Document change record

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Author</th>
<th>Change Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>04.04.2018</td>
<td>1.0</td>
<td>Sabin Larrañaga</td>
<td>Document creation</td>
</tr>
<tr>
<td>06.11.2018</td>
<td>1.1</td>
<td>Sabin Larrañaga</td>
<td>Minor updates: latest recommendations, re-wording of some paragraphs, footnotes, and correction of typos.</td>
</tr>
</tbody>
</table>

References

General
- SAP HANA Developer Guide: Covers the how-to of building applications using SAP HANA: model data, write procedures and application logic in SAP HANA Extended application services, classic model
- SAP HANA XSA Developer Guide: Covers the development in SAP HANA Extended application services, advanced model
- SAP HANA Security Guide: Contains a separate chapter for SAP HANA Extended application services, advanced model
- SAP HANA Security Checklist: Contains a separate chapter for SAP HANA Extended application services, advanced model
- SAP Note 2465027 - Deprecation of SAP HANA extended application services, classic model and SAP HANA Repository

Migration
- SAP HANA XS Advanced Migration Guide: Focuses on migrating development objects and not on changes of the application lifecycle

Glossary

Following abbreviations will be used throughout the document:

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDI</td>
<td>SAP HANA Deployment Infrastructure</td>
</tr>
<tr>
<td>LCM</td>
<td>SAP HANA Lifecycle Management</td>
</tr>
<tr>
<td>XSA</td>
<td>SAP HANA Extended application services, advanced model</td>
</tr>
<tr>
<td>XSC</td>
<td>SAP HANA Extended application services, classic model</td>
</tr>
</tbody>
</table>
Table of Contents

REFERENCES ............................................................................................................................................ 2
GLOSSARY .................................................................................................................................................. 2

TABLE OF CONTENTS .................................................................................................................................. 3

1 INTRODUCTION ...................................................................................................................................... 5
2 PREPARATION ........................................................................................................................................ 5
2.1 PREREQUISITES .................................................................................................................................. 5
2.2 SAP HANA RELEASE ......................................................................................................................... 5
3 WHAT’S CHANGED FROM XSC TO XSA? ............................................................................................. 5
3.1 DEVELOPMENT SCENARIOS ................................................................................................................ 5
3.1.1 HANA and XSC .............................................................................................................................. 5
3.1.2 HANA and XSA .............................................................................................................................. 6
3.2 ORGS, SPACES, AND PROJECTS ......................................................................................................... 7
3.3 DATABASE ROLES ............................................................................................................................... 7
3.3.1 Design-time roles created in XSC .................................................................................................... 8
3.3.2 Design-time roles created in XSA .................................................................................................... 8
3.3.3 Comparing design-time roles between XSC and XSA ................................................................... 8
4 HANA DEPLOYMENT INFRASTRUCTURE ............................................................................................ 9
4.1 HDI CONTAINER MODEL .................................................................................................................... 10
4.2 HDI CONTAINER GROUPS ................................................................................................................... 10
4.3 HDI CONTAINER ISOLATION ................................................................................................................ 10
4.4 BREAKING THE HDI CONTAINER ISOLATION .................................................................................. 11
5 ROLE DEVELOPMENT IN XSA .............................................................................................................. 12
5.1 THE STEPS BEFORE DEVELOPMENT CAN START ............................................................................ 12
5.2 EQUIP THE HDI CONTAINER WITH EXTERNAL PRIVILEGES ........................................................... 12
5.2.1 Between HDI containers in the same space .................................................................................. 12
5.2.2 Between HDI containers in different spaces and external objects ............................................... 13
5.3 ROLE DEVELOPMENT ....................................................................................................................... 16
6 BEST PRACTICES AND RECOMMENDATIONS FOR ROLE DEVELOPMENT IN XSA ......................... 17
6.1 ORGANIZATION AND SPACES ......................................................................................................... 17
6.2 RECOMMENDED SETUP FOR DEV ENVIRONMENTS .................................................................... 17
6.3 RECOMMENDED SETUP FOR PROD ENVIRONMENTS ................................................................... 17
6.4 OTHER CONSIDERATIONS AND HINTS ........................................................................................... 18
7 MIGRATING ROLES FROM XSC TO XSA .......................................................................................... 18
7.1.1 Migration ........................................................................................................................................ 18
8 ROLE TEMPLATES ................................................................................................................................ 18
8.1 GUIDING PRINCIPLES IN DESIGNING THE ROLES ........................................................................ 19
8.2 PREREQUISITES ................................................................................................................................. 19
8.3 SETUP PROJECT FOR THE ROLES ..................................................................................................... 19
8.3.1 Create a new project ....................................................................................................................... 19
8.3.2 Create a database module for the project ....................................................................................... 20
8.3.3 Adjust the HDI namespace configuration ....................................................................................... 21

Best practices and recommendations for developing roles in SAP® HANA, March 2018 (PUBLIC) 3
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.3.4</td>
<td>Create a user provided service to equip HDI container</td>
<td>21</td>
</tr>
<tr>
<td>8.3.5</td>
<td>Bound the user provided service to the database module</td>
<td>22</td>
</tr>
<tr>
<td>8.3.6</td>
<td>Create the .hdbgrants file</td>
<td>23</td>
</tr>
<tr>
<td>8.3.7</td>
<td>Prepare the needed privileges in SAP HANA Database</td>
<td>25</td>
</tr>
<tr>
<td>8.4</td>
<td>CREATE DESIGN-TIME OBJECTS</td>
<td>31</td>
</tr>
<tr>
<td>8.4.1</td>
<td>Synonyms</td>
<td>31</td>
</tr>
<tr>
<td>8.4.2</td>
<td>Roles</td>
<td>33</td>
</tr>
<tr>
<td>8.4.3</td>
<td>Procedures</td>
<td>48</td>
</tr>
<tr>
<td>8.5</td>
<td>DEPLOY THE PROJECT TO SAP HANA</td>
<td>51</td>
</tr>
</tbody>
</table>
1 Introduction

The objective of this document is to describe the best practices and general recommendations on the role development process for SAP HANA database access roles with SAP HANA and SAP HANA Extended application services, advanced model (XSA).

SAP HANA extended application services, advanced model, (XS advanced) provides a comprehensive platform for the development and execution of native data-intensive applications that run efficiently in SAP HANA. It thus succeeds XS classic (XSC) as the default application programming model for SAP HANA. This implies that XSA defined and deployed roles will substitute the XSC defined and activated design-time roles.

The purpose of this document is not to replace the SAP HANA XSA Developer Guide, but to provide additional information and recommendations on how to address the common challenges on the role development process with XSA.

Additionally, in this document, we deliver and describe HDI-based role templates for administrative tasks in SAP HANA database. This HDI-based role templates can be used by customers as a starting point for the creation of their own roles.

2 Preparation

2.1 Prerequisites

This document aims at administrators and role developers that

- are experienced with SAP HANA and
- are familiar with SAP HANA privileges and roles and
- are familiar with XSC and
- have basic knowledge about XSA (e.g. have attended the latest openSAP course on SAP HANA development).

2.2 SAP HANA release

SAP HANA XSA is shipped as an optional component for the following releases of SAP HANA, so this document only applies in case we have one of the following releases installed or are planning to move or upgrade to one of the following SAP HANA releases:

- SAP HANA 1.0 SPS12
- SAP HANA 2.0 SPS00 or higher

3 What’s changed from XSC to XSA?

A couple of things have changed when moving from SAP HANA Extended application services, classic model (XSC) to SAP HANA Extended application services, advanced model (XSA). In this section, we describe the most important changes related to role development. Please also check out the developer guide and the XSA guides mentioned in the references section.

3.1 Development Scenarios

3.1.1 HANA and XSC

Previously, with HANA and XSC, the development of HANA database artifacts was done through SAP HANA Studio or with the SAP HANA Web-based Development Workbench.

In this scenario, a developer using one of those tools creates a design-time version of an object, such as a view, a procedure or a role, and this design-time version is saved within the SAP HANA Repository. By activating the design-time object, a runtime version of the object (also known as catalog object) is created by the _SYS_REPO technical user.

In this scenario, there is only one repository known as SAP HANA Repository and it is owned by the _SYS_REPO user. All objects generated from the SAP HANA Repository are also owned by the _SYS_REPO user.
What’s changed from XSC to XSA?

3.1.2 HANA and XSA

With the arrival of XSA, it has changed the way we develop HANA database objects compared to XSC. Developers will use the SAP Web IDE in XSA to create the design-time version of a HANA database object, such as a view, procedure or role. The design-time objects are created within a project that it is stored in a GIT repository.

When the developer deploys the project, a runtime version of the objects is created in a HANA Deployment Infrastructure (HDI) Container within the HANA database.

An HDI container can be seen as a database schema and we can have multiple HDI Containers within the SAP HANA database.

All database objects deployed within the container are owned by container-specific technical object owner.
3.2 Orgs, Spaces, and Projects

In XSC all the development objects were stored in the SAP HANA Repository which was composed by a hierarchy of packages. There were no granular admin privileges and we could use “package privileges” to allow developers to create, edit or activate objects within a specific package and its sub-packages.

With XSA, a new controller model has been introduced which is composed of a hierarchy of organizations and spaces. This new model allows us to separate administration privileges per organization and space, and to isolate resources and privileges between applications that are in different spaces.

The controller model of XS Advanced introduces the following terms:

- **Organization**: can be used e.g. to logically separate customers or development areas. One initial organization is always created during the installation. Organizations can be used to
  - Separate line of business development
  - Separate security-related development from other development
  - Separate sets of privileges for role-building

- **Space**: an organization can contain one or more spaces. Spaces can be used to physically separate applications. Services are always available for all applications in the same space. The first space created during installation is named SAP and contains all services needed to develop and deploy applications with XSA. Spaces can be used to
  - Separate security-related development from other development
  - Separate sets of privileges for role-building

- **Application**: in a space, one or more applications can be deployed. They share the services of their space.

- **Projects**: can be used to separate e.g. sets of roles for different target systems. A developer can create multiple projects within a space.

3.3 Database Roles

In SAP HANA we have two ways of creating database roles:

- Create roles directly at run-time. These roles are then known as catalog roles.
- Create roles using a role definition. These roles are then known as design-time roles.

To understand the difference between this two type of roles, we can check the [SAP HANA Security Guide](http://example.com). SAP recommends working with design-time roles and thanks to XSA and the new HDI concept, there is a significant difference in the way they are created and managed compared with the design-time roles in XSC.
3.3.1 Design-time roles created in XSC

In XSC, the runtime versions of the design-time roles are global in the database. As there is a single repository in the database, all the design-time roles are owned by the _SYS_REPO user and can only be equipped with privileges that are granted to the _SYS_REPO user with grant/admin option. This means that the _SYS_REPO user holds almost all privileges on the system.

Granting or revoking a design-time role is only done via database procedures owned by the _SYS_REPO user. E.g. CALL_GRANT_ACTIVATED_ROLE. These roles cannot be granted by a user with ROLE ADMIN privilege.

Additionally, application development and role development are mixed up. As there is only one repository that it’s owned by the _SYS_REPO user and developers can create different type of objects, application developers can take advantage of this setup to create roles with wider privileges for their applications.

3.3.2 Design-time roles created in XSA

On the other hand, in XSA, runtime versions of the design-time roles are created at schema-level due to containerization. These roles are owned by the container-specific technical user and can be compound by privileges granted with admin/grantable option to the container-specific technical owner.

Thanks to this containerization, it is possible to separate role development from the application development as this can be organized within different spaces in XSA and thereby possible to only assign the relevant privileges to the respective container-specific technical users.

To grant or revoke a role created with XSA there are two possibilities:
- Using a database procedure which is provided as part of HDI container API by default.
- Using the system privileges ROLE ADMIN that allows to grant/revoke roles system-wide.

3.3.3 Comparing design-time roles between XSC and XSA

As mentioned before, there are many differences between roles developed on XSC (repository roles) and the roles developed on XSA (HDI-based roles). The principal differences are:

<table>
<thead>
<tr>
<th>XS Classic/Repository-Roles</th>
<th>XS Advanced/HDI Roles</th>
</tr>
</thead>
<tbody>
<tr>
<td>All roles are global roles in the database</td>
<td>Roles are now created on schema-level due to HDI containerization</td>
</tr>
<tr>
<td>All roles are owned by _SYS_REPO</td>
<td>Roles are owned by a schema-level technical user</td>
</tr>
<tr>
<td>All roles are build with privileges of _SYS_REPO</td>
<td>The schema-level technical users can be equipped with different sets of privileges</td>
</tr>
<tr>
<td>Application development and role development are mixed up</td>
<td>Application development and role development can be separated</td>
</tr>
<tr>
<td>All roles are granted via a procedure owned by _SYS_REPO.</td>
<td>Role granting can be separated per container (via API) or be system-wide using the ROLE ADMIN system privilege</td>
</tr>
<tr>
<td>Role History cannot be completely tracked</td>
<td>Complete role history is available in GIT</td>
</tr>
</tbody>
</table>

In the following diagram, we can see the differences between roles (design-time & run-time roles) in XSC and in XSA.
There are also differences in the way a role can be granted depending if they are global roles (XSC) or schema-local roles (XSA). Global roles cannot be granted or revoked with ROLE ADMIN system privilege, but through the SYS_REPO API, thus the execute privilege on the procedures is required.

Conversely, schema-local roles can be granted or revoked with ROLE ADMIN system privilege and/or through the container-local API. This characteristic allows us to grant all the roles in the system by having ROLE ADMIN system privilege or only grant roles that are contained in a specific container where we have the execute privilege on container-local API.

<table>
<thead>
<tr>
<th>PRIVILEGE</th>
<th>XSC (Global Role)</th>
<th>XSA (Schema-local Role)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Owner: _SYS_REPO</td>
<td>No</td>
<td>Yes – for all roles</td>
</tr>
<tr>
<td>Execute on _SYS_REPO API</td>
<td>Yes – for all roles</td>
<td>Yes – for roles in schema</td>
</tr>
<tr>
<td>Execute on container-local API</td>
<td></td>
<td>CALL ContainerGRANT_SCHEMA.Roles</td>
</tr>
</tbody>
</table>

4 HANA Deployment Infrastructure

With XSA it was also introduced the HANA Deployment Infrastructure (HDI) which is a service layer of the SAP HANA database that simplifies the deployment of HANA database artifacts.
4.1 HDI container model

As discussed previously, all SAP HANA database artifacts, such as roles or tables, that are created through XS advanced are deployed in the SAP HANA Database within an HDI container.

An HDI container is essentially a database schema. Each container has its own container object owner (also known as #OO user), some APIs and a schema for storage.

4.2 HDI Container Groups

HDI containers can be grouped into HDI container groups for administrative purposes. An HDI container administrator can create container groups, assign containers to a container group, and manage container group administrative privileges.

A container group role is created per container group. This role is granted by default to all the container object owners that belong to the same container group. Container group roles were released with SAP HANA 2.0 SPS03.

4.3 HDI Container Isolation

The HDI containers provide security isolation as it is based on a “zero” privileges principle by default. All objects within a container are owned by the #OO user. This user has the following characteristics:
The object owner is a restricted database user.

The object owner has CREATE ANY privileges on the schema.

However, the object owner has no external privileges by default.

By default, references to external objects are not allowed. For example, view X has no direct reference to table ERP.Y:

![Diagram showing HDI container isolation]

The HDI container isolation also affects the creation of database roles through XSA. When we create a database role, using XSA, the database role will be deployed within an HDI container and it will be owned by the database object owner. Thus, to be able to deploy the roles, the role can only contain privileges that are assigned to the #OO user with GRANT / ADMIN option. Otherwise, the deployment will fail due to a “Missing Authorization Error”.

### 4.4 Breaking the HDI container isolation

To break the HDI container isolation, we need to grant the privilege (with GRANT/ADMIN option) of the external object to the #OO user.

In the previous example, to create a view X as table ERP.Y, the following actions need to be performed in advance:

- Grant SELECT privilege on ERP.Y to the #OO user with GRANT/ADMIN option.
- Create a synonym Y that points to table ERP.Y

![Diagram showing breaking HDI container isolation]

In case of database roles, the approach is similar. To deploy successfully a database role, all the privileges that are included in the role should be granted to the #OO user with GRANT / ADMIN option in advance.
There are different alternative options how to grant the database-level privileges to the #OO user. These options are described in the following Equip HDI container with external privileges section.

5 Role development in XSA

For the development of roles, no real application is created, only database artifacts – the roles – are used. For this, developers will use SAP Web IDE to work on a project within a space in XSA where they will create the design-time objects. In the space, only two types of services are needed to create roles:

- HDI Container
- User Provided Service (Optional)

In this section, we are going to describe the steps needed to develop transportable HANA database roles using XS Advance.

5.1 The Steps before development can start

The initial step is to setup the and prepare XSA + SAP Web IDE application. Once the setup is ready, we need to setup our organization and spaces in a way that suits the needs of our organization.

How to create spaces, setup users and call the SAP Web IDE can be found in chapter Post-Installation Administration Tasks in the SAP Web IDE for SAP HANA Installation and Upgrade Guide.

Our recommendation is to setup at least a dedicated space for role developing. You can find more details in the Best practices and recommendations section in this document.

5.2 Equip the HDI container with external privileges

In the following section are described the different alternatives to grant database-level privileges to the container object owners. These privileges can be:

- System privileges
- Object privileges
- Analytic privileges

For illustration purposes, let's assume that we would like to deploy our database roles within the HDI container named “C1” and we need to grant in advance all the external privileges to the container object owner user (C1#OO user) with GRANT /ADMIN option.

5.2.1 Between HDI containers in the same space
When we have two or more containers living in the same space, every HDI container is shared (accessible) to other HDI containers living in the same space. This is possible as it exists a so-called trusted setup between all the resources living in the same space.

In this case, there is a shortcut mechanism where the container C1 can itself get granted privileges for the other container within the same space, e.g. C2. This is possible thanks to the trusted relationship between services and applications living in the same space.

5.2.2 Between HDI containers in different spaces and external objects

In the case of an external schema (X schema) or an HDI container living in a different space, it is more complex since the owner of the object needs to explicitly grant the privilege. The same applies to system privileges or another type of objects such as tables or views.

The following options are available to grant the privileges of an external object to the C1#OO user

5.2.2.1 GRANT to #OO user directly

This option consists of granting the external privilege or role directly to the #OO user. After this, we will be able to deploy the database role within the HDI container.

This approach is similar to the one used with XSC when we needed to grant to the _SYS_REPO user the privilege with GRANT / ADMIN option to include it in a design-time role.

When using this option, the following challenges exist:

- The #OO is automatically created during the “Build” process when the HDI container is also created. This is a problem when transporting the roles as the first time when we build and deploy the project, as #OO will be created with minimum privileges and we will get a “Missing authorizations” error.
- In a development scenario, we can potentially have more than one container for role building (one per developer) so we will need to manually grant the privilege or role to each #OO user.

5.2.2.2 GRANT to global role _SYS_DI_OO_DEFAULTS

This option consists of granting the external privilege to the _SYS_DI_OO_DEFAULTS role instead of granting it directly to the #OO user.

The _SYS_DI_OO_DEFAULTS role contains the set of default privileges that are granted to all HDI container object owner users.

By using this option, we can prepare in advance the required external privileges for our database roles by granting them in advance to the _SYS_DI_OO_DEFAULTS role. This will solve the issue when we are deploying the project for the first time or when transporting the roles, as the role will be granted to the #OO when it is created.
and the deployment will succeed. Also, there won’t be any additional work in scenarios with multiples containers as the role will be assigned to all the #OO users.

On the negative side, as the _SYS_DI_OO_DEFAULTS role is assigned to all #OO users, this will break, at HANA level, the separation created by different spaces at XSA level.

This option is not recommended. Even though it solves the problems to transport or manage multiple containers mentioned in the previous option, this role will be granted to all #OO users and then we lose the separation between spaces.

5.2.2.3 Using a user provided service with a granting user

In the following option, we will explain the possibility to equip the #OO user with the required external privileges by using a granting user at database level and a user provided service in our XSA space.

The granting user, for example user A, is a standard database user that have all the required privileges for role developing with GRANT / ADMIN option. This user account needs to be activated and it will be exposed to the XSA space.

We should not use the user SYSTEM as the granting user

Additionally, in the XSA space, we need to define a user provided service with the granting user credentials. This user provided service will be used by the HDI deployer (application used to “prepare” the HDI container) to grant the required privileges to the #OO user.

To define which external privileges are going to be granted to the #OO user, we need to maintain in the XSA project the .hdbgrants file specifying the user provided service as granting service.

On startup, the HDI deployer will process the .hdbgrants file, connects to the database using the user provided service’s credentials (user A credentials), and grants the specified privileges to the #OO user.
We need to make sure that the privileges are owned by the granting user with GRANT / ADMIN option, otherwise, we will get an authorization error.

The downside of this option is that the credentials of the user A need to be shared for the definition of the user provided service, and, in cases like role developing, this user will need to have wide privileges in the system which represents a security issue.

5.2.2.4 Using a user provided service with a procedure grantor

This option is like Using a user provided service with a granting user, but, it implements additional features to mitigate the security risk discussed in that section.

At HANA database level, we will need to setup two different user accounts:

- **User A:** This is a standard user that have all the required privileges with GRANT / ADMIN option. This user owns a database procedure running in definer mode that will be used to grant the privileges to the #OO user. This user account can be deactivated and there is no need to share its credentials.

- **User B:** This is a standard user account with privileges to EXECUTE the database procedure owned by the user A. This user needs to be activated and the credentials need to be shared as it will be exposed to the XSA space.

If User A will be used for building roles only containing privileges already granted to SYSTEM by default (e.g. SYSTEM PRIVILEGES), the SYSTEM user could be used as User A. If User A will be used for building roles containing privileges of e.g. developed objects, a dedicated user should be created and used as User A instead of SYSTEM user.

Additionally, in the XSA space, we need to define a user provided service with the user B credentials. This user provided service will be used by the HDI deployer (application used to “prepare” the HDI container) to provide automatically the required privileges to the #OO user.

To define which external privileges are going to be granted to the #OO user, we need to maintain in the XSA project the .hdbgrants file, specifying the user provided service as granting service.

On startup, the HDI deployer will process the .hdbgrants file, connect to the database using the user provided service’s credentials (user B credentials), and execute the procedure that grants the privileges to the #OO user.
We should not use the user SYSTEM as the user A, except for role-building with privileges granted to the SYSTEM user by default. E.g. SYSTEM PRIVILEGES

In comparison with option *Using a user provided service with a granting user*, in this option, the user account that has all the required privileges (user A) can be deactivated and the credentials are not shared. Instead, the credentials of the user B, with very limited privileges, are shared to define the user provided service.

Additionally, further security validations can be implemented within the procedure’s logic to avoid granting the privileges to a different user than container object owners. For example, validating that the grantee user is deactivated and the username ends with “#OO” characters.

The procedure grantor mechanism is supported as of version 3.4.1 of the @sap/hdi-deploy component in XSA.

5.3 Role development process

Role development is done by the developers in the Web IDE. There the developer Alice will create a new project in the space created for the roles.

Alice creates an HDB module which automatically deploys to an HDI container on the HANA DB.

The services needed for role developing needs to be made known to the project. This is done in the `mta.yaml` file. After this, Alice can now:

- Create some file for granting privileges via the user provided service - `.hdbgrants`
- Create some synonyms for object privileges - `.hdbsynonyms`
- Create role definitions - `.hdbrole`
- And perhaps some role configuration files - `.hdbroleconfig`

For further details on how to create the different development objects needed for role developing, we can take a look at the [SAP HANA Developer Guide for SAP HANA XS Advanced Model](https://help.sap.com/doc/8a90581a0ba1b47c017000000b56e89a/1.0.0/en-US/index.html)
Best practices and Recommendations for role development in XSA

6.1 Organization and spaces
For role development, there is no unique setup of organizations and spaces that are ideal for this purpose. The possibility to organize privileges and roles into organization and spaces will give us the flexibility to cover all the different scenarios we could find.

We should take into consideration that, while more spaces and organizations in the XSA environment, the complexity also increases. So, let’s keep it simple.

As a principle, we should clearly differentiate security-related development, like role development, from application development. Application developers will need to create roles to access their developed artifacts, but those roles should only include privileges related those artifacts and not privileges for database administration for example. We can avoid this by having a dedicated organization or a dedicated space within our organization for security-related development.

Another recommendation is to separate into different organizations or spaces the roles for database administration from the roles for a specific functional area, for example, roles for HR applications.

6.2 Recommended setup for DEV environments
In development environments, it is recommended to use the option Using a user provided service with a procedure grantor as a mechanism to grant the external privileges to the #OO user of the container where the database roles are being deployed.

With this mechanism, no activated user need to have all the granted privileges. Only the call of the procedure is exposed and the procedure can contain various additional security checks.

To avoid delays in the development process due to missing authorizations in the container, depending on your scenario the following recommendations apply:

- When developing roles only containing privileges already granted to SYSTEM by default (e.g. SYSTEM PRIVILEGES), the SYSTEM user should be used as the user holding all the privileges (User A as explained in section Using a user provided service with a procedure grantor).

  In this case, the space where the User Provided Service is exposed should be used exclusively for role development and not for application development.

- When developing roles containing privileges of e.g. developed objects, it is recommended to grant wide privileges to the database user holding all the privileges.

Afterwards, developers can assign needed privileges to the HDI container through the grant file definition (.hdbgrants).

6.3 Recommended setup for PROD environments
Also in a productive environment, it is recommended to use the option Using a user provided service with a procedure grantor as the mechanism to grant the privileges to the #OO user.

With this mechanism, no activated user need to have all the granted privileges. Only the call of the procedure is exposed and the procedure can contain various additional security checks.

For a production environment, depending on your scenario the following recommendations apply:

- When developing roles only containing privileges already granted to SYSTEM by default (e.g. SYSTEM PRIVILEGES), the SYSTEM user should be used as the user holding all the privileges (User A as explained in section Using a user provided service with a procedure grantor).

  In this case, the space where the User Provided Service is exposed should be used exclusively for role development and not for application development.

- When developing roles containing privileges of e.g. developed objects, it is recommended to grant to the database user (user A in section Using a user provided service with a procedure grantor) only the privileges required to deploy the project. The list of required privilege can be obtained from the grant file definition (.hdbgrants) in the DEV environment.
6.4 Other considerations and hints

- Use SYSTEM user to initially set up the user providing the privileges.
- Be aware, that a role developer for HANA DB roles will have critical privileges if we allow him to test the roles in the database.
- Segregate different sets of privileges in different spaces or even different organizations.
- Separate sets of roles meant for different target systems in different projects. E.g. test + prod system.
- Do not mix XSC and XSA role development. E.g. do not include XSA built roles in XSC defined roles.
- Everything discussed in this chapter holds for all databases in your system, i.e. all Tenants and the SystemDB. Technically this is solved by mapping different spaces with different databases.

7 Migrating roles from XSC to XSA

7.1.1 Migration

For the migration, we assume that you currently develop your roles in XS classic and you have an existing role concept for HANA for development and administration tasks. Now you want to make the switch from XS classic to XS advanced and therefore you also have to migrate your existing roles both for development and for administration.

In this scenario, there is no need to migrate your old developer roles as they are specific to XSC. The new developer roles will be created and managed with the XSA admin tools. The rest of the roles used to access HANA (e.g. Admin) can be migrated.

To migrate the roles the SAP HANA XS Advance Migration Assistance is available to convert the XSC roles into XSA roles. The migration assistant tool use as input a delivery unit (DU) in the XSC system where the roles are included and it creates an XSA project containing the roles and all the necessary files.

You can find more information about the migration assistant and how to run it in the SAP HANA XS Advanced Migration Guide.

For the migration, we recommend to setup a dedicated XSA space for roles to be migrated. The idea is to do the initial migration of the roles in a single HDI container and, after the migration, we could reorganize and segregate the roles and privileges into different HDI containers, spaces or organizations to fit our requirements.

The HDI container to be used for the deployment of the migrated roles should be equipped with similar privileges as the _SYS_REPO user to be able to deploy the roles.

Based on the previous recommendations, a possible migration process could be the following:

1. Use the migration assistant tool to create the XSA roles project.
2. Import the XSA roles project in SAP Web IDE.
3. Equip the HDI container with similar privileges as the _SYS_REPO user.
4. Review/fine-tune the roles and deploy on the _SYS_REPO like HDI container.

8 Role templates

In the following section, we provide different role templates and we describe the steps needed to implement them using XS advanced.

The roles described in this section are considered templates. That is, that customers can use them as a base to create their own version of the roles to cover their needs. The proposed role templates are for roles in the areas of:

- Database administration
- Security administration
- User administration
- Support task
8.1 Guiding principles in designing the roles

When designing the roles described in this document, the following guiding principles were followed:

- Strong security requirements.
  - Granular structure.
  - User management is strictly separated from role assignment.
  - Strong control over granting of roles to users (e.g. only allow granting of end-user roles that have been designed by the security team and deployed into HANA using the same HDI container).
  - Case for granting roles to roles at HANA level is not permitted. This should be done via HDI.
- We only work actively with HDI roles.
  - Only HDI roles are newly created.
  - Only HDI roles may be granted to users.
  - The “ROLE ADMIN” privilege is not granted to any role or user.

8.2 Prerequisites

To implement the role templates, the following prerequisites are needed:

- In XSA:
  1. Setup and prepare XSA + SAP Web IDE application in our development system.
  2. Setup an XSA space for the role building scenario. This space is where the project is going to be run.
  3. Map the XSA space to database where the roles will be deployed.
  4. Grant the following privileges to the developer user account in XSA:
     - XSA space developer rights
     - SAP Web IDE development permissions

How to create spaces, setup user and call the SAP Web IDE can be found in chapter Post-Installation Administration Tasks in the SAP Web IDE for SAP HANA Installation and Upgrade Guide

- In SAP HANA Database:
  1. Credentials of the SYSTEM user.

It is recommended to do the initial setup at HANA database level with SYSTEM user as this user already holds all the required privileges with GRANT / ADMIN option. Once all bootstrapping is properly done it is recommended to deactivate the SYSTEM user.

8.3 Setup project for the roles

In the following steps, we are going to describe the steps to setup a multi-target application project to implement the role templates.

8.3.1 Create a new project

Using SAP Web IDE, we create a new multi-target application project named “database_roles”.

New Multi-Target Application Project
Basic Information

Project Name: database_roles
Provide a description and select a space where we want to run our application project.

Click on “Finish”.

**8.3.2 Create a database module for the project**

Create a SAP HANA Database Module for the “database_role” project.

Set the module name to “db”.

In the step of the wizard, we set the schema name to “DATABASE_ROLES” and select our SAP HANA database version.

Click on Finish.
8.3.3 Adjust the HDI namespace configuration

In the role templates, we are not using the namespace for the name of the objects in runtime. Thus, we need to modify the HDI namespace configuration that is created by default when creating a SAP HANA Database module. For this, we need to find the .hdinamespace file in and delete it from our project.

The .hdinamespace file is hidden by default. To find it, we need to select the option “Show hidden files” in the View menu.

8.3.4 Create a user provided service to equip HDI container

This step is only required if we plan to equip our HDI container with the required external privileges using the following methods:

Using a user provided service with a granting user
Using a user provided service with a procedure grantor

If we are implementing a different method, we skip this step.

We will now create a user-provided service called PRIVILEGE_SERVICE. This service will be used during the deployment of the project to grant all the required privileges to the #OO user. The list of privileges granted to the #OO needs to be defined in a .hdbgrants file.

Execute the following command depending on the selected method.

8.3.4.1 Using a user provided service with a granting user

Open the XS client and execute a command in the XSA space where we are running the project.

8.3.4.2 Using a user provided service with a procedure grantor

The procedure grantor mechanism is supported as of version 3.4.1 of the @sap/hdi-deploy component in XSA.

Open the XS client and execute a command in the XSA space where we are running the project.
8.3.5 Bound the user provided service to the database module.

This step is only required if we plan to equip our HDI container with the required external privileges using the following methods:

- Using a user provided service with a granting user
- Using a user provided service with a procedure grantor

If we are implementing a different method, we skip this step.

We need to bound the user provided service named PRIVILEGE_SERVICE to the database module by modifying the MTA development descriptor file (mta.yaml) of our project.

Open the mta.yaml file with the “Code Editor” and replace its content with the following code.

```
ID: database_roles
_schema-version: '2.0'
version: 0.0.1

modules:
  - name: database-roles-db
type: hdb
path: db
requires:
  - name: database-roles-db-hdi-container
properties:
  TARGET_CONTAINER: ~{service-name}
  - name: database-roles-db-privileges
resources:
  - name: database-roles-db-hdi-container
type: com.sap.xs.hdi-container
properties:
  service-name: ${service-name}
parameters:
  config:
    schema: DATABASE_ROLES
  - name: database-roles-db-privileges
type: org.cloudfoundry.existing-service
parameters:
  service-name: PRIVILEGE_SERVICE
```
Now the `mta.yaml` file contains one module named “database-role-db” of type “hdb” which reflects an HDI container. The hdb module is bound to two additional resources from the project:

- “database-roles-db-hdi-container” that is for the HDI container that is created when we deploy the project. It has a configuration to set the schema name of the HDI container to “DATABASE_ROLES”.
- “database-roles-db-privileges” that is for a user provided service named “PRIVILEGE_SERVICE” on the XSA space.

### 8.3.6 Create the `.hdbgrants` file

This step is only required if we plan to equip our HDI container with the required external privileges using the following methods:

- Using a user provided service with a granting user
- Using a user provided service with a procedure grantor

If we are implementing a different method, we skip this step.

To assign privileges automatically to the object owner and (or) the application binding users, the HDI Deployer provides `.hdbgrants` files, which use a syntax that is similar to the `.hdbrole` artifact.

As a developer, we can use the `.hdbgrants` file to automatically grant privilege to our HDI container before the content is deployed.

Now we need to create the PRIVILEGE_SERVICE.hdbgrants file. The recommended path for this file is `database_roles -> db ->cfg -> grants`

Open the PRIVILEGE_SERVICE.hdbgrants file with the “Code Editor” and copy the following code.

```json
{
    "PRIVILEGE_SERVICE": {
        "object_owner": {
            "schema_privileges": [
                {
                    "schema": "_SYS_STATISTICS",
                    "privileges_with_grant_option": ["SELECT"]
                }
            ],
            "roles": [],
            "object_privileges": [
                {
                    "schema": "_SYS_SECURITY",
                    "privileges_with_grant_option": ["SELECT"]
                }
            ]
        }
    }
}
```
Now the PRIVILEGE_SERVICE.hdbgrants file specifies that the user provided service named "PRIVILEGE_SERVICE" should be used to grant to the #OO user the specified privileges.
8.3.7 Prepare the needed privileges in SAP HANA Database

As described previously in the guide, to deploy the HDI roles, we need to grant the external privileges with GRANT / ADMIN option to the #OO user of the HDI container. To do this, there are different alternatives described in the section Equip the HDI Container with external privileges. The project with the role templates can be deployed using any of the described methods.

In this section, we are describing the different steps needed for each of the methods.

The external privileges included in the role templates are the following:

<table>
<thead>
<tr>
<th>Privilege</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT on schema _SYS_STATISTICS</td>
<td>Object privilege</td>
</tr>
<tr>
<td>SELECT, INSERT, UPDATE and DELETE on table _SYS_SECURITY._SYS_PASSWORD_BLACKLIST</td>
<td>Object privilege</td>
</tr>
<tr>
<td>CATALOG READ</td>
<td>System privilege</td>
</tr>
<tr>
<td>SERVICE ADMIN</td>
<td>System privilege</td>
</tr>
<tr>
<td>INIFILE ADMIN</td>
<td>System privilege</td>
</tr>
<tr>
<td>TRACE ADMIN</td>
<td>System privilege</td>
</tr>
<tr>
<td>SESSION ADMIN</td>
<td>System privilege</td>
</tr>
<tr>
<td>VERSION ADMIN</td>
<td>System privilege</td>
</tr>
<tr>
<td>LICENSE ADMIN</td>
<td>System privilege</td>
</tr>
<tr>
<td>SAVEPOINT ADMIN</td>
<td>System privilege</td>
</tr>
<tr>
<td>RESOURCE ADMIN</td>
<td>System privilege</td>
</tr>
<tr>
<td>BACKUP OPERATOR</td>
<td>System privilege</td>
</tr>
<tr>
<td>BACKUP ADMIN</td>
<td>System privilege</td>
</tr>
<tr>
<td>CREATE SCHEMA</td>
<td>System privilege</td>
</tr>
<tr>
<td>EXPORT</td>
<td>System privilege</td>
</tr>
<tr>
<td>IMPORT</td>
<td>System privilege</td>
</tr>
<tr>
<td>AUDIT ADMIN</td>
<td>System privilege</td>
</tr>
<tr>
<td>AUDIT OPERATOR</td>
<td>System privilege</td>
</tr>
<tr>
<td>USER ADMIN</td>
<td>System privilege</td>
</tr>
<tr>
<td>ENCRYPTION ROOT KEY ADMIN</td>
<td>System privilege</td>
</tr>
<tr>
<td>SSL ADMIN</td>
<td>System privilege</td>
</tr>
<tr>
<td>TRUST ADMIN</td>
<td>System privilege</td>
</tr>
<tr>
<td>CERTIFICATE ADMIN</td>
<td>System privilege</td>
</tr>
<tr>
<td>LOG ADMIN</td>
<td>System privilege</td>
</tr>
</tbody>
</table>

8.3.7.1.1 GRANT privileges to #OO user directly

The description of this method can be found in section GRANT privilege to #OO user directly.

The #OO user is created for the first time when the project is deployed and the HDI container is created in the database. Thus, we need to deploy the project at least one time to create the #OO user. To deploy the project, we can check the Deploy the roles to HANA section.
With this option, if the project we are deploying for the first time already contains design-time roles, the deployment will fail throwing a missing authorization error.

As the schema name was configured with the name DATABASE_ROLES, when we deploy the project for the first time, the HDI container should be named DATABASE_ROLES_1 and the object owner user (#OO) should be DATABASE_ROLES_1#OO

Once the #OO exists in the database, we can grant to this user all the required privileges. For this, we will need to execute the following script as SYSTEM user.

```
GRANT SELECT ON SCHEMA _SYS_STATISTICS TO DATABASE_ROLES_1#OO WITH GRANT OPTION;
GRANT SELECT, INSERT, UPDATE, DELETE ON _SYS_SECURITY._SYS_PASSWORD_BLACKLIST TO DATABASE_ROLES_1#OO WITH GRANT OPTION;
GRANT CATALOG READ TO DATABASE_ROLES_1#OO WITH ADMIN OPTION;
GRANT SERVICE ADMIN TO DATABASE_ROLES_1#OO WITH ADMIN OPTION;
GRANT INIFILE ADMIN TO DATABASE_ROLES_1#OO WITH ADMIN OPTION;
GRANT TRACE ADMIN TO DATABASE_ROLES_1#OO WITH ADMIN OPTION;
GRANT SESSION ADMIN TO DATABASE_ROLES_1#OO WITH ADMIN OPTION;
GRANT VERSION ADMIN TO DATABASE_ROLES_1#OO WITH ADMIN OPTION;
GRANT LICENSE ADMIN TO DATABASE_ROLES_1#OO WITH ADMIN OPTION;
GRANT Savepoint ADMIN TO DATABASE_ROLES_1#OO WITH ADMIN OPTION;
GRANT Resource ADMIN TO DATABASE_ROLES_1#OO WITH ADMIN OPTION;
GRANT Backup Operator TO DATABASE_ROLES_1#OO WITH ADMIN OPTION;
GRANT Backup ADMIN TO DATABASE_ROLES_1#OO WITH ADMIN OPTION;
GRANT Create Schema TO DATABASE_ROLES_1#OO WITH ADMIN OPTION;
GRANT Export TO DATABASE_ROLES_1#OO WITH ADMIN OPTION;
GRANT Import TO DATABASE_ROLES_1#OO WITH ADMIN OPTION;
GRANT Audit ADMIN TO DATABASE_ROLES_1#OO WITH ADMIN OPTION;
GRANT Audit OPERATOR TO DATABASE_ROLES_1#OO WITH ADMIN OPTION;
GRANT User ADMIN TO DATABASE_ROLES_1#OO WITH ADMIN OPTION;
GRANT Encryption Root Key ADMIN TO DATABASE_ROLES_1#OO WITH ADMIN OPTION;
GRANT SSL ADMIN TO DATABASE_ROLES_1#OO WITH ADMIN OPTION;
GRANT Trust ADMIN TO DATABASE_ROLES_1#OO WITH ADMIN OPTION;
GRANT Certificate ADMIN TO DATABASE_ROLES_1#OO WITH ADMIN OPTION;
GRANT Log ADMIN TO DATABASE_ROLES_1#OO WITH ADMIN OPTION;
```

8.3.7.1.2 GRANT to global role _SYS_DI_OO_DEFAULTS

The description of this method can be found in section GRANT to global role _SYS_DI_OO_DEFAULTS.

To grant the required privilege to the #OO user through the _SYS_DI_OO_DEFAULTS role, we need to execute the following script as SYSTEM user.

```
GRANT SELECT ON SCHEMA _SYS_STATISTICS TO _SYS_DI_OO_DEFAULTS WITH GRANT OPTION;
GRANT SELECT, INSERT, UPDATE, DELETE ON _SYS_SECURITY._SYS_PASSWORD_BLACKLIST TO _SYS_DI_OO_DEFAULTS WITH GRANT OPTION;
GRANT CATALOG READ TO _SYS_DI_OO_DEFAULTS WITH ADMIN OPTION;
GRANT SERVICE ADMIN TO _SYS_DI_OO_DEFAULTS WITH ADMIN OPTION;
GRANT INIFILE ADMIN TO _SYS_DI_OO_DEFAULTS WITH ADMIN OPTION;
GRANT TRACE ADMIN TO _SYS_DI_OO_DEFAULTS WITH ADMIN OPTION;
GRANT SESSION ADMIN TO _SYS_DI_OO_DEFAULTS WITH ADMIN OPTION;
```
8.3.7.1.3 Using a user provided service with a granting user

The description of this method can be found in section Using a user provided service with a granting user.

For this method, we will create a database user named PRIVILEGE_GRANTOR_USER and we will grant to this user all the required privileges for role developing with GRANT / ADMIN option. For this, we need to execute the following script as SYSTEM user at database level.

```
CREATE USER PRIVILEGE_GRANTOR_USER PASSWORD ABCD1234xyz NO FORCE_FIRST_PASSWORD_CHANGE;
GRANT SELECT ON SCHEMA _SYS_STATISTICS TO PRIVILEGE_GRANTOR_USER WITH GRANT OPTION;
GRANT SELECT, INSERT, UPDATE, DELETE ON _SYS_SECURITY._SYS_PASSWORD_BLACKLIST TO PRIVILEGE_GRANTOR_USER WITH GRANT OPTION;
GRANT CATALOG READ TO PRIVILEGE_GRANTOR_USER WITH ADMIN OPTION;
GRANT INIFILE ADMIN TO PRIVILEGE_GRANTOR_USER WITH ADMIN OPTION;
GRANT TRACE ADMIN TO PRIVILEGE_GRANTOR_USER WITH ADMIN OPTION;
GRANT SESSION ADMIN TO PRIVILEGE_GRANTOR_USER WITH ADMIN OPTION;
GRANT VERSION ADMIN TO PRIVILEGE_GRANTOR_USER WITH ADMIN OPTION;
GRANT LICENSE ADMIN TO PRIVILEGE_GRANTOR_USER WITH ADMIN OPTION;
GRANT RESOURCE ADMIN TO PRIVILEGE_GRANTOR_USER WITH ADMIN OPTION;
GRANT BACKUP OPERATOR TO PRIVILEGE_GRANTOR_USER WITH ADMIN OPTION;
GRANT BACKUP ADMIN TO PRIVILEGE_GRANTOR_USER WITH ADMIN OPTION;
GRANT CREATE SCHEMA TO PRIVILEGE_GRANTOR_USER WITH ADMIN OPTION;
GRANT EXPORT TO PRIVILEGE_GRANTOR_USER WITH ADMIN OPTION;
GRANT AUDIT ADMIN TO PRIVILEGE_GRANTOR_USER WITH ADMIN OPTION;
GRANT AUDIT OPERATOR TO PRIVILEGE_GRANTOR_USER WITH ADMIN OPTION;
GRANT USER ADMIN TO PRIVILEGE_GRANTOR_USER WITH ADMIN OPTION;
GRANT ENCRYPTION ROOT KEY ADMIN TO PRIVILEGE_GRANTOR_USER WITH ADMIN OPTION;
GRANT SSL ADMIN TO PRIVILEGE_GRANTOR_USER WITH ADMIN OPTION;
GRANT TRUST ADMIN TO PRIVILEGE_GRANTOR_USER WITH ADMIN OPTION;
GRANT CERTIFICATE ADMIN TO PRIVILEGE_GRANTOR_USER WITH ADMIN OPTION;
GRANT LOG ADMIN TO PRIVILEGE_GRANTOR_USER WITH ADMIN OPTION;
```
8.3.7.1.4 Using a user provided service with a procedure grantor

The description of this method can be found in section Using a user provided service with a procedure grantor.

For this method, we will create two database users. The first user will be named PRIVILEGE_PROCEDURE_GRANTOR_USER and we will grant to this user all the required privileges for role developing with GRANT / ADMIN option. This user will also create a database procedure named PRIVILEGE_PROCEDURE_GRANTOR_DEFINER.GRANT to be used by the user provided service.

The second database user will be named PRIVILEGE_PROCEDURE_GRANTOR_USER and this user will be granted only the EXECUTE privilege on procedure PRIVILEGE_PROCEDURE_GRANTOR_DEFINER.GRANT. This user will be exposed to the XSA space on the user provided service.

For this, we need to execute the following script as SYSTEM user at database level.

```sql
CREATE USER PRIVILEGE_PROCEDURE_GRANTOR_USER PASSWORD ABCD1234xyz NO FORCE_FIRST_PASSWORD_CHANGE;
CREATE USER PRIVILEGE_PROCEDURE_GRANTOR_DEFINER PASSWORD ABCD1234xyz NO FORCE_FIRST_PASSWORD_CHANGE;
GRANT SELECT ON SCHEMA _SYS_STATISTICS TO PRIVILEGE_PROCEDURE_GRANTOR_DEFINER WITH GRANT OPTION;
GRANT SELECT, INSERT, UPDATE, DELETE ON _SYS_SECURITY._SYS_PASSWORD_BLACKLIST TO PRIVILEGE_PROCEDURE_GRANTOR_DEFINER WITH GRANT OPTION;
GRANT CATALOG READ TO PRIVILEGE_PROCEDURE_GRANTOR_DEFINER WITH ADMIN OPTION;
GRANT SERVICE ADMIN TO PRIVILEGE_PROCEDURE_GRANTOR_DEFINER WITH ADMIN OPTION;
GRANT INIFILE ADMIN TO PRIVILEGE_PROCEDURE_GRANTOR_DEFINER WITH ADMIN OPTION;
GRANT TRACE ADMIN TO PRIVILEGE_PROCEDURE_GRANTOR_DEFINER WITH ADMIN OPTION;
GRANT SESSION ADMIN TO PRIVILEGE_PROCEDURE_GRANTOR_DEFINER WITH ADMIN OPTION;
GRANT VERSION ADMIN TO PRIVILEGE_PROCEDURE_GRANTOR_DEFINER WITH ADMIN OPTION;
GRANT LICENSE ADMIN TO PRIVILEGE_PROCEDURE_GRANTOR_DEFINER WITH ADMIN OPTION;
GRANT SAVEPOINT ADMIN TO PRIVILEGE_PROCEDURE_GRANTOR_DEFINER WITH ADMIN OPTION;
GRANT RESOURCE ADMIN TO PRIVILEGE_PROCEDURE_GRANTOR_DEFINER WITH ADMIN OPTION;
GRANT BACKUP OPERATOR TO PRIVILEGE_PROCEDURE_GRANTOR_DEFINER WITH ADMIN OPTION;
GRANT BACKUP ADMIN TO PRIVILEGE_PROCEDURE_GRANTOR_DEFINER WITH ADMIN OPTION;
GRANT CREATE SCHEMA TO PRIVILEGE_PROCEDURE_GRANTOR_DEFINER WITH ADMIN OPTION;
GRANT SYSTEM ADMIN TO PRIVILEGE_PROCEDURE_GRANTOR_DEFINER WITH ADMIN OPTION;
GRANT APPLICATION ADMIN TO PRIVILEGE_PROCEDURE_GRANTOR_DEFINER WITH ADMIN OPTION;
GRANT USER ADMIN TO PRIVILEGE_PROCEDURE_GRANTOR_DEFINER WITH ADMIN OPTION;
GRANT ENCRYPTION ROOT KEY ADMIN TO PRIVILEGE_PROCEDURE_GRANTOR_DEFINER WITH ADMIN OPTION;
```
GRANT SSL ADMIN TO PRIVILEGE_PROCEDURE_GRANTOR_DEFINER WITH ADMIN OPTION;
GRANT TRUST ADMIN TO PRIVILEGE_PROCEDURE_GRANTOR_DEFINER WITH ADMIN OPTION;
GRANT CERTIFICATE ADMIN TO PRIVILEGE_PROCEDURE_GRANTOR_DEFINER WITH ADMIN OPTION;
GRANT LOG ADMIN TO PRIVILEGE_PROCEDURE_GRANTOR_DEFINER WITH ADMIN OPTION;

CONNECT PRIVILEGE_PROCEDURE_GRANTOR_DEFINER PASSWORD ABCD1234xyz;

CREATE PROCEDURE PRIVILEGE_PROCEDURE_GRANTOR_DEFINER.GRANT(
  IN PRIVILEGES TABLE (  
    PRIVILEGE_TYPE NVARCHAR(128), -- 'SCHEMA_OBJECT_PRIVILEGE'  
      -- 'GLOBAL_OBJECT_PRIVILEGE'  
      -- 'SCHEMA_ROLE'  
      -- 'GLOBAL_ROLE'  
      -- 'SCHEMA_PRIVILEGE'  
      -- 'SYSTEM_PRIVILEGE'  
    PRIVILEGE_NAME NVARCHAR(256), -- cf. SYS.PRIVILEGES  
    OBJECT_SCHEMA NVARCHAR(256), -- NULL or schema  
    OBJECT_NAME NVARCHAR(256),  
    OBJECT_TYPE NVARCHAR(128), -- NULL or 'REMOTE SOURCE'  
    GRANTEE_SCHEMA NVARCHAR(256), -- NULL or schema  
    GRANTEE_NAME NVARCHAR(256),  
    GRANTABLE NVARCHAR(5)         -- 'TRUE' or 'FALSE'  
  )
)
LANGUAGE SQLSCRIPT
SQL SECURITY DEFINER
AS
BEGIN
  DECLARE ERROR CONDITION FOR SQL_ERROR_CODE 10000;
  DECLARE CURSOR PRIVILEGES_CURSOR FOR SELECT * FROM :PRIVILEGES;

  -- TODO: add checks for valid grantees, e.g. check with _SYS.Di#<group>.M_CONTAINER_SCHEMAS
  -- or with SYS.USERS and creator and grantee like '%#OO'
  -- TODO: keep only functionality that should be allowed, e.g. only allow to grant schema-local
  -- roles, but no object privileges, etc.

  FOR PRIVILEGE AS PRIVILEGES_CURSOR
    DO
      DECLARE TO_GRANTEE_CLAUSE NVARCHAR(512);
      DECLARE GRANTABLE_CLAUSE NVARCHAR(512) = '';

      IF PRIVILEGE.GRANTEE_SCHEMA IS NULL THEN
        TO_GRANTEE_CLAUSE = ' TO '' || ESCAPE_DOUBLE_QUOTES(PRIVILEGE.GRANTEE_NAME) || '';
      ELSE
        TO_GRANTEE_CLAUSE = ' TO '' || ESCAPE_DOUBLE_QUOTES(PRIVILEGE.GRANTEE_SCHEMA) || ''';
      END IF;
  END FOR;
IF PRIVILEGE.GRANTABLE = 'TRUE' THEN
  IF PRIVILEGE.PRIVILEGE_TYPE = 'SYSTEM_PRIVILEGE' OR
     PRIVILEGE.PRIVILEGE_TYPE = 'GLOBAL_ROLE' OR
     PRIVILEGE.PRIVILEGE_TYPE = 'SCHEMA_ROLE' THEN
    GRANTABLE_CLAUSE = ' WITH ADMIN OPTION';
  ELSE
    GRANTABLE_CLAUSE = ' WITH GRANT OPTION';
  END IF;
ELSEIF PRIVILEGE.GRANTABLE != 'FALSE' THEN
  SIGNAL ERROR SET MESSAGE_TEXT = 'unsupported value for GRANTABLE: ' || PRIVILEGE.GRANTABLE;
END IF;

IF PRIVILEGE.PRIVILEGE_TYPE = 'SCHEMA_OBJECT_PRIVILEGE' THEN
  EXEC 'GRANT ' || ESCAPE_DOUBLE_QUOTES(PRIVILEGE.PRIVILEGE_NAME) || ' ON ' || ESCAPE_DOUBLE_QUOTES(PRIVILEGE.OBJECT_SCHEMA) || '.' || ESCAPE_DOUBLE_QUOTES(PRIVILEGE.OBJECT_NAME) || ' ' || TO_GRANTEE_CLAUSE || GRANTABLE_CLAUSE;
ELSEIF PRIVILEGE.PRIVILEGE_TYPE = 'GLOBAL_OBJECT_PRIVILEGE' THEN
  IF PRIVILEGE.OBJECT_TYPE = 'REMOTE SOURCE' THEN
    EXEC 'GRANT ' || ESCAPE_DOUBLE_QUOTES(PRIVILEGE.PRIVILEGE_NAME) || ' ON ' || PRIVILEGE.OBJECT_TYPE || ' ' || ESCAPE_DOUBLE_QUOTES(PRIVILEGE.OBJECT_NAME) || ' ' || TO_GRANTEE_CLAUSE || GRANTABLE_CLAUSE;
  ELSE
    SIGNAL ERROR SET MESSAGE_TEXT = 'unsupported value for OBJECT_TYPE for GLOBAL_OBJECT_PRIVILEGE: ' || PRIVILEGE.OBJECT_TYPE;
  END IF;
ELSEIF PRIVILEGE.PRIVILEGE_TYPE = 'SCHEMA_ROLE' THEN
  EXEC 'GRANT ' || ESCAPE_DOUBLE_QUOTES(PRIVILEGE.OBJECT_SCHEMA) || '.' || ESCAPE_DOUBLE_QUOTES(PRIVILEGE.OBJECT_NAME) || ' ' || TO_GRANTEE_CLAUSE || GRANTABLE_CLAUSE;
ELSEIF PRIVILEGE.PRIVILEGE_TYPE = 'GLOBAL_ROLE' THEN
  EXEC 'GRANT ' || ESCAPE_DOUBLE_QUOTES(PRIVILEGE.OBJECT_NAME) || ' ' || TO_GRANTEE_CLAUSE || GRANTABLE_CLAUSE;
ELSEIF PRIVILEGE.PRIVILEGE_TYPE = 'SCHEMA_PRIVILEGE' THEN
  EXEC 'GRANT ' || ESCAPE_DOUBLE_QUOTES(PRIVILEGE.PRIVILEGE_NAME) || ' ' || TO_GRANTEE_CLAUSE || GRANTABLE_CLAUSE;
END IF;
8.4 Create design-time objects

In the following section, we will find the description and the definitions of all the design-time objects needed for the deployment and management of the template roles in the HANA database. These objects are:

- Synonyms
- Procedures
- Roles

We recommend creating the following structure of folders within the project to organize all the design-time objects.

```
8.4.1 Synonyms

Database synonyms are created using a synonym definition file (hdbsynonym) and are needed to refer to external objects, such as tables, views, procedures, etc. Synonyms. In role development, synonyms are needed to refer to objects privileges.

In the database-role project, one synonym definition file was created per schema. The following synonym definitions have been created.
```
8.4.1.1 _SYS_SECURITY

This synonym declaration contains all the definition of the synonyms to reference objects from the _SYS_SECURITY schema.

The following synonyms are defined:

<table>
<thead>
<tr>
<th>Privilege</th>
<th>Object</th>
<th>Schema Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>_sys_security._sys_password_blacklist</td>
<td>_SYS_PASSWORD_BLACKLIST</td>
<td>_SYS_SECURITY</td>
</tr>
</tbody>
</table>

8.4.1.1.1 _SYS_SECURITY.hdbsynonym

```json
{
    "_sys_security._sys_password_blacklist": {
        "target": {
            "object": "_SYS_PASSWORD_BLACKLIST",
            "schema": "_SYS_SECURITY"
        }
    }
}
```

8.4.1.2 SYS

This synonym declaration contains all the definition of the synonyms to reference objects from the SYS schema.

The following synonyms are defined:

<table>
<thead>
<tr>
<th>Privilege</th>
<th>Object</th>
<th>Schema Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>sys.users</td>
<td>USERS</td>
<td>SYS</td>
</tr>
<tr>
<td>sys.roles</td>
<td>ROLES</td>
<td>SYS</td>
</tr>
<tr>
<td>sys.dummy</td>
<td>DUMMY</td>
<td>SYS</td>
</tr>
<tr>
<td>sys.m_services</td>
<td>M_SERVICES</td>
<td>SYS</td>
</tr>
<tr>
<td>sys.m_services_memory</td>
<td>M_SERVICES_MEMORY</td>
<td>SYS</td>
</tr>
<tr>
<td>sys.m_services_statistics</td>
<td>M_SERVICES_STATISTICS</td>
<td>SYS</td>
</tr>
<tr>
<td>sys.m_heap_memory_reset</td>
<td>M_HEAP_MEMORY_RESET</td>
<td>SYS</td>
</tr>
</tbody>
</table>

8.4.1.2.1 SYS.hdbsynonym

```json
{
    "sys.users": {
        "target": {
            "schema": "SYS",
            "object": "USERS"
        }
    },
    "sys.roles": {
        "target": {
            "schema": "SYS",
            "object": "ROLES"
        }
    }
}
```
8.4.2 Roles
We have designed the role templates rather detailed on purpose. The high granularity supports a highly specialized IT team setup and even if the roles may not fit perfectly the requirements of a given IT team, it will be easy to create appropriate roles for most circumstances.

At the same time, most IT teams will not require the full granularity that we introduce. Acknowledging this, we also introduce collection roles which we expect to be a good match in most cases.

8.4.2.1 Granular Roles
The following granular roles are created to group privileges needed in multiple end-user roles and for simplification of the maintenance. Granular roles are not designed to be granted to end-users but to be included in end-user roles.
8.4.2.1.1 SELECT__SYS_STATISTICS
Role to grant read-only access to schema _SYS_STATISTICS.

8.4.2.1.1.1 Granted privileges

<table>
<thead>
<tr>
<th>Privilege</th>
<th>What does it do?</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT on schema _SYS_STATISTICS</td>
<td>View alerts from the statistics server.</td>
</tr>
</tbody>
</table>

8.4.2.1.2 SELECT__SYS_STATISTICS.hdbroleconfig

```
{
  "SELECT__SYS_STATISTICS": {
    "_SYS_STATISTICS_schema": {
      "schema": "_SYS_STATISTICS"
    }
  }
}
```

8.4.2.1.3 SELECT__SYS_STATISTICS.hdbrole

```
{
  "role": {
    "name": "SELECT__SYS_STATISTICS",
    "schema_privileges": {
      "reference": "_SYS_STATISTICS_schema",
      "privileges": ["SELECT"]
    }
  }
}
```

8.4.2.2 DB Admin Roles
The following roles are for database administrators:
- **DB_SYSTEM_ADMIN_GENERIC** which in itself is composed of three further roles:
  - **DB_BACKUP_ADMIN**
  - **DB_BASIC_ADMIN**
  - **DB_PERSISTANCE_ADMIN**
- **DB_BACKUP_OPERATOR**
- **DB_DATA_ADMIN**
- **DB_DATABASE_MONITORING**

8.4.2.2.1 DB_BACKUP_ADMIN
Simple role that allows all backup-related tasks, such as creating a database backup or managing the backup catalog or deleting backups from disk.

8.4.2.2.1.1 Granted Privileges

<table>
<thead>
<tr>
<th>Privilege</th>
<th>What does it do?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
System privilege CATALOG READ | Authorizes unfiltered access to the data in the system views that a user has already been granted the SELECT privilege on.

System privilege BACKUP ADMIN | Authorizes BACKUP and RECOVERY statements for defining and initiating backup and recovery procedures. It also authorizes changing system configuration options with respect to backup and recovery.

8.4.2.2.1.2 DB_BACKUP_ADMIN.hdbrole

```
{
  "role": {
    "name": "DB_BACKUP_ADMIN",
    "system_privileges": [
      "CATALOG READ",
      "BACKUP ADMIN"
    ]
  }
}
```

8.4.2.2.2 DB_BACKUP_OPERATOR

Minimal role for a user who can only create data backups, and do nothing else.

8.4.2.2.2.1 Granted Privileges

<table>
<thead>
<tr>
<th>Privilege</th>
<th>What does it do?</th>
</tr>
</thead>
<tbody>
<tr>
<td>System privilege BACKUP OPERATOR</td>
<td>Enable creation of data backups.</td>
</tr>
</tbody>
</table>

8.4.2.2.2.2 DB_BACKUP_OPERATOR.hdbrole

```
{
  "role": {
    "name": "DB_BACKUP_OPERATOR",
    "system_privileges": [
      "BACKUP OPERATOR"
    ]
  }
}
```

8.4.2.2.3 DB_BASIC_ADMIN

This role collects all actions that any DB administrator will expect they are allowed to do.

8.4.2.2.3.1 Granted Privileges

<table>
<thead>
<tr>
<th>Privilege</th>
<th>What does it do?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Role SELECT__SYS_STATISTICS</td>
<td>Role to grant read-only access to schema _SYS_STATISTICS.</td>
</tr>
</tbody>
</table>
CATALOG READ system privilege  Authorizes unfiltered access to the data in the system views that a user has already been granted the SELECT privilege on.

SERVICE ADMIN system privilege  Authorizes the ALTER SYSTEM [START|CANCEL|RECONFIGURE] statements for administering system services of the database.

INIFILE ADMIN system privilege  Authorizes making changes to system settings.

TRACE ADMIN system privilege  Authorizes the use of the ALTER SYSTEM...TRACES statements for operations on database trace files and authorizes changing trace system settings.

SESSION ADMIN system privilege  Authorizes the ALTER SYSTEM commands concerning sessions to stop or disconnect a user session or to change session variables.

VERSION ADMIN system privilege  Authorizes the use of the ALTER SYSTEM RECLAIM VERSION SPACE statement of the multi-version concurrency control (MVCC) feature.

LICENSE ADMIN system privilege  Authorizes the use of the SET SYSTEM LICENSE statement to install a new license.

8.4.2.2.3.2  DB_BASIC_ADMIN.hdbrole

```json
{
  "role": {
    "name": "DB_BASIC_ADMIN",
    "schema_roles": [
      {
        "names": [
          "SELECT__SYS_STATISTICS"
        ]
      }
    ],
    "system_privileges": [
      "CATALOG_READ",
      "SERVICE_ADMIN",
      "INIFILE_ADMIN",
      "TRACE_ADMIN",
      "SESSION_ADMIN",
      "VERSION_ADMIN",
      "LICENSE_ADMIN"
    ]
  }
}
```

8.4.2.2.4  DB_DATA_ADMIN

This is a fancy role – it defines a user who can:

- Create new database schemas directly in the catalog
- Export catalog objects to the DB server (binary or csv)
- Export catalog objects to the client machine (binary or csv)
- Import catalog objects from the DB server (binary or csv)
• Import catalog objects from the client machine (binary or csv)

Due to the concept of object ownership, this role defines a very powerful user, because the creator of a DB schema owns the schema and has full access to all objects in the schema.

Similarly, the user who imports catalog objects is the object owner and thus has full access to these objects.

We suggest that this role should only be used in test and development systems, in which developer might need to be able to create their own data objects for trial purposes.

It might be useful in productive systems. However, note that the IMPORT feature basically allows overwriting all catalog objects in the database (IMPORT ... WITH REPLACE).

In our eyes, also the EXPORT privilege should be avoided, because there will normally not be any justification for exporting tables or other objects out of a production system.

CREATE SCHEMA should also not be needed in production systems, except for initial system setup, e.g. for setting up targets for data services.

### 8.4.2.4.1 Granted Privileges

<table>
<thead>
<tr>
<th>Privilege</th>
<th>What does it do?</th>
</tr>
</thead>
<tbody>
<tr>
<td>System privilege CREATE SCHEMA</td>
<td>Create new schemas directly in the database catalog.</td>
</tr>
<tr>
<td>System privilege EXPORT</td>
<td>Export catalog objects to the DB server (csv/binary) or to the client machine.</td>
</tr>
<tr>
<td>System privilege IMPORT</td>
<td>Import catalog objects from the DB server (csv/binary) or from the client machine.</td>
</tr>
</tbody>
</table>

### 8.4.2.4.2 DB_DATA_ADMIN.hdbrole

```
{
   "role": {
      "name": "DB_DATA_ADMIN",
      "system_privileges": [
         "CREATE SCHEMA",
         "EXPORT",
         "IMPORT"
      ]
   }
}
```

### 8.4.2.5 DB_DATABASE_MONITORING

Role for a read-only user who can perform all typical DB monitoring tasks, such as monitoring memory and other resource consumption, system alerts, view traces, view system configuration, etc.

### 8.4.2.5.1 Granted Privileges

<table>
<thead>
<tr>
<th>Privilege</th>
<th>What does it do?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Role SELECT__SYS_STATISTICS</td>
<td>Role to grant read-only access to schema _SYS_STATISTICS.</td>
</tr>
<tr>
<td>CATALOG READ system privilege</td>
<td>Authorizes unfiltered access to the data in the system views that a user has already been granted the SELECT privilege on.</td>
</tr>
<tr>
<td>SELECT on synonym sys.m_services</td>
<td>Read the status of all services.</td>
</tr>
<tr>
<td>SELECT on synonym sys.m_service_memory</td>
<td>Read detailed information on memory utilization by services.</td>
</tr>
</tbody>
</table>
### Role templates

<table>
<thead>
<tr>
<th>SELECT on synonym sys.m_service_statistics</th>
<th>Read statistics on active services.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT on synonym sys.m_heap_memory_reset</td>
<td>Read memory allocator statistics since the last reset.</td>
</tr>
</tbody>
</table>

#### 8.4.2.2.5.2 DB_DATABASE_MONITORING.hdbrole

```json
{
    "role": {
        "name": "DB_DATABASE_MONITORING",
        "system_privileges": [
            "CATALOG READ"
        ],
        "schema_roles": [
            {
                "names": [
                    "SELECT__SYS_STATISTICS"
                ]
            }
        ],
        "object_privileges": [
            {
                "name": "sys.m_services",
                "type": "VIEW",
                "privileges": ["SELECT"]
            },
            {
                "name": "sys.m_service_memory",
                "type": "VIEW",
                "privileges": ["SELECT"]
            },
            {
                "name": "sys.m_service_statistics",
                "type": "VIEW",
                "privileges": ["SELECT"]
            },
            {
                "name": "sys.m_heap_memory_reset",
                "type": "VIEW",
                "privileges": ["SELECT"]
            }
        ]
    }
}
```
### 8.4.2.2.6 DB_PERSISTENCE_ADMIN

This role enables persistence-related tasks such as cleaning up free (unused) space in data and log volume, or enforcing a savepoint.

#### 8.4.2.2.6.1 Granted Privileges

<table>
<thead>
<tr>
<th>Privilege</th>
<th>What does it do?</th>
</tr>
</thead>
<tbody>
<tr>
<td>System privilege CATALOG READ</td>
<td>Authorizes unfiltered access to the data in the system views that a user has already been granted the SELECT privilege on.</td>
</tr>
<tr>
<td>System Privilege SAVEPOINT ADMIN</td>
<td>Authorizes the execution of a savepoint using the ALTER SYSTEM SAVEPOINT statement.</td>
</tr>
<tr>
<td>System Privilege RESOURCE ADMIN</td>
<td>Authorizes statements concerning system resources (for example, the ALTER SYSTEM RECLAIM DATAVOLUME and ALTER SYSTEM RESET MONITORING VIEW statements). It also authorizes many of the statements available in the Management Console.</td>
</tr>
<tr>
<td>System Privilege LOG ADMIN</td>
<td>Authorizes the use of the ALTER SYSTEM LOGGING [ON</td>
</tr>
</tbody>
</table>

```
8.4.2.2.6.2 DB_PERSISTENCE_ADMIN.hdbrole
{
    "role": {
        "name": "DB_PERSISTENCE_ADMIN",
        "system_privileges": [
            "CATALOG READ",
            "SAVEPOINT ADMIN",
            "RESOURCE ADMIN",
            "LOG ADMIN"
        ]
    }
}
```

### 8.4.2.2.7 DB_SYSTEM_ADMIN_GENERIC

For convenience, we are combining the three roles DB_BASIC_ADMIN, DB_PERSISTENCE_ADMIN and DB_BACKUP_ADMIN into one typical generic administrator role.

#### 8.4.2.2.7.1 Granted Privileges

<table>
<thead>
<tr>
<th>Privilege</th>
<th>What does it do?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Role DB_BASIC_ADMIN</td>
<td>See role description.</td>
</tr>
<tr>
<td>Role DB_PERSISTENCE_ADMIN</td>
<td>See role description.</td>
</tr>
<tr>
<td>Role DB_BACKUP_ADMIN</td>
<td>See role description.</td>
</tr>
</tbody>
</table>

```
8.4.2.2.7.2 DB_SYSTEM_ADMIN GENERIC.hdbrole
{
    "role": {
```

Best practices and recommendations for developing roles in SAP® HANA, March 2018 (PUBLIC) 39
8.4.2.3 Security Roles

The following roles are for security administrators:

- **SECURITY_ADMIN_EXTENDED** which in itself is composed of two further roles
  - **SECURITY_ADMIN** which in itself is composed of two further roles
    - SECURITY_ADMIN_BASIC
    - SECURITY_ADMIN_TROUBLESHOOTING
  - AUDIT_OPERATOR
  - SECURITY_ADMIN_CERTIFICATES
  - SECURITY_ADMIN_DISK_ENCRYPTION

8.4.2.3.1 AUDIT_OPERATOR

Defining the security policies and monitoring the security log should generally be separated. If we write audit logs to the syslog daemon, we get the separation by default (only depending on how we configure the syslog itself). If we write audit entries to the trail target “database table”, our proposed security admin roles do not have read or write access to this audit trail. That’s why we introduce the additional role AUDIT_OPERATOR.

The role lacks read access to the audit configuration, because today HANA does not offer a privilege to only grant read without write on the audit config.

We also do not add catalog read to the audit operator role, which we might add if we think it is needed.

8.4.2.3.1.1 Granted Privileges

<table>
<thead>
<tr>
<th>Privilege</th>
<th>What does it do?</th>
</tr>
</thead>
<tbody>
<tr>
<td>System privilege AUDIT OPERATOR</td>
<td>Authorizes the execution of the following statement: ALTER SYSTEM CLEAR AUDIT LOG. It also allows access to the AUDIT_LOG system view.</td>
</tr>
</tbody>
</table>

8.4.2.3.1.2 AUDIT_OPERATOR.hdbrole
8.4.2.3.2 SECURITY_ADMIN_AUDIT
Our basic security admin has no privileges to manage the database’s audit setting. Instead, we propose a dedicated role for this purpose.

Note that we separate administration of the audit from reading (and managing) the content of the audit trail. See role AUDIT_OPERATOR for that purpose.

8.4.2.3.2.1 Granted Privileges

<table>
<thead>
<tr>
<th>Privilege</th>
<th>What does it do?</th>
</tr>
</thead>
<tbody>
<tr>
<td>System privilege CATALOG READ</td>
<td>Authorizes unfiltered access to the data in the system views that a user has already been granted the SELECT privilege on.</td>
</tr>
<tr>
<td>System privilege AUDIT ADMIN</td>
<td>Controls the execution of the following auditing-related statements: CREATE AUDIT POLICY, DROP AUDIT POLICY, and ALTER AUDIT POLICY, as well as changes to the auditing configuration. It also allows access to the AUDIT_LOG system view.</td>
</tr>
</tbody>
</table>

8.4.2.3.2 SECURITY_ADMIN_AUDIT.hdbrole

```json
{
  "role": {
    "name": "SECURITY_ADMIN_AUDIT",
    "system_privileges": [
      "CATALOG READ",
      "AUDIT ADMIN"
    ]
  }
}
```

8.4.2.3.3 SECURITY_ADMIN_BASIC
For a security admin, we identified a number of essential privileges, and a few optional ones. This role SECURITY_ADMIN_BASIC collects the essential privileges. Below we find roles SECURITY_ADMIN_TROUBLESHOOTING and SECURITY_ADMIN_AUDIT, which enable further actions a security admin might need to perform.

We also implemented two collection roles, named SECURITY_ADMIN and SECURITY_ADMIN_EXTENDED, both of which include SECURITY_ADMIN_BASIC.

Our security admin (basic) can modify security-related properties of the database, such as modifying the password policy, or managing the password blacklist.

8.4.2.3.3.1 Granted Privileges

<table>
<thead>
<tr>
<th>Privilege</th>
<th>What does it do?</th>
</tr>
</thead>
<tbody>
<tr>
<td>System privilege CATALOG READ</td>
<td>Authorizes unfiltered access to the data in the system views that a user has already been granted the SELECT privilege on.</td>
</tr>
<tr>
<td>System privilege INIFILE ADMIN</td>
<td>Authorizes making changes to system settings.</td>
</tr>
<tr>
<td>SELECT, INSERT, UPDATE and DELETE</td>
<td>Authorizes making changes to system settings.</td>
</tr>
<tr>
<td>on synonym _sys_security__sys_password_blacklist</td>
<td>Authorizes making changes to system settings.</td>
</tr>
</tbody>
</table>

8.4.2.3.3.2 SECURITY_ADMIN_BASIC.hdbrole

```json
{
  "role": {
    "name": "SECURITY_ADMIN_BASIC",
    "system_privileges": [
      "CATALOG READ",
      "INIFILE ADMIN",
      "SELECT",
      "INSERT",
      "UPDATE",
      "DELETE"
    ]
  }
}
```
8.4.2.3.4 SECURITY_ADMIN_CERTIFICATES
This role assembles further privileges that can be useful for a security admin, but are not necessarily essential. The role enables the management of certificates and certificate collections in the system.

8.4.2.3.4.1 Granted Privileges

<table>
<thead>
<tr>
<th>Privilege</th>
<th>What does it do?</th>
</tr>
</thead>
<tbody>
<tr>
<td>System privilege CATALOG READ</td>
<td>Authorizes unfiltered access to the data in the system views that a user has already been granted the SELECT privilege on.</td>
</tr>
<tr>
<td>SSL ADMIN</td>
<td>Authorizes the use of the SET...PURPOSE SSL statement. It also allows access to the PSES system view.</td>
</tr>
<tr>
<td>TRUST ADMIN</td>
<td>Authorizes the use of statements to update the trust store.</td>
</tr>
<tr>
<td>CERTIFICATE ADMIN</td>
<td>Authorizes the changing of certificates and certificate collections that are stored in the database.</td>
</tr>
</tbody>
</table>

8.4.2.3.4.2 SECURITY_ADMIN_CERTIFICATES.hdbrole

```json
"role":{
    "name": "SECURITY_ADMIN_CERTIFICATES",
    "system_privileges": [
        "CATALOG READ",
        "SSL ADMIN",
        "TRUST ADMIN",
        "CERTIFICATE ADMIN"
    ]
}
```
8.4.2.3.5 SECURITY_ADMIN_DISK_ENCRYPTION
This role assembles further privileges that can be useful for a security admin but are not necessarily essential. Presently, the role allows switching on and off disk encryption (encryption of the data volumes). The required “RESOURCE ADMIN” privilege has the side effect of allowing further data volume administrative tasks.

8.4.2.3.5.1 Granted Privileges

<table>
<thead>
<tr>
<th>Privilege</th>
<th>What does it do?</th>
</tr>
</thead>
<tbody>
<tr>
<td>System privilege CATALOG READ</td>
<td>Authorizes unfiltered access to the data in the system views that a user has already been granted the SELECT privilege on.</td>
</tr>
<tr>
<td>System Privilege RESOURCE ADMIN</td>
<td>Authorizes statements concerning system resources (for example, the ALTER SYSTEM RECLAIM DATAVOLUME and ALTER SYSTEM RESET MONITORING VIEW statements). It also authorizes many of the statements available in the Management Console.</td>
</tr>
<tr>
<td>System Privilege ENCRYPTION ROOT KEY ADMIN</td>
<td>Authorizes all statements related to management of root keys. Allows access to the system views pertaining to encryption (for example, ENCRYPTION_ROOT_KEYS, M_ENCRYPTION_OVERVIEW, M_PERSISTENCE_ENCRYPTION_STATUS, M_PERSISTENCE_ENCRYPTION_KEYS, and so on).</td>
</tr>
</tbody>
</table>

8.4.2.3.5.2 SECURITY_ADMIN_DISK_ENCRYPTION.hdbrole

```json
{
"role": {
"name": "SECURITY_ADMIN_DISK_ENCRYPTION",
"system_privileges": [
"CATALOG READ",
"RESOURCE ADMIN",
"ENCRYPTION_ROOT_KEY_ADMIN"
]
}
}
```

8.4.2.3.6 SECURITY_ADMIN_TROUBLESHOOTING
This role assembles further privileges that can be useful for a security admin but are not necessarily essential. Presently, the role empowers the user to start and stop traces — the security admin would be especially interested in authorization traces or traces related to the handling of analytic privilege (both part of the DB trace or user-specific trace), but this cannot be controlled at that granularity, so the security admin with this privilege will have full administrative rights on the database traces.

8.4.2.3.6.1 Granted Privileges

<table>
<thead>
<tr>
<th>Privilege</th>
<th>What does it do?</th>
</tr>
</thead>
<tbody>
<tr>
<td>System privilege CATALOG READ</td>
<td>Authorizes unfiltered access to the data in the system views that a user has already been granted the SELECT privilege on.</td>
</tr>
</tbody>
</table>
8.4.2.3.6.2 SECURITY_ADMIN_TROUBLESHOOTING.hdbrole

```
"role": {
    "name": "SECURITY_ADMIN_TROUBLESHOOTING",
    "system_privileges": [
        "CATALOG READ",
        "TRACE ADMIN"
    ]
}
```

8.4.2.3.7 SECURITY_ADMIN

We assume that in most cases we will at least grant the basic and the troubleshooting privileges to our security admins. Hence, we provide a role which combines both roles and might be the standard role for security admins.

8.4.2.3.7.1 Granted Privileges

<table>
<thead>
<tr>
<th>Privilege</th>
<th>What does it do?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Role SECURITY_ADMIN_BASIC</td>
<td>See role description.</td>
</tr>
<tr>
<td>Role SECURITY_ADMIN_TROUBLESHOOTING</td>
<td>See role description.</td>
</tr>
</tbody>
</table>

8.4.2.3.8 SECURITY_ADMIN_EXTENDED

We assume that in most cases we will grant the basic and the optional privileges to our security admins. Hence, we provide a role which combines both roles and might be standard the role for security admins.

8.4.2.3.8.1 Granted Privileges

<table>
<thead>
<tr>
<th>Privilege</th>
<th>What does it do?</th>
</tr>
</thead>
</table>
8.4.2.3.8.2 SECURITY_ADMIN_EXTENDED.hdbrole

```
{
    "role": {
        "name": "SECURITY_ADMIN_EXTENDED",
        "schema_roles": [
            {
                "names": [
                    "SECURITY_ADMIN",
                    "SECURITY_ADMIN_AUDIT"
                ]
            }
        ]
    }
}
```

8.4.2.4 Support Roles

We foresee the following roles for support staff that is concerned with the database in SAP HANA:

- **BASIS_SUPPORT_READ**
- **BASIS_SUPPORT_TRACE**

Even though SAP HANA comes with a very special and powerful role called SAP_INTERNAL_HANA_SUPPORT, this role should only be granted to SAP development support staff in exceptional circumstances. Hence, this role should not be used for support activities unless SAP development support asks you to.

8.4.2.4.1 BASIS_SUPPORT_READ

Role for a read-only support user who can perform all typical DB monitoring tasks, such as monitoring memory and other resource consumption, system alerts, view traces, view system configuration, etc. but has no access to the repository.

This role is simply a reference to our basis role DB_DATABASE_MONITORING. We introduce this redundancy, so we can have dedicated role to be granted to support staff.

This will normally be a safe role to give to internal database support as well as to SAP support.

**8.4.2.4.1.1 Granted Privileges**

<table>
<thead>
<tr>
<th>Privilege</th>
<th>What does it do?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Role DB_DATABASE_MONITORING</td>
<td>See role details.</td>
</tr>
</tbody>
</table>

8.4.2.4.2 BASIS_SUPPORT_READ.hdbrole

```
{
    "role": {
        "name": "BASIS_SUPPORT_READ",
        "schema_roles": [
            {
                "names": [
                    "BASIS_SUPPORT_READ",
                    "SECURITY_ADMIN"
                ]
            }
        ]
    }
}
```
8.4.2.4.2 BASIS_SUPPORT_TRACE

Role for a read-only support user who can start/stop traces and perform all typical DB monitoring tasks, such as monitoring memory and other resource consumption, system alerts, view traces, view system configuration, etc.

The role extends role BASIS_SUPPORT_READ and adds the TRACE ADMIN privilege.

Because of the trace management capabilities, we suggest that usage of this role should be restricted in more critical system landscapes, especially when working with external support such as SAP support staff.

8.4.2.4.2.1 Granted Privileges

<table>
<thead>
<tr>
<th>Privilege</th>
<th>What does it do?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Role BASIS_SUPPORT_READ</td>
<td>See role details.</td>
</tr>
<tr>
<td>System privilege TRACE ADMIN</td>
<td>Authorizes the use of the ALTER SYSTEM...TRACES statements for operations on database trace files and authorizes changing trace system settings.</td>
</tr>
</tbody>
</table>

8.4.2.5 User Admin Roles

For user administration tasks, it is recommended to separate the user creation from the role assignment. Based on this, the following roles don’t share any privilege and separate these two activities:

- CONTAINER_ROLE_ADMIN
- USER_ADMIN
8.4.2.5.1 CONTAINER_ROLE_ADMIN
Role for granting and revoking database roles that are deployed within the same HDI schema. The ROLE ADMIN is not included to avoid granting catalog roles or roles from different HDI schema.

This role does not include USER ADMIN system privileges so it does not allow user creation and management.

8.4.2.5.1.1 Granted Privileges

<table>
<thead>
<tr>
<th>Privilege</th>
<th>What does it do?</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXECUTE on procedure GRANT_ROLE_TO_USER</td>
<td>Grant to a database user any HDI role created within the same HDI schema.</td>
</tr>
<tr>
<td>EXECUTE on procedure REVOKE_ROLE_TO_USER</td>
<td>Revoke from a database user any HDI role created within the same HDI schema.</td>
</tr>
</tbody>
</table>

8.4.2.5.1.2 CONTAINER_ROLE_ADMIN.hdbrole

```json
{
    "role": {
        "name": "CONTAINER_ROLE_ADMIN",
        "object_privileges": [
            {
                "name": "GRANT_ROLE_TO_USER",
                "type": "PROCEDURE",
                "privileges": [ "EXECUTE" ]
            },
            {
                "name": "REVOKE_ROLE_FROM_USER",
                "type": "PROCEDURE",
                "privileges": [ "EXECUTE" ]
            }
        ]
    }
}
```

8.4.2.5.2 USER_ADMIN
Role to allow creation and modification of database user.

This role does not allow granting roles to users.

8.4.2.5.2.1 Granted Privileges

<table>
<thead>
<tr>
<th>Privilege</th>
<th>What does it do?</th>
</tr>
</thead>
<tbody>
<tr>
<td>System privileges USER_ADMIN</td>
<td>Authorizes the creation and modification of users using the CREATE USER, ALTER USER, and DROP USER commands.</td>
</tr>
</tbody>
</table>

8.4.2.5.2.2 USER_ADMIN.hdbrole

```json
{
    "role": {
    
    }
}
```
8.4.3 Procedures

8.4.3.1 Grant Roles to User
This procedure can be used to grant to a database user any HDI role created within the same HDI schema. It accepts two parameters:

- role_name
  This is the name of the role would like to be granted. There is no need to include the HDI schema name.
- grantee
  Name of the database user to which we want to grant the role.

Within the procedure the following conditions are checked, throwing errors if they are violated:

- Grantee must exist error code 11001
- Grantee must be different from grantor error code 11002
- Role must exist error code 11003

We include the EXECUTE privilege for this role in our suggested role GRANT_ROLE_TO_USER.

8.4.3.1.1 GRANT_ROLE_TO_USER.hdbprocedure

```
PROEDURE "GRANT_ROLE_TO_USER" ( 
  IN role_name NVARCHAR(256),
  IN grantee NVARCHAR(256)
)

LANGUAGE SQLSCRIPT
SQL SECURITY DEFINER
AS
  -- SQL statement we're going to execute
  v_statement NVARCHAR(1024);
  hdi_oo NVARCHAR (256);
  role_schema NVARCHAR (256);
  counter INTEGER := 0;
  error_code INTEGER;
  error_message NVARCHAR(1024);

BEGIN
  -- prepare error handling in case of invalid arguments
  DECLARE USERNOTEXIST CONDITION FOR SQL_ERROR_CODE 11001;
  DECLARE GRANTSELF CONDITION FOR SQL_ERROR_CODE 11002;
  DECLARE ROLENOTEXIST CONDITION FOR SQL_ERROR_CODE 11003;
  DECLARE EXIT HANDLER FOR USERNOTEXIST RESIGNAL;
  DECLARE EXIT HANDLER FOR GRANTSELF RESIGNAL;
  DECLARE EXIT HANDLER FOR ROLENOTEXIST RESIGNAL;
```
hdi_oo := ::CURRENT_OBJECT_SCHEMA || '#OO';
role_schema := ::CURRENT_OBJECT_SCHEMA;

-- check if role exists
SELECT COUNT(*) INTO counter FROM (SELECT * FROM "sys.roles" WHERE role_name = :role_name AND role_schema_name = :role_schema AND creator = :hdi_oo);
IF ( counter != 1 ) THEN
    SIGNAL ROLENOTEXIST SET MESSAGE_TEXT = 'Role does not exist: ' || :role_name;
END IF;

-- check input parameter user:
-- does grantee exist?
SELECT COUNT(*) INTO counter FROM (SELECT * FROM "sys.users" WHERE user_name = :grantee);
IF ( counter != 1 ) THEN
    SIGNAL USERNOTEXIST SET MESSAGE_TEXT = 'User does not exist: ' || :grantee;
END IF;

-- self grant?
IF :grantee = SESSION_USER
    THEN SIGNAL GRANTSELF SET MESSAGE_TEXT = 'Self-grant not allowed';
END IF;

-- assemble grant statement: we have to call the
v_statement := 'GRANT "' || ESCAPE_DOUBLE_QUOTES(:role_schema) || '"."' || ESCAPE_DOUBLE_QUOTES(:role_name) || '" TO "' || ESCAPE_DOUBLE_QUOTES(:grantee) || '"';

-- and run the statement:
EXEC v_statement;
END;

Invocation of the procedure:
call <HDI_schema_name>.GRANT_ROLE_TO_USER ('<role-name without schema-name>','<username>')

We will get the following warning message when the call statement is executed:
java.sql.SQLWarning: Not recommended feature: DDL statement is used in Dynamic SQL (current dynamic_sql_ddl_error_level = 1)

This message can be ignored.

8.4.3.2 Revoke Roles from Users
This procedure can be used to revoke from a database user any HDI role created within the same HDI schema. It accepts two parameters:

- role_name
  This is the name of the role would like to be revoked. There is no need to include the HDI schema name.
- grantee

Best practices and recommendations for developing roles in SAP® HANA, March 2018 (PUBLIC) 49
Name of the database user from which we want to revoke the role.

Within the procedure the following conditions are checked, throwing errors if they are violated:

- Grantee must exist error code 11001
- Grantee must be different from grantor error code 11002
- Role must exist error code 11003

We include the EXECUTE privilege for this role in our suggested role GRANT_ROLE_TO_USER.

### 8.4.3.2.1 REVOKE_ROLE_FROM_USER.hdbprocedure

```sql
PROCEDURE "REVOKE_ROLE_FROM_USER" (  
    IN role_name NVARCHAR(256),  
    IN grantee NVARCHAR(256)  
)

LANGUAGE SQLSCRIPT
SQL SECURITY DEFINER
--DEFAULT SCHEMA <default_schema_name>
AS
    -- SQL statement we're going to execute
    v_statement NVARCHAR(1024);
    hdi_oo NVARCHAR (256);
    role_schema NVARCHAR (256);
    counter INTEGER := 0;
    error_code INTEGER;
    error_message NVARCHAR(1024);

BEGIN
    -- prepare error handling in case of invalid arguments
    DECLARE USERNOTEXIST CONDITION FOR SQL_ERROR_CODE 11001;
    DECLARE GRANTSELF CONDITION FOR SQL_ERROR_CODE 11002;
    DECLARE ROLENOTEXIST CONDITION FOR SQL_ERROR_CODE 11003;
    DECLARE EXIT HANDLER FOR USERNOTEXIST RESIGNAL;
    DECLARE EXIT HANDLER FOR GRANTSELF RESIGNAL;
    DECLARE EXIT HANDLER FOR ROLENOTEXIST RESIGNAL;

    hdi_oo := ::CURRENT_OBJECT_SCHEMA || '#OO';
    role_schema := ::CURRENT_OBJECT_SCHEMA;

    -- check input parameter user:
    -- does grantee exist?
    SELECT COUNT (*) INTO counter FROM (SELECT * FROM "sys.users" WHERE user_name = :grantee);
    IF (counter != 1) THEN
        SIGNAL USERNOTEXIST SET MESSAGE_TEXT = 'User does not exist: ' || :grantee;
    END IF;
```

8.5 Deploy the project to SAP HANA

To deploy the project click on the “Build” option from the context menu of the “db” module folder.
After the successful deployment we should be able to see the new HDI schema and all the HDI roles in the SAP HANA database.
Role templates

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

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

The information contained herein may be changed without prior notice. Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

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

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

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies.