, 368, 471
 - enable auditing* 461
 - navigation* 219
- SAP HANA Web-Based Development Workbench editor 200, 205, 220, 258, 356
- SAP HANA XSA 28
- SAP Landscape Transformation 280
- SAP logon tickets 397–398
- SAP NetWeaver 73, 510
- SAP NetWeaver Application Server 397
- SAP system ID 47
- SAP Web Dispatcher 396, 410, 412
- SAP Web IDE 34, 362
- Savepoints 420
- Schema nodes 92
- Schema object privileges 75
- Schema objects 86
- Schema prefix 172
- Schema privileges 212, 282
 - granting multiple* 281
 - revoking multiple* 282
- Schema-level access 177
- Schemas 48, 75, 269, 283
 - creation* 88
 - SQL* 280
- Script server 34
- Script-based repository roles 258, 287
- Secure Shell (SSH) 422
- Secure Sockets Layer (SSL) 46, 403
- Secure store in the file system (SSFS) 420
- Secured models 320
- Security console 71, 462
- Security lifecycle management 427
 - best practices* 428
 - content packages* 429
 - multiple environments* 428
 - role dependencies* 430
 - rollback plan* 430
 - test and validate* 429

Security management task auditing 483
 Security manager 125
 Security model 36, 167, 427
 test plans 432
 testing changes 430
 tracking changes 430
 validating 431
 Security model dependencies 100
 Security packages 66, 442
 Security tracing 487
 Sensitive data 37
 Sequences 86, 275
 Server-side data encryption 420
 Service accounts 510
 Service provider (SP) 393
 Simple and Protected GSSAPI Negotiation
 Mechanism (SPNEGO) 391
 Single sign-on (SSO) 375, 396, 509
 Software provisioning manager 90
 SQL 194
 SQL console 69, 71, 91, 133, 178, 231, 280
 SQL privileges 75
 SQL statements 133, 364
 SQL-based analytic privileges 77, 320,
 323, 332
 SQL-based dynamic analytic privileges 343,
 349
 defining 341
 mapping table 336
 SQLScript 335, 340, 372
 SSL access 438
 SSL certificates 403, 417
 SSL communications 417
 SSL encryption 413, 416
 SSL protocols 403
 SSL secured connection 418
 Standard calculation views 319
 Standard roles 150, 194
 Standard user accounts 116
 application accounts 116
 individual accounts 116
 service accounts 116
 Standard users 507
 Star join calculation views 319
 Static analytic privileges 320, 322, 326, 332
 Static filter condition 345
 Static SQL-based analytic privileges 324
 creation 331
 Static XML-based analytic privileges 330
 creation 326
 Stored procedure 82, 87, 93, 151, 161,
 165, 179, 231, 337, 344
 configuration 334
 Structured privileges ... 137–138, 183, 215, 354,
 499
 creation 244
 Subcatalog objects 49
 Subpackages 298, 438
 creation 298
 Synonyms 86
 SYS schema 126, 172
 _SYS_BI_CP_ALL 325
 _SYS_BIC schema 517
 _SYS_REPO ... 69, 85, 94, 117, 145, 178, 190, 200,
 202, 244, 277, 279, 288, 300, 512
 SYS_REPO 211, 233, 337, 354
 SYS_REPO schema 352
 Syslog 458
 SYSTEM account 43, 117, 520
 System account 73
 System auditing role 266
 System change auditing 482
 System privileges ... 74, 133, 142, 147, 179, 188,
 193, 211, 222, 241, 261, 408, 512
 case study 262
 default 242
 definition 241
 developer-related 242
 environment monitoring-related 252
 granting 253
 repository roles 257
 revoking 257
 SAP HANA Studio 254
 SAP HANA Web-Based Development
 Workbench 256
 security admin-related 243
 SQL 253
 system admin-related 246
 System users 522
 System utilization 456

T

Table types 335
 Tables 86, 282, 318
 Tailored Datacenter Integration (TDI) ... 32, 522
 TCP/IP connectivity port 34
 TCP/IP packets 404, 412
 TCP/IP port 445
 Technical user accounts 117
 Third-party authentication 389
 Trace
 deletion 491
 disable 492
 Trace file 418, 493
 Trace file viewer 494
 Trace level 492
 Transport Layer Security (TLS) 403, 511
 Transport route 446
 Triggers 86
 Troubleshooting 487
 case study 504

U

User accounts 115, 142
 authentication 400
 creation and management 119
 default settings 121
 deletion 129
 SAP HANA Studio 121
 SQL statements 120
 system views 126
 User admin system privilege 512
 User attributes 126, 129
 User credentials 375
 User management 70
 User management role 262
 User roles
 assignment 149
 User schemas 76
 User traces
 SQL configuration 491
 User-specific trace 487

V

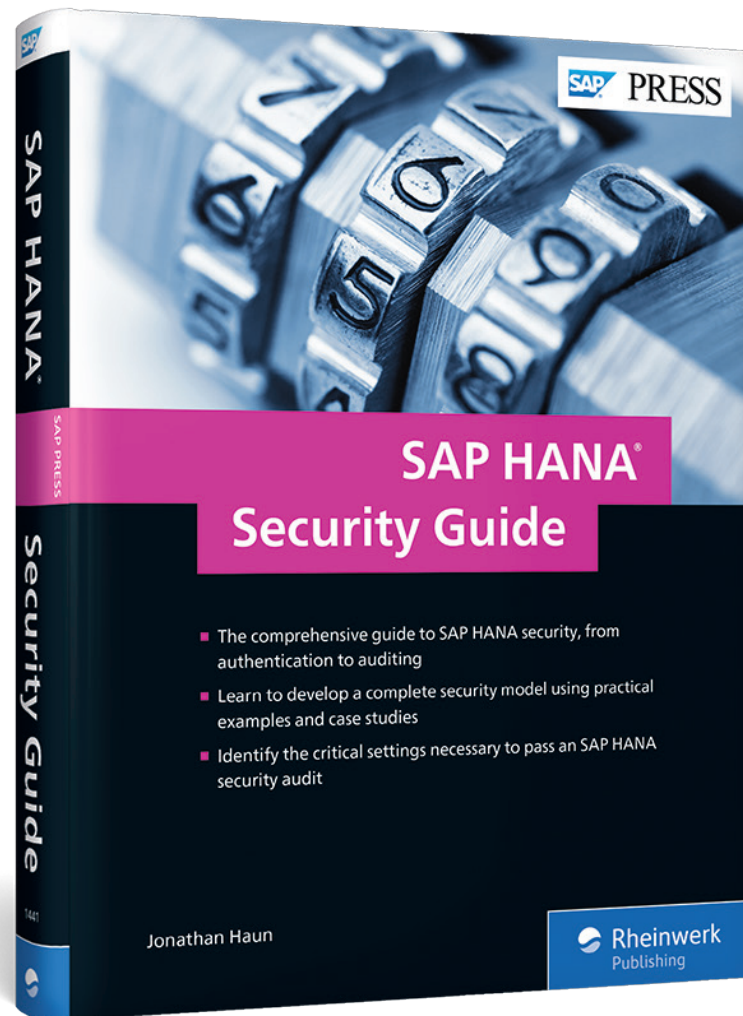
Validity period 329
 Views 86

W

Workspaces 41

X

X.509 375, 396, 403
 X509 certificate 120
 XML-based analytic privileges 77, 320, 517
 stored procedure 337
 XML-based dynamic analytic privileges 336
 mapping table 334
 stored procedure 335
 XS Artifact Administration 415
 XS engine 28, 33, 39, 119, 234, 377, 397, 409,
 414, 428
 web-based applications 415
 XSA engine 33



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 Proti-viti'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>.

We hope you have enjoyed this reading sample. You may recommend or pass it on to others, but only in its entirety, including all pages. This reading sample and all its parts are protected by copyright law. All usage and exploitation rights are reserved by the author and the publisher.