



# Oracle Database 19c Security

## *Database Security*

- **Agenda**

- **Day 1)**
  - Understand basic Oracle security solution
  - Creating basic Oracle users
  - Introduction into new Oracle sys accounts
  - Configure strong database authentication
  - Control access using Oracle Label Security
- **Day 2)**
  - Configure strong database authentication
  - Control access using Oracle Label Security
  - Assign database privileges
- **Day 3)**
  - Introduction to Oracle's Database Vault
  - Introduction into Data Redaction
  - Introduction into Transparent Data Encryption (TDE)

- **Agenda**

- Day 4)
  - Introduction into Data Masking
  - Audit Database activity
  - Oracle's network security
- Day 5)
  - Oracle's database roles and proxies
  - Security using PL/SQL
  - Definers rights and Invokers rights
  - Remap data using datapump

## *Creating a password profile*

- You'll need an existing user who has create profile privilege.
- Also, you'll need an unlocked user account.

*Make sure that the `resource_limit` parameter is set to `true`.*

## *Creating a password profile*

1. Use OS authentication to connect to the database.
2. Create a password profile with the name **userprofile** that has the following restrictions:
  - The system allows four login attempts before locking a user account (`failed_login_attempts`)
  - After locking a user account, it will remain locked for two days (`password_lock_time`)
  - A password for the user can remain unchanged for 180 days - after which the password will expire, and the user will have to change the password for his next login (`password_life_time`)
3. Assign a newly created password profile to the user.
4. Alter the default password profile with the **failed\_login\_attempts** restriction.

## *Creating a password profile*

- In Oracle Database 19c, there are three password verify functions:
  - `verify_function_11G` (carried over)
  - `ora19c_verify_function` (new)
  - `ora19c_strong_verify_function` (new)
- If password complexity checking is not enabled, you should run the `utlpwdmg.sql` script provided by Oracle.
- If you want to choose which verify function will be used in the default profile:

```
alter profile default limit password_verify_function ora19c_strong_verify_function;
```

## *Creating password-authenticated users*

- You'll need an existing user who has **create user** privilege.
- You'll use **Oracle Enterprise Manager Database Express 19c (EM Express)**.

## *Creating password-authenticated users*

- 
1. Use OS authentication to connect to the database.
  2. Create a password-authenticated user **jessica** with simpler password.
  3. Create a password-authenticated user **tom** with more complex password.

*Both of these users are using the default password profile.*

4. Create a password-authenticated user with the assigned password profile **userprofile**.

## *Creating password-authenticated users*

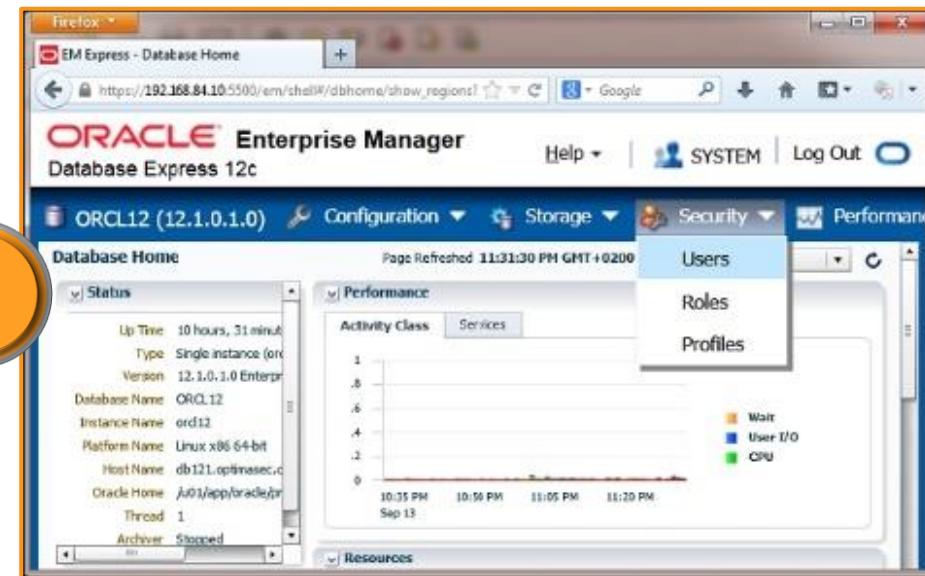
5. Create user **john**. This user has to change his password at the first database login.
  
6. Create the user **richard**. In the **create user** statement, **quota unlimited on users** means that you want to let the user allocate space in the tablespace without bound.

# How to create a user using EM Express



1

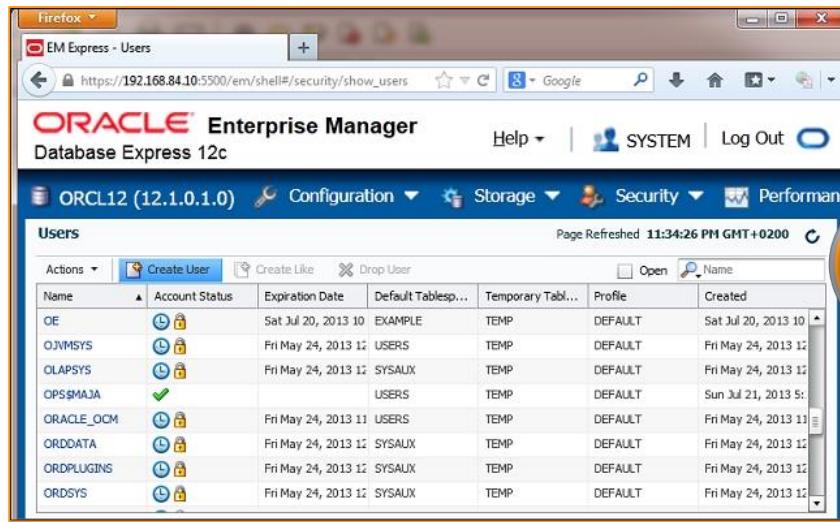
Start EM Express and log in to it using the user that has either **EM\_EXPRESS\_BASIC** or **EM\_EXPRESS\_ALL** role



2

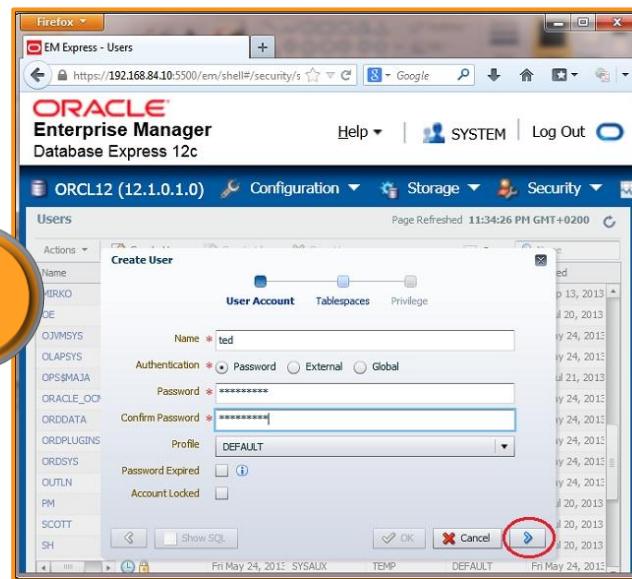
Select **Users** from the **Security** drop-down menu

# How to create a user using EM Express



3

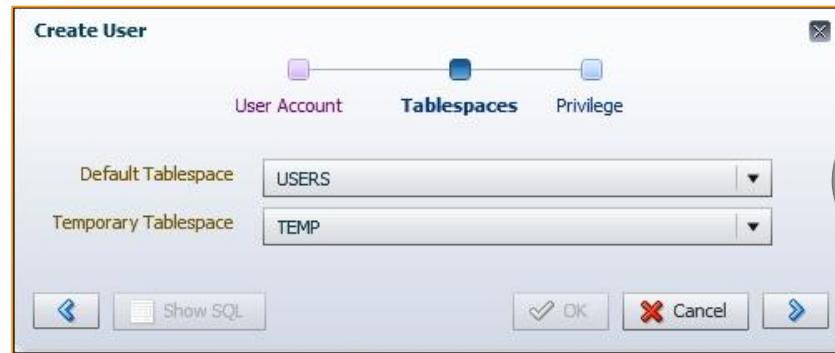
Click on the **Create User** tab



Enter user details in the pop-up dialog  
leave the default values and click on  
the **Next** button

4

# How to create a user using EM Express

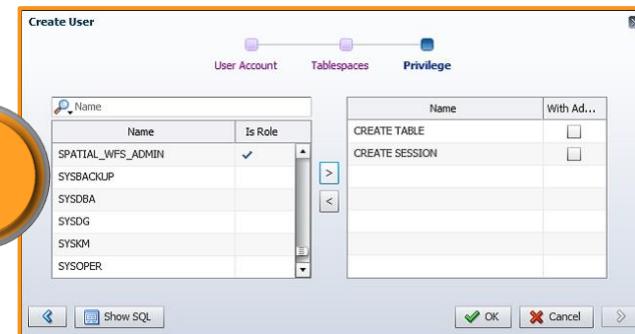


5

Choose default tablespace and temporary tablespace from the drop-down lists. Leave the default values

You can grant privileges to user by selecting them in the left panel and moving them to the right pane (use > button).

6



7

A pop-up window confirmation should appear with the following message: **SQL statement has been processed successfully.** Click on the **OK** button to close the window.

## *Changing a user's password*

- You'll need an existing user who has `alter user` privilege (you may use OS-authenticated user who has the DBA role) and other existing users (for example, `jessica` and `tom`).

## *Changing a user's password*

- 
1. Use OS authentication to connect to the database.
  2. Privilege user changes **jessica's** password.
  3. **tom** changes his own password.

## *Changing a user's password*

- There is another way to change the user's password using the **alter user** statement as follows:

```
SQL> alter user jessica identified by oracle_2;
```

*This approach is not recommended because password remains in the command-line history.*

## *Creating a user with the same credentials on another database*

- You'll need:

- An existing user who has dba role in the first database (you can use an OS-authenticated user)
- An existing user in the first database (for example, jessica)
- An existing (for example, password-authenticated) user, who has create user privilege, in the second database (for example, ernesto)

## *Creating a user with the same credentials on another database*

1. Use OS authentication to connect to the database.
2. Find a DDL statement that has been used for user creation. This DDL statement may contain **default** and **temporary** tablespace assignments.

```
SQL> select dbms_metadata.get_ddl('USER', 'JESSICA') from dual;  
  
DBMS_METADATA.GET_DDL('USER','JESSICA')  
-----  
CREATE USER "JESSICA" IDENTIFIED BY VALUES  
'S:D82E6EF961F2EA7A878BCDDBC7E5C542BC148C4759D19A720A96BBF65658;H:F297A50FD538EF4AB1  
19EB0278C9E72D;C50B1E9C9AA52EC2'  
DEFAULT TABLESPACE "USERS"      TEMPORARY TABLESPACE "TEMP"
```

3. Use only the first part of this DDL to create a user on the second database.

## *Creating a user with the same credentials on another database*

- There is another way to accomplish the task. This way requires **select** on the **sys.user\$** table:

1. Connect to the first database as a user who has the **select** privilege on the **sys.user\$** table:

```
$ sqlplus / as sysdba
```

2. Find the hash value of a user's password (for example, user **jessica**):

```
SQL> select spare4  
from user$  
where name='JESSICA';
```

## *Creating a user with the same credentials on another database*

3. Connect to the second database as a user who has **create user** privilege:  

```
$ sqlplus ernesto@orcl2
```
4. Create a user with the same username (for example, **jessica**) using the hash value of the password:

```
SQL> create user jessica identified by values  
'S:2724193130FC67E7E23E3E44E33AF143F7A6C36489792B  
5856133DCB331D;H:184895E50EA2FBCC2311ED76A3E5CF35;  
T:BECCD5FC6F6E62BC34DF1C826AEE899EC6A6025FA0D5071659DA  
7DD1ABB37763483B5C821E5A34C1184A56BE4B1C92CED79639D11101D  
61B86ACBE60A30F19CC277D5753F7D3756DC1B7705C0ACE81F3';
```

## *Locking a user account*

- You'll need an existing (for example, OS-authenticated) user who has `alter user` privilege (you may use user who has a DBA role) and another existing user (for example, `steve`).

# *Locking a user account*

1. Use OS authentication to connect to the database.
2. Lock the account of user **steve**:

```
SQL> alter user steve account lock;  
User altered  
SQL> connect steve/welcome1  
ERROR: ORA-28000: the account is locked
```

However, objects in steve's schema are available, so users can access them (considering that they have necessary privileges):

```
SQL> select a, b from steve.table1;  
      A      B  
-----  
      1      3  
      2      4  
      4      9
```

## *Locking a user account*

3. Unlock the account of user **steve**. Now user **steve** can successfully connect to the database:

```
SQL> alter user steve account unlock;  
User altered.  
SQL> conn steve/welcome1  
Connected.
```

## *Expiring a user's password*

- You'll need an existing (for example, OS-authenticated) user who has `alter user` privilege (you may use user who has a DBA role) and another existing user (for example, `steve`).

## *Expiring a user's password*

1. Use OS authentication to connect to the database.
2. Expire password for the user **steve**:

```
SQL> alter user steve password expire;
```

```
User altered.
```

```
SQL> conn steve/welcome1
```

```
ERROR: ORA-28001: the password has expired
```

```
Changing password for steve
```

```
New password:
```

```
Retype new password:
```

```
Password changed
```

```
Connected.
```

## *Creating and using OS-authenticated users*

- You'll need an existing user who has a dba role, for example, **johndba**.
- It is assumed that you are working on Linux.

# *Creating and using OS-authenticated users*

- In OS authentication, database delegates user authentication to the operating system.
- This means that in order for OS authentication to work, user must exist as the user of the operating system, these users are created with a prefix that is defined in the **os\_authent\_prefix** parameter (default is **ops\$**).
- If an OS-authenticated user has the **create session** privilege, he or she can connect to the database using the following syntax:

```
SQL> connect /  
Connected.
```

```
SQL> show user  
USER is "OPS$ernesto"
```

## *Creating and using OS-authenticated users*

Note that you cannot grant a **sysdba**, **sysoper**, **sysbackup**, **sysdg**, or **syskm** privilege to users that are identified externally, using a **grant** statement:

```
SQL> grant sysdba to ops$ernesto;  
grant sysdba to ops$ernesto
```

```
ERROR at line 1: ORA-01997: GRANT failed: user  
'OPS$ernesto' identified externally
```

# *Creating and using OS-authenticated users*

- If you want to connect as **sysdba** using OS authentication, you have to add OS user **ernesto** to OS group DBA:

```
[root@db121 ~]# usermod -a -G dba ernesto
[root@db121 ~]# su - ernesto
[ernesto@db121 ~]$ sqlplus / as sysdba
SQL*Plus: Release 12.1.0.1.0 Production on Fri Sep 03 20:14:03 2013
Copyright (c) 1982, 2013, Oracle. All rights reserved.
Connected to:
Oracle Database 19c Enterprise Edition Release 12.1.0.1.0 - 64 bit
Production With the Partitioning, OLAP, Advanced Analytics and Real Application Testing options
```

- You can change the **os\_authent\_prefix** parameter with custom value.

## *Creating and using proxy users*

- You'll need an existing (for example, OS-authenticated) user who has a DBA role and another existing user (for example, **steve**)

## *Creating and using proxy users*

1. Proxy authentication is best-suited type of authentication for three-tiered environments.
2. The middle tier is represented as a proxy user in the database and this user can authenticate end-users in such a way that these end users can be audited by the database.
3. Create a user **appserver** (to be the proxy user). Grant this user only the **create session** privilege.

3. Authorize user **steve** to connect through proxy user **appserver**.

```
SQL> connect appserver[steve]
Enter password:
Connected.
SQL> show user
USER is "steve"
SQL> select sys_context('USERENV','PROXY_USER') from dual;
SYS_CONTEXT('USERENV','PROXY_USER')
-----
APP SERVER
```

## *Creating and using proxy users*

4. To see proxy users, you can query the **proxy\_users** view:

```
SQL> select * from proxy_users;  
  
PROXY    CLIENT AUT FLAGS-----  
APP SERVER steve   NO  PROXY MAY ACTIVATE ALL CLIENT ROLES
```

5. Revoke authorization from user **steve** to connect through proxy user **appserver**.

## *Creating and using proxy users*

- You can control which roles the proxy user can activate for user.
- If you want the proxy user to activate only particular roles (or no roles) for a user, you can do that by adding the **WITH ROLES <role1, role2, .., roleN>** (or **WITH NO ROLES**) clause at the end of the **alter user** statement.

# *Creating and using proxy users*

- For instance, if the user **steve** has many roles (including **usr\_role**), and you want him to have only **usr\_role** when he is connected through proxy user **appserver**:

```
SQL> alter user steve grant connect through appserver with roles usr_role;  
User altered.
```

```
SQL> connect appserver[steve]
```

```
Enter password:
```

```
Connected.
```

```
SQL> select * from session_roles;
```

```
ROLE
```

```
-----
```

```
USR_ROLE
```

```
SQL> connect steve
```

```
Enter password:
```

```
Connected.
```

```
SQL> select count(*) from session_roles;
```

```
COUNT(*)
```

```
-----
```

```
25
```

## *Creating and using proxy users*

- You can request reauthentication of a user to the database.
- This is done by using the **authentication required** clause at the end of **alter user** statement:

```
SQL> alter user steve grant connect through appserver authentication required;  
User altered.
```

## *Creating and using database roles*

- You will need an existing (for example, OS-authenticated) user that has a dba role and another three existing users (for example, **steve**, **tom**, and **jessica**).
- It is assumed that sample schemas are installed.

## *Creating and using database roles*

1. Use OS authentication to connect to the database.
2. Grant system privileges and object privileges to the role **usr\_role**.
3. Practice using database roles; grant the following:
  - A role to another role
  - System and object privileges to role
  - Roles to users
4. Revoke privileges and roles by using a **revoke** statement.  
For example:

```
SQL> revoke usr_role from steve;
```

# *Creating and using database roles*

```
SQL> grant role1 to role2;  
Grant succeeded.
```

```
SQL> grant role2 to role1;  
grant role2 to role1  
*  
ERROR at line 1: ORA-01934: circular role grant detected
```

## *Creating and using database roles*

- Suppose that user **steve** grants object privilege to user **jessica** with a grant option and user **jessica** grants that privilege to user **tom**.
- If user **steve** revokes that privilege from **jessica**, it will be automatically revoked from **tom**.

# *Creating and using database roles*

```
SQL> grant select on hr.employees to jessica with grant option;
```

```
Grant succeeded.
```

```
SQL> connect jessica
```

```
Enter password:
```

```
Connected.
```

```
SQL> grant select on hr.employees to tom;
```

```
Grant succeeded.
```

```
SQL> connect tom/oracle_123
```

```
Connected.
```

```
SQL> select count(*) from hr.employees;
```

```
COUNT(*)
```

```
-----  
107
```

```
SQL> connect steve/welcome1
```

```
Connected.
```

```
SQL> revoke select on hr.employees from jessica;
```

```
Revoke succeeded.
```

```
SQL> connect tom/oracle_123
```

```
Connected.
```

```
SQL> select count(*) from hr.employees;
```

```
select count(*) from hr.employees
```

```
*
```

```
ERROR at line 1:
```

```
ORA-00942: table or view does not exist
```

# The sysbackup privilege – how, when, and why should you use it?

- You'll need:

- An existing database user (for example, tom) and a password file in 19c format, if you want to complete it using a password-authenticated user
- An existing OS user (for example, john), who belongs to the backupdba OS group, in order to connect to the database using OS authentication

## The sysbackup privilege – how, when, and why should you use it?

- You can use either Oracle Recovery Manager (RMAN) or SQL\*Plus to perform the operations.
- When you connect to the database as **sysbackup**, you are connected as a predefined user **sysbackup**. If you want to check this, run the following statement:

```
SQL> select user from dual;
```

- Otherwise, the following statement:

```
SQL> show user
```

## The sysbackup privilege – how, when, and why should you use it?

- Using the **sysbackup** privilege, you can connect to the database even when it is not open. This privilege enables better **separation of duties** and the implementation of the **least privilege principle**.

## The sysbackup privilege – how, when, and why should you use it?

- To view the list of privileges a user can exercise when connected to the database using **sysbackup** privilege, you can create a user (for example, **tom**) and grant the user only **sysbackup** privileges.
- The next step is to connect to the database as user **tom**, using the **sysbackup** privilege and the **execute** statement:

```
select * from session_privs;
```

# The sysbackup privilege – how, when, and why should you use it?

- These privileges are shown in the following table:

Privileges (output from the previous statement)			
sysbackup	select any transaction	select any dictionary	resumable
create any directory	alter database	audit any	create any cluster
create any table	unlimited tablespace	drop tablespace	alter tablespace
alter session	alter system		

# The sysbackup privilege – how, when, and why should you use it?

- This is how you can check **enabled** roles:

```
SQL> select * from session_roles;
```

```
ROLE
```

```
-----  
SELECT_CATALOG_ROLE  
HS_ADMIN_SELECT_ROLE
```

*HS\_ADMIN\_SELECT\_ROLE* is granted to *SELECT\_CATALOG\_ROLE*.

If you want to view the roles and privileges granted to **sysbackup**, you can query **DBA\_ROLE\_PRIVS** and **DBA\_SYS\_PRIVS**:

```
SQL> select * from dba_role_privs where grantee='SYSBACKUP';  
SQL> select * from dba_sys_privs where grantee='SYSBACKUP';
```

# The sysbackup privilege – how, when, and why should you use it?

- Also, this new administrative privilege enables you to select, insert, delete, execute, and perform operations:

SELECT	PERFORM operations
X\$ tables	STARTUP, SHUTDOWN
V\$ and GV\$ views	CREATE PFILE, CREATE SPFILE
APPQOSSYS.WLM_CLASSIFIER_PLAN	CREATE CONTROLFILE
SYSTEM.LOGSTDBY\$PARAMETERS	FLASHBACK DATABASE
INSERT/DELETE	DROP DATABASE
SYS.APPLY\$_SOURCE_SCHEMA	CREATE/DROP RESTORE POINT (including GUARANTEED restore points)
SYSTEM.LOGSTDBY\$PARAMETERS	
EXECUTE	
SYS.DBMS_BACKUP_RESTORE	SYS.DBMS_DATAPUMP
SYS.DBMS_RCMAN	SYS.DBMS_IR
SYS.DBMS_PIPE	SYS.SYS_ERROR
SYS.DBMS_TTS	SYS.DBMS_TDB
SYS.DBMS_PLUGTS	SYS.DBMS_PLUGTSP

## The sysbackup privilege – how, when, and why should you use it?

- You can't drop user **sysbackup**.
- In a multitenant environment, you can restrict a user to be able to perform backups only for the PDB it can connect to.
- You can accomplish that by creating a local user in the PDB and granting the **sysbackup** privilege to the user.

# The sysbackup privilege – how, when, and why should you use it?

- When you are connected to the database as the **sysbackup**, you are connected as **sysbackup** user to **SYS** schema:

```
SQL> connect / as sysbackup  
Connected.
```

```
SQL> show user  
USER is "SYSBACKUP"
```

```
SQL> select sys_context( 'userenv', 'current_schema' ) from dual;  
SYS_CONTEXT('USERENV','CURRENT_SCHEMA')-----SYS
```

# *The syskm privilege – how, when, and why should you use it?*

- You'll need:

- An existing database user (for example, jessica) and a password file in the 19c format, if you want to complete it using a password-authenticated user
- An existing OS user (for example, bob), who belongs to the kmdba OS group, in order to connect to the database using OS authentication

## *The syskm privilege – how, when, and why should you use it?*

1. When you connect to the database as **syskm**, you are connected as a predefined user, **syskm**.
2. In most circumstances when using TDE, you don't have to have **syskm** administrative privilege.
3. You can perform operations related to managing the TDE keystore.

# The syskm privilege – how, when, and why should you use it?

- Privileges you can use when connected as **syskm**:

- ADMINISTER KEY MANAGEMENT
- CREATE SESSION
- SELECT on V\$ (and GV\$) views:
  - SYS.V\$ENCRYPTED\_TABLESPACES
  - SYS.V\$ENCRYPTION\_WALLET
  - SYS.V\$WALLET
  - SYS.V\$ENCRYPTION\_KEYS
  - SYS.V\$CLIENT\_SECRETS
  - SYS.DBA\_ENCRYPTION\_KEY\_USAGE
  - SYS DATABASE KEY INFO

# The syskm privilege – how, when, and why should you use it?

- You can't drop user **syskm**.
- When you are connected to the database as **syskm**, you are connected as the **syskm** user to **SYS** schema:

```
SQL> connect / as syskm  
Connected.
```

```
SQL> show user  
USER is "SYSKM"
```

```
SQL> select sys_context( 'userenv', 'current_schema' ) from dual;  
SYS_CONTEXT('USERENV','CURRENT_SCHEMA')  
-----  
SYS
```

# *The sysdg privilege – how, when, and why should you use it?*

- You'll need:

- An existing database user (for example, steve) and a password file in the 19c format if you want to complete it using a password-authenticated user
- An existing OS user (for example, kelly), who belongs to the dgdba OS group in order to connect to the database using OS authentication

## *The sysdg privilege – how, when, and why should you use it?*

1. When you connect to the database as **sysdg**, you are connected as a predefined user, **sysdg**.
  
2. User can grant/revoke sysdg privilege to/from another existing user.

# The sysdg privilege – how, when, and why should you use it?

3. After you connect to the database using the **sysdg** administrative privilege, you can perform the following operations:

Operations	
STARTUP, SHUTDOWN	CREATE SESSION
ALTER SESSION	SELECT ANY DICTIONARY
ALTER DATABASE	FLASHBACK DATABASE
ALTER SYSTEM	EXECUTE SYS.DBMS_DRS
CREATE/DROP RESTORE POINT (including GUARANTEED restore points)	SELECT X\$ tables, V\$ and GV\$ views
DELETE APPQOSSYS.WLM_CLASSIFIER_PL AN	SELECT APPQOSSYS.WLM_CLASSIFIER_PL AN

# The sysdg privilege – how, when, and why should you use it?

- You can't drop user **sysdg**.
- When you are connected to the database as **sysdg**, you are connected as **sysdg** user to the **SYS** schema:

```
SQL> connect / as sysdg  
Connected.
```

```
SQL> show user  
USER is "SYSDG"
```

```
SQL> select sys_context( 'userenv', 'current_schema' ) from dual;  
SYS_CONTEXT('USERENV','CURRENT_SCHEMA')
```

---

```
SYS
```



# Oracle Database 19c Security

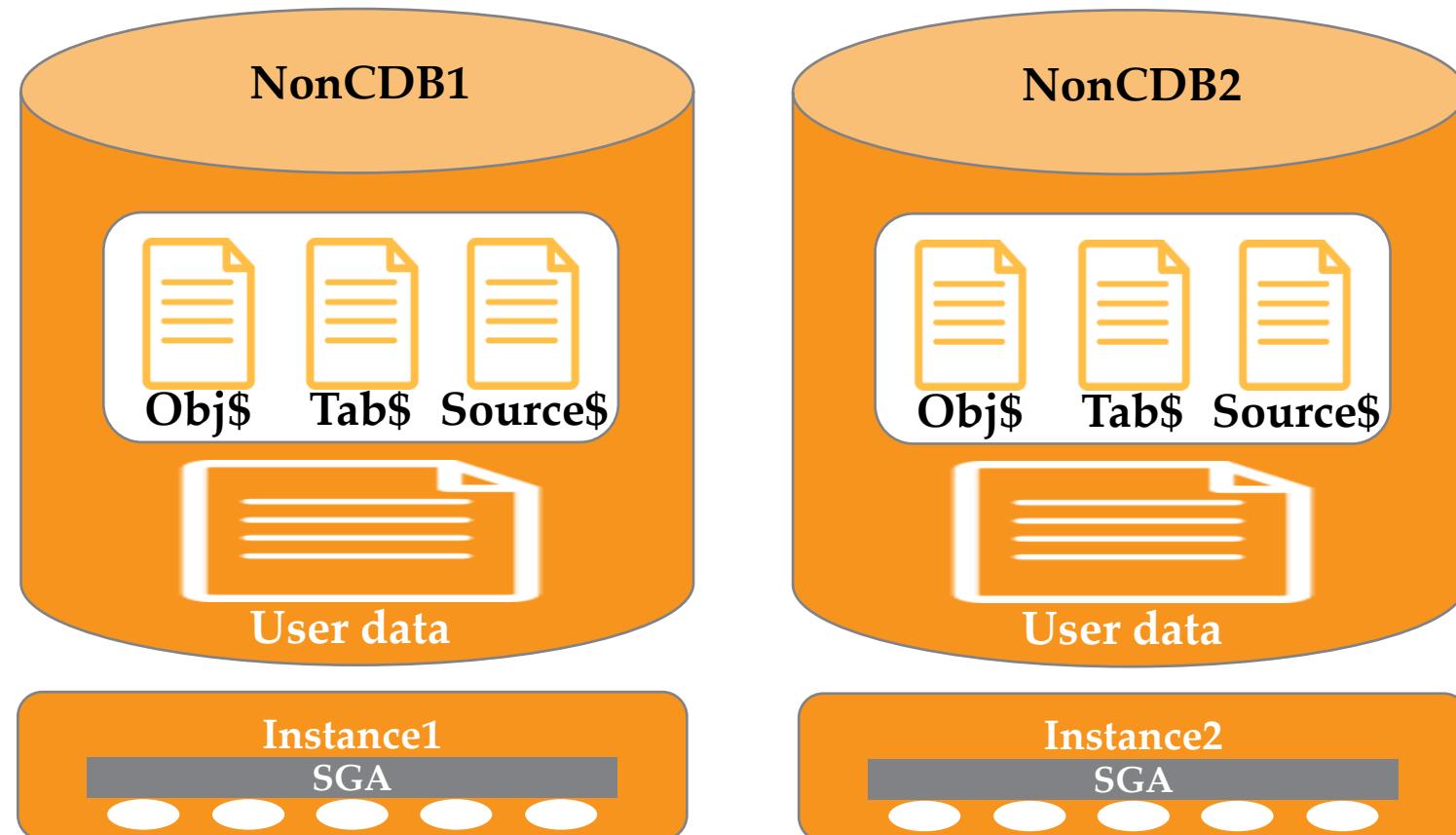
*Security Considerations in Multitenant Environment*

- We will cover the following:

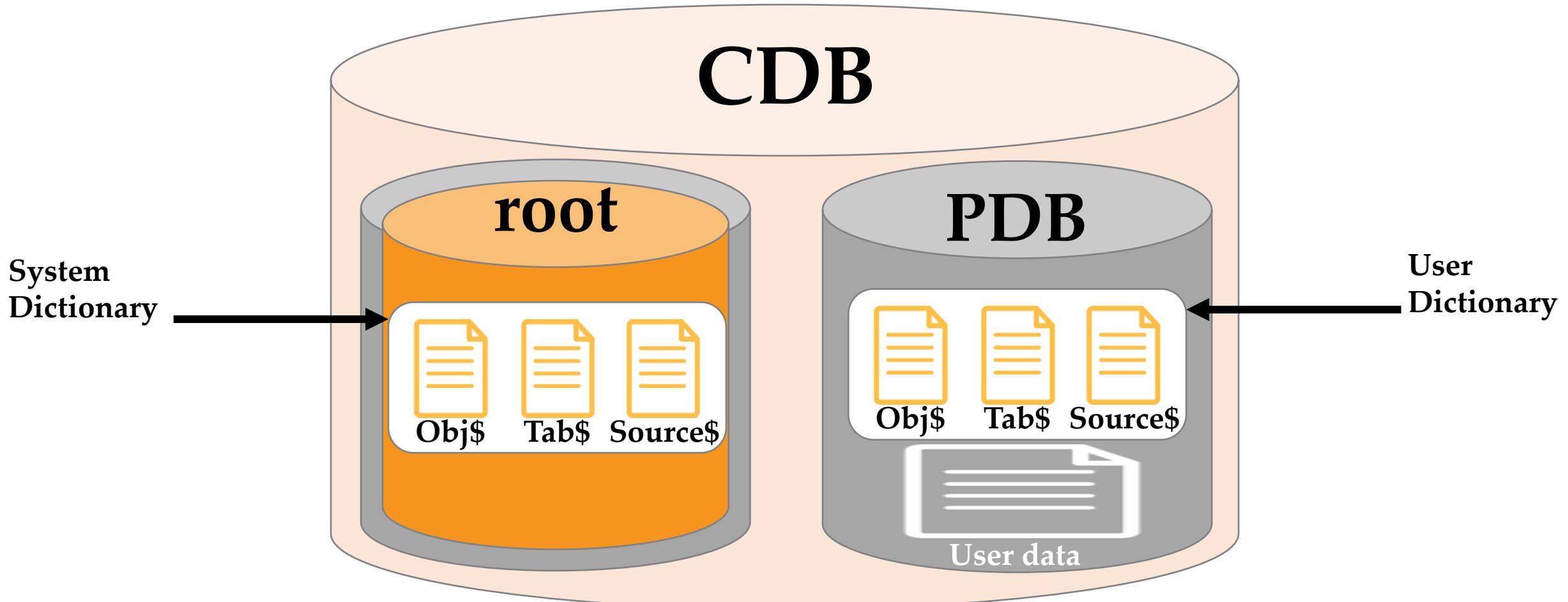
- Creating a common user
- Creating a local user
- Creating a common role
- Creating a local role
- Granting privileges commonly
- Granting privileges locally
- The effects of plugging/unplugging operations on users, roles, and privileges

- The Oracle multitenant environment is a new architecture of Oracle Database, introduced in version 19c .
- This chapter is focused on some of the security considerations concerning common and local users, roles, and privileges.

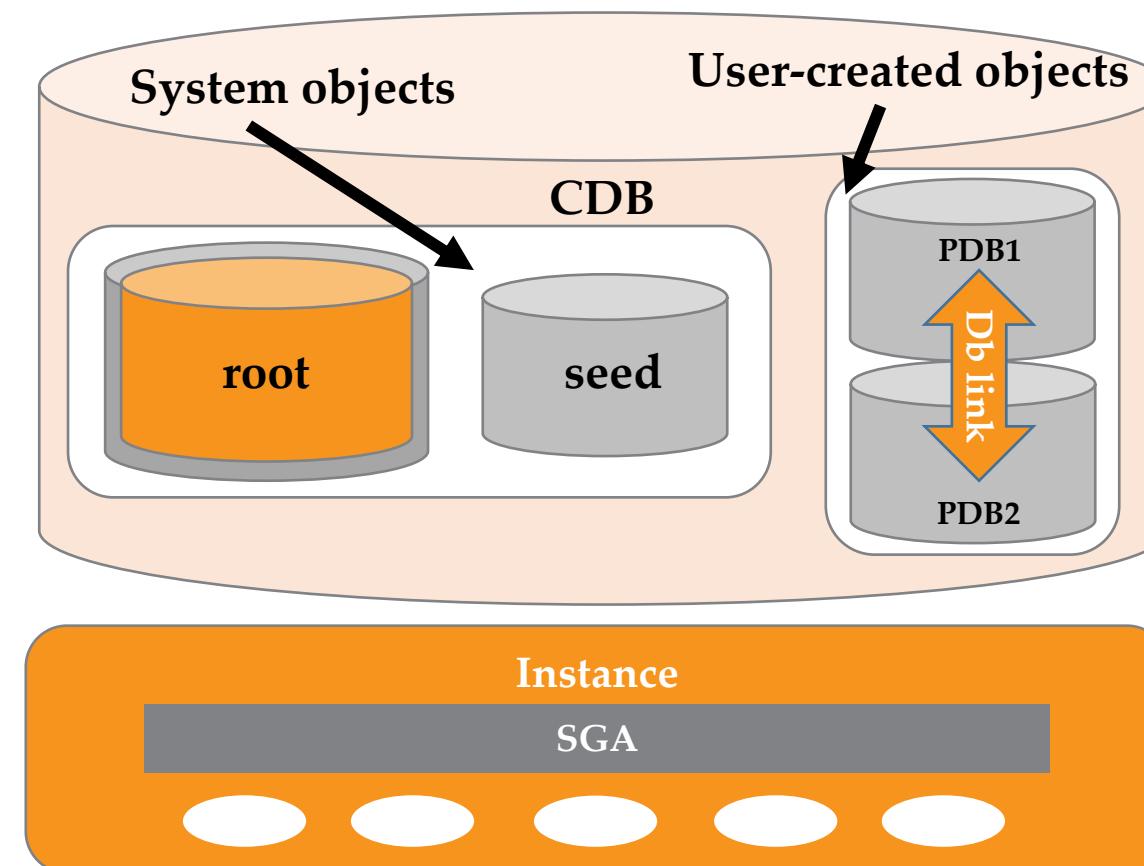
- Traditional architecture of Oracle Database.



- Data Dictionary separation



- A multitenant architecture

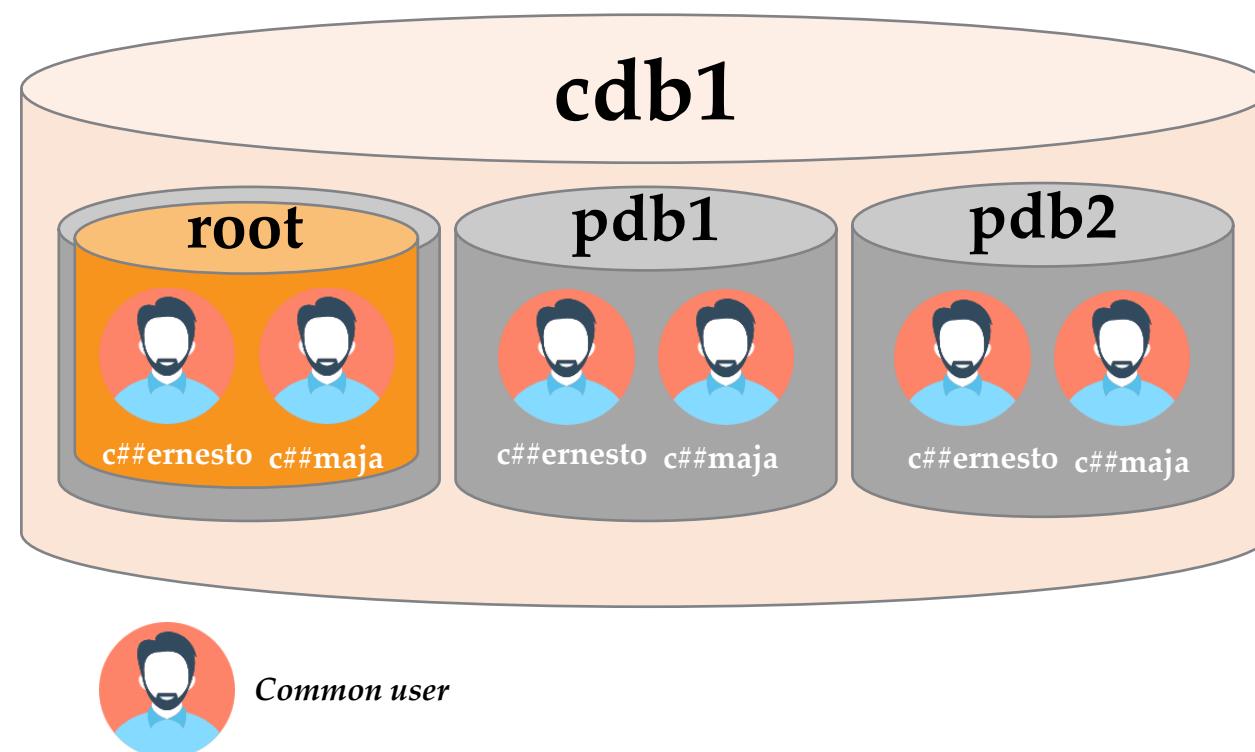


## *Creating a common user*

- A **common user** is a user created in the root container, which has the same identity across all containers.
- The main purpose of a common user is to perform "infrastructure" administrative tasks.
- There are two types of common users: Oracle-supplied (for example, **SYS** and **SYSTEM**) and user-created common users.
- You'll need an existing common user who has **create user** privilege granted commonly.

# *Creating a common user*

- `c##maja` is actually not a single user, but each container has a user named `c##maja` and the passwords must be the same.



## *Rules/guidelines for creating and managing common users*

- There are a few rules you should be aware of:

- The name of a common user must be unique across all containers. In version 12.1.0.1, it must begin with c## or C## unless you change the internal parameter common\_user\_prefix.
- A common user can have different privileges in different containers.
- The schemas for a common user may contain different objects in different containers.

## *Creating a common user*

- The column **oracle\_maintained** (in **DBA\_USERS**) provides information as to whether a user is created and maintained by Oracle-supplied scripts:

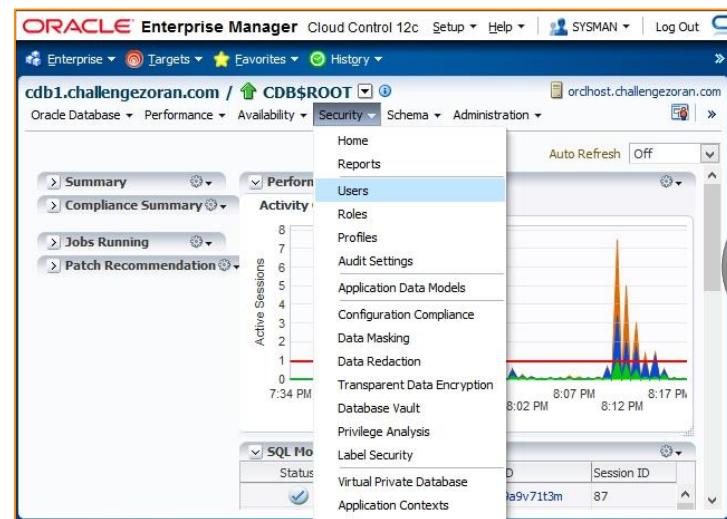
```
c##ernesto@CDB1> select username, oracle_maintained from dba_users where username='SYSTEM'  
or username='C##ernesto';
```

USERNAME	ORACLE_MAINTAINED
-----	-
SYSTEM	Y
C##ernesto	N

# How to create a common user using OEM 19c

From the **Databases** page, select the root database in which you want to create a common user.

- 1 Start OEM 19c and log in using user **SYSMAN** or **SYSTEM**.
- 2
- 3 From the **Administration** menu, select **Security** and then **Users**



# How to create a common user using OEM 19c

The screenshot shows the 'Database Login' dialog box. The 'Database' dropdown is set to 'CDB\$ROOT'. Under 'Credential', the 'Named' radio button is selected, while 'New' is highlighted with a red box. The 'Username' field contains 'c##zoran' and the 'Password' field contains '\*\*\*\*\*'. The 'Role' dropdown is set to 'Normal'. A 'Save As' checkbox is unchecked. At the bottom are 'Login' and 'Cancel' buttons.

4

If prompted, log in to the root as a common user who has a **create user** privilege

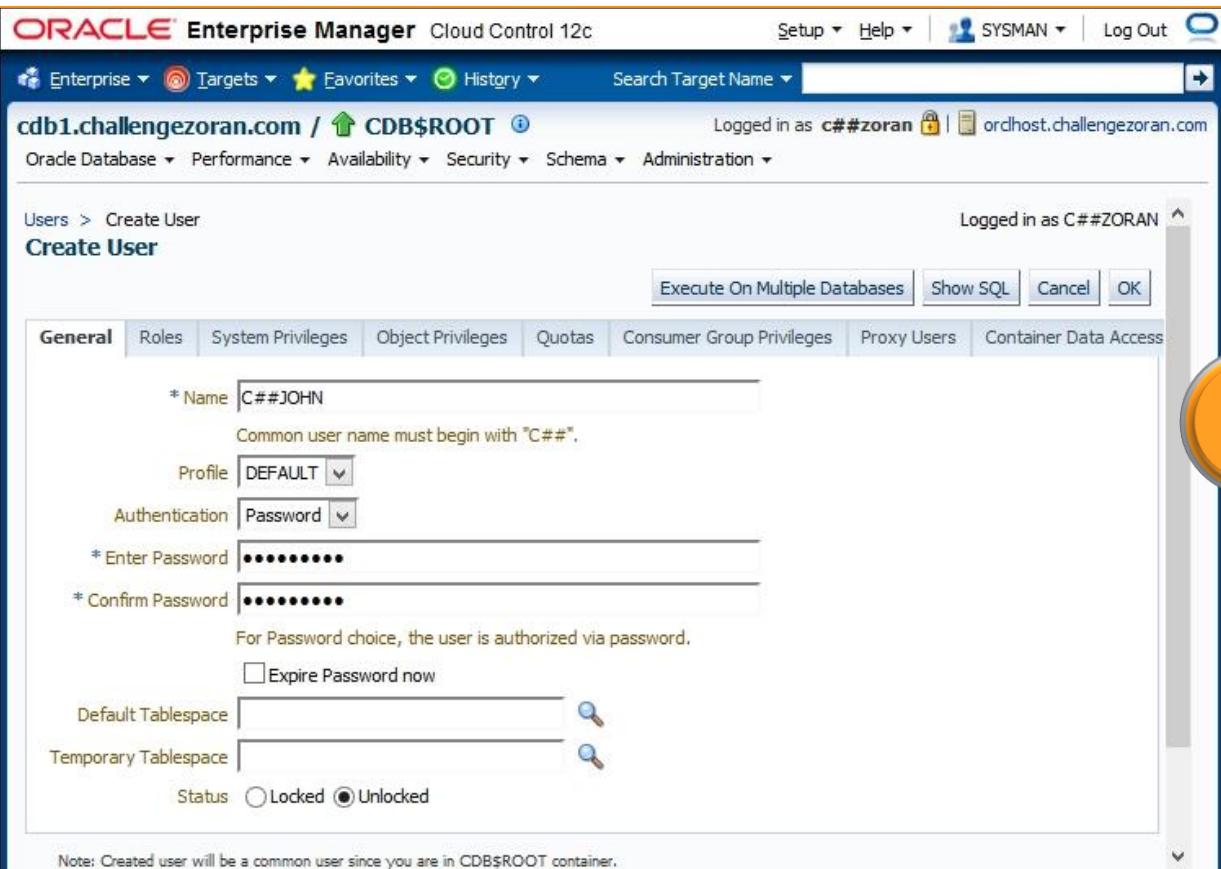
The screenshot shows the 'Users' page in OEM. The top status bar indicates 'Logged in as c##zoran'. The 'Object Type' dropdown is set to 'User'. On the right, there is a 'Create' button highlighted with a red box. Below it is a search bar with 'Object Name' and a 'Go' button. The main area displays a table of users, with one row selected for 'ANONYMOUS'. The table columns include Select, UserName, Account Status, Expiration Date, Default Tablespace, Temporary Tablespace, Profile, and Created.

Select	UserName	Account Status	Expiration Date	Default Tablespace	Temporary Tablespace	Profile	Created
<input checked="" type="radio"/>	ANONYMOUS	EXPIRED & LOCKED	May 24, 2013 1:20:02 PM CEST	SYSAUX	TEMP	DEFAULT	May 24, 2013 12:06:10 PM CEST

Click on the **Create** button

5

# How to create a common user using OEM 19c



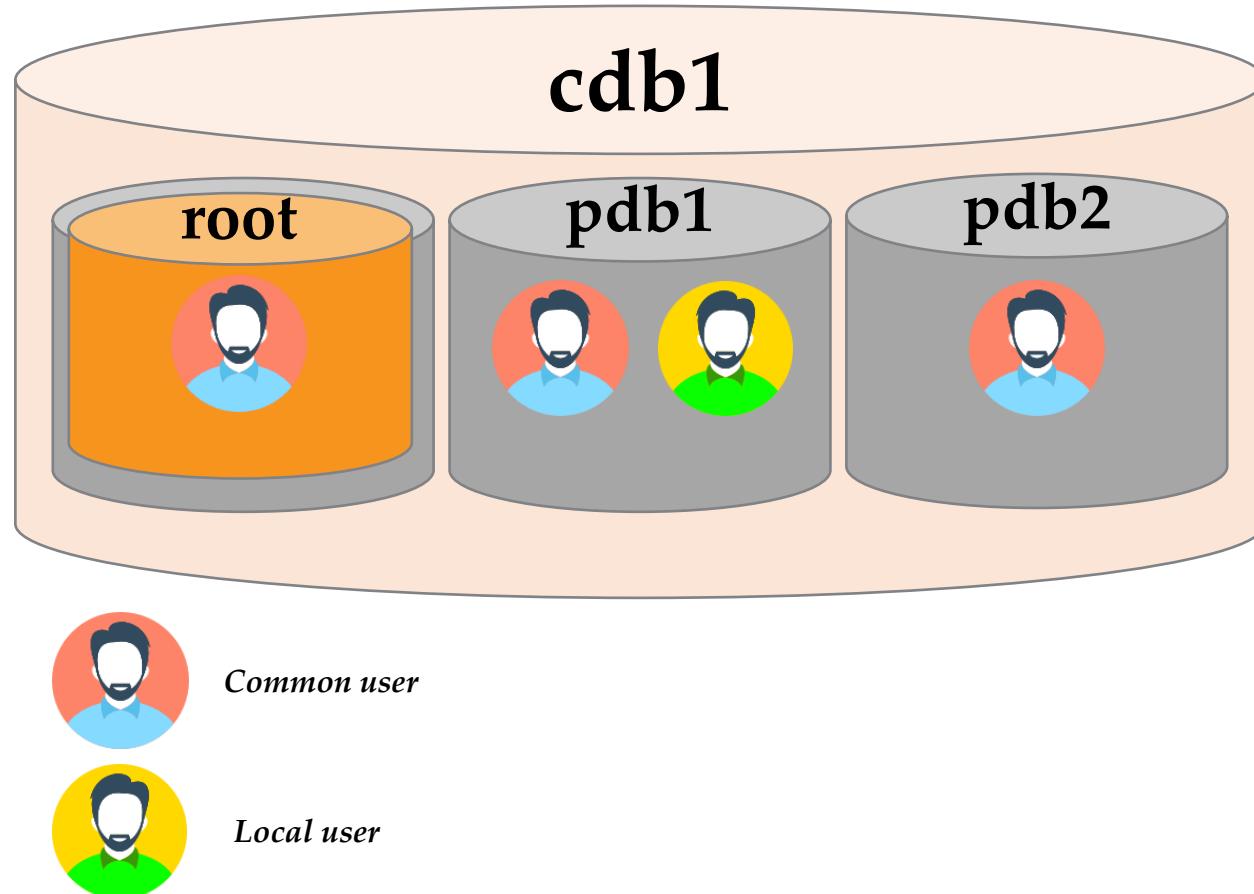
4

To create a common user, it is enough to fill out the following fields on the **General** tab: **Name**, **Enter Password**, and **Confirm Password** and then click on the **OK** button:

## *Creating a local user*

- A **local user** is a user that is created and that exists in only one PDB. A local user can't be created in the root container.
- You'll need an existing user (either common or local) who has **create user** privilege in that pluggable database.

# *Creating a local user*



## Rules/guidelines for creating and managing local users

- There are a few rules you should be aware of:

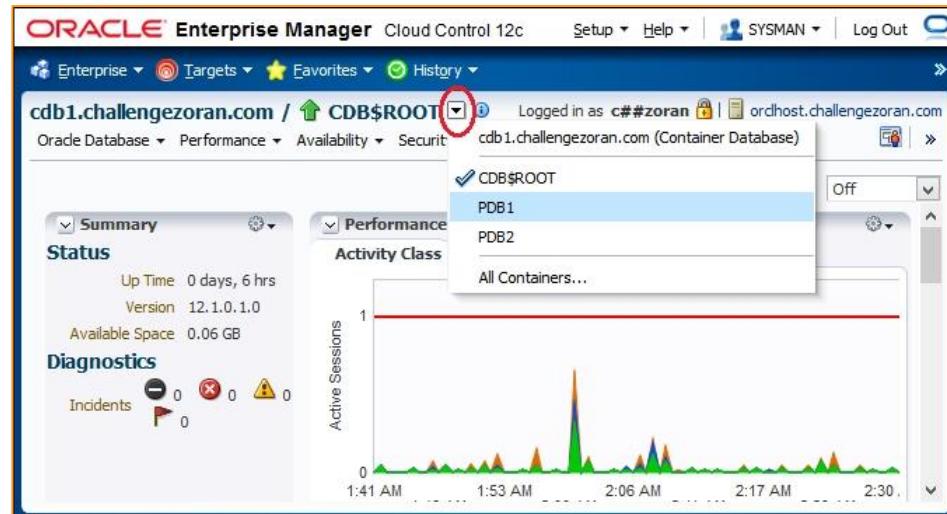
- The name of a local user must be unique within its pluggable database and it must not begin with c## or C##
- A local user cannot be created in the root
- A local user exists in one and only one PDB and owns a schema in that PDB

# How to create a local user using OEM 19c

From the **Databases** page, select the **pluggable database** in which you want to create a common user.

1 Start OEM 19c and log in using user **SYSMAN** or **SYSTEM**.

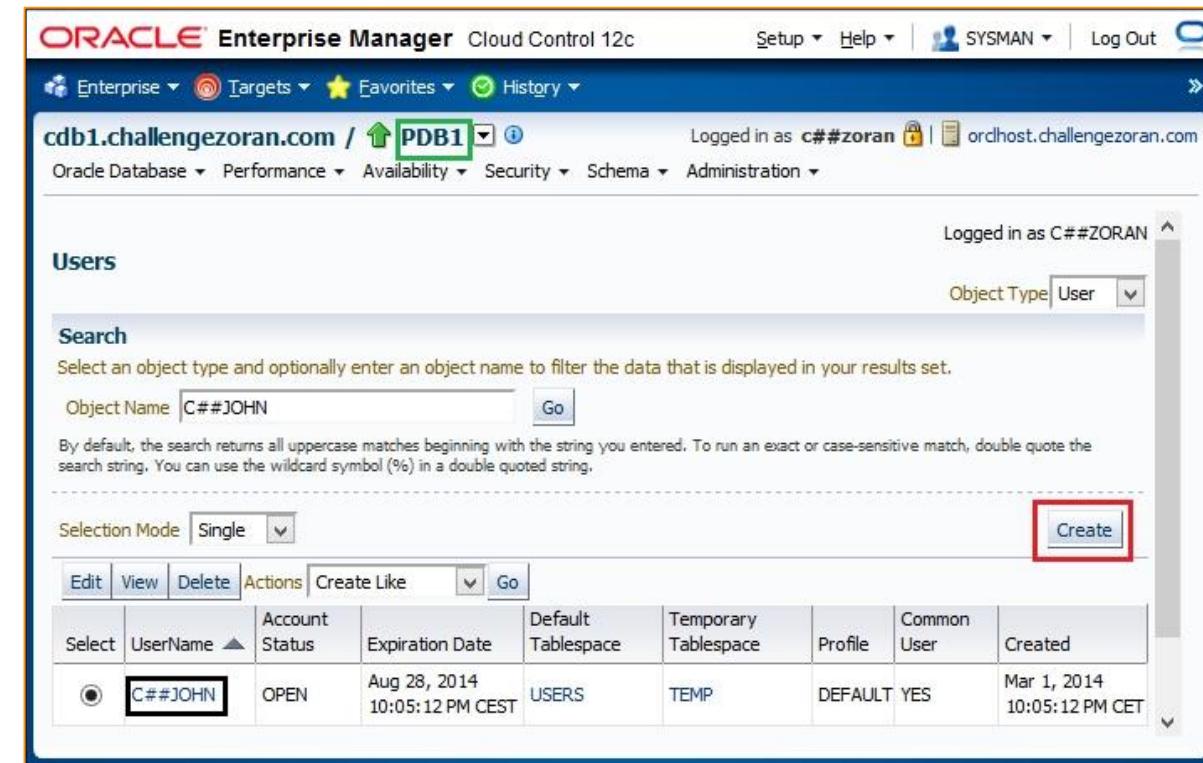
2



3

If you want to switch container, you should click on **Container Switcher** and a drop-down menu will open

# How to create a local user using OEM 19c



The screenshot shows the Oracle Enterprise Manager interface for a CDB. The top navigation bar includes 'Enterprise Manager' and 'Cloud Control 12c'. The main title is 'cdb1.challengezoran.com / PDB1'. The user is logged in as 'c##zoran'. The 'Users' section is displayed, with a search bar containing 'C##JOHN'. A table lists a single user entry: 'C##JOHN' (Status: OPEN, Expiration Date: Aug 28, 2014, Default Tablespace: USERS, Temporary Tablespace: TEMP, Profile: DEFAULT, Common User: YES, Created: Mar 1, 2014). A red box highlights the 'Create' button in the toolbar above the table.

Select	UserName	Status	Expiration Date	Default Tablespace	Temporary Tablespace	Profile	Common User	Created
<input checked="" type="radio"/>	C##JOHN	OPEN	Aug 28, 2014 10:05:12 PM CEST	USERS	TEMP	DEFAULT	YES	Mar 1, 2014 10:05:12 PM CET

4

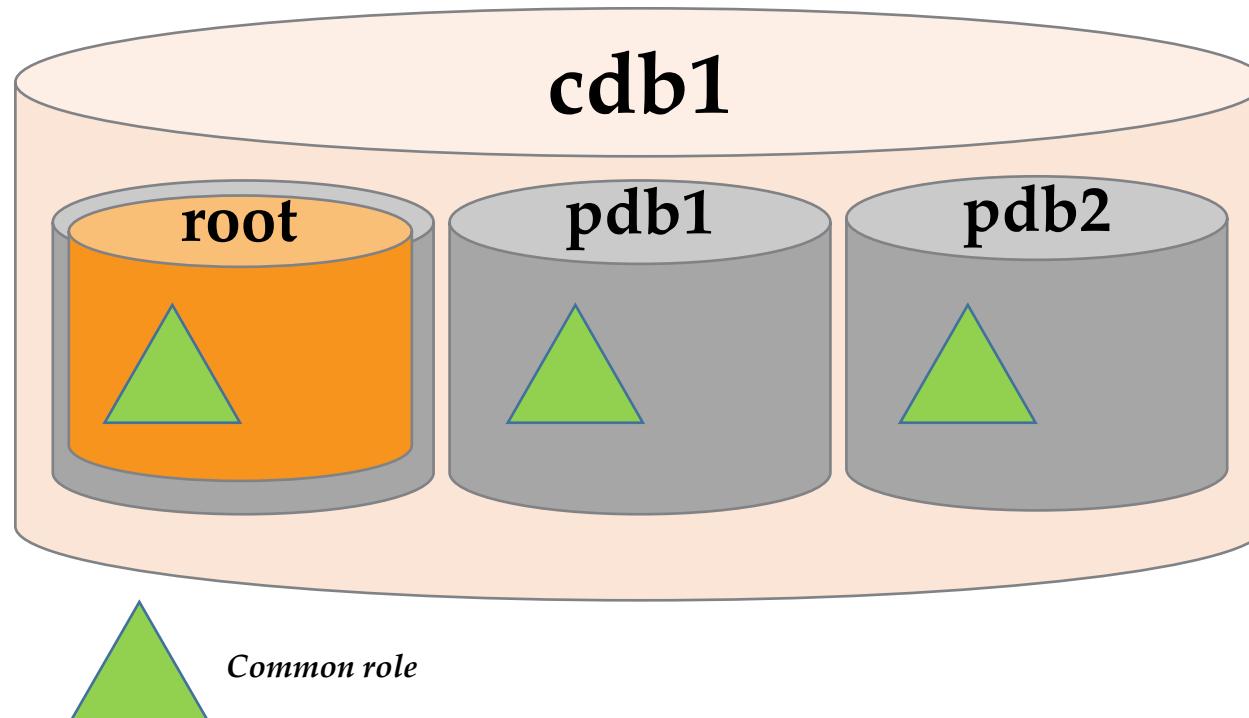
The common user is created in all pluggable databases that reside in the CDB and will be created in all future PDBs. By clicking on the **Create** button, you can create (only) a local user:

## *Creating a common role*

- **Common roles** are roles created in the root container and they exist in all containers.
- These roles can have a different set of privileges in different containers and they can be granted to either common or local users or roles.
- You'll need an existing common user who has **create role** privilege granted commonly.

## *Creating a common role*

- When you create a common role, that role exists in all containers in that database.

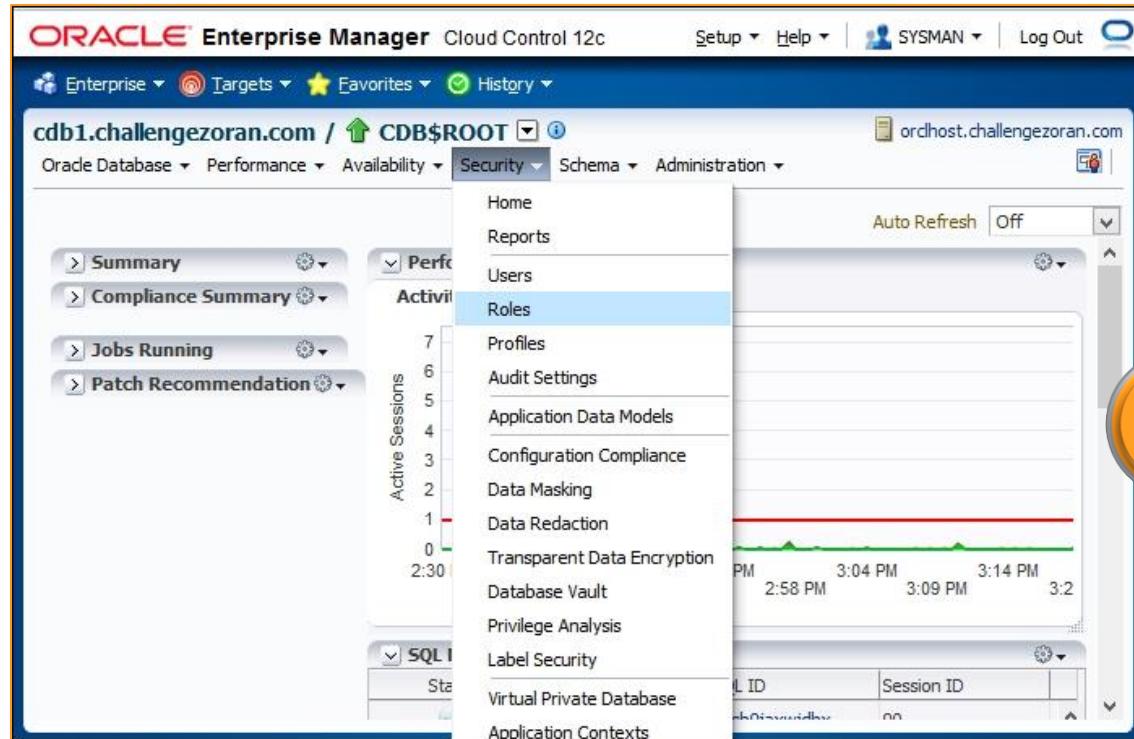


# *Creating a common role*

```
c##ernesto@CDB1> select * from dba_roles where role='C##ROLE1';
ROLE          PASSWORD AUTHENTICAT COM O
-----
C##ROLE1      NO      NONE      YES N
c##ernesto@CDB1> connect c##ernesto/oracle@pdb1
Connected.
c##ernesto@PDB1> select * from dba_roles where role='C##ROLE1';
ROLE          PASSWORD AUTHENTICAT COM O
-----
C##ROLE1      NO      NONE      YES N
c##ernesto@PDB1> connect c##ernesto/oracle@pdb2
Connected.
c##ernesto@PDB2> select * from dba_roles where role='C##ROLE1';
ROLE          PASSWORD AUTHENTICAT COM O
-----
C##ROLE1      NO      NONE      YES N
```

# How to create a common role using OEM 19c

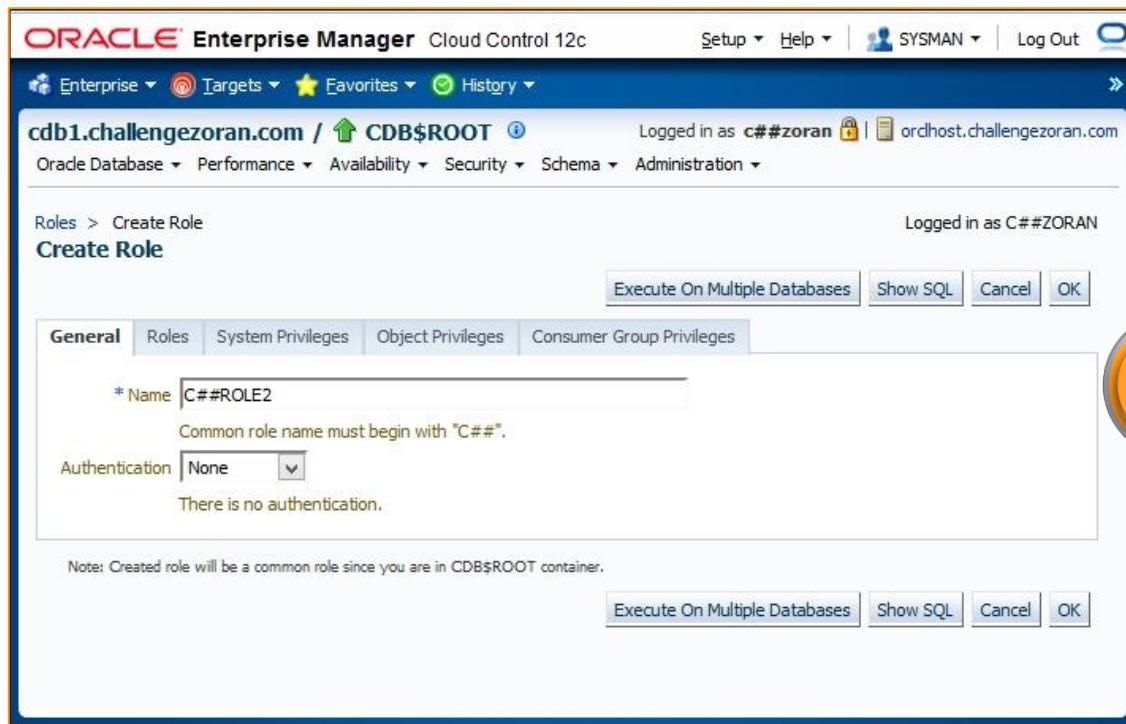
- You should connect to the root (**CDB\$ROOT**) as a common user who has **create role** privilege granted commonly.



1

From the **Administration** menu,  
select **Security** (drop-down menu) and  
then **Roles**

# How to create a common role using OEM 19c



2

On the **Roles** page, click on the **Create** button and the **Create Role** page appears

On the **Create Role** page, you name the role on the **General** tab. Also, you may grant other roles and privileges to the role. Click on the **OK** button to create it.

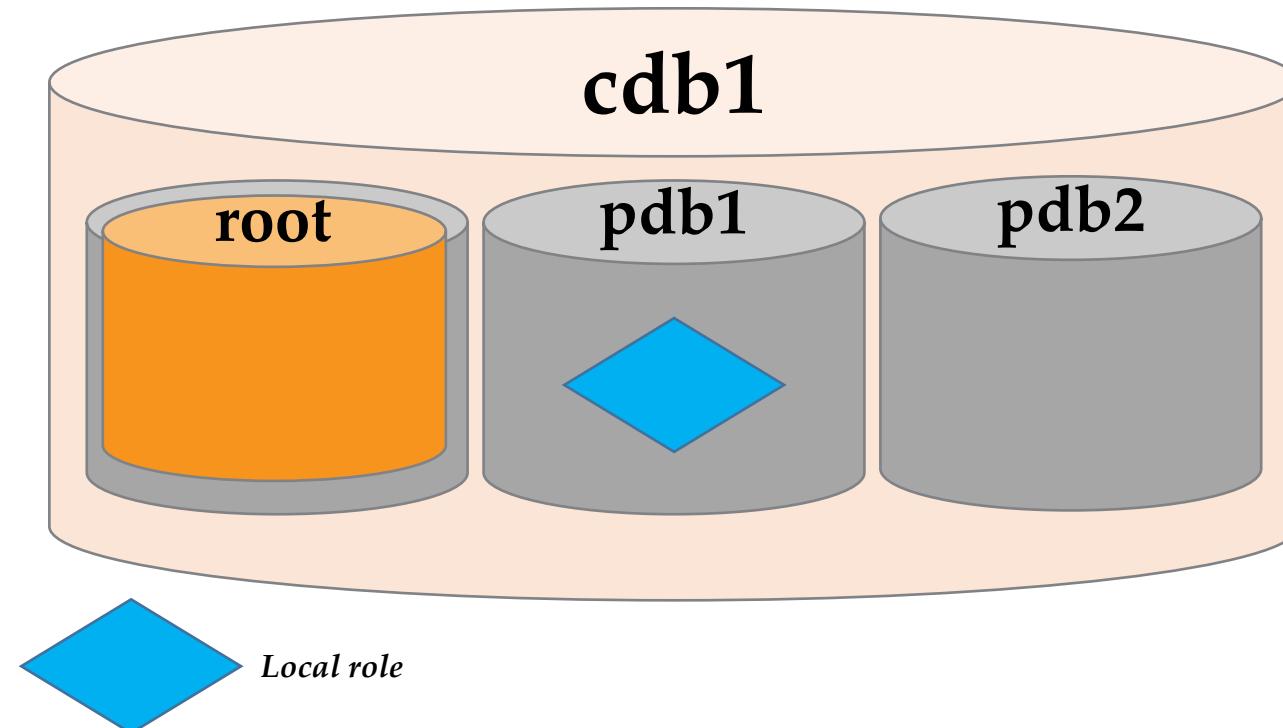
3

## *Creating a local role*

- **Local roles** are roles created in PDB and they exist only in that PDB. These roles can be granted **only locally** to either common or local users or roles.
- You'll need a pluggable database open. And an existing user (either common or local) who has **create role** privilege in that pluggable database.

## *Creating a local role*

- When you create a local role, that role exists only in the pluggable database in which it is created.



# *Creating a local role*

```
c##maja@CDB1> select * from dba_roles where role='LOCAL_ROLE1';
no rows selected
c##maja@CDB1> connect c##maja/oracle@pdb1
Connected.
c##maja@PDB1> select * from dba_roles where role='LOCAL_ROLE1';
ROLE          PASSWORD AUTHENTICAT COM O
-----
LOCAL_ROLE1      NO      NONE      NO  N
c##maja@PDB1> connect c##maja/oracle@pdb2
Connected.
c##maja@PDB2> select * from dba_roles where role='LOCAL_ROLE1';
no rows selected
```

## *How to create a local role using OEM 19c*

- You should connect to PDB as a common or local user who has **create role** privilege in that PDB.
- All the remaining steps are done in the same way as in the **How to create a common role using OEM 19c** section.

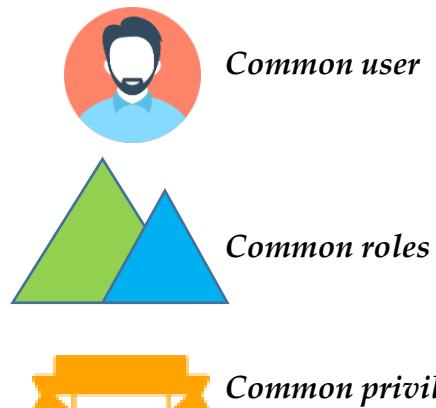
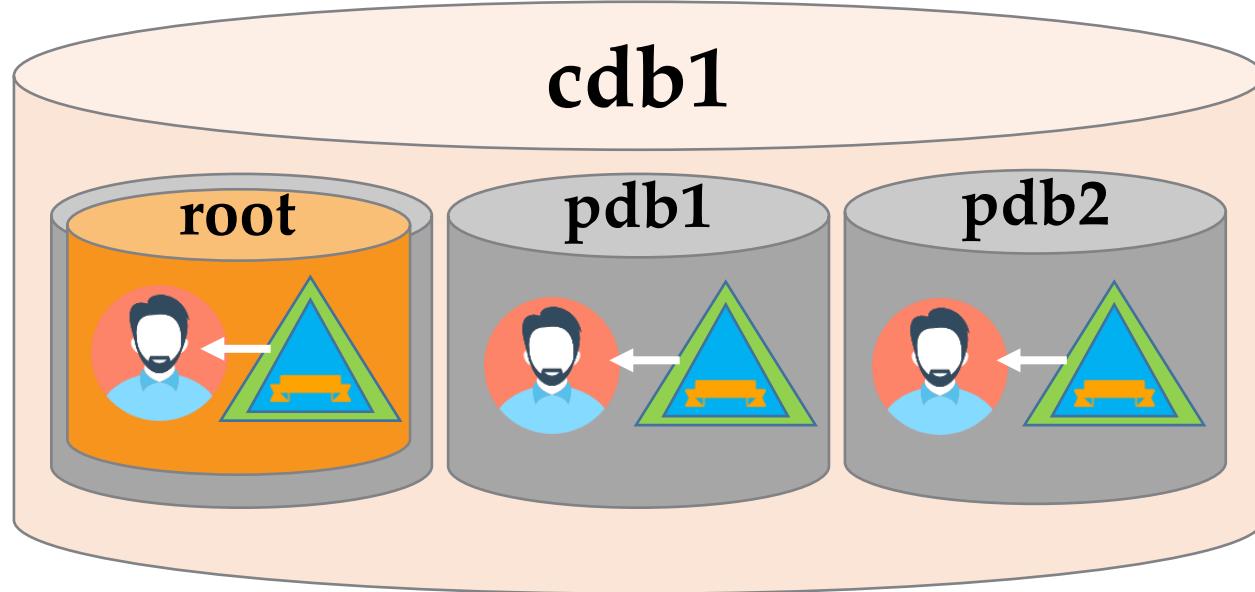
## *Granting privileges and roles commonly*

- The common privilege is a privilege that can be exercised across all containers in a container database.
- Depending only on the way it is granted, a privilege becomes common or local.
- When you grant a privilege commonly (across all containers) it becomes a common privilege.

## *Granting privileges and roles commonly*

- You will need to connect to the root container as an existing common user who is able to grant a specific privilege or existing role to another existing common user.
- You will use the following:
  - Common users `c##maja` and `c##ernesto` with the dba role granted commonly
  - Common user `c##john`
  - Common roles `c##role1` and `c##role2`

# Granting privileges and roles commonly



1. You can grant privileges or common roles commonly only to a common user. You need to connect to the root container as a common user who is able to grant a specific privilege or role.

## *Granting privileges and roles commonly*

2. System privilege **create session** is granted to the common user **c##john commonly** by adding a container=all clause to the grant statement.

```
SQL> connect c##john/oracle@cdb1
Connected.
c##john@CDB1> connect c##john/oracle@pdb1
Connected.
c##john@PDB1> connect c##john/oracle@pdb2
Connected.
c##john@PDB2>
```

# Granting privileges and roles commonly

3. System privilege **select any table** is granted to the common role **c##role1** commonly.

```
c##ernesto@CDB1> select * from role_sys_privs where role='C##ROLE1';
  ROLE          PRIVILEGE      ADM  COM
-----  -----  ---  ---
C##ROLE1      SELECT ANY TABLE      NO  YES
c##ernesto@CDB1> connect c##ernesto/oracle@pdb1
Connected.
c##ernesto@PDB1> select * from role_sys_privs where role='C##ROLE1';
  ROLE          PRIVILEGE      ADM  COM
-----  -----  ---  ---
C##ROLE1      SELECT ANY TABLE      NO  YES
c##ernesto@PDB1> connect c##ernesto/oracle@pdb2
Connected.
c##ernesto@PDB2> select * from role_sys_privs where role='C##ROLE1';
  ROLE          PRIVILEGE      ADM  COM
-----  -----  ---  ---
C##ROLE1      SELECT ANY TABLE      NO  YES
```

# Granting privileges and roles commonly

4. The common role **c##role1** is granted to another common role **c##role2** commonly.

```
c##ernesto@CDB1> select * from role_role_privs where role='C##ROLE2';
  ROLE          GRANTED_ROLE      ADM  COM
-----  -----
C##ROLE2      C##ROLE1        NO   YES

c##ernesto@CDB1> connect c##ernesto/oracle@pdb1
Connected.

c##ernesto@PDB1> select * from role_role_privs where role='C##ROLE2';
  ROLE          GRANTED_ROLE      ADM  COM
-----  -----
C##ROLE2      C##ROLE1        NO   YES

c##ernesto@PDB1> connect c##ernesto/oracle@pdb2
Connected.

c##ernesto@PDB2> select * from role_role_privs where role='C##ROLE2';
  ROLE          GRANTED_ROLE      ADM  COM
-----  -----
C##ROLE2      C##ROLE1        NO   YES
```

## Granting privileges and roles commonly

5. The common role `c##role2` is granted to the common user `c##john` commonly.
  - Consequently, the user `c##john` can use the `select any table` privilege in all containers in this container database:

```
c##john@CDB1> select count(*) from c##ernesto.t1;
  COUNT(*)
-----
  4
c##john@CDB1> connect c##john/oracle@pdb1
Connected.
c##john@PDB1> select count(*) from hr.employees;
  COUNT(*)
-----
  107
c##john@PDB1> connect c##john/oracle@pdb2
Connected.
c##john@PDB2> select count(*) from sh.sales;
  COUNT(*)
-----
  918843
```

## *Granting privileges and roles locally*

- A local privilege is a privilege than can be exercised only in a container in which it is granted.
- Depending only on the way it is granted, a privilege becomes common or local.
- When you grant privilege locally (in the current container), it becomes a local privilege.

## *Granting privileges and roles locally*

- You'll need an existing user who can grant some privileges and roles in a specific container to existing users and roles.
- Local grants are valid only in the current container even though the granted user (or role) is common.
- Consequently, common users and common roles can have a different set of privileges in different containers.

## *Effects of plugging/unplugging operations on users, roles, and privileges*

- The purpose is to show what is going to happen to users, roles, and privileges when you unplug a pluggable database from one container database and plug it into some other container database.
- You will need the following:

- Two container databases (**cdb1** and **cdb2**)
- One pluggable database (**pdb1**) in the container database **cdb1**
- Local user **steve** in the pluggable database **pdb1** with the local create session privilege
- The common user **c##john** with the create session common privilege and create synonym local privilege on the pluggable database **pdb1**

## Effects of plugging/unplugging operations on users, roles, and privileges

1. Unplug **pdb1** from **cdb1** and plugged it into **cdb2**.

The following is how you try to connect to **pdb1** as a local user:

```
SQL> connect steve@pdb1
```

2. All local privileges are migrated even if they are granted to common users/roles.

However, if you try to connect to **pdb1** as a previously created common user, **c##john**, you'll get an error, as follows:

```
SQL> connect c##john@pdb1
ERROR:
ORA-28000: the account is locked
Warning: You are no longer connected to ORACLE.
```

## Effects of plugging/unplugging operations on users, roles, and privileges

3. This happened because, after migration, common users are migrated in a pluggable database as locked accounts.
- You can continue to use objects in these users' schemas, or you can create these users in a root container of a new CDB.
  - To do this, we first need to close **pdb1**.

```
sys@CDB2> alter pluggable database pdb1 close;
Pluggable database altered.

sys@CDB2> create user c##john identified by oracle container=all;
User created.

sys@CDB2> alter pluggable database pdb1 open;
Pluggable database altered.
```

## Effects of plugging/unplugging operations on users, roles, and privileges

4. If we try to connect to **pdb1** as the user **c##john**, we will get the following error:

```
SQL> conn c##john/oracle@pdb1
ERROR:
ORA-01045: user C##JOHN lacks CREATE SESSION privilege; logon denied
Warning: You are no longer connected to ORACLE.
```

5. Even though **c##john** had the **create session** common privilege in **cdb1**, he cannot connect to the migrated PDB.
  - So, we need to give the **create session** privilege (either common or local) to the user **c##john**, as follows:

```
sys@cDB2> grant create session to c##john container=all;
Grant succeeded.
```

## Effects of plugging/unplugging operations on users, roles, and privileges

6. We grant a **create synonym** local privilege to a user, **c##john**.

- Let's try this privilege on the migrated **pdb2**:

```
c##john@PDB1> create synonym emp for hr.employees;  
Synonym created.
```

- This proves that local privileges are always migrated.



# Oracle Database 19c Security

*19c Security New Features*

## 19c Security New Features

- New ALTER SYSTEM Clause FLUSH PASSWORDFILE\_METADATA\_CACHE
- Transparent Online Conversion Support for Auto-Renaming in Non-Oracle-Managed Files Mode
- Support for Additional Algorithms for Offline Tablespace Encryption
- Key Management of Encrypted Oracle-Managed Tablespaces in Transparent Data Encryption

- Support for Host Name-Based Partial DN Matching for Host Certificates
- New PDB\_GUID Audit Record Field for SYSLOG and the Windows Event Viewer
- New EVENT\_TIMESTAMP\_UTC Column in the UNIFIED\_AUDIT\_TRAIL View
- Passwords Removed from Oracle Database Accounts
- Signature-Based Security for LOB Locators

- Unified Auditing Top-Level Statements
- Privilege Analysis Now Available in Oracle Database Enterprise Edition
- Support for Oracle Native Encryption and SSL Authentication for Different Users Concurrently
- Ability to Grant or Revoke Administrative Privileges to and from Schema-Only Accounts

- Automatic Support for Both SASL and Non-SASL Active Directory Connections
- Database Vault Operations Control for Infrastructure Database Administrators
- Database Vault Command Rule Support for Unified Audit Policies

- ALTER SYSTEM Clause FLUSH  
PASSWORDFILE\_METADATA\_CACHE
  - Refreshes the metadata cache with the latest details of the database password file.
  - V\$PASSWORDFILE\_INFO contains the latest details of the database password file.
  - Useful when the database password file name or location is changed, and the metadata cache needs to be refreshed with the details of the updated database password file.

- Transparent Online Conversion Support for Auto-Renaming in Non-Oracle-Managed Files Mode
  - Transparent Data Encryption online conversion in non-Oracle-managed files mode, you are no longer forced to include the FILE\_NAME\_CONVERT clause in the ADMINISTER KEY MANAGEMENT SQL statement. The file name retains its original name.

## *19c Security New Features*

- Support for Additional Algorithms for Offline Tablespace Encryption
  - Adds support for the AES192 and AES256 encryption algorithms, as well as for the ARIA, GOST, and 3DES encryption algorithms for offline tablespace encryption.

- Key Management of Encrypted Oracle-Managed Tablespaces in Transparent Data Encryption
  - A closed Transparent Data Encryption (TDE) encryption keystore has no impact on internal operations to Oracle-managed tablespaces.
  - Internal processes can access a keystore when the keystore is closed, which allows the internal process to continue and successfully
  - Closing the TDE keystore has no effect on queries of an encrypted SYSTEM, SYSAUX, TEMP, and UNDO tablespace
  - User initiated operations such as decrypt on any encrypted Oracle-managed tablespace still require the TDE keystore to be in the OPEN state.

- **Support for Host Name-Based Partial DN Matching for Host Certificates**
  - New support for partial distinguished name (DN) matching that adds the ability for the client to further verify the server certificate.
  - The client supports both full and partial DN matching. If the server DN matching is enabled, then partial DN matching is the default.
  - Partial and full DN matching for certificate verification enables more flexibility based on how the certificates were created

- **New PDB\_GUID Audit Record Field for SYSLOG and the Windows Event Viewer**
  - The audit record fields for SYSLOG and the Windows Event Viewer now include a new field, PDB\_GUID
  - In a multitenant container database (CDB) deployment, the pluggable database that generated a unified audit trail record must be identified in the audit trail. The new field captures this information starting with this release. The data type is VARCHAR2

- **New EVENT\_TIMESTAMP\_UTC Column in the UNIFIED\_AUDIT\_TRAIL View**
  - The new EVENT\_TIMESTAMP\_UTC column appears in the UNIFIED\_AUDIT\_TRAIL view.
  - The EVENT\_TIMESTAMP\_UTC column helps partition pruning, improving the read performance of the UNIFIED\_AUDIT\_TRAIL view.

- **Passwords Removed from Oracle Database Accounts**
  - Oracle Database supplied schema-only accounts now have their passwords removed to prevent users from authenticating to these accounts.
  - This enhancement does not affect the sample schemas
  - Administrators can still assign passwords to the default schema-only accounts.
  - The benefit of this feature is that administrators no longer have to periodically rotate the passwords for these Oracle Database provided schemas

- **Signature-Based Security for LOB Locators**
  - Configure signature-based security for large object (LOB) locators.
  - LOB signature keys are in both multitenant pluggable databases (PDBs) or in standalone, non-multitenant databases. You can enable the encryption of the LOB signature key credentials by executing the ALTER DATABASE DICTIONARY ENCRYPT CREDENTIALS SQL statement

- **Unified Auditing Top-Level Statements**

- Unified auditing top-level feature enables auditing top-level user (direct user) activities in the
- Use this feature to audit only the events generated by top-level users, without the overhead of creating audit records for indirect SQL statements

- Privilege Analysis Now Available in Oracle Database Enterprise Edition
- Support for Oracle Native Encryption and SSL Authentication for Different Users Concurrently
  - You can set the new SQLNET.IGNORE\_ANO\_ENCRYPTION\_FOR\_TCPS parameter to TRUE. This setting ignores the SQLNET.ENCRYPTION\_CLIENT or SQLNET.ENCRYPTION\_SERVER when a TCPS client is used and either of these two parameters are set to REQUIRED.

- Ability to Grant or Revoke Administrative Privileges to and from Schema-Only Accounts
  - Grant administrative privileges, such as SYSOPER and SYSBACKUP, to schema-only accounts.
- Automatic Support for Both SASL and Non-SASL Active Directory Connections
  - Support is available for both Simple Authentication and Security Layer (SASL) and Transport Layer Security (TLS) binds for Microsoft Active Directory connections.

- **Database Vault Operations Control for Infrastructure Database Administrators**
  - You can now use Oracle Database Vault to block common users (infrastructure database administrators, for example) from accessing local data in pluggable databases (PDBs).
- **Database Vault Command Rule Support for Unified Audit Policies**
  - Oracle Database Vault command rules for unified audit policies.



# Oracle Database 19c Security

## *PL/SQL Security*

- **We will cover the following:**
  - Creating and using definer's rights procedures
  - Creating and using invoker's rights procedures
  - Using code-based access control
  - Restricting access to program units by using accessible by

- **Definer** is the owner of a procedure.
- **Invoker** is a user who uses a procedure, but is not the definer of the procedure.
- **Definer's rights procedure** is a procedure (or a program unit) that executes with the privileges of its definer.
- **Invoker's rights procedure** is a procedure (or a program unit) that executes with the privileges of the invoker.

- **Code base access control** is a new feature, introduced in Oracle Database 19c.
- It enables you to grant database roles to PL/SQL functions, procedures, or packages.
- The purpose of the accessible by clause is to limit the calling set of program units to be those in the accessible by clause and the unit itself.

## *Creating and using definer's rights procedures*

- You'll learn to create and use definer's rights procedures.
- You'll use a user who has a DBA role.

## *Creating and using definer's rights procedures*

1. Definer's rights procedures are executed by using privileges that are granted to the owner of the procedure.
2. In our example, we have two users: **procowner** - a user who is the owner of the procedure and has privilege to update table **ernesto.tbl** and **procuser** - a user who just executes the procedure.
3. **procuser** creates procedure by using the **AUTHID DEFINER** clause, which means that this procedure will be definer's rights procedure.
4. This is a default behavior.

## *Creating and using definer's rights procedures*

5. **procuser** tries to update table **ernesto.tbl** directly, but it gets an error:

```
SQL> UPDATE ernesto.TBL SET B = 'value1' WHERE A = 1;  
UPDATE ernesto.TBL SET B = 'value1' WHERE A = 1  
*  
ERROR at line 1:  
ORA-01031: insufficient privileges
```

6. This is the expected behavior, considering that **procuser** doesn't have an update privilege on **ernesto.tbl**.
7. When **procuser** executes the procedure, the table is updated because the privilege of the definer is applied.

## *Creating and using invoker's right procedures*

- You'll learn to create and use invoker's rights procedures.
- They can be useful when creating PL/SQL procedures in a highly privileged schema.
- Also, when there is no SQL code in the PL/SQL procedure and the procedure is available to other users, invoker's rights procedure will be executed more efficiently.
- There are no changes in the values of current schema and currently enabled roles during the execution.
- You'll need a user who has the DBA role.

## *Creating and using invoker's right procedures*

- Invoker's rights procedures are executed by using privileges that are granted to the user that executes the procedure.
- The user **ernesto** creates an invoker's rights procedure by specifying the **AUTHID CURRENT\_USER** clause.
- When **procuser1** executes that procedure, he or she succeeds because **update** privilege is granted to **procuser1**, but when **procuser2** tries to execute it, he or she gets an error because **procuser2** lacks the **update** privilege.

## *Creating and using invoker's right procedures*

- Let's consider this security problem.
- 1. Connect as a user who has a DBA role (for example, ernesto).
- Create a new user maluser and grant him the privileges create session and create procedure.

```
SQL> create user maluser identified by oracle1;  
SQL> grant create session, create procedure to maluser;
```

## *Creating and using invoker's right procedures*

2. Connect as the user **maluser** and create the following "malicious" procedure with the purpose of granting him the DBA role:

```
SQL> connect maluser/oracle1
create or replace procedure mal_proc
authid current_user
as
begin
  execute immediate 'grant dba to maluser';
end;
/
```

## *Creating and using invoker's right procedures*

3. Connect as a user who has a DBA role (for example, **ernesto**) and execute the procedure you created in the previous step:

```
SQL> connect ernesto
SQL> EXEC maluser.mal_proc;
PL/SQL procedure successfully completed.
```

4. Connect as **maluser** and check whether the DBA role is granted:

```
SQL> connect maluser
SQL> select * from session_roles where role= 'DBA';
```

## *Creating and using invoker's right procedures*

- We've seen that a low-privileged user can trick the DBA user to grant him the DBA role, by tricking the DBA user (in this case, **ernesto**) to execute an invoker's rights procedure that was created by low-privileged user (in this case, **maluser**).
- The user **ernesto** can avoid this scenario by examining code that he is executing using his own privileges and specifying users whose procedures he wants to execute using his own privileges.
- The latter can be done by granting the **INHERIT PRIVILEGE** privilege to these users.

# *Creating and using invoker's right procedures*

- Let's try this:

1. Connect as a user **ernesto** and revoke **inherit** privileges from public user:

```
SQL> connect ernesto  
SQL> revoke inherit privileges on user ernesto from public;
```



ernesto



```
SQL> REVOKE INHERIT PRIVILEGES ON USER ernesto  
FROM PUBLIC;
```



public



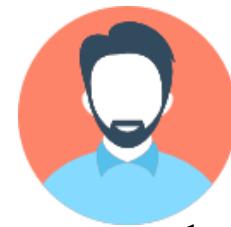
# Creating and using invoker's right procedures

## 2. Try to execute mal\_proc:

```
SQL> EXEC maluser.mal_proc;
BEGIN maluser.mal_proc; END;
*
ERROR at line 1:
ORA-06598: insufficient INHERIT PRIVILEGES privilege
ORA-06512: at "MALUSER.MAL_PROC", line 1
ORA-06512: at line 1
```



maluser



ernesto

```
SQL> create or replace procedure mal_proc
  2 authid current_user
  3 as
  4 begin
  5 execute immediate 'grant dba to maluser'
  6 end;
  7 /
```

EXECUTE



```
ERROR at line 1:
ORA-06598: insufficient INHERIT PRIVILEGES privilege
ORA-06512: at "maluser.mal_proc", line 1
ORA-06512: at line 1
```

# *Creating and using invoker's right procedures*

## 3. Grant inherit rights privileges to maluser:

```
SQL> grant inherit privileges on user ernesto to maluser;
```



maluser

```
SQL> create or replace procedure mal_proc  
2 authid current_user  
3 as  
4 begin  
5 execute immediate 'grant dba to maluser'  
6 end;  
7 /
```

```
SQL> GRANT INHERIT PRIVILEGES ON USER ernesto TO maluser;
```



ernesto

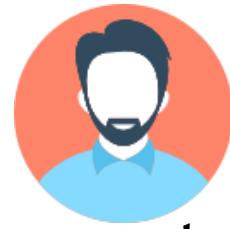
# *Creating and using invoker's right procedures*

4. Try again to execute **mal\_proc**:

```
SQL> EXEC maluser.mal_proc;  
PL/SQL procedure successfully completed.
```



maluser



ernesto

```
SQL> create or replace procedure mal_proc  
2 authid current_user  
3 as  
4 begin  
5 execute immediate 'grant dba to maluser'  
6 end;  
7/
```

EXECUTE



```
PL/SQL procedure successfully completed.
```

# *Creating and using invoker's right procedures*

- Users who have **the inherit any privileges** system privilege are exempted from this rule.
1. Connect as a user who has a DBA role (for example, **ernesto**), create two users, and grant them the following privileges:

```
SQL> connect ernesto
SQL> create user super_user identified by oracle1;
SQL> create user regular_user identified by oracle2;
SQL> grant create session, create procedure to super_user;
SQL> grant create session, create procedure to
    regular_user;
```

## *Creating and using invoker's right procedures*

2. Grant **inherit any privileges** only to **super\_user**:

```
SQL> grant inherit any privileges to super_user;
```

3. Connect as **regular\_user** and create the following procedure:

```
SQL> connect regular_user
create or replace procedure reg_proc
  authid current_user
  as
begin
  execute immediate 'grant dba to regular_user';
end;
/
```

## *Creating and using invoker's right procedures*

4. Connect as **super\_user** and create the following procedure:

```
SQL> connect super_user
create or replace procedure sup_proc
  authid current_user
  as
  begin
    execute immediate 'grant dba to super_user';
  end;
  /
```

## *Creating and using invoker's right procedures*

5. Connect as user **ernesto** and try to execute **reg\_proc** from **regular\_user**:

```
SQL> connect ernesto/oracle_4U
Connected.
SQL> EXEC regular_user.reg_proc;
BEGIN regular_user.reg_proc; END;
*
ERROR at line 1:
ORA-06598: insufficient INHERIT PRIVILEGES privilege
ORA-06512: at "REGULAR_USER.REG_PROC", line 1
ORA-06512: at line 1
```

6. Try to execute **sup\_proc** from **super\_user**:

```
SQL> EXEC super_user.sup_proc;
PL/SQL procedure successfully completed.
```

## *Using code-based access control*

- In this chapter, you'll use code base access control with invoker's rights procedure.
- You'll use a user who has a DBA role.

## *Using code-based access control*

- Code-based access control allows us to grant a role to a PL/SQL procedure, function, or package.
- It works with both definer's rights and invoker's rights procedures.
- The invoker's rights procedure is used to update content.
- This is an invoker's rights procedure, meaning that it is executed by using privileges granted to invoker (`proc_user`).
- However, `proc_user` doesn't have `update` privilege, but he can still execute it successfully because procedure itself contains `update` privilege, granted through the role `proc_role`.

## Using code-based access control

- Remember that, in some cases, privileges granted to users via roles are not active during the PL/SQL calls. Let's try this:
1. Connect as a user who has a DBA role (for example, **ernesto**), create the user **plsusr**, and grant him the **create session** and **create procedure** privileges:

```
SQL> create user plsusr identified by oracle1;
SQL> grant create session, create procedure to plsusr;
```

2. Create the role **plsrole1** and grant the **create table** privilege to it:

```
SQL> create role plsrole1;
SQL> grant create table to plsrole1;
```

### 3. Grant **plsrole1** to the user **plsusr**:

```
SQL> grant plsrole1 to plsusr;
```

### 4. Connect as **plsusr** and create the procedure **cr\_table**:

```
SQL> connect plsusr
create or replace procedure cr_table
authid definer
as
begin
  execute immediate 'create table test2(a int)';
end;
/
```

## Using code-based access control

5. Create the table **test1** to check whether the **plsusr** user has a **create table** privilege:

```
SQL> create table test1(a int);
Table created.
```

6. Execute the **cr\_table** procedure and observe the **insufficient privileges** error.

```
SQL> exec cr_table;
BEGIN cr_table; END;
*
ERROR at line 1:
ORA-01031: insufficient privileges
ORA-06512: at "PLSUSR.CR_TABLE", line 5
ORA-06512: at line 1
```

## *Using code-based access control*

7. Connect as a user who has the DBA role and grant the **create table** privilege directly to the user **plsusr**:

```
SQL> connect ernesto
SQL> grant create table to plsusr;
```

8. Connect as the user **plsusr** and try to execute the procedure **cr\_table** again.

```
SQL> connect plsusr/oracle1
SQL> exec cr_table;
PL/SQL procedure successfully completed.
```

## *Restricting access to program units by using accessible by*

- You'll learn about the effects of using the **accessible by** clause.
- You'll use a user who has the **create procedure** privilege.

## Restricting access to program units by using accessible by

1. An **accessible by** clause enables us to specify which packages can access procedures and functions of another package.
2. This process is called **white listing**.
3. We create the **protected\_pkg** package and we specify that procedures and functions of this package can be accessed only by procedures and functions of **public\_pkg** package.
4. We execute the **public\_proc** procedure from the **public\_pkg** package and, in output, we can observe that the **protected\_proc** procedure has been successfully executed.

## *Restricting access to program units by using accessible by*

5. However, if we try to execute `protected_proc` directly, we get an `insufficient privileges` error because the `accessible by` clause restricts execution of this procedure.
  
6. Even if we try to create a new package with the procedure that calls the `protected_proc` procedure, we get an `insufficient privileges` error.



# Oracle Database 19c Security

*Virtual Private Database*

- In this chapter, we will cover the following:
  - Creating different policy functions
  - Creating Oracle Virtual Private Database row-level policies
  - Creating column-level policies
  - Creating a driving context
  - Creating policy groups
  - Setting context as a driving context
  - Adding a policy to a group
  - Exempting users from VPD policies

- Oracle **Virtual Private Database (VPD)** is a security feature, introduced in Oracle Database 8i.
- Discretionary access control (DAC) grants/restricts access to data at an object level.
- VPD enables you more granular control over security of your data.

- There are five types of policies based on how often a policy function is evaluated:

- DBMS\_RLS.DYNAMIC
- DMBS\_RLS STATIC
- DBMS\_RLS.SHARED\_STATIC
- DBMS\_RLS.CONTEXT\_SENSITIVE
- DBMS\_RLS.SHARED\_CONTEXT\_SENSITIVE

# *Introduction*

- Steps to implement the VPD policy

Create a PL/SQL package that sets application context



Create an application context



Create a policy function



*Create a VPD policy*

# *Introduction*

- A driving context is an application context that has at least one attribute and its purpose is to determine which group of policies will be applied.

# Introduction

- Steps to implement policy groups

Determine the default policies



Create and set the driving context



Create a policy group for each application



*Add policies to the appropriate policy groups*

# *Creating different policy functions*

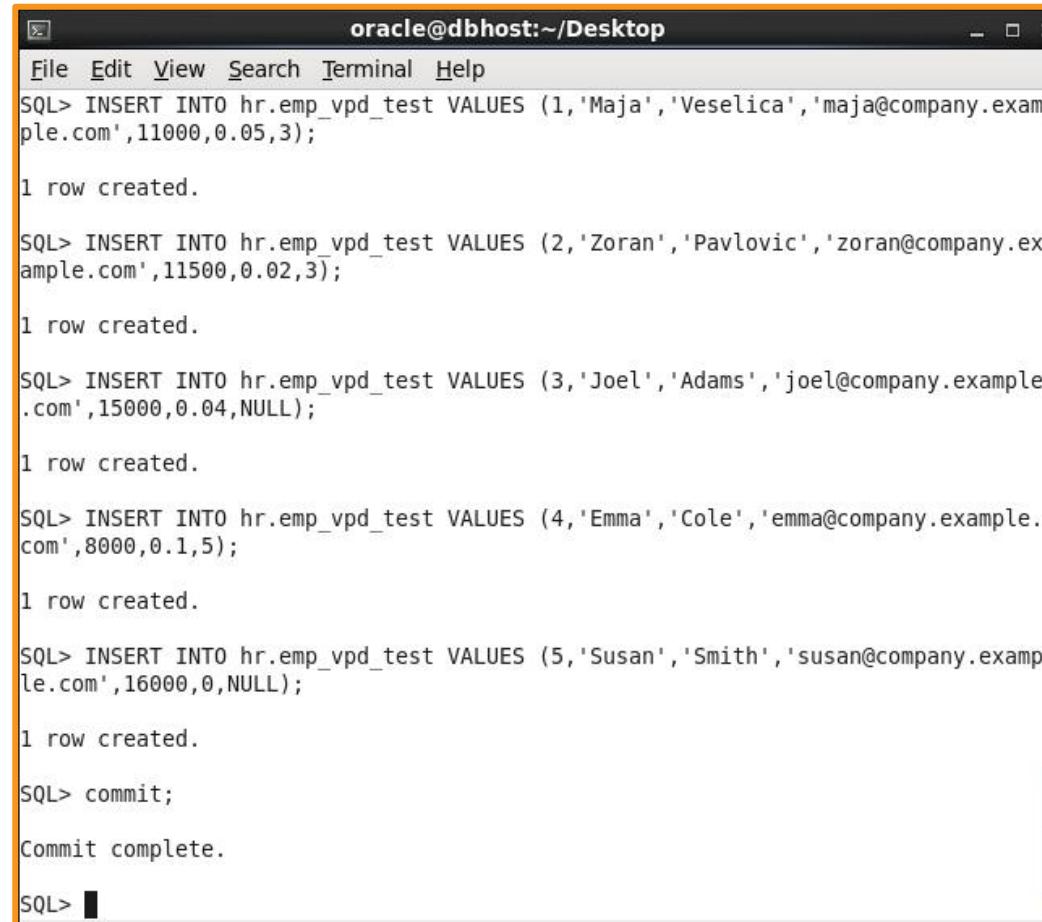
- The purpose of a policy function is to return a predicate that will be applied in WHERE clause of the statement (except for INSERT operation).
- You'll create several simple policy functions, based on different business and security requirements.
- You'll need to create the table `hr.emp_vpd_test`, insert several values into that table, and create several user.

```
SQL> CREATE TABLE hr.emp_vpd_test (
  2 emp_id NUMBER(6) NOT NULL,
  3 first_name VARCHAR2(30) NOT NULL,
  4 last_name VARCHAR2(30) NOT NULL,
  5 email VARCHAR2(30) NOT NULL,
  6 salary NUMBER(8,2),
  7 comm_pct NUMBER(2,2),
  8 mgr_id NUMBER(6));
```

Table created.

# *Creating different policy functions*

- Test data in the table `hr.emp_vpd_test`



The screenshot shows a terminal window titled "oracle@dbhost:~/Desktop". The window contains the following SQL commands and their results:

```
File Edit View Search Terminal Help
oracle@dbhost:~/Desktop
SQL> INSERT INTO hr.emp_vpd_test VALUES (1,'Maja','Veselica','maja@company.example.com',11000,0.05,3);
1 row created.

SQL> INSERT INTO hr.emp_vpd_test VALUES (2,'Zoran','Pavlovic','zoran@company.example.com',11500,0.02,3);
1 row created.

SQL> INSERT INTO hr.emp_vpd_test VALUES (3,'Joel','Adams','joel@company.example.com',15000,0.04,NULL);
1 row created.

SQL> INSERT INTO hr.emp_vpd_test VALUES (4,'Emma','Cole','emma@company.example.com',8000,0.1,5);
1 row created.

SQL> INSERT INTO hr.emp_vpd_test VALUES (5,'Susan','Smith','susan@company.example.com',16000,0,NULL);
1 row created.

SQL> commit;
Commit complete.

SQL>
```

## *Creating different policy functions*

- Create a policy function that uses the application context you created, whereas other policy functions you created use built-in application contexts.

*A policy function can be part of a package or is standalone.*

# *Creating different policy functions*

- To test whether a policy function where a user can't access data in a table and other users can access entire data in the table:
  1. Connect to the database as the user **maja** and execute the following statement:

```
SQL> select no_access('a','b') from dual;  
NO_ACCESS('A','B')  
-----  
1=1
```

## *Creating different policy functions*

2. Grant the user **susan** execute on **no\_access** function and connect to the database as the user **susan**.

```
SQL> grant execute on no_access to susan;  
Grant succeeded.  
  
SQL> connect susan  
Enter password:  
Connected.
```

3. Execute the following statement:

```
select maja.no_access('a','b') from dual;
```

```
SQL> select maja.no_access('a','b') from dual;  
MAJA.NO_ACCESS('A','B')  
-----  
1=2
```

## *Creating different policy functions*

4. Connect to the database as the user **maja** and revoke execute on **no\_access** function from the user **susan**.

```
SQL> connect maja
Enter password:
Connected.
SQL> revoke execute on no_access from susan;

Revoke succeeded.
```

## *Creating Oracle Virtual Private Database row-level policies*

- Oracle VPD row-level policies restrict users' access per row for a protected object.
- This means that two users who execute the same query against, for example, a table may, as a result, receive different number of rows.

## *Creating Oracle Virtual Private Database row-level policies*

- You define two VPD policies on the same table, and they are both enabled.
- The first one only restricts the user **susan** from accessing the table, whereas the other one affects all users connected to the database.
- If **susan** connects to the database, both policies will determine whether she can access the data and if **yes**, which data.
- The way the policies are defined, she won't be able to view data in the table.

## *Creating column-level policies*

- When you create a column-level VPD policy, you define sensitive columns, and if those columns are referenced in a query, statement will be rewritten.
- To create a column-level VPD policy, you also use the **DBMS\_RLS.ADD\_POLICY** procedure.

## *Creating column-level policies*

1. Create the **test\_col** VPD policy.
2. The user is granted the role (**HREMP\_TEST**) that will allow him to view entire data.
3. Restrict displayed rows by **TEST\_POL2**, so user can view only his data.

## *Creating column-level policies*

4. Disable the TEST\_POL2 policy using the DBMS\_RLS.ENABLE\_POLICY procedure. The syntax is:

```
DBMS_RLS.ENABLE_POLICY (
object_schema IN VARCHAR2 NULL,
object_name   IN VARCHAR2,
policy_name   IN VARCHAR2,
enable        IN BOOLEAN TRUE)
```

5. Other user can view all rows, but cannot view salary and comm\_pct, because he doesn't have the HREMP\_TEST role.

## *Creating a driving context*

- Having multiple VPD policies is harder to manage, and it can lead to unexpected/unwanted results.
- For example, you have two applications and want to create two policy groups.
- If the first application accesses the table, the `test_pol1` and `test_col` policies should be enforced, and if second application accesses the table, the `test_pol2` policies should be applied.

## *Creating a driving context*

- You'll create an application context and set it.
- You'll need an existing user who can create an application context.

## *Creating policy groups*

- You'll create two policy groups that will be applied to table `hr.emp_vpd_test`.
- You'll need an existing user who has appropriate privileges.

## *Setting context as a driving context*

- You'll make an existing application context a driving context (you'll associate it with the protected object).
- You'll need an existing application context and an existing user who has appropriate privileges.

## *Adding policy to a group*

- Create VPD policies as part of a policy group.
- You'll need an existing user who has appropriate privileges. Drop all VPD policies using the **DBMS\_RLS.DROP\_POLICY** procedure.
- Drop policies

```
SQL> BEGIN
 2  DBMS_RLS.DROP_POLICY('HR','EMP_VPD_TEST','TEST_POL1');
 3  DBMS_RLS.DROP_POLICY('HR','EMP_VPD_TEST','TEST_POL2');
 4  DBMS_RLS.DROP_POLICY('HR','EMP_VPD_TEST','TEST_COL');
 5  END;
 6  /
```

PL/SQL procedure successfully completed.

## *Exempting users from VPD policies*

- VPD policies are not enforced for users who connect as **sysdba**, during direct path export, and for users who have the **EXEMPT ACCESS POLICY** privilege.
- You'll connect to the database as **SYS** user and grant **EXEMPT ACCESS POLICY** to an existing user.



# Oracle Database 19c Security

## *Data Redaction*

We will cover the following:

- Creating a redaction policy when using full redaction
- Creating a redaction policy when using partial redaction
- Creating a redaction policy when using random redaction
- Creating a redaction policy when using regular expression redaction
- Using Oracle Enterprise Manager Cloud Control 19c to manage redaction policies
- Changing the function parameters for a specified column
- Adding a column to the redaction policy
- Enabling, disabling, and dropping a redaction policy
- Exempting users from data redaction policies

- **Oracle Data Redaction** is a new security feature, introduced in Oracle Database 19c.
- Oracle Data Redaction masks sensitive data just before the results of the SQL query are returned to the application that issued the query.
- Data stored in the database is **NOT** changed in any way.

- Oracle Data Redaction and Oracle Data Masking are both used to mask sensitive data, but these solutions are completely different.
- **Oracle Data Masking** enables organizations to use production data in development and test environments by changing production data with realistic data (transformation is done by using masking rules).

- When you implement Oracle Data Redaction, you have to decide the following:

- What data should be redacted
- Which redaction method is most suitable for the identified data
- In which situations the redaction should take place

# *The parts of a redaction policy*

<b>WHAT</b>	Schema, object, column
<b>HOW</b>	Redaction type, required parameters
<b>WHEN</b>	SQL expression

# *The types of redaction*

None	Full	Partial	Regular Expression	Random
<ul style="list-style-type: none"><li>Redaction is NOT applied</li></ul>	<ul style="list-style-type: none"><li>Columns are redacted to <b>constant values</b> depending on column data type</li></ul>	<ul style="list-style-type: none"><li>User-specified <b>positions</b> are replaced by a user-specified <b>character</b></li></ul>	<ul style="list-style-type: none"><li>Pattern for matching and replacing is defined and used for redaction</li></ul>	<ul style="list-style-type: none"><li>Preserves data types</li><li>Randomizes output</li></ul>

- To view which data redaction policies are defined and whether they are enabled, you can query the **redaction\_policies** view.
- Also, it is very useful to query the **redaction\_columns** view, which shows which columns will be masked and what type of redaction will be used.
- We assume that database is up and running, and each user has at least a **create session** privilege.

## *Creating a redaction policy when using full redaction*

- You will create a redaction policy on the **income\_level** column, find the default values for different data types, and change the default value for the **varchar2** data type.

## *Creating a redaction policy when using full redaction*

- You'll need the following:

- An existing user who can view data in OE.CUSTOMERS sample table but doesn't have exempt redaction policy privilege (for example, oe)
- To connect as a **SYS** user to the database
- To restart the database

## *Creating a redaction policy when using full redaction*

1. In order to manage redaction policies, you need to connect to a database as a user who has an execute privilege on the **dbms\_redact** package.
2. Define redaction policy **CUST\_POL**.
3. Creating a new redaction policy is done by using the **ADD\_POLICY** procedure in the **DBMS\_REDACT** package.

## *Creating a redaction policy when using full redaction*

4. The **WHAT** section defines on which the column redaction policy should be applied.
5. The **HOW** section defines the redaction type.
6. The **WHEN** section defines the conditions when protected data will be masked.

# *Creating a redaction policy when using full redaction*



SQL> SELECT \* FROM EMPLOYEES;

NAME	SALARY
frank	0
emily	0
grace	0

EMPLOYEES

NAME	SALARY
frank	8800
emily	9600
grace	11300

```
DBMS_REDACT.ADD_POLICY  
(object_schema => 'GLDB'  
object_name => 'EMPLOYEES'  
policy_name => 'SAL_POLICY'  
column_name => 'SALARY'  
function_type => DBMS_REDACT.FULL,  
expression => '7=7');
```

# Creating a redaction policy when using full redaction

```
oracle@dbhost:~/Desktop
File Edit View Search Terminal Help
SQL> connect / as sysdba
Connected.
SQL> desc REDACTION_VALUES_FOR_TYPE_FULL
Name          Null?    Type
-----        -----
NUMBER_VALUE      NOT NULL NUMBER
BINARY_FLOAT_VALUE NOT NULL BINARY_FLOAT
BINARY_DOUBLE_VALUE NOT NULL BINARY_DOUBLE
CHAR_VALUE        VARCHAR2(1)
VARCHAR_VALUE     VARCHAR2(1)
NCHAR_VALUE       NCHAR(1)
NVARCHAR_VALUE   NVARCHAR2(1)
DATE_VALUE        NOT NULL DATE
TIMESTAMP_VALUE   NOT NULL TIMESTAMP(6)
TIMESTAMP_WITH_TIME_ZONE_VALUE NOT NULL TIMESTAMP(6) WITH TIME ZONE
BLOB_VALUE         BLOB
CLOB_VALUE         CLOB
NCLOB_VALUE        NCLOB
SQL> select date_value from redaction_values_for_type_full;
DATE_VALU
-----
01-JAN-01
```

1

To find out default values for other data types, query **REDACTION\_VALUES\_FOR\_TYPE\_FULL**. Finding the default value for **DATE** data type is depicted

# *How to change the default value*



```
oracle@dbhost:~/Desktop
File Edit View Search Terminal Help
SQL> connect secmgr
Enter password:
Connected.
SQL> exec dbms_redact.update_full_redaction_values (varchar_val => 'T')
BEGIN dbms_redact.update_full_redaction_values (varchar_val => 'T'); END;
*
ERROR at line 1:
ORA-00942: table or view does not exist
ORA-06512: at "SYS.DBMS_REDACT", line 363
ORA-06512: at line 1
```

1

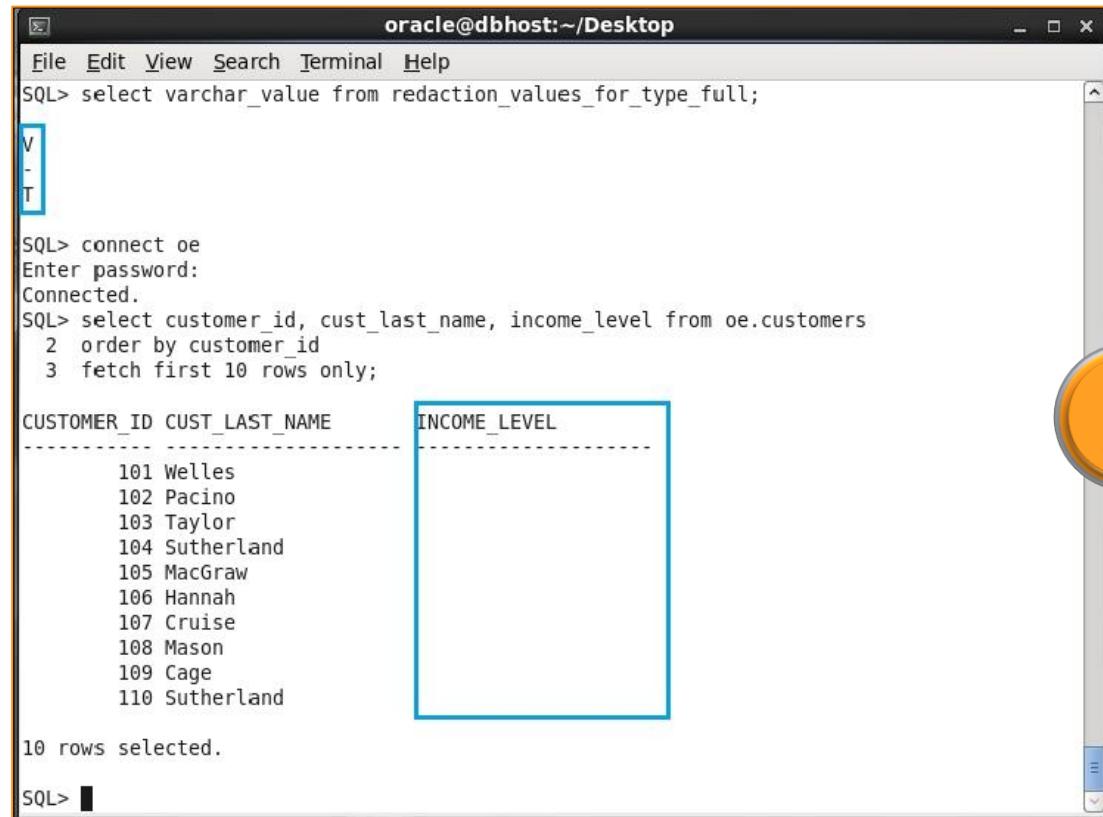
An unsuccessful change of default value

Connect to the database as the **SYS** user and change the default value.

2

```
SQL> exec dbms_redact.update_full_redaction_values (varchar_val => 'T')
PL/SQL procedure successfully completed.
```

# How to change the default value



```
oracle@dbhost:~/Desktop
File Edit View Search Terminal Help
SQL> select varchar_value from redaction_values_for_type_full;
V
-
T

SQL> connect oe
Enter password:
Connected.
SQL> select customer_id, cust_last_name, income_level from oe.customers
  2 order by customer_id
  3 fetch first 10 rows only;

CUSTOMER_ID CUST_LAST_NAME      INCOME_LEVEL
-----  -----
101 Welles
102 Pacino
103 Taylor
104 Sutherland
105 MacGraw
106 Hannah
107 Cruise
108 Mason
109 Cage
110 Sutherland

10 rows selected.

SQL>
```

3

Verify that the default value is changed and that there is **no effect before you restart the database.**

# How to change the default value



```
oracle@dbhost:~/Desktop
File Edit View Search Terminal Help
SQL> connect / as sysdba
Connected.
SQL> shutdown immediate
Database closed.
Database dismounted.
ORACLE instance shut down.
SQL> startup
ORACLE instance started.

Total System Global Area 1073741824 bytes
Fixed Size          2932632 bytes
Variable Size       692060264 bytes
Database Buffers   373293056 bytes
Redo Buffers        5455872 bytes
Database mounted.
Database opened.
SQL> connect oe
Enter password:
Connected.

SQL> select customer_id, cust_last_name, income_level from oe.customers
  2 order by customer_id
  3 fetch first 10 rows only;

CUSTOMER_ID CUST_LAST_NAME      INCOME_LEVEL
-----  -----
  101 Welles                      T
  102 Pacino                     T
  103 Taylor                      T
  104 Sutherland                  T
  105 MacGraw                     T
  106 Hannah                      T
  107 Cruise                      T
  108 Mason                       T
  109 Cage                        T
  110 Sutherland                  T

10 rows selected.
```

4

Restart the database and verify that the modified default value is displayed.

## *Creating a redaction policy when using partial redaction*

- You will implement partial redaction on columns of two different types: Number and Varchar2.
- Partial redaction means that only part of the data in a specified column will be masked whereas the other part of the data will be visible to the user.

## *Creating a redaction policy when using partial redaction*

- In order to manage redaction policies and also to create some test tables, you can connect to a database as a user who has a dba role.
- If you just need to manage redaction policies, you can connect with user who has the execute privilege on the **dbms\_redact** package.

## *Creating a redaction policy when using partial redaction*

- Creation of redaction policy for number type column:
  1. Create a redaction policy named **a\_tbl\_partial**.
  2. Creating a new redaction policy is done by using the **ADD\_POLICY** procedure in the **DBMS\_REDACT** package.
  3. A policy consists of several distinct sections, which column our redaction policy should be applied, redaction type, function parameters, condition when protected data will be masked.
  4. Evaluate using the following expression:  
**SYS\_CONTEXT("SYS\_SESSION\_ROLES","MYROLE") = "FALSE"**

## *Creating a redaction policy when using partial redaction*

- Creation of redaction policy for the Varchar2 column type.
  1. The difference is on function parameters and on condition, which is always TRUE.

# Creating a redaction policy when using partial redaction



SQL> SELECT \* FROM CUSTOMERS;

NAME	CREDIT_CARD
tom	#####9132
steve	#####5691
john	#####5806

CUSTOMERS	
NAME	CREDIT_CARD
tom	3455647456589132
steve	3734982321225691
john	3472586894975806

```
DBMS_REDACT.ADD_POLICY  
(object_schema => 'ERNESTO'  
object_name => 'CUSTOMERS'  
policy_name => 'CCN_POLICY'  
column_name => 'CREDIT_CARD'  
function_type => DBMS_REDACT.PARTIAL,  
function_parameters => 'VVVVVVVVVVVVVVVV,  
VVVVVVVVVVVVVVVV, #,1,12'  
expression => '1=1');
```

# Creating a redaction policy when using partial redaction

- Even though users can't see unmasked data, they can use redacted columns in `where` clause:

```
SQL> select * from ernesto.customers;
NAME          CREDIT_CARD
-----
tom           #####9132
steve         #####5691
john          #####5806
SQL> select * from ernesto.customers where credit_card like
      '34%';
NAME          CREDIT_CARD
-----
tom           #####9132
john          #####5806
```

## *Creating a redaction policy when using random redaction*

- Random redaction type is usually used for the number and date-time data types because for these data types, it is hard to make a distinction between the redacted (random) and real data.
- You will create redaction policy **EMP\_POL** using random redaction type on **hr.employees** table, column salary, by using SQL\*Plus.
- You will modify the **EMP\_POL** redaction policy.

## *Creating a redaction policy when using random redaction*

- To complete this, you'll need:

- An existing user who can view data in the HR.EMPLOYEES sample table but doesn't have an exempt redaction policy privilege (for example, hr)
- The secmgr user created in the Creating a redaction policy using full redaction recipe or another user who can create redaction policies (has execute on the dbms\_redact package)

## *Creating a redaction policy when using random redaction*

1. Create the redaction policy **EMP\_POL** by using the procedure **ADD\_POLICY** in the **DBMS\_REDACT** package.
2. Define that random redaction type will be used to redact data.
3. Policy expression, specifies the blacklist (which contains only user HR).
4. To define a whitelist change operator **=** to operator  **$\diamond$**  and define left and right operand according to your needs.

## *Creating a redaction policy when using random redaction*

- When the number data type is redacted using random redaction type, the redacted value will belong to the interval  $[0, |n|]$ , where  $|n|$  is the absolute value of the original data.
- The only exception to this rule is when original data is an integer between **-1** and **9**, and in that case, the redacted value will belong to the interval  $[0, 9]$ .

## *Creating a redaction policy when using regular expression redaction*

- A regular expression redaction type enables you to create and implement flexible redaction rules.
- You define patterns that will be used in order to match and replace data, as well as some other parameters of the search.
- You will create the redaction policy **SHORT\_POL**, which will be used to mask customers' phone numbers.

## *Creating a redaction policy when using regular expression redaction*

- You'll need:

- An existing user who can view data in the SH.CUSTOMERS sample table but doesn't have an exempt redaction policy privilege (for example, sh)
- The secmgr user you created in the Creating redaction policy using full redaction recipe or another user who can create redaction policies (has execute on dbms\_redact package)

## *Creating a redaction policy when using regular expression redaction*

- When creating redaction policies that use regular expression redaction type, you can choose between redaction shortcuts and the creation of custom regular expressions.

- You will perform several tasks with Data Redaction policies using Oracle Enterprise Manager Cloud Control 19c, including creation, modification, and deletion.
- You need Enterprise Manager Cloud Control 19c and **HR** sample schema in the database.

## *Changing the function parameters for a specified column*

- There are several ways in which you can change an existing redaction policy. You will:

- Change the function parameters for a specified column (the `a_tbl_partial` policy)
- Add a column (`commission_pct` in the `hr.employees` table) to the redaction policy `EMP_POL`

## Changing the function parameters for a specified column

- Also, it is possible to remove column from the redaction policy, alter the policy expression, and modify the type of redaction for a specified column.
- You concluded that the `a_tbl_partial` redaction policy doesn't satisfy the requirements for your application anymore because it redacts first four digits with 0 and leading zeros are not displayed in the application.
- You decide to alter the `a_tbl_partial` policy.

## *Changing the function parameters for a specified column*

- Before doing this, you should have completed the **Creating a redaction policy when using partial redaction** section.
- You will use the **secmgr** user you created in the **Creating a redaction policy when using full redaction** section.

## Add a column to the redaction policy

- You have to modify the existing redaction policy in order to redact more than one column in the table.
- In the table **HR.EMPLOYEES**, besides the column **SALARY**, you want to redact the column **COMMISSION\_PCT**.
- You will modify the redaction policy **EMP\_POL**.
- You decide that you want to use full redaction type for the column **COMMISSION\_PCT**.

## Add a column to the redaction policy

- Before doing this, you should have completed the **Creating redaction policy when using random redaction** section.
- You will use the **secmgr** user you created in the **Creating redaction policy when using full redaction** section.

## Add a column to the redaction policy

- Use the procedure `ALTER_POLICY` in the PL/SQL package `DMBS_REDACT` to change redaction policies.
- Specify value for the `ACTION` parameter, which defines what kind of change will happen.

## *Enabling, disabling, and dropping redaction policy*

- You will perform the three basic tasks: enabling, disabling, and dropping the same redaction policy (**CUST\_POL**).
- Also, you will check which redaction policies exist in the database and whether they are enforced (enabled).

## *Enabling, disabling, and dropping redaction policy*

- Before doing this, you should have completed the **Creating a redaction policy when using full redaction** section.

## *Exempting users from data redaction policies*

- You will create a user and then exempt that user from Data Redaction.
- This user will be exempted from all redaction policies in the database.

## *Exempting users from data redaction policies*

- Before doing this, you should have completed the **Creating a redaction policy when using the partial redaction section**.

## *Exempting users from data redaction policies*

- There is a new system privilege that is used to control which users will be exempted from data redaction in Oracle Database.
- This privilege is **EXEMPT REDACTION POLICY**.
- Users who are granted this privilege will be able to see clear (unmasked) data in the whole database if they have (select) privilege to access that data.

## *Exempting users from data redaction policies*

- Backup/restore as well as import and export operations are not subject to data redaction.
- However, data redaction policies are included in export and import operations.



# Oracle Database 19c Security

*Transparent Sensitive Data Protection*

In this chapter, we will cover the following tasks:

- Creating a sensitive type
- Determining sensitive columns
- Creating transparent sensitive data protection policy
- Associating transparent sensitive data protection policy with sensitive type
- Enabling, disabling, and dropping policy
- Altering transparent sensitive data protection policy

- Oracle Transparent Sensitive Data Protection (TSDP) is a new security feature, introduced in Oracle Database 19c.
- TSDP provides a way to create classes of sensitive data and enables more centralized control of how sensitive data is protected.

- In database versions 12.1.0.1 and 12.1.0.2, it leverages two Oracle security mechanisms:
  - Oracle Virtual Private Database (VPD), described in Chapter 4 , Virtual Private Database
  - Oracle Data Redaction, explained in Chapter 5 , Data Redaction

## *Steps to implement TSDP*

Create a sensitive data type

Associate sensitive columns with that type

Create TSDP policy

Associate the policy with the type

Enable the policy

- We assume that the database is up and running and each user has at least a create session privilege. I
- It is assumed that the user has a DBA role and it executes privileges on the following packages:
  - DBMS\_TSDP\_MANAGE
  - DBMS\_TSDP\_PROTECT
  - DBMS\_RLS
  - DBMS\_REDACT

## *Creating a sensitive type*

- To create a sensitive type, you can use Oracle Enterprise Manager or a command-line interface.
- You'll use the command-line interface to execute a PL/SQL procedure.
- You decided that you want to protect e-mail addresses stored in your database, so first you are going to create sensitive type `email_type`.

## *Creating a sensitive type*

- You'll need an existing user who can create a sensitive type.

## *Creating a sensitive type*

- You create a sensitive type (for example, `email_type`), which you can use to consistently mask (protect).
- By creating a sensitive type, you only define that in the database, there exists a class of sensitive data and you name it.
- You'll define where that sensitive data resides (in which columns) and the way that data will be protected.

# *Creating a sensitive type*

- To view existing sensitive types, execute the following query:

```
select name from DBA_SENSITIVE_COLUMN_TYPES;
```

```
SQL> SELECT NAME FROM DBA_SENSITIVE_COLUMN_TYPES;
NAME
-----
email_type
SQL> █
```

1

Finding information about defined  
sensitive types

## *Determining sensitive columns*

- After you decide which data is sensitive, you'll need to find all places where that data is stored.
- Once you do that, you'll classify the data (associate sensitive columns with sensitive types).
- You'll associate two sensitive columns (from two tables) with sensitive type you created previously.

## *Determining sensitive columns*

- To complete this recipe, create a user `challengeernesto`, create table `T1`, and insert several values into the table or use your own table.
- Also, you'll need an existing user who has an execute privilege on `dbms_tsdp_manage` package (for example, `c##ernesto`).

# Determining sensitive columns

- Creating table T1

```
SQL> CREATE TABLE CHALLENGEERNESTO.T1 (NAME VARCHAR2(30), EMAIL_ADDRESS VARCHAR2(40));
Table created.

SQL> INSERT INTO CHALLENGEERNESTO.T1 VALUES ('ERNESTO PAVLOVIC', 'ERNESTO.PAVLOVIC@CHALLENGEERNESTO.COM');
1 row created.

SQL > INSERT INTO CHALLENGEERNESTO.T1 VALUES ('MAJA VESELICA', 'MAJA.VESELICA@CHALLENGEERNESTO.COM');
1 row created.

SQL> COMMIT;
Commit complete.

SQL> SELECT * FROM CHALLENGEERNESTO.T1;
NAME                           EMAIL_ADDRESS
-----                         -----
ERNESTO PAVLOVIC               ERNESTO.PAVLOVIC@CHALLENGEERNESTO.COM
MAJA VESELICA                  MAJA.VESELICA@CHALLENGEERNESTO.COM
SQL
```

## *Determining sensitive columns*

- Define where sensitive data resides and associate it with previously created sensitive data type.

*You can associate a column with only one sensitive type. If you try to associate it with another type, you'll receive ORA-45607.*

## *Creating transparent sensitive data protection policy*

- This step defines the way you want to protect sensitive data.
- You can use **Data Redaction** or VPD settings for your TSDP policy.
- In this recipe, you'll use regular expression redaction to protect previously defined sensitive data.

## *Creating transparent sensitive data protection policy*

- To complete this, you'll need an existing user who has the execute privilege on the **dbms\_tsdp\_protect** package.

## *Creating transparent sensitive data protection policy*

1. Define variables `redact_features_options` and `policy_conditions`.
2. Data redaction settings, for TSPD policy, are defined by using `redact_features_options` variable that holds parameter-value pairs that correspond with the parameters in `DBMS_REDACT.ADD_POLICY` procedure.
3. Specify that data type of protected columns should be `VARCHAR2` in order for redaction settings to be applied on the column.

## Associating transparent sensitive data protection policy with sensitive type

- You'll associate TSDP policy and sensitive type you created previously.
- You'll need an existing user who has the execute privilege on the `dbms_tsdp_protect` package.

## Associating transparent sensitive data protection policy with sensitive type

- To verify that you successfully associated the TSDP policy and the sensitive type, execute the following query:

```
SQL> SELECT POLICY_NAME, SENSITIVE_TYPE FROM DBA_TSDP_POLICY_TYPE;
```

## *Enabling, disabling, and dropping policy*

- You'll learn to enable, disable, and drop transparent sensitive data protection policies.
- You'll need two existing users-one to manage TSDP policies and the other to view sensitive data.

## *Enabling, disabling, and dropping policy*

- Correct result-column `email_address` in schema `challengeernesto` is masked like specified in the policy and full redaction is applied on all values in column `email` in schema `HR` where data isn't matched to the specified pattern.
- Before you drop the policy, you don't have to disable it.

# *Enabling, disabling, and dropping policy*

- Another way to enable/disable protection is to use procedures `enable_protection_column` (`disable_protection_column`):

```
SQL>begin
  DBMS_TSDP_PROTECT.ENABLE_PROTECTION_COLUMN(
    schema_name =>'CHALLENGErnesto',
    table_name =>'T1',
    column_name =>'EMAIL_ADDRESS',
    policy => 'redact_regex_email');
  end;
/
SQL>begin
  DBMS_TSDP_PROTECT.ENABLE_PROTECTION_COLUMN(
    schema_name =>'HR',
    table_name =>'EMPLOYEES',
    column_name =>'EMAIL',
    policy => 'redact_regex_email');
  end;
/
SQL>begin
  DBMS_TSDP_PROTECT.DISABLE_PROTECTION_COLUMN(
    schema_name =>'CHALLENGErnesto',
    table_name =>'T1',
    column_name =>'EMAIL_ADDRESS',
    policy => 'redact_regex_email');
  end;
/
SQL>begin
  DBMS_TSDP_PROTECT.DISABLE_PROTECTION_COLUMN(
    schema_name =>'HR',
    table_name =>'EMPLOYEES',
    column_name =>'EMAIL',
    policy => 'redact_regex_email');
  end;
/
```

## *Altering transparent sensitive data protection policy*

- You'll alter policy you created in recipe **Creating transparent sensitive data protection policy** and enable it.

# Altering transparent sensitive data protection policy

- You'll need two existing users. Also, update the table `hr.employees`, as shown:

```
SQL> UPDATE HR.EMPLOYEES SET EMAIL = EMAIL || '@example.com' WHERE 1=1;  
107 rows updated.  
SQL> commit;  
Commit complete.  
SQL> █
```

## *Altering transparent sensitive data protection policy*

- After you alter the policy, you have to manually enable it (it isn't automatically enabled).



# Oracle Database 19c Security

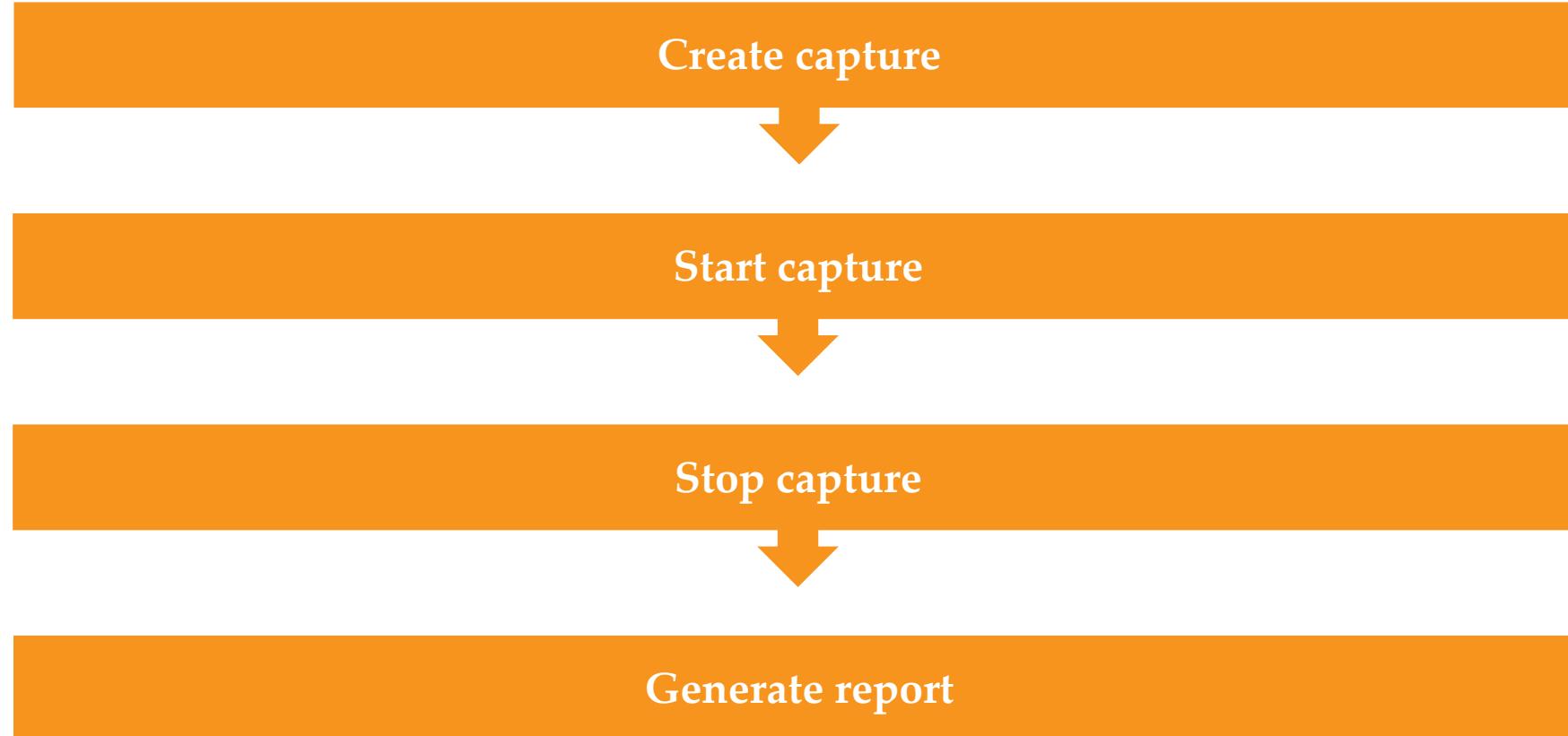
## *Privilege Analysis*

In this chapter, we will cover the following tasks:

- [Creating a database analysis policy](#)
- [Creating a role analysis policy](#)
- [Creating a context analysis policy](#)
- [Creating a combined analysis policy](#)
- [Starting and stopping privilege analysis](#)
- [Reporting on used system privileges](#)
- [Reporting on used object privileges](#)
- [Reporting on unused system privileges](#)
- [Reporting on unused object privileges](#)
- [How to revoke unused privileges](#)
- [Dropping the analysis](#)

- **Privilege analysis** is a new security feature, introduced in Oracle Database 19c.
- Privilege analysis is very useful to implement and maintain the least privilege principle by identifying both privileges that users are actually using (used privileges) and those that are only granted to them (unused privileges).

# *The steps to analyze the used and unused privileges*



# Introduction

- In this chapter, it is assumed that all users have a **create session** privilege, and in the following table, other privileges and roles granted to the users and roles are listed:

USER/ROLE	HR.EMPLOYEES	OE.ORDERS	ROLES/SYS.PRIVS.
BARBARA			P1_ROLE
NICK			DBA
ALAN	SELECT, INSERT, UP DATE, DELETE		
STEVE			P2_ROLE
P1_ROLE	SELECT		
P2_ROLE		SELECT, INSERT, UP DATE, DELETE	SELECT ANY TABLE, CREATE TABLE

- Depending on your needs, you can create and use four different types of privilege analysis policies that differ in the scope of the analysis. This scope can be:

- An entire database
- Role-based
- Context-based
- Role- and context-based

## *Creating database analysis policy*

- You'll learn to create database privilege analysis policy.
- It analyzes privileges in the whole database (except privileges used by **SYS** user).
- You can use SQL\*Plus and Enterprise Manager Cloud Control 12.1.0.3+ (in our case, EM19cR4) to create privilege analysis policies.

## *Creating database analysis policy*

- You'll need an existing user who can create a privilege analysis policy (has CAPTURE\_ADMIN role and SELECT ANY DICTIONARY privilege), for example, SYSTEM user.

## *Creating database analysis policy*

1. Create database-wide policy that will capture privileges, which are used by users (except the SYS user).
  
2. However, to start gathering data about privilege usage, you have to enable the policy.

# *Creating database analysis policy*



1

Login to EM as a user who has appropriate privileges and select **Privilege Analysis** from **Security** drop-down menu:

# Creating database analysis policy

Log in to the database as **SYSTEM** user or a user who has appropriate privileges (**CAPTURE\_ADMIN** role and **SELECT ANY DICTIONARY** privilege).

2

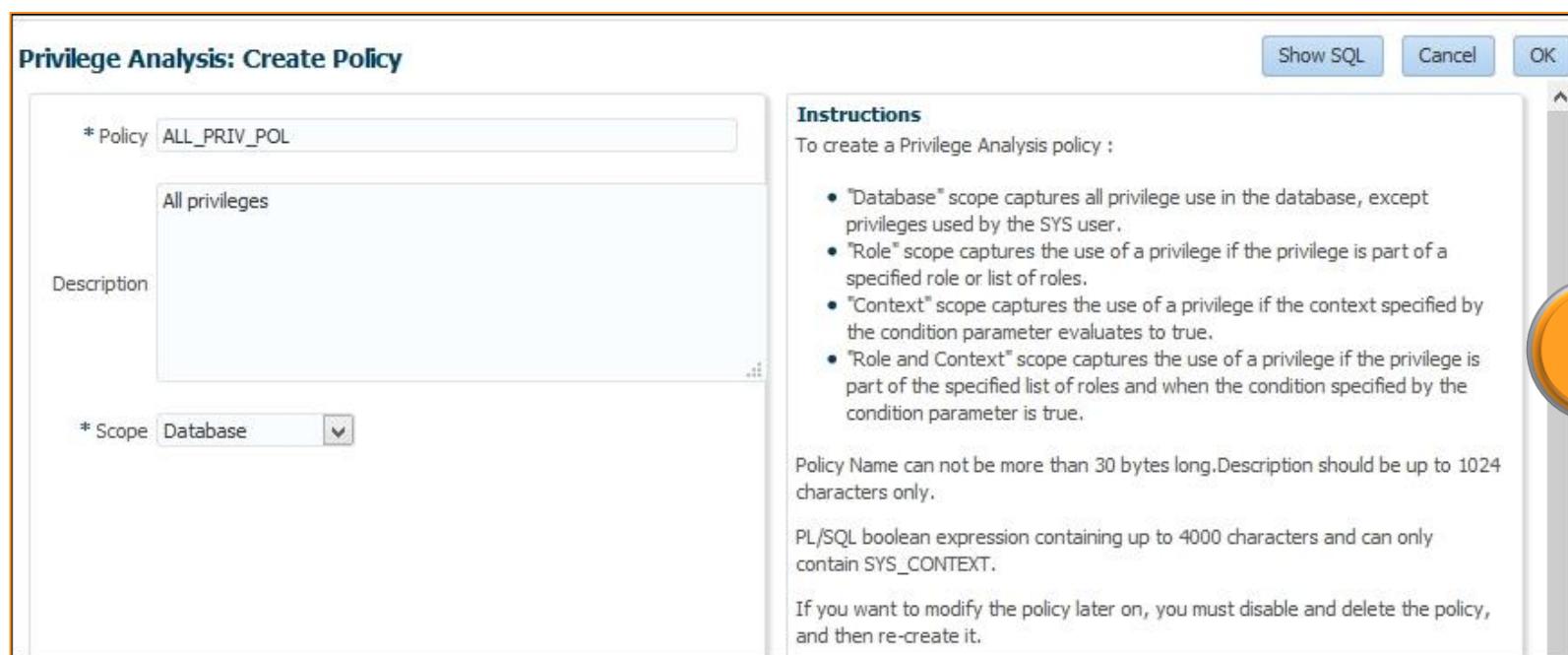


Policies					
		Actions		View	
		Create	Start Capture	Stop Capture	Generate Report
Policy	Active	Type	First Start Time	Last End Time	Capture Scope
ORA\$DEPENDENCY		Database			

3

Click on the **Create** button in the **Policy** section

# Creating database analysis policy



4

To create a database policy, choose that scope is **Database**, name the policy, and optionally write a description. Click on the **OK** button.

# Creating database analysis policy

You should receive a confirmation message and see your newly created policy listed in the table

5

The screenshot shows a confirmation message and a table of policies.

**Confirmation:**  
Privilege Analysis policy ALL\_PRIV\_POL has been created successfully.

**Privilege Analysis:**  
Privilege Analysis enables you to find information about privilege usage for a database according to a specified condition, such as privileges to run an application module or privileges used in a given user session. It analyzes both system privileges and object privileges. To monitor the privileges used for a user's action, you must create and enable a Privilege Analysis policy. Afterward, you can generate a report that describes the used and unused privileges and then from there, revoke (and regrant) privileges as necessary. However, you cannot use privilege analysis to analyze the use of SYS user privileges. Privilege analysis is licensed as part of Oracle Database Vault, but you do not need to enable Database Vault to use it.

**Policies**

Policy	Active	Type	First Start Time	Last End Time	Total Capture Duration	User
ALL_PRIV_POL		Database				
ORA\$DEPENDENCY		Database				

## *Creating role analysis policy*

- You'll create a **role analysis policy** using SQL\*Plus and Enterprise Manager Cloud Control 19c (EM).
- The usage of directly and indirectly granted privileges to the roles listed in the policy, will be captured if the roles are active for the session.

## *Creating role analysis policy*

- You'll need an existing user who can create a privilege analysis policy (has a CAPTURE\_ADMIN role and a SELECT ANY DICTIONARY privilege), for example, SYSTEM user.

# *Creating role analysis policy*

Log in to the database as **SYSTEM** user or a user who has appropriate privileges (**CAPTURE\_ADMIN** role and **SELECT ANY DICTIONARY** privilege).



Login to EM as a user who has appropriate privileges and select **Privilege Analysis** from **Security** drop-down menu:



Click on the **Create** button in the **Policy** section

# *Creating role analysis policy*

## Privilege Analysis: Create Policy

\* Policy ROLE\_PRIV\_POL

Usage of privileges granted through listed roles

Description

\* Scope Role

### Available Roles

DV\_PUBLIC  
DV\_PATCH\_ADMIN  
DV\_STREAMS\_ADMIN  
DV\_GOLDENGATE\_ADMIN  
DV\_XSTREAM\_ADMIN  
DV\_GOLDENGATE\_REDO\_ACCESS  
DV\_AUDIT\_CLEANUP  
DV\_DATAPUMP\_NETWORK\_LINK  
DV\_REALM\_RESOURCE  
DV\_REALM\_OWNER  
P2\_ROLE

\* Roles

### Selected Roles

DBA  
P1\_ROLE

4

Name the policy, select roles, optionally write a description, and click on OK button

# *Creating role analysis policy*

You should receive a confirmation message and see your newly created policy listed in the table

5

The screenshot shows a confirmation message and a table of policies.

**Confirmation:** Privilege Analysis policy ROLE\_PRIV\_POL has been created successfully.

**Privilege Analysis:** Describes the purpose of Privilege Analysis, mentioning it enables finding information about privilege usage for a database according to a specified condition, such as privileges to run an application module or privileges used in a given user session. It analyzes both system privileges and object privileges. To monitor the privileges used for a user's action, you must create and enable a Privilege Analysis policy. Afterward, you can generate a report that describes the used and unused privileges and then from there, revoke (and regrant) privileges as necessary. However, you cannot use privilege analysis to analyze the use of SYS user privileges. Privilege analysis is licensed as part of Oracle Database Vault, but you do not need to enable Database Vault to use it.

**Policies:**

Actions	View	Create	Start Capture	Stop Capture	Generate Report	Delete
Policy	Active	Type	First Start Time	Last End Time	Capture Scope	T
ALL_PRIV_POL		Database				
ORA\$DEPENDENCY		Database				
ROLE_PRIV_POL		Role				

## *Creating context analysis policy*

- You'll create a context analysis policy.
- After the policy is enabled, it will capture privileges when the condition specified in the policy evaluates to **true**.
- You'll need an existing user who can create a privilege analysis policy (has the **CAPTURE\_ADMIN** role and the **SELECT ANY DICTIONARY** privilege), for example, the **SYSTEM** user.

# *Creating context analysis policy*

Log in to the database as **SYSTEM** user or a user who has appropriate privileges (**CAPTURE\_ADMIN** role and **SELECT ANY DICTIONARY** privilege).



Login to EM as a user who has appropriate privileges and select **Privilege Analysis** from **Security** drop-down menu:



Click on the **Create** button in the **Policy** section

# *Creating context analysis policy*

**Privilege Analysis: Create Policy**

\* Policy

Privileges used by Steve

Description

\* Scope

SYS\_CONTEXT ('USERENV','CURRENT\_SCHEMA') = 'SYSTEM'

\* Condition

Examples:

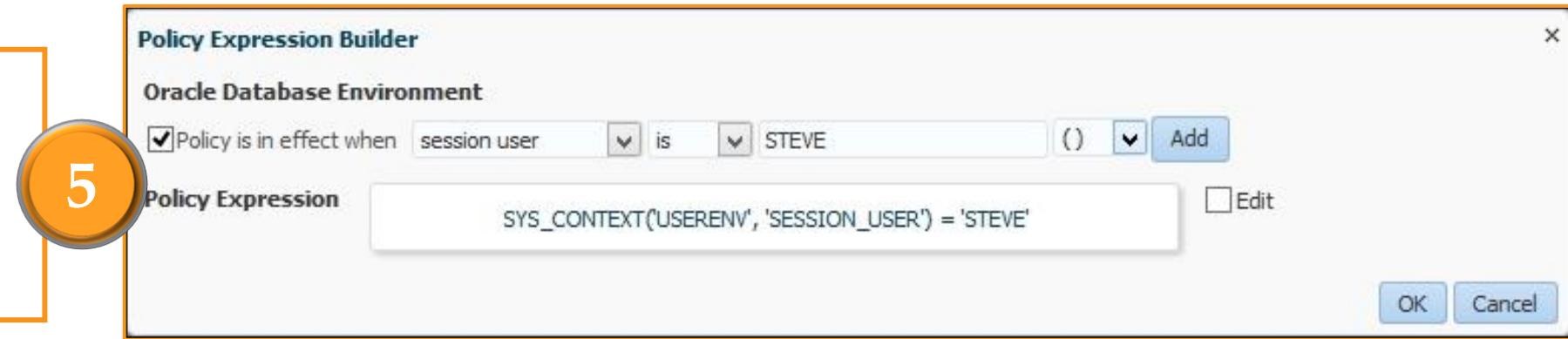
SYS\_CONTEXT ('USERENV', 'HOST') NOT IN ('sales\_24','sales\_12')  
SYS\_CONTEXT ('USERENV','CURRENT\_SCHEMA') = 'SYS'

4

Name the policy and optionally write a description

# *Creating context analysis policy*

Click on the Build Context Expression button. You can enter expression manually or use the built-in help. Click on the OK button.



# *Creating context analysis policy*

Privilege Analysis: Create Policy

\* Policy CONT\_PRIV\_POL

Privileges used by Steve

Description

\* Scope Context

SYS\_CONTEXT('USERENV', 'SESSION\_USER') = 'STEVE'

\* Condition

Examples:

SYS\_CONTEXT ('USERENV', 'HOST') NOT IN ('sales\_24','sales\_12')  
SYS\_CONTEXT ('USERENV','CURRENT\_SCHEMA') = 'SYS'

6

Make sure that you chose options you wanted and then click on the **OK** button

# *Creating context analysis policy*

You should receive a confirmation message and see your newly created policy listed in the table.

7

The screenshot shows a software interface for managing privilege analysis policies. At the top, a yellow banner displays the title "Creating context analysis policy". Below the banner, a large orange arrow points to the left. A callout box on the left contains the text: "You should receive a confirmation message and see your newly created policy listed in the table." A large orange circle containing the number "7" is positioned next to the callout box. The main area of the interface is titled "Confirmation" and displays the message: "Privilege Analysis policy CONT\_PRIV\_POL has been created successfully." Below this, the "Privilege Analysis" section provides a brief description of what it does. The bottom half of the screen shows a table titled "Policies" with the following data:

Policy	Active	Type	First Start Time	Last End	Capture
ALL_PRIV_POL		Database			
CONT_PRIV_POL		Context			
ORA\$DEPENDENCY		Database			
ROLE_PRIV_POL		Role			

## *Creating combined analysis policy*

- You'll create a combined analysis policy.
- This type of policy defines that the usage of directly and indirectly granted privilege to specified roles will be gathered if roles are enabled in the session and the context condition is satisfied.
- The context condition can consist of one or more conditions (you can use the AND or OR Boolean operators).

## *Creating combined analysis policy*

- You'll need an existing user who can create a privilege analysis policy (has the CAPTURE\_ADMIN role and the SELECT ANY DICTIONARY privilege), for example, the SYSTEM user.

# *Creating combined analysis policy*

Log in to the database as **SYSTEM** user or a user who has appropriate privileges (**CAPTURE\_ADMIN** role and **SELECT ANY DICTIONARY** privilege).

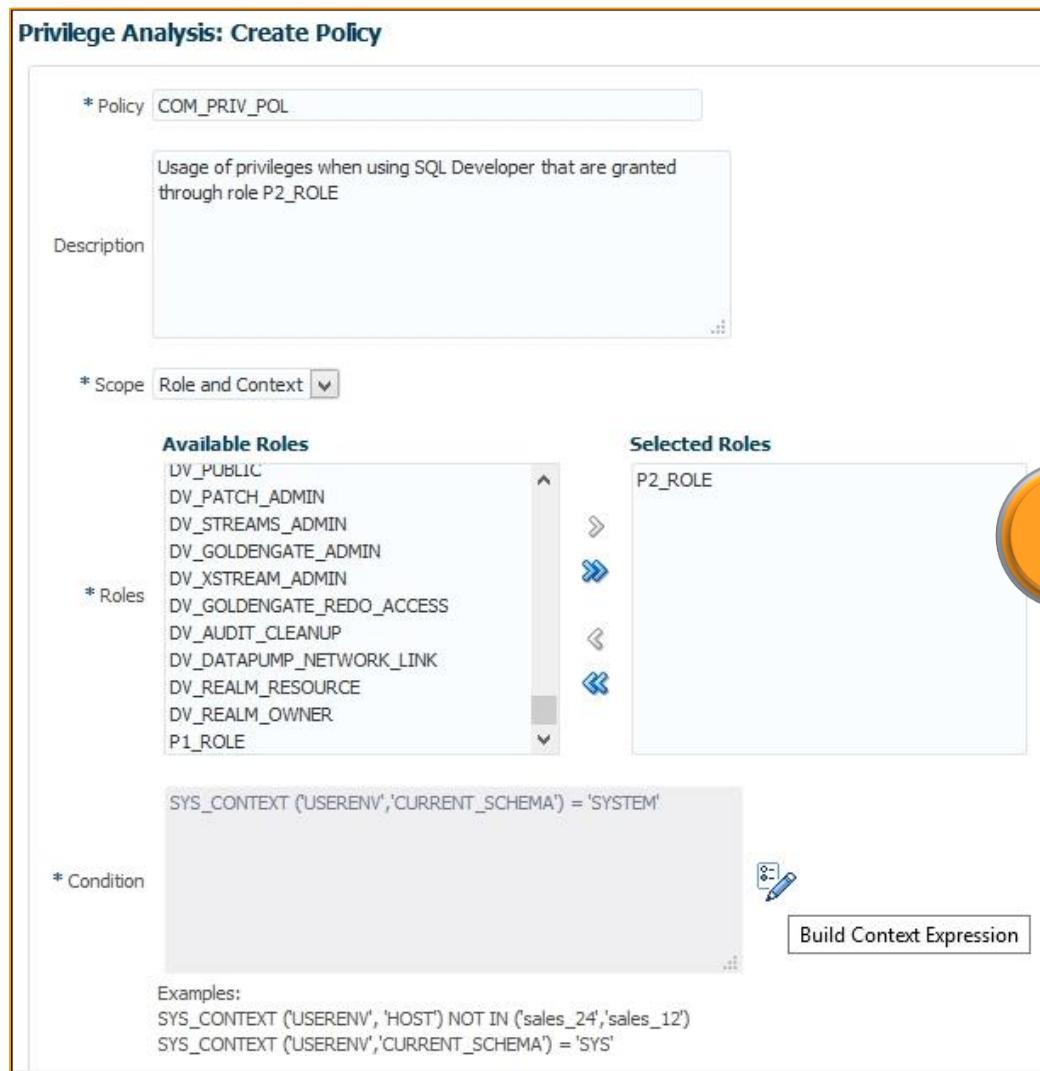


Login to EM as a user who has appropriate privileges and select **Privilege Analysis** from **Security** drop-down menu:



Click on the **Create** button in the **Policy** section

# Creating combined analysis policy



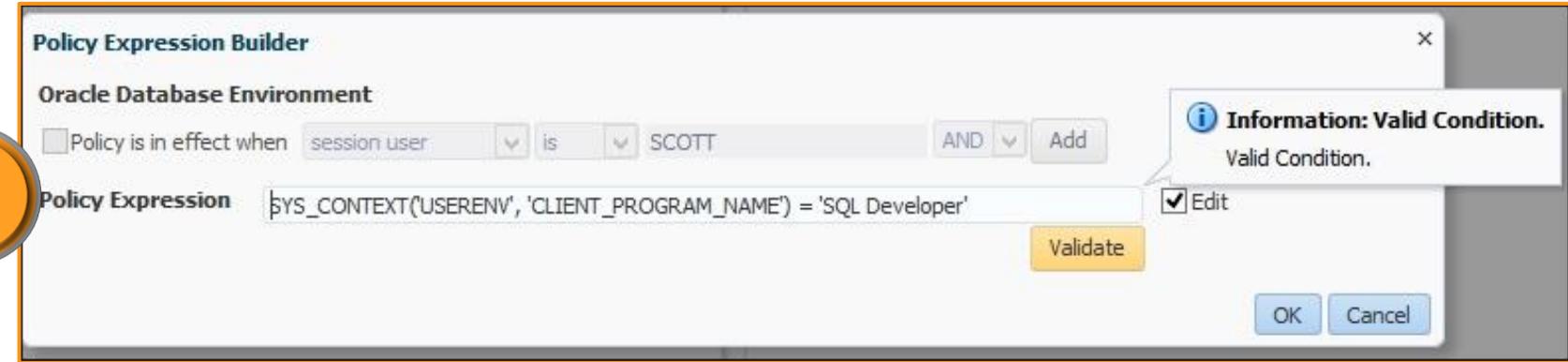
4

Name the policy, select roles, and optionally write a description.  
Click on **Build Context Expression**

# *Creating combined analysis policy*

Manually write the policy expression. Click on the **Validate** button and then on the **OK** button

5



# Creating combined analysis policy

The screenshot shows the Oracle Database Vault Policies page. At the top, there is a confirmation message: "Privilege Analysis policy COM\_PRIV\_POL has been created successfully." Below this, the "Privilege Analysis" section provides a brief overview of what it does. The main area is titled "Policies" and contains a table with the following data:

Policy	Active	Type	First Start Time	Last End	Capture
ALL_PRIV_POL		Database			
COM_PRIV_POL		Role and Context			
CONT_PRIV_POL		Context			
ORA\$DEPENDENCY		Database			
ROLE_PRIV_POL		Role			

A large orange arrow points from the top right towards the confirmation message.

6

You should receive a confirmation message and see your newly created policy listed in the table.

## *Starting and stopping privilege analysis*

- To start capturing privileges, you'll enable privilege analysis policies you created previously.
- You'll need an existing user who can manage privilege analysis policies (has the CAPTURE\_ADMIN role and the SELECT ANY DICTIONARY privilege), for example, the SYSTEM user.

## *Starting and stopping privilege analysis*

1. Start capturing privileges according to the policy **ALL\_PRIV\_POL**.
2. Then, execute several statements as the user **ALAN**.
3. The point of those statements is to generate records.

## *Starting and stopping privilege analysis*

4. Stop capturing the privilege usage.
5. Populate `DBA_USED_XXX` and `DBA_UNUSED_XXX` data dictionary views.

# Starting and stopping privilege analysis

Log in to the database as **SYSTEM** user or a user who has appropriate privileges (**CAPTURE\_ADMIN** role and **SELECT ANY DICTIONARY** privilege).

1

Login to EM as a user who has appropriate privileges and select **Privilege Analysis** from **Security** drop-down menu:

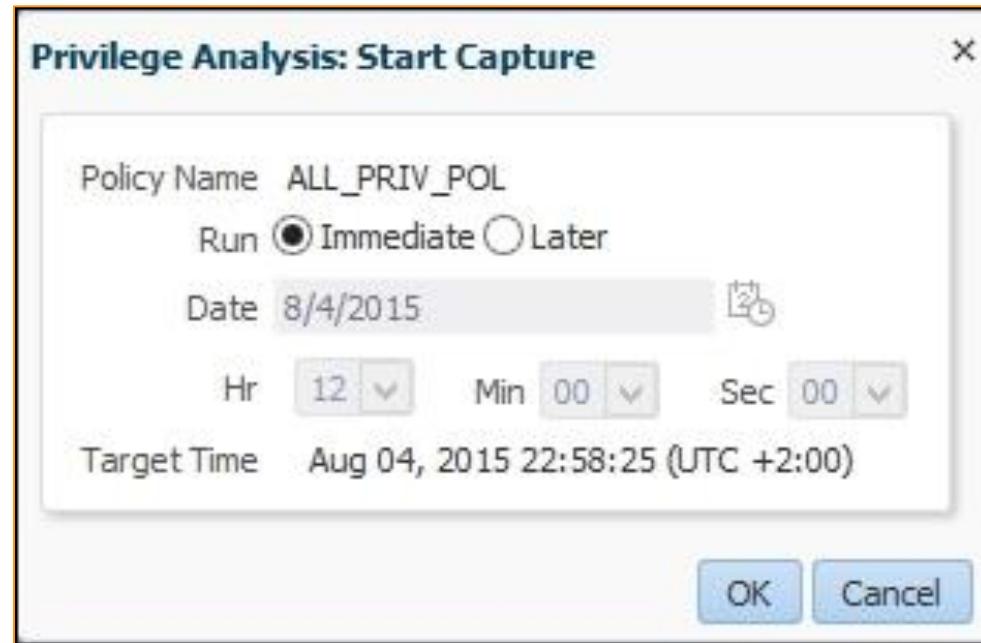
2

Policies					
Actions		View		Actions	
		Create	Start Capture	Stop Capture	Generate Report
Policy	Active	Type	First Start Time	Last End Time	Cap
ALL_PRIV_POL		Database			
COM_PRIV_POL		Role and Context			
CONT_PRIV_POL		Context			
ORA\$DEPENDENCY		Database			
ROLE_PRIV_POL		Role			

3

Select the database policy and click on the **Start Capture** button

# Starting and stopping privilege analysis



4

You can either start capture immediately or schedule it. Leave the defaults and click on the **OK** button

# Starting and stopping privilege analysis

You should receive a confirmation message and see that your policy is active

5

The screenshot shows the Oracle Database Control interface with a confirmation message and a policies list.

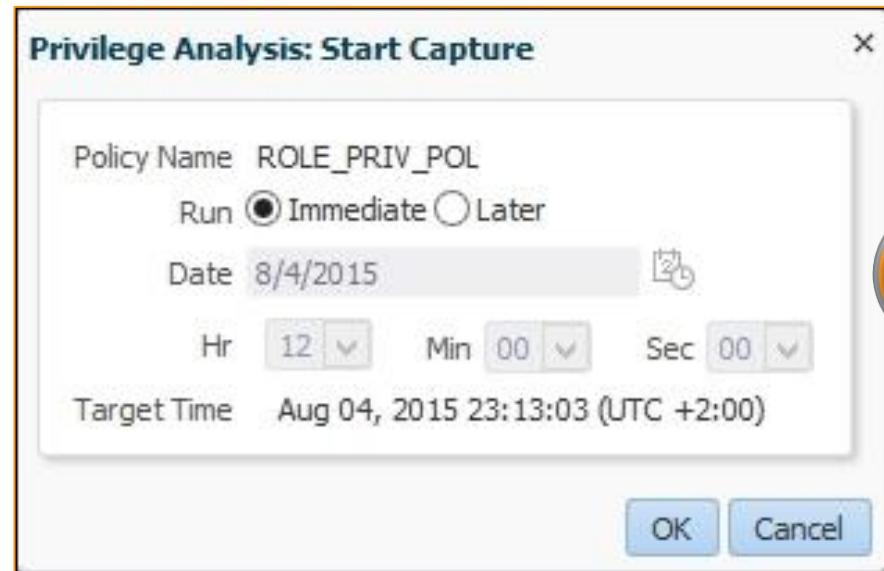
**Confirmation:**  
Started capture for Privilege Analysis policy ALL\_PRIV\_POL.

**Privilege Analysis:**  
Privilege Analysis enables you to find information about privilege usage for a database according to a specified condition, such as privileges to run an application module or privileges used in a given user session. It analyzes both system privileges and object privileges. To monitor the privileges used for a user's action, you must create and enable a Privilege Analysis policy. Afterward, you can generate a report that describes the used and unused privileges and then from there, revoke (and regrant) privileges as necessary. However, you cannot use privilege analysis to analyze the use of SYS user privileges. Privilege analysis is licensed as part of Oracle Database Vault, but you do not need to enable Database Vault to use it.

**Policies:**

Actions	View	Create	Start Capture	Stop Capture	Generate Report	Delete	Capture Scope
Policy	Active	Type	First Start Time	Last End Time			
ALL_PRIV_POL	↑	Database	Aug 04, 2015 11:00 PM				
COM_PRIV_POL		Role and Context					
CONT_PRIV_POL		Context					
ORA\$DEPENDENCY		Database					
ROLE_PRIV_POL		Role					

# Starting and stopping privilege analysis



6

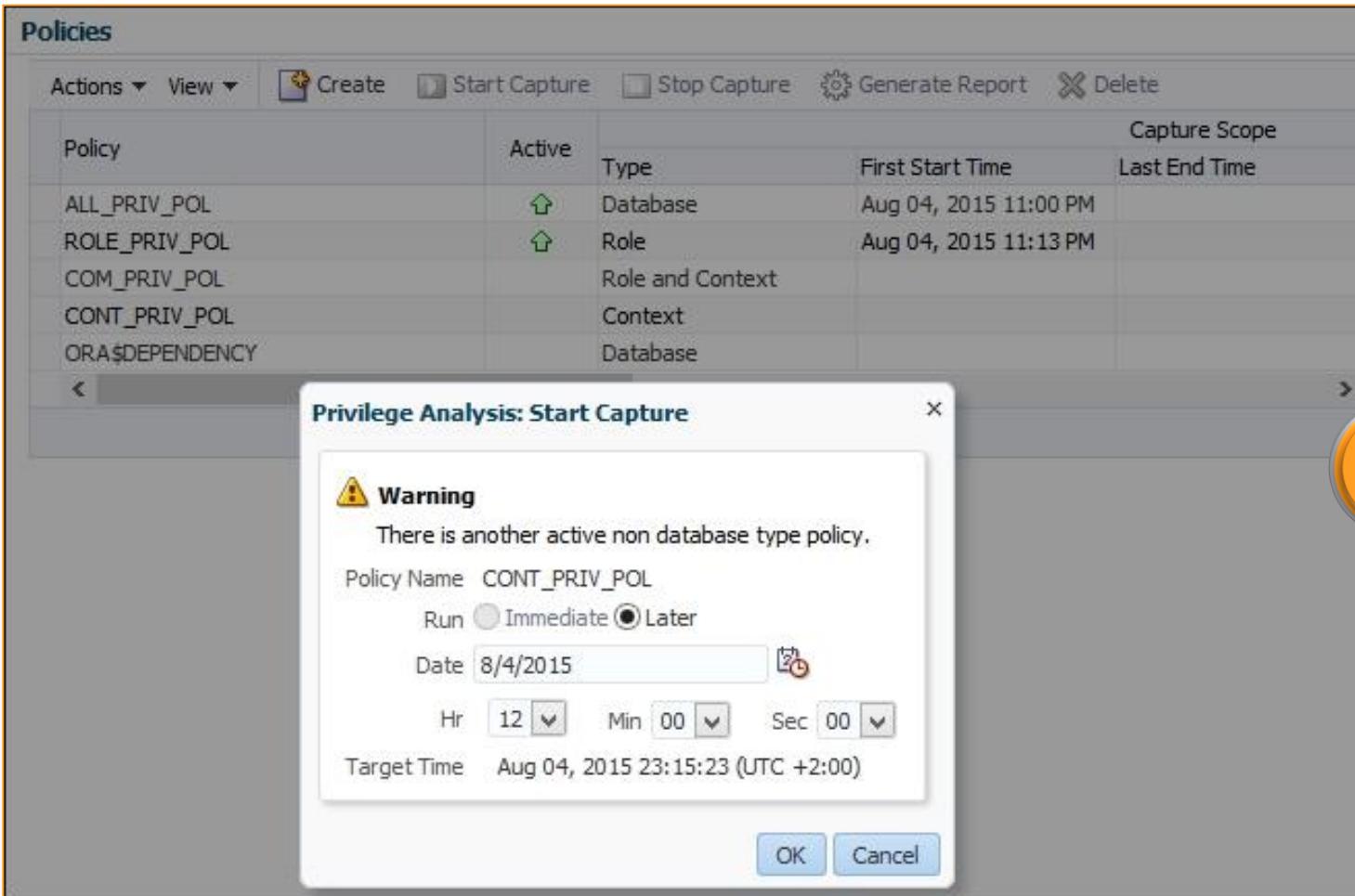
Select the role policy and click on **Start Capture**

You should see under the **Policies** section that both policies are active

7

Policies				
Actions ▾ View ▾		Create	Start Capture	Stop Capture
Policy	Active	Type	First Start Time	Capture Scope
ALL_PRIV_POL	↑	Database	Aug 04, 2015 11:00 PM	
ROLE_PRIV_POL	↑	Role	Aug 04, 2015 11:13 PM	
COM_PRIV_POL		Role and Context		
CONT_PRIV_POL		Context		
ORA\$DEPENDENCY		Database		

# Starting and stopping privilege analysis



8

Verify that you can't enable another non-database policy while role policy is active. Select **CONT\_PRIV\_POL** and click on the **Start Capture** button. You'll receive warning message, and you'll only be able to schedule job to run at later point in time

# Starting and stopping privilege analysis

To disable capture, select an active policy (for example, ALL\_PRIV\_POL) and click on the button Stop Capture

9

Policies	Actions	View	Create	Start Capture	Stop Capture	Generate Report	Delete
Policy	Active	Type	Stop Capture Time	Capture Scope	Last End Time	Total Capture Duration	
ALL_PRIV_POL		Database	Aug 04, 2015 11:00 PM			7hr 26Min	
ROLE_PRIV_POL		Role	Aug 04, 2015 11:13 PM			7hr 13Min	
COM_PRIV_POL		Role and Context					
CONT_PRIV_POL		Context					
ORA\$DEPENDENCY		Database					

Privilege Analysis: Stop Capture

Policy Name: ALL\_PRIV\_POL

Run  Immediate  Later

Date: 8/5/2015

Hr: 12 Min: 00 Sec: 00

Target Time: Aug 05, 2015 06:28:58 (UTC +2:00)

Generate Report

OK Cancel

10

Choose to immediately stop capture and tick generate report checkbox. Click on the OK button

# Starting and stopping privilege analysis

The screenshot shows the Oracle Database Control interface with the 'Policies' tab selected under 'Privilege Analysis'. A confirmation message box is displayed at the top left:

**Confirmation**  
Capture for Privilege Analysis policy ALL\_PRIV\_POL has been stopped and Oracle Scheduler Job "I\_ALL\_PRIV\_POL" has been submitted to generate report.

**Privilege Analysis**

Privilege Analysis enables you to find information about privilege usage for a database according to a specified condition, such as privileges to run an application module or privileges used in a given user session. It analyzes both system privileges and object privileges. To monitor the privileges used for a user's action, you must create and enable a Privilege Analysis policy. Afterward, you can generate a report that describes the used and unused privileges and then from there, revoke (and regrant) privileges as necessary. However, you cannot use privilege analysis to analyze the use of SYS user privileges. Privilege analysis is licensed as part of Oracle Database Vault, but you do not need to enable Database Vault to use it.

**Policies**

Policy	Active	Capture Scope			
		Type	First Start Time	Last End Time	Total Capture Duration
ROLE_PRIV_POL	Role	Role	Aug 04, 2015 11:13 PM		7hr 17Min
COM_PRIV_POL	Role and Context	Role and Context			
CONT_PRIV_POL	Context	Context			
ORA\$DEPENDENCY	Database	Database			
ALL_PRIV_POL	Database	Database	Aug 04, 2015 11:00 PM	Aug 05, 2015 06:30 AM	7hr 30Min

11

You should receive confirmation message that capture has been stopped and that job has been submitted

# Starting and stopping privilege analysis

Refresh page

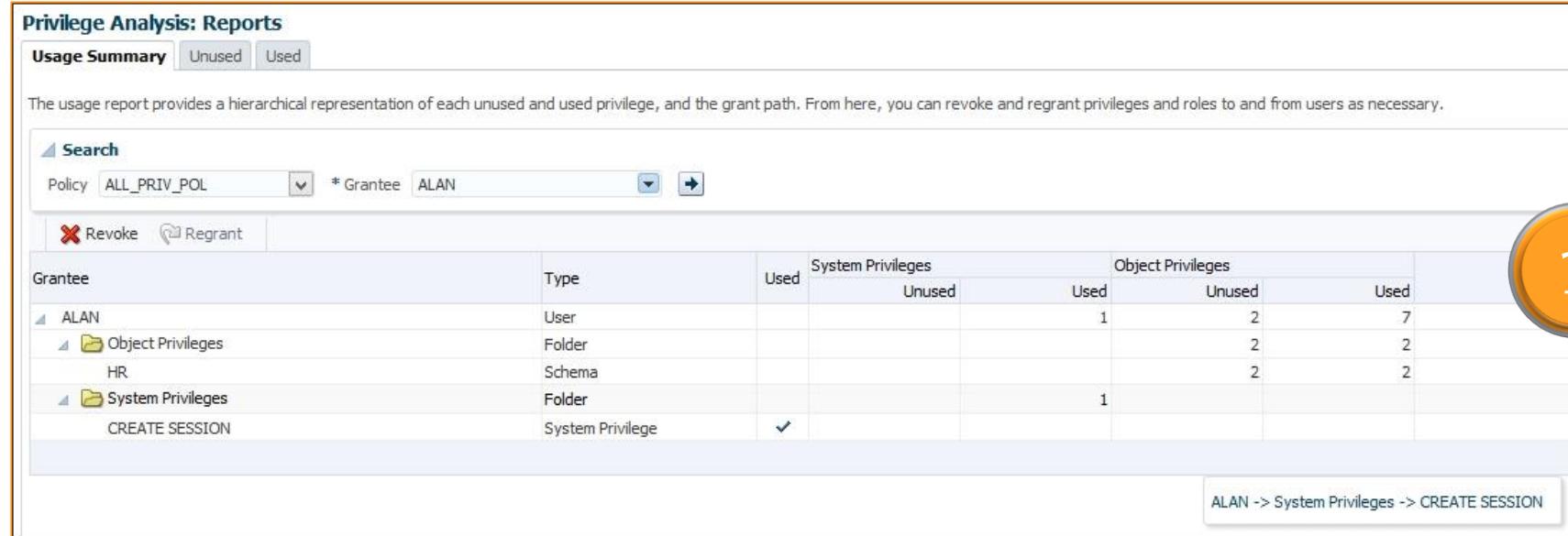
12

Policies												
Policy	Active	Type	Capture Scope			Total Capture Duration	Unused Privileges			Scheduled Jobs	Most Recent Job	
			First Start Time	Last End Time	Users		System	Object	Public		Status	Type
ROLE_PRIV_POL	↑	Role	Aug 04, 2015 11:13 PM			7Hr 18Min						
COM_PRIV_POL		Role and Context										
CONT_PRIV_POL		Context										
ORA\$DEPENDENCY		Database					19					
ALL_PRIV_POL		Database	Aug 04, 2015 11:00 PM	Aug 05, 2015 06:30 AM	20	7Hr 30Min	492	13683			✓	Report Generation

## *Reporting on used system privileges*

- You'll view collected data about the usage of system privileges during a capture interval.
- You'll need an existing user who can create a privilege analysis policy (has the CAPTURE\_ADMIN role and the SELECT ANY DICTIONARY privilege), for example, the SYSTEM user.

# Reporting on used system privileges



The usage report provides a hierarchical representation of each unused and used privilege, and the grant path. From here, you can revoke and regrant privileges and roles to and from users as necessary.

**Search**

Policy ALL\_PRIV\_POL \* Grantee ALAN

**Revoke** **Regrant**

Grantee	Type	Used	System Privileges		Object Privileges		
			Unused	Used	Unused	Used	
ALAN	User		1	2	7		
Object Privileges	Folder			2	2		
HR	Schema			2	2		
System Privileges	Folder		1				
CREATE SESSION	System Privilege	✓					

ALAN -> System Privileges -> CREATE SESSION

1

In EM 19c, after you have generated the report, select the policy and from Actions dropdown menu, select Reports. The Usage Summary report will open

# Reporting on used system privileges

Click on the tab **Used** and choose **All** for Match radio button, **Policy: ALL\_PRIV\_POL**, **User Name: ALAN**, and click on the **Search** button.

Privilege Analysis: Reports

Usage Summary    Unused    **Used**

Used Privileges

Search

Match:  All  Any

Policy: ALL\_PRIV\_POL

User Name: ALAN

System Privileges:

Object Name:

Advanced    Saved Search    UsedPrevVOCriteria

Search    Reset    Save...

SL#	Policy	User Name	Used Role	System Privileges	Path	Admin/Grant Option
1	ALL_PRIV_POL	ALAN	ALAN	CREATE SESSION	ALAN	False
2	ALL_PRIV_POL	ALAN	PUBLIC		PUBLIC	False
3	ALL_PRIV_POL	ALAN	PUBLIC		PUBLIC	False
4	ALL_PRIV_POL	ALAN	PUBLIC		PUBLIC	False
5	ALL_PRIV_POL	ALAN	PUBLIC		PUBLIC	False
6	ALL_PRIV_POL	ALAN	ALAN		ALAN	False
7	ALL_PRIV_POL	ALAN	PUBLIC		PUBLIC	False
8	ALL_PRIV_POL	ALAN	ALAN		ALAN	False

Columns Hidden 10

# Reporting on used system privileges

The screenshot shows the 'Privilege Analysis: Reports' interface. The top navigation bar has tabs for 'Usage Summary', 'Unused', and 'Used'. The 'Used' tab is selected. On the left, there's a sidebar with 'Used Privileges' and a search section. The search section includes fields for 'Policy' (set to 'ROLE\_PRIV\_POL'), 'User Name' (set to 'NICK'), and 'System Privileges' and 'Object Name' fields. Below the search is a table with columns: SL#, Policy, User Name, Used Role, Path, System Privileges, Object Owner, and Object Name. The table contains four rows corresponding to the policy 'ROLE\_PRIV\_POL' assigned to user 'NICK'. The first row grants 'CREATE SESSION' privilege. The second and third rows grant 'CREATE TABLE' privilege. The fourth row grants 'SELECT ANY TABLE' privilege on the 'CUSTOMERS' table.

SL#	Policy	User Name	Used Role	Path	System Privileges	Object Owner	Object Name
1	ROLE_PRIV_POL	NICK	EM_EXPRESS_BASIC	NICK,DBA,EM_EXPRESS_ALL,EM_EXPRESS_BASIC	CREATE SESSION		
2	ROLE_PRIV_POL	NICK	DATAPUMP_EXP_FULL_DATABASE	NICK,DBA,DATAPUMP_EXP_FULL_DATABASE,EXP_FULL_D...	CREATE TABLE		
3	ROLE_PRIV_POL	NICK	DATAPUMP_EXP_FULL_DATABASE	NICK,DBA,DATAPUMP_EXP_FULL_DATABASE	CREATE TABLE		
4	ROLE_PRIV_POL	NICK	OLAP_DBA	NICK,DBA,OLAP_DBA	SELECT ANY TABLE	OE	CUSTOMERS

3

If you haven't generated report for the role policy, do it now and return to this tab (the **Used** tab). Find all records generated by **ROLE\_PRIV\_POL** for user **Nick** (who has a DBA role).

## *Reporting on used object privileges*

- You'll view collected data about the usage of object privileges during the capture interval.
- You'll need an existing user who can create a privilege analysis policy (has the CAPTURE\_ADMIN role and the SELECT ANY DICTIONARY privilege), for example, the SYSTEM user.

# Reporting on used object privileges

Privilege Analysis: Reports

Usage Summary    Unused    **Used**

Used Privileges

**Search**

Match  All  Any

Policy ALL\_PRIV\_POL

User Name ALAN

System Privileges

Object Name

Advanced    Saved Search    UsedPrevVOCriteria

Search    Reset    Save...

View    Export to Spreadsheet

SL#	Policy	User Name	Object Privileges	Used Role	Path	Object Owner	Object Name
1	ALL_PRIV_POL	ALAN	SELECT	ALAN	ALAN	HR	EMPLOYEES
2	ALL_PRIV_POL	ALAN	DELETE	ALAN	ALAN	HR	EMPLOYEES
3	ALL_PRIV_POL	ALAN	SELECT	PUBLIC	PUBLIC	SYS	DUAL
4	ALL_PRIV_POL	ALAN	SELECT	PUBLIC	PUBLIC	SYS	DUAL
5	ALL_PRIV_POL	ALAN	EXECUTE	PUBLIC	PUBLIC	SYS	DBMS_APPLICATION_INFO
6	ALL_PRIV_POL	ALAN	EXECUTE	PUBLIC	PUBLIC	SYS	DBMS_OUTPUT
7	ALL_PRIV_POL	ALAN	SELECT	PUBLIC	PUBLIC	SYSTEM	PRODUCT_PRIVS
8	ALL_PRIV_POL	ALAN		ALAN	ALAN		

Columns Hidden 9

1

In EM 19c, after you have generated the report, select the policy, and from **Actions** drop-down menu, select **Reports**. The **Usage Summary** report will open. Click on the **Used** tab and verify that the user **Alan** has used the **SELECT** and **DELETE** privileges while **ALL\_PRIV\_POL** has been active

## *Reporting on unused system privileges*

- You'll view the collected data about the unused system privileges during the capture interval.
- You'll need an existing user who can create a privilege analysis policy (has the CAPTURE\_ADMIN role and the SELECT ANY DICTIONARY privilege), for example, the SYSTEM user.

## *Reporting on unused system privileges*

- To view report about the unused system privileges in EM19c, see instructions to view the used system privileges and under **Privilege Analysis: Reports**, choose the **Unused** tab instead of the **Used** tab.

## *Reporting on unused object privileges*

- You'll view collected data about the unused object privileges during the capture interval.
- You'll need an existing user who can create a privilege analysis policy (has the CAPTURE\_ADMIN role and the SELECT ANY DICTIONARY privilege), for example, the SYSTEM user.

# Reporting on unused object privileges

Privilege Analysis: Reports

Usage Summary    **Unused**    Used

Unused Privileges

Search Advanced Saved Search DbaUnusedPrvsVOCriteria

Match:  All  Any

Policy: ALL\_PRIV\_POL

Grantee: ALAN

System Privileges:

Object Owner:

Object Name:

Search Reset Save...

View Export to Spreadsheet

SL#	Policy	Grantee	Grantee Type	System Privileges	Object Privileges	Object Owner	Object Name	Object Type	Path
1	ALL_PRIV_POL	ALAN	USER		INSERT	HR	EMPLOYEES	TABLE	ALAN
2	ALL_PRIV_POL	ALAN	USER		UPDATE	HR	EMPLOYEES	TABLE	ALAN

Columns Hidden 3

SL#	Policy	Grantee	Grantee Type	System Privileges	Object Privileges	Object Owner	Object Name	Object Type	Path
1	ALL_PRIV_POL	ALAN	USER		INSERT	HR	EMPLOYEES	TABLE	ALAN
2	ALL_PRIV_POL	ALAN	USER		UPDATE	HR	EMPLOYEES	TABLE	ALAN

1

In EM 19c, after you have generated the report, select the policy, and from **Actions** drop-down menu, select **Reports**. The **Usage Summary** report will open. Click on the **Unused** tab and verify that the user **Alan** hasn't used the **INSERT** and **UPDATE** privileges while **ALL\_PRIV\_POL** has been active.

## *How to revoke unused privileges*

- You can manually revoke unused privileges one by one from users, write your own scripts to complete that task, or use Enterprise Manager Cloud Control 19c.
- You'll use EM19c to efficiently revoke unused privileges based on reports you generated in the previous recipes.

## *How to revoke unused privileges*

- In EM 19c, there is another excellent option to create a new role based on privilege analysis results.
- This way, you won't change an existing role (and affect other users and roles who have that role), but create a new one and afterwards revoke the old role and grant a newly created one.

# How to revoke unused privileges

## Privilege Analysis: Create Role

Use this feature to create a new role from Privilege Analysis results. The role can have used or unused system/object privileges and roles.

Note: If the logged in user does not have sufficient privileges, then the user will be prompted to provide SYSDBA credentials.

\* Policy Name ALL\_PRIV\_POL

\* Role Name

Unused

Used

Directly Granted System Privileges  All  None  Customize

Directly Granted Object Privileges  All  None  Customize

Directly Granted Roles  All  None  Customize

Cancel

OK

1

You can select it from the **Actions** menu (**Create Role**). The configuration part of a process for creating a new role is shown

## *Dropping the analysis*

- In this recipe, you'll drop an existing privilege analysis policy.
- It has to be disabled before dropping; otherwise, you'll receive an error.
- You'll need an existing user who can manage privilege analysis policies (has the **CAPTURE\_ADMIN** role and the **SELECT ANY DICTIONARY** privilege), for example, the **SYSTEM** user and an existing privilege analysis policy.

## *Dropping the analysis*

- In EM 19c under the **Policies** section, select policy you want to drop and click on the **Delete** button.



# Oracle Database 19c Security

## *Transparent Data Encryption*

# *Transparent Data Encryption*

We will cover the following tasks:

- Configuring a keystore location in sqlnet.ora
- Creating and opening the keystore
- Setting a master encryption key in a software keystore
- Column encryption - adding a new encrypted column to a table
- Column encryption - creating a new table that has encrypted column(s)
- Using salt and MAC
- Column encryption - encrypting the existing column
- Autologin keystore
- Encrypting tablespace
- Rekeying
- Backup and recovery

- Encryption is a very important security mechanism used to enforce confidentiality of data.
- There are two types of encryption that can be used in the Oracle Database.
- The first type is application-based encryption, which is implemented using the **DBMS\_CRYPTO** PL/SQL package.

- The second type is **Transparent Data Encryption (TDE)**.
- TDE is a part of Advanced Security option of Oracle Database Enterprise Edition.
- It can be used to encrypt data in rest (table columns and tablespaces inside the database) and in transit (network, **Recovery Manager (RMAN)** backups, and Data Pump Exports).

- The word **transparent** in Transparent Data Encryption means that application is not aware that data is encrypted in any way.
- The application will never see the encrypted data-if user is not authorized to see the data, error will be shown.
- The only way that a user will see encrypted data is if he or she tries to avoid Oracle Database Access Controls, by reading data files directly.

- TDE should never be used as a mechanism of access control.
- There are two types of TDE: column and tablespace.

- In column encryption, only user-selected columns (in user-selected tables) are encrypted.
- This encryption type is more suitable for systems with small number of columns that need to be encrypted.
- Encrypting large number of columns can lead to significant performance degradation.

- There are several encryption algorithms that can be chosen from: AES128, AES192, AES256, and 3DES168.
- The default one is AES192.
- Because these are block cyphers, each row that is going to be encrypted need to be padded to a multiple of 16 bytes.
- By default, salt and MAC are used.

There are several restrictions of column encryption:

- Foreign key constraints are not supported because each table has a different table key
- B-Tree indexes are not supported when using salt
- Bitmap indexes are not supported

- Transportable tablespaces are not supported
- Synchronous **Change Data Capture (CDC)** is not supported
- External Large Objects (LOBs) are not supported
- **SYS** schema objects cannot be encrypted

- Tablespace encryption is the second type of TDE, which has better performance and has fewer restrictions.
- This type of TDE is usually more suitable for systems that need to encrypt large portion of data in the database.
- Using this type, all data that resides inside encrypted tablespace is encrypted (no restrictions on data types).

- 
- Encryption/decryption is performed on the I/O level, so performance overhead can be expected to be seen on that level.
  - Tablespace encryption doesn't require additional storage.

Unlike column encryption, tablespace encryption supports the following:

- Foreign keys
- Bitmap indexes
- Transportable tablespaces (as long as platforms are of the same endian and the same keystore exists on both locations)
- All data types

However, there are still some limitations:

- BFILE cannot be encrypted
- External tables cannot be encrypted
- UNDO tablespace cannot be encrypted

- TEMP tablespace cannot be encrypted
- SYSTEM tablespace cannot be encrypted
- Key for tablespace cannot be rekeyed

- TDE uses two-tier key architecture.
- For column encryption, columns are encrypted using column (also known as table) keys.
- There is only one key per table regardless of number of columns that are encrypted in that particular table.

- For tablespace encryption, tablespaces are encrypted using tablespace keys.
- Both table and tablespace keys are stored in data dictionary inside Oracle Database.
- These keys are encrypted using a master key.
- There is only one master key per database.

- This master key is stored in a keystore outside the Oracle Database.
- This keystore can be a software keystore or a hardware keystore.
- A keystore is secured by a password, which is used during maintenance operations.

## *Configuring keystore location in sqlnet.ora*

- You're going to configure the location of a software keystore in a regular file system.

## *Creating and opening the keystore*

- You're going to create a password-based keystore.
- Open it and learn to check its status.
- It is assumed that the keystore location is already configured.
- You'll grant, as the **SYS** user, administer key management privilege, or **SYSKYM** administrative privilege to an existing user (for example, **maja**).

## *Creating and opening the keystore*

- Create a new wallet, which is a file with **.p12** extension, in a wallet directory.
- Open the keystore.
- It will remain open until you manually close it.

## *Creating and opening the keystore*

- Verify that the keystore has been successfully created by checking that the file **ewallet.p12** exists in the directory you specified as a keystore location (**ENCRYPTION\_WALLET\_LOCATION** parameter in **sqlnet.ora**).
- You should get the similar result to the one shown next:

```
[oracle@dbhost Desktop]$ ls -l /u01/app/oracle/admin/ora12cR1/wallet
total 4
-rw-r--r--. 1 oracle oinstall 2408 Oct 11 23:54 ewallet.p12
```

## *Creating and opening the keystore*

- To view the status of the keystore execute the following statements:

```
$ sqlplus / as syskmSQL>
SELECT STATUS, WALLET_TYPE FROM V$ENCRYPTION_WALLET;
```

- You should get the similar result to the one shown next:

SQL> select status, wallet_type from v\$encryption_wallet;	
STATUS	WALLET_TYPE
OPEN_NO_MASTER_KEY	PASSWORD

## *Setting master encryption key in software keystore*

- You're going to create the first master key for the password-based software keystore you created and opened previously.

## *Setting master encryption key in software keystore*

- It is assumed that software keystore is already opened.
- You'll need an existing user who has the **SYSKM** administrative or administer key management privilege (for example, **maja**)

## *Setting master encryption key in software keystore*

- The **WITH BACKUP** clause instructs Oracle Database to create a backup of a keystore before the creation of a master key.
- This backup is created in the same directory where keystore resides and is created in the form **ewallet\_timestamp.p12**.

# *Setting master encryption key in software keystore*

- Verify the status of the keystore

```
SQL> select status, wallet_type from v$encryption_wallet;
```

STATUS	WALLET_TYPE
OPEN	PASSWORD

## *Column encryption - adding new encrypted column to table*

- You'll add a new column, which will be encrypted using a nondefault encryption algorithm, to an existing table.
- It is assumed that a keystore is opened and a master key is created.

## Column encryption - creating new table that has encrypted column(s)

- You're going to learn to use TDE column encryption to encrypt columns in a newly created table.
- It is assumed that a keystore is opened and a master key is created.

## *Using salt and MAC*

- You'll understand when you should use salt and MAC.
- It is assumed that a keystore is opened and a master key is created.

## *Using salt and MAC*

- Encrypt the `last_name` column using the AES256 algorithm with salt and use MAC
- Encrypt the `cust_street_address` column using the AES256 algorithm with no salt and use MAC

## *Using salt and MAC*

- In general, you have to use same encryption algorithm for all encrypted columns at the same time.
- You can choose a **SALT** option on the encrypted column level in a table, but you have to choose either the **Mac** or **NOMAC** option on a table level.

## *Using salt and MAC*

- To understand why salt is important, let's consider a basic scenario that doesn't use salt.
- For example, if we have 100 rows and they contain only values **A**, **B**, **C**, and **D**, this will mean that there are only 4 different values in 100 rows.

## *Using salt and MAC*

- If we know that value **A** exists in 3 rows, value **B** exists in 20, value **C** exists in 30, and value **D** exists in 47 rows, we can then check **ciphertexts**.
- By evaluating it, we can find that one ciphertext that exists in 3 rows will be value **A**, one that exists in 20 rows will be value **B**, and so on.
- In order to avoid this problem, we can introduce salt.

## *Using salt and MAC*

- Salt is used to ensure that each encrypted row has different ciphertext regardless of number of same values in plaintext rows.
- In our previous example, if we used salt, even though there were only 4 different plaintext values in 100 rows, there will be 100 different ciphertext values in 100 rows.
- Which will be almost impossible for attacker to presume which value corresponds to which row.

## *Using salt and MAC*

- Consequently, there is no need for salt if plaintext values are unique. There is additional storage cost of 16 bytes per row for salt.
- Salt cannot be used on indexed columns.

## *Using salt and MAC*

- MAC (short for **Message Authentication Code**) is a hash value computed on encrypted data, which is used for data integrity verification.
- There is the additional storage cost of 20 bytes per row.

## *Using salt and MAC*

- TDE column restriction

```
SQL> alter table hr.test modify (ID ENCRYPT);
alter table hr.test modify (ID ENCRYPT)
*
ERROR at line 1:
ORA-28338: Column(s) cannot be both indexed and encrypted with salt
```

- It is shown that column ID (which is primary key) cannot be encrypted with salt.

## *Using salt and MAC*

- Encrypted primary key with no salt

```
SQL> alter table hr.test modify (ID ENCRYPT NO SALT);  
Table altered.
```

- After changing attribute to **NOSALT**, the primary key column is successfully encrypted.

## *Column encryption - encrypting existing column*

- It is common case that organizations first create database and later decide that they want to implement encryption.
- You're going to encrypt an existing column using TDE column encryption.
- It is assumed that a keystore is opened and a master key is created.

## *Column encryption - encrypting existing column*

- You can't use TDE column encryption to encrypt column, which is a foreign key.
- If you need to encrypt that kind of column, use TDE tablespace encryption.

# Column encryption - encrypting existing column

\$ sqlplus oe

1

Connect to the database as a user who can select data from a table, for example, **OE.ORDERS** (for example, the **oe** user):

Select data from the foreign key column you want to encrypt, just to verify the user can view it.

2

```
SQL> select distinct(customer_id) from oe.orders  
2  order by order_total desc  
3  fetch first 8 rows only;
```

CUSTOMER\_ID

-----  
147  
150  
149  
148  
108  
122  
117  
104

8 rows selected.

# Column encryption - encrypting existing column

SQL> connect maja

3

Connect to the database as a user who has administer key privilege or **SYSKM** privilege (for example, **maja**)

Encrypt the **customer\_id** column in the **oe.orders** table using the default encryption algorithm (AES192).

4

## Column encryption - encrypting existing column

- A TDE column encryption restriction

```
SQL> ALTER TABLE OE.ORDERS MODIFY (CUSTOMER_ID ENCRYPT);
ALTER TABLE OE.ORDERS MODIFY (CUSTOMER_ID ENCRYPT)
*
ERROR at line 1:
ORA-28335: referenced or referencing FK constraint column cannot be encrypted
```

## *Auto-login keystore*

- Autologin keystore is a type of keystore that doesn't need to be manually opened.
- The local autologin keystore can be opened only from computer where it has been created.
- Autologin keystores have system-generated passwords.
- It is assumed that password-based software keystore is created.

## *Auto-login keystore*

- The **cwallet.sso** file is created.
- That file represents autologin keystore.

## *Encrypting tablespace*

- It is not possible to encrypt an existing tablespace using TDE tablespace encryption.
- You'll create a new encrypted tablespace.
- It is assumed that a keystore is opened and a master key is created.

# Encrypting tablespace

- Create an encrypted tablespace TEST\_ENC.
- To find information about encrypted tablespaces, you can query the V\$ENCRYPTED\_TABLESPACES view.

```
SQL> connect / as sysdba
Connected.
SQL> desc v$encrypted_tablespaces
Name          Null?    Type
-----
TS#           NUMBER
ENCRYPTIONALG VARCHAR2(7)
ENCRYPTEDTS  VARCHAR2(3)
ENCRYPTEDKEY RAW(32)
MASTERKEYID   RAW(16)
BLOCKS_ENCRYPTED NUMBER
BLOCKS_DECRYPTED NUMBER
CON_ID        NUMBER

SQL> select encryptedts, encryptionalg from v$encrypted_tablespaces;
ENC ENCRYPT
-----
YES AES192
```

## *Encrypting tablespace*

- You can import existing tables into encrypted tablespace using Oracle Data Pump.
- Another option is to use SQL statements, for example, **CTAS** (short for **CREATE TABLE AS**).

## *Rekeying*

- You can change (rekey) a master key and table keys.
- You cannot rekey tablespace keys.
- It is assumed that a keystore is opened and a master key is created.

# *Rekeying*

- When you change a table key, all encrypted data in the oe.customers table is decrypted and then encrypted using the new table key and the new encryption algorithm.
- If you just want to change key and use the same algorithm as before, syntax for rekeying is:

```
ALTER TABLE table_name REKEY;
```

## *Rekeying*

- Create a backup of the keystore and create a new master key in the keystore.
- Old master keys are held in the keystore.
- It is extremely important to have backup of the keystore.

RMAN supports three encryption modes:

- Transparent mode
- Password mode
- Dual mode

You're going to learn to create encrypted backups using RMAN.

- If a backup is created in transparent mode, it can be restored only by using a key that is used to create the backup.
- If the backup is created in password mode, it can be restored only by using a password that is provided during the backup.
- If the backup is created in dual mode, it can be restored by either key that is stored in the external keystore or the password that is provided during the backup.



# Oracle Database 19c Security

## *Unified Auditing*

In this chapter, we will cover the following tasks:

- Enabling the Unified Auditing mode
- Configuring whether loss of audit data is acceptable
- Which roles do you need to have to be able to create audit policies and to view audit data?
- Auditing RMAN operations
- Auditing Data Pump operations
- Auditing Database Vault operations
- Creating audit policies to audit privileges, actions, and roles under specified conditions
- Enabling an audit policy
- Finding information about audit policies and audited data
- Auditing application contexts
- Purging audit trail
- Disabling and dropping audit policies

- Unified Auditing is a new feature in Oracle Database 19c, and it introduces new auditing architecture.
- Some of the characteristics of unified auditing are:
  - A single audit trail
  - Being based on a read-only table
  - Extensible Audit Framework for additional columns
  - The separation of audit administration with new roles
  - Auditing performance is better, especially when used in the queued-write mode

## Traditional Audit Trails

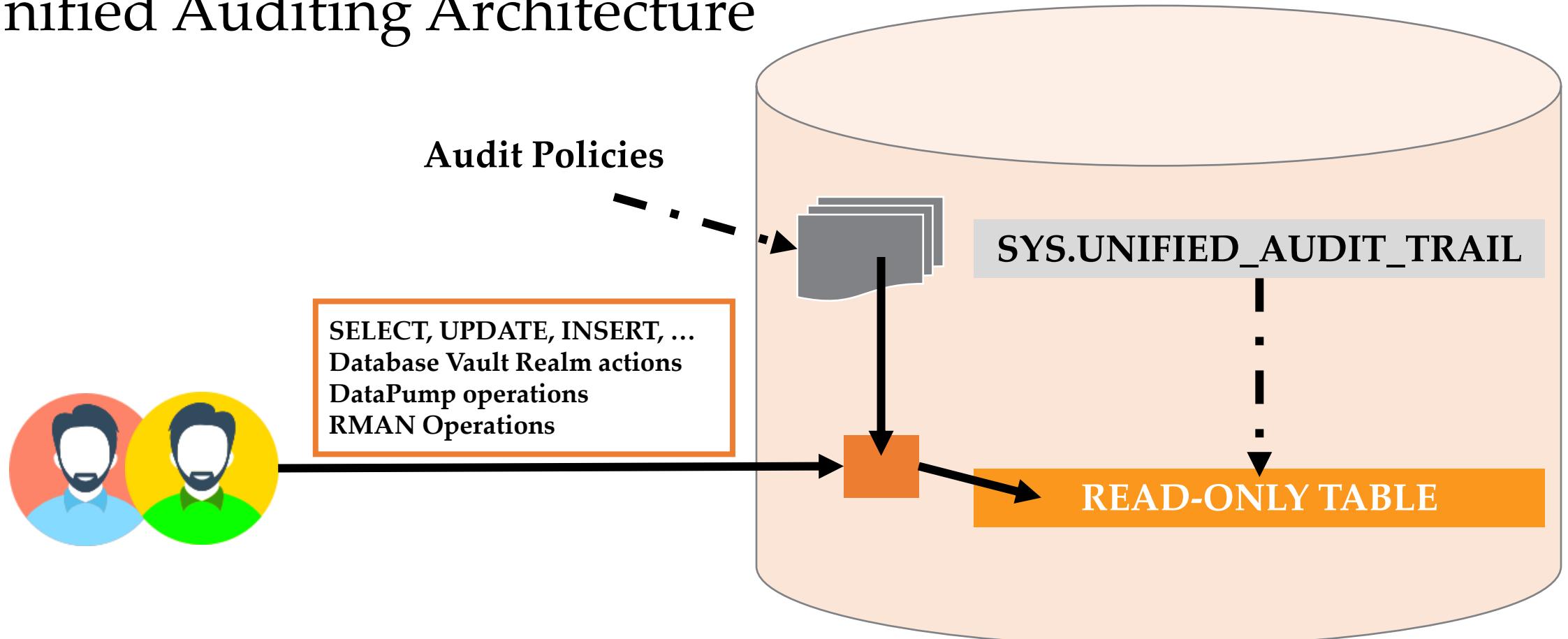
SYS.AUD\$  
SYS.FGA\_LOG\$  
V\$XML\_AUDIT\_TRAIL  
DBA\_COMMON\_AUDIT\_TRAIL  
DVSYS.AUDIT\_TRAIL\$  
OS files



## Unified Audit Trails

SYS.UNIFIED\_AUDIT\_TRAIL

- Unified Auditing Architecture



## *Enabling Unified Auditing mode*

- In Oracle Database 19c, unified auditing is not enabled by default.
- The process of enabling it is simple and equivalent to enabling of other database options.
- You'll need to shut down the database.

## *Enabling Unified Auditing mode*

- When database is upgraded to 19c, by default, it uses the traditional way of auditing.
- However, when you directly install a new database 19c, default auditing is set to mixed auditing mode.
- In both cases, the procedure to enable the unified auditing mode is the same.

## *Enabling Unified Auditing mode*

- After you enable the unified auditing mode, traditional auditing doesn't work anymore.
- Old audit instance parameters (`AUDIT_TRAIL`, `AUDIT_FILE_DEST`, `AUDIT_SYSLOG_LEVEL`, and `AUDIT_SYS_OPERATIONS`) are disregarded.
- Predefined unified audit policies that are enabled by default are:
  - `ORA_SECURECONFIG` (database versions: 12.1.0.1, 12.1.0.2)
  - `ORA_LOGON_FAILURES` (Oracle Database 12.1.0.2)

# *Enabling Unified Auditing mode*

- Predefined unified audit policies

Predefined audit policy	Oracle Database 12.1.0.1	Oracle Database 12.1.0.2
ORA_RAS_POLICY_MGMT	Yes	Yes
ORA_DATABASE_PARAMETER	Yes	Yes
ORA_RAS_SESSION_MGMT	Yes	Yes
ORA_ACCOUNT_MGMT	Yes	Yes
ORA_SECURECONFIG	Yes	Yes
ORA_LOGON_FAILURES	No	Yes
ORA_CIS_RECOMMENDATIONS	No	Yes
ORA_DV_AUDPOL	No	Yes

## *Enabling Unified Auditing mode*

- If you execute the following statement in both 12.1.0.1 and 12.1.0.2 database versions, as a user who has the **audit\_admin** or **dba** role:

```
SQL> select audit_option from audit_unified_policies  
where policy_name='ORA_SECURECONFIG'  
order by 1;
```

## *Enabling Unified Auditing mode*

- In Oracle Database 19cR1 **Standard Edition (SE)**, when you enable unified auditing mode and query the v\$option view to verify that it's enabled you may see the following:

PARAMETER	VALUE
<hr/>	
Unified Auditing	FALSE

## Configuring whether loss of audit data is acceptable

- You'll learn to set whether audit data is queued in memory or is immediately written to audit trail.
- You'll need an existing user who has the **audit\_admin** role.

## Configuring whether loss of audit data is acceptable

- The default value for a write mode is the queued-write mode.
- In this mode, audit data is stored in SGA queues and later automatically persisted in the read-only table in the **AUDSYS** schema in the **SYSAUX** tablespace.
- You can also manually flush content of memory queues to the disk:

```
SQL>EXEC SYS.DBMS_AUDIT_MGMT.FLUSH_UNIFIED_AUDIT_TRAIL;
```

Which roles do you need to have to be able to create audit policies and to view audit data?

- In this recipe, you're going to create two users (for example, **jack** and **jill**).
- Jack's job is to implement auditing requirements and to make sure that auditing is functioning properly.
- Jill is an auditor and her job is to analyze audit data.
- You'll need an existing user who has the DBA role.

Which roles do you need to have to be able to create audit policies and to view audit data?



**AUDIT\_ADMIN**

Manages audit configuration & audit trail.



**AUDIT\_VIEWER**

Analysis audit data.

Which roles do you need to have to be able to create audit policies and to view audit data?

- Grant the **AUDIT\_ADMIN** role to the newly created user **jack** because that role enables him to create, alter, enable, disable, and drop audit policies, view audit data, and manage the unified audit trail.
- Grant the **AUDIT\_VIEWER** role to the user **jill** because that role enables her to view audit data.

## Which roles do you need to have to be able to create audit policies and to view audit data?

- To test what can and can't be done as a user who has the **audit\_viewer** role, connect to the database as **jill** and try to create the unified audit policy **jill\_policy**:

```
SQL> connect jill
```

```
SQL> create audit policy jill_policy  
actions delete on oe.orders;  
actions delete on oe.orders  
*
```

```
ERROR at line 2:  
ORA-00942: table or view does not exist
```



*Which roles do you need to have to be able to create audit policies and to view audit data?*

- Even if you grant object privileges on the **oe.orders** table to **jill**, she won't be able to create unified audit policy because she doesn't have the **audit\_admin** role or the **AUDIT SYSTEM** system privilege:

```
SQL> conn maja
```

```
SQL> grant select,delete on oe.orders to jill;
```

```
SQL> connect jill
```

Which roles do you need to have to be able to create audit policies and to view audit data?

```
SQL> create audit policy jill_policy  
actions delete on oe.orders;  
actions delete on oe.orders  
*  
ERROR at line 2:  
ORA-01031: insufficient privileges
```

Which roles do you need to have to be able to create audit policies and to view audit data?

- Revoke select and delete on the **oe.orders** table from Jill:

```
SQL> connect maja
```

```
SQL> revoke select,delete on oe.orders from jill;
```

```
Revoke succeeded.
```

Which roles do you need to have to be able to create audit policies and to view audit data?

- Grant the AUDIT SYSTEM privilege to jill and again try to create the audit policy jill\_policy:

```
SQL> grant audit system to jill;  
SQL> connect jill
```

```
SQL> create audit policy jill_policy  
actions delete on oe.orders;  
Audit policy created.
```

Which roles do you need to have to be able to create audit policies and to view audit data?

- Drop the unified audit policy **jill\_policy** and revoke the **AUDIT SYSTEM** privilege from **jill**:

```
SQL> drop audit policy jill_policy;
```

Audit Policy dropped.

```
SQL> connect maja
```

```
SQL> revoke audit system from jill;
```

Which roles do you need to have to be able to create audit policies and to view audit data?

- View audit data:

```
SQL> connect jill
```

```
SQL> select dbusername, action_name from unified_audit_trail  
where unified_audit_policies='ORA_SECURECONFIG';
```

- Also, a user who has the audit\_viewer role can access information about defined and enabled unified audit policies.

## Which roles do you need to have to be able to create audit policies and to view audit data?

- Throughout this chapter, you'll use a user who has the **audit\_admin** role (for example, **jack**).
- So only test you'll do right now is to enable the predefined audit policy **ORA\_ACCOUNT\_MGMT** and then to disable it:

```
SQL> connect jack
```

```
SQL> audit policy ora_account_mgmt;  
Audit succeeded.  
SQL> noaudit policy ora_account_mgmt;  
Noaudit succeeded.
```

## *Auditing RMAN operations*

- You'll see that RMAN operations are audited by default.
- In this, we assume that database is in the ARCHIVELOG mode.
- You'll need an existing user who has the SYSBACKUP privilege (for example, `tom`) and an existing user who has the dba role (for example, `maja`).

## *Auditing RMAN operations*

- When the mixed or unified auditing mode is enabled, RMAN operations are automatically audited.
- Perform the backup of the tablespace **EXAMPLE**.
- Then intentionally cause a problem by removing the datafile.
- Afterwards, perform restore and recover RMAN operations.

## *Auditing RMAN operations*

- The whole point of the example is to execute several RMAN operations.
- In the `unified_audit_trail` data dictionary view, there are several columns that contain data pertaining to the RMAN events.
- Their names start with RMAN, so it's easy to find them.

# *Auditing RMAN operations*

- You should get similar result to this one:

DBUSERNAME	RMAN_OPERATION
TOM	Backup
TOM	List
TOM	Restore
TOM	Recover

## *Auditing Data Pump operations*

- You can audit Data Pump export, import, or both export and import operations by creating audit policies.
- You'll need an existing user who has the `audit_admin` role (for example, **jack**).

## Auditing Data Pump operations

- Also, it is assumed that directory for export operations (for example, `my_dir`) is created and a user (for example, `maja`) who is going to perform the Data Pump export has read and write privileges on the directory.

```
SQL> CREATE DIRECTORY my_dir AS '/u01/app/oracle/oradata/export';
```

```
SQL> grant read, write ON DIRECTORY my_dir to maja;
```

## *Auditing Database Vault operations*

- You'll learn to audit Oracle Database Vault events.
- You'll need to use Oracle Database 19c, which has Oracle Database Vault enabled and at least some of the components configured.
- Also, you'll need an existing user who has the **audit\_admin** role.

## *Auditing Database Vault operations*

- To create an audit policy that captures Oracle Database Vault events, specify **ACTIONS COMPONENT = DV <action> ON <object>**.
- Define the audit policy **dbv\_policy** that encapsulates the rules.
- In the unified audit trail, Oracle Database Vault-specific audit data is stored in the columns whose name starts with **DV\_**.

## *Auditing Database Vault operations*

- When you are using Oracle Database Vault, you can also additionally secure your auditing infrastructure by creating a realm around the **AUDIT\_ADMIN** and **AUDIT\_VIEWER** roles.
- This allows you to control who can grant those roles.

## *Creating audit policies to audit privileges, actions and roles under specified conditions*

- You will create several unified audit policies.
- You'll need two existing users:

- A user who has the `audit_admin` role (for example, `jack`)
- A user who has the `create session` privilege (for example, `john`)

## *Creating audit policies to audit privileges, actions and roles under specified conditions*

- Also, you should create the roles **hr\_role** and **oe\_role** as stated here and grant **hr\_role** to the user **john**.

```
SQL> create role hr_role;
SQL> grant select any table, create table to hr_role;
SQL> grant insert on hr.departments to hr_role;
SQL> create role oe_role;
SQL> grant drop any table to oe_role;
SQL> grant select, update on oe.orders to oe_role;
SQL> grant oe_role to hr_role;
SQL> grant hr_role to john;
```

## *Creating audit policies to audit privileges, actions and roles under specified conditions*

- When you create a unified audit policy, it is stored in the first-class object owned by **SYS** schema.
- Create the audit policy **my\_policy1**.
- Create the audit policy **role\_con\_policy**.

## *Creating audit policies to audit privileges, actions and roles under specified conditions*

- The Role **HR\_ROLE** has to exist at the time audit policy **role\_con\_policy** is created because if it doesn't exist you will get an error message:

ERROR at line 2:  
ORA-01919: role 'HR\_ROLE' does not exist

- Create the audit policy **hr\_policy**, the way it is written, audit records will be generated for **select**, **insert**, and **update** operations on all objects and for delete operations on **hr.departments**.

## *Creating audit policies to audit privileges, actions and roles under specified conditions*

- Create the audit policy **oe\_policy**, which will be used in order to audit all actions on table orders in oe schema.
- In Oracle Database 12.1.0.1 due to the bug, audit records for this policy are not generated.
- Workaround is to specify one by one actions instead of using keyword **ALL**.

## *Enabling audit policy*

- You will learn to use different options to enable unified audit policies.
- You'll need an existing user who has the **audit\_admin** role (for example, **jack**) and several other existing users (for example, **john**, **maja**, and **zoran**).

## *Enabling audit policy*

- Define **BY** list, which means that only user(s) listed on that list will be affected by the policy.
- Define that audit policy **hr\_policy** is applied to all users, but only successful operations will generate audit records.
- Define **EXCEPT** list.

## *Enabling audit policy*

- Enable audit policy using default options, which means that `role_con_policy` will affect all users for both successful and unsuccessful events.
- You can't use both `BY` and `EXCEPT` lists for the same policy statement.

## *Finding information about audit policies and audited data*

- You will view audited data and find information about unified audit policies.

## *Finding information about audit policies and audited data*

- You'll need three existing users:

- A user who has `audit_admin` role (for example, `jack`)
- A user who has `hr_role` and `oe_role` (for example, `john`), created in recipe `Creating audit policies to audit privileges, actions and roles under specified conditions`
- A user who has `admin_viewer` role (for example, `jill`)

- Also, you'll need to connect to the database as **SYS** user.

## *Auditing application contexts*

- You will configure auditing of information contained in an application context.
- You'll need an existing (or predefined) application context and a user who has the **audit\_admin** role (for example, **jack**).

# *Auditing application contexts*

- Audit custom application contexts:

NAMESPACE	ATTRIBUTE	USER_NAME
USERENV	HOST	JILL
USERENV	SERVICE_NAME	ALL USERS
USERENV	SESSION_USER	ALL USERS

## *Auditing application contexts*

- If needed, execute the following statement as the user **jack**:

```
SQL>EXEC SYS.DBMS_AUDIT_MGMT.FLUSH_UNIFIED_AUDIT_TRAIL;
```

# *Auditing application contexts*

- Audit records:

## APPLICATION\_CONTEXTS

```
(USERENV,SERVICE_NAME=SYS$USERS); (USERENV,SESSION_USER=JACK)  
(USERENV,SERVICE_NAME=SYS$USERS); (USERENV,SESSION_USER=JILL);  
(USERENV,HOST= dbhost.orapassion.com)
```

## *Auditing application contexts*

- To disable auditing of application contexts, you should use the **NOAUDIT** command:

```
SQL> connect jack
```

```
SQL> NOAUDIT CONTEXT NAMESPACE USERENV  
ATTRIBUTES HOST BY jill;
```

## *Purging audit trail*

- You can clean up audit data manually or by scheduling clean up job.
- You'll need a user who has the **audit\_admin** role (for example, **jack**).

## *Purging audit trail*

- By default, `USE_LAST_ARCH_TIMESTAMP` is set to `TRUE`.
- It means that only records created before that time will be deleted.
- If you set that parameter to `FALSE`, all records will be deleted.

## *Purging audit trail*

- In multitenant environment, use CONTAINER clause as well (CONTAINER => DBMS\_AUDIT\_MGMT.CONTAINER\_CURRENT or DBMS\_AUDIT\_MGMT.CONTAINER\_ALL).

## *Disabling and dropping audit policies*

- You will learn to disable and drop audit policies.
- You'll need an enabled unified audit policy (for example, `oe_policy`) and a user who has the `audit_admin` role (for example, `jack`).

## *Disabling and dropping audit policies*

- Check that the audit policy `oe_policy` is enabled.
- Disable it.
- To be able to drop audit policy, you have to disable it first.
- Drop the audit policy `oe_policy`.



# Oracle Database 19c Security

## *Additional Topics*

In this chapter, we will cover the following tasks:

- Exporting data using Oracle Data Pump in the Oracle Database Vault environment
- Creating factors in Oracle Database Vault
- Using TDE in a multitenant environment

- An Oracle Database Vault component **factor** is a named variable, which can have one or more values, assigned in several ways.
- The actual value of factor is named **identity**.
- Each factor has a **factor type**.

- A factor type is used only for classification purposes.
- Factors are building blocks for configuring security policies.
- They can be used in rules/rule sets.
- You can configure factors by using Oracle Enterprise Manager or the Database Vault API.

- In Oracle Database 19c, it is possible to perform Oracle Data Pump regular and transportable export and import operations in the Oracle Database Vault environment.
- You'll export data that resides in a schema that is protected by a realm.

- It is assumed that:

- You are using Oracle Database 12.1.0.2 (the traditional architecture) on Linux
- Sample schemas are installed (you'll use HR schema in this recipe)
- Database Vault is enabled and configured (a Database Vault owner is user dbv\_owner, account manager is user dbv\_acctmgr, and realm that protects HR schema is created).

- This is one way how you can create **HR** realm:

```
SQL> connect dbv_owner
SQL> BEGIN
  DBMS_MACADM.CREATE_REALM(
    realm_name  => 'HR Realm',
    description => 'Protects HR schema',
    enabled      => DBMS_MACUTL.G_YES,
    audit_options=> DBMS_MACUTL.G_REALM_AUDIT_OFF,
    realm_type   => 0);
END;
/
```

PL/SQL procedure successfully completed.

```
SQL> BEGIN
  DBMS_MACADM.ADD_OBJECT_TO_REALM(
    realm_name  => 'HR Realm',
    object_owner => 'HR',
    object_name  => '%',
    object_type  => '%');
END;
/
```

PL/SQL procedure successfully completed.

- A directory for export operations (for example, `dp_dir`) is created and a user (for example, `piter`) who is going to perform Data Pump export has read and write privileges on the directory.
- Also, the `DATAPUMP_EXP_FULL_DATABASE` role and the `CREATE TABLE` and `UNLIMITED TABLESPACE` privileges have been granted to the user:

```
SQL> connect system
SQL> CREATE DIRECTORY dp_dir AS '/u01/app/oracle/oradata/dp_exp';
SQL> connect dbv_acctmgr
SQL> create user piter identified by T2abc_4z1;
SQL> grant create session to piter;
SQL> connect / as sysdba
SQL> grant create table, unlimited tablespace to piter;
SQL> grant read, write ON DIRECTORY dp_dir to piter;
SQL> grant DATAPUMP_EXP_FULL_DATABASE to piter;
```

- To be able to export data that is protected by Database Vault mechanisms, user has to be authorized.
- You can authorize a user to perform export and import operations:
  1. On specific database object in a schema, such as table.

```
SQL> EXEC DBMS_MACADM.AUTHORIZE_DATAPUMP_USER ('AMY', 'HR',  
'EMPLOYEES');
```

2. On specific schema.

3. For entire database.

```
SQL> EXEC DBMS_MACADM.AUTHORIZE_DATAPUMP_USER ('KIM');
```

```
SQL> grant DV_OWNER to kim;
```

- If you want to encrypt Oracle Data Pump export, using its encryption features, Oracle Advanced Security option has to be enabled.
- To encrypt export, specify appropriate value for **ENCRYPTION** parameter.
- These are allowed values for the parameter:

**ENCRYPTION=[ALL|DATA\_ONLY|ENCRYPTED\_COLUMNS\_ONLY|MATERIALIZE\_ONLY|NONE]**

## *Creating factors in Oracle Database Vault*

- You'll create three factors (**Day**, **Holiday**, and **NonWorkingDay**).
- The factor **Day** will return name of the day based on **sysdate**.

# *Creating factors in Oracle Database Vault*

- It is assumed that:

- You are using Oracle Database 12.1.0.2 (the traditional architecture) on Linux and Oracle Enterprise Manager Cloud Control 19c
- Database Vault is enabled and configured (the Database Vault owner is the user `dbv_owner` and account manager is the user `dbv_acctmgr`).
- The user `dbv_owner` has been granted the **SELECT ANY DICTIONARY** privilege

# *Creating factors in Oracle Database Vault*

- The user **piter** exists, the function **piter.get\_function** has been created, and **DVSYS** has been granted the **EXECUTE** privilege on the function:

```
SQL> connect system
SQL> create or replace function piter.get_holiday
  2 return varchar2
  3 IS
  4 holiday varchar2(10);
  5 begin
  6 IF (RTRIM(TO_CHAR(SYSDATE,'DD-MON')) IN ('1-JAN', '4-JUL',
  '15-NOV')) THEN
  7 holiday := 'TRUE';
  8 ELSE
  9 holiday := 'FALSE';
 10 END IF;
 11 RETURN holiday;
 12 end;
 13 /
Function created.
SQL> grant execute on piter.get_holiday to dvsys;
Grant succeeded.
```

## *Creating factors in Oracle Database Vault*

- The identity of a factor can be assigned by:
  - Method
  - Constant
  - Factors
- The process of assigning identity to a factor is named factor identification.

## *Creating factors in Oracle Database Vault*

- You created two factors (**Day** and **Holiday**) whose identities were assigned by methods and one factor (**NonWorkingDay**) whose identity was assigned by factors.
- Factors can be evaluated.
- Because you created factors that can change during a session, you chose evaluation by access.

## *Creating factors in Oracle Database Vault*

- Verify that the factor **NonWorkingDay** got the value **COMPANY\_HOLIDAY**.
- In our case, **COMPANY\_HOLIDAY** was matched, so **WEEKEND** wasn't evaluated.
- In general, it is better to try to avoid overlapping conditions (if possible) because maintenance is easier and the risk of making a mistake is smaller.

# Creating factors in Oracle Database Vault

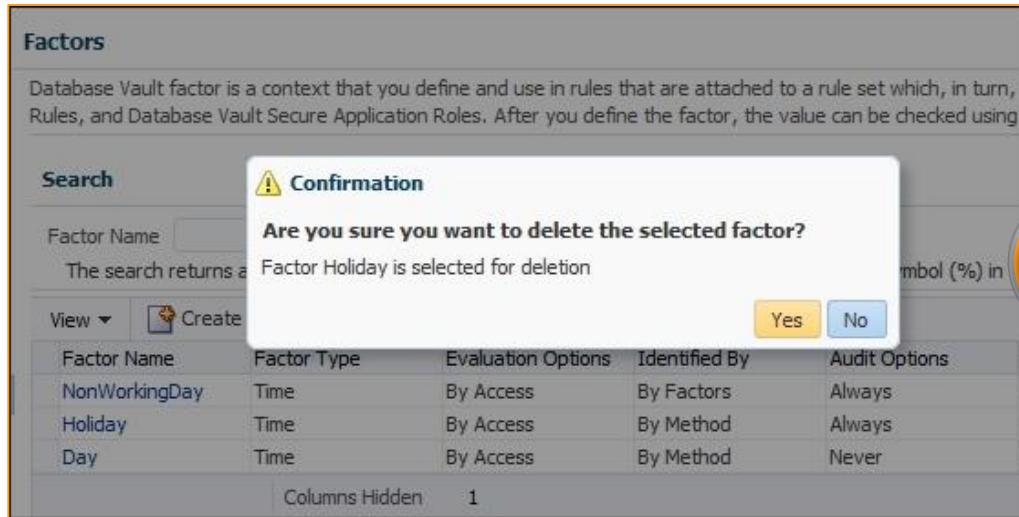
- From Factors page in EM19c verify that you can't delete the factor **Holiday** because you use it to resolve identities for the factor **NonWorkingDay**

Factor Name	Factor Type	Evaluation Options	Identified By	Audit Options	Fail Options
NonWorkingDay	Time	By Access	By Factors	Always	Show Error Message
Holiday	Time	By Access	By Method	Always	Show Error Message
Day	Time	By Access	By Method	Never	Show Error Message

1

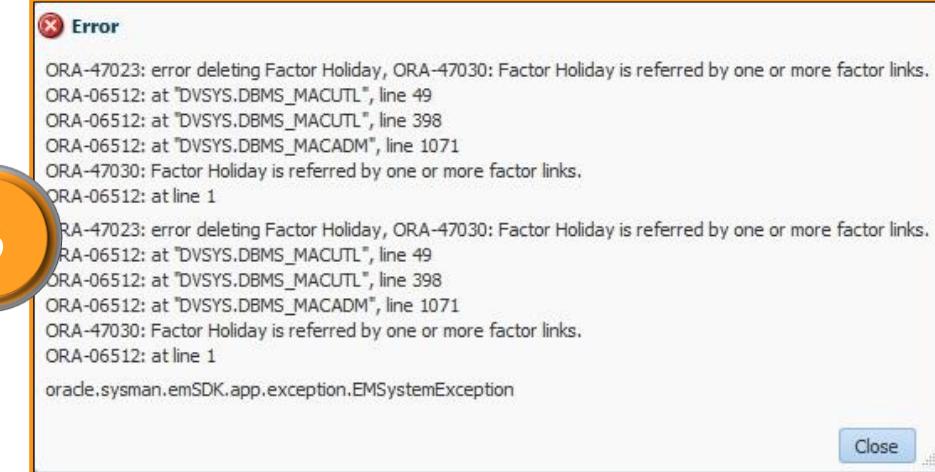
Select the factor **Holiday** and click on the **Delete** button.

# Creating factors in Oracle Database Vault



2

Click on the button Yes



You will receive an error message

3

## *Using TDE in a multitenant environment*

- You will perform different operations using Transparent Data Encryption in a multitenant environment.
- It is assumed that:
  - You have two container databases (the multitenant architecture), version 12.1.0.2 in the same host.
  - You have at least one pluggable database in each container database
  - You have sample schemes installed.

## *Using TDE in a multitenant environment*

- Creation of keystore and master key in root container.
- Opening and creation of master key in the pluggable database.
- Another way of creation of the keystore and master key (in the second container database).

## *Using TDE in a multitenant environment*

- User with the **SYSKM** system privilege is used, and opening of keystore as well as the creation of master keys are done by using **container=all** clause.
- Because there is only one keystore per container database but multiple master keys, if database needs to be unplugged and plugged into another container database, a master key needs to be exported and imported into the target database also.

## *Using TDE in a multitenant environment*

- Export a master key and unplug the database.
- Plugging this database into another container database (cdb2).
- We are importing a master key.
- After importing a master key, restart that pluggable database.



# Oracle Database 19c Security

## *Appendix – Application Contexts*

In this chapter, we will cover the following tasks:

- Exploring and using built-in contexts
- Creating an application context
- Setting application context attributes
- Using an application context

- An **application context** is a memory container that holds a set of key-value pairs.
- Also, an application context is a namespace because in different application contexts, attributes that have the same name can exist.

# *The steps to implement a local application context*

Create an application context

Create a PL/SQL package that sets the context

Set application context attributes by calling the package

*Use application context attributes in your application*

## *Exploring and using built-in contexts*

- The **USERENV** application context is a built-in context that contains information about the current session.
- You'll learn to retrieve values from built-in contexts.
- You'll need an existing user who can get values from built-in namespaces by using the **SYS\_CONTEXT** function (for example, user **maja**).

## *Exploring and using built-in contexts*

- You use the `SYS_CONTEXT` function to get values of several parameters from the `USERENV` context.
- You can use that function in both SQL and PL/SQL statements.

## *Exploring and using built-in contexts*

- The **UNIFIED\_AUDIT\_SESSIONID** attribute (parameter) is introduced in Oracle Database 12.1.0.2.
- The value of that parameter is **unified audit session ID** if the database uses unified auditing mode or mixed auditing mode, and **NULL** if the database uses traditional auditing.

# *The value of the UNIFIED\_AUDIT\_SESSIONID*

ATTRIBUTE  
**UNIFIED\_AUDIT\_SESSIONID**



VALUE

**unified audit sesión ID,  
unified or mixed auditing  
mode**

**NULL,  
traditional auditing mode**

## *Exploring and using built-in contexts*

- Another built-in context is **SYS\_SESSION\_ROLES**.
- You can use it to check whether a specified role is currently enabled for the session.
- Afterwards, you'll verify that **zoran** has the **test\_role** role by using the **SYS\_CONTEXT** function.
- When working in the multitenant environment, some useful attributes are **CON\_ID**, **CON\_NAME**, and **CDB\_NAME**.

# *Exploring and using built-in contexts*

```
SQL> create role test_role;
Role created.

SQL> grant select on hr.employees to test_role;
Grant succeeded.

SQL> grant test_role to ernesto
Grant succeeded.

SQL> select sys_context('SYS_SESSION_ROLES', 'TEST_ROLE') from dual;
SYS_CONTEXT('SYS_SESSION_ROLES', 'TEST_ROLE')
-----
FALSE

SQL> connect ernesto
Enter password:
Connected.
SQL> select sys_context('SYS_SESSION_ROLES', 'TEST_ROLE') from dual;
SYS_CONTEXT('SYS_SESSION_ROLES', 'TEST_ROLE')
-----
TRUE
```

## *Creating an application context*

- In this recipe, you'll create a local application context (for example, `sh_client`).
- In the next techniques, you will use it to store clients' identifiers.
- You'll need an existing user who can create an application context (it needs the `CREATE ANY CONTEXT` privilege or a DBA role), for example, the user `maja`.

## *Creating an application context*

- Create application context `sh_client` and defined that the PL/SQL package `sh_ctx_pkg` will be used to create and set application context attributes.
- At this moment, attributes aren't set in the application context.

## *Setting application context attributes*

- You'll create the PL/SQL package (for example, `sh_ctx_pkg`) that will set application context attributes for the application context you created in the previous technique (for example, `sh_client`).
- Also, you'll create a logon trigger.

## *Setting application context attributes*

- You'll need an existing user who can create `sh_ctx_pkg`.
- Make sure that the user has direct privileges on the `sh.customers` table (even if he/she has a DBA role).

## Setting application context attributes

- Create a logon trigger so that every user who connects to the database is going to have an application context set.
- It is very important to note that if you try to set or change key-value pairs outside the package you specified when you created application context, you will receive the error **insufficient privileges**.

```
SQL> exec DBMS_SESSION.SET_CONTEXT('sh_client','cust_id',101);
BEGIN DBMS_SESSION.SET_CONTEXT('sh_client','cust_id',101); END;
*
ERROR at line 1:
ORA-01031: insufficient privileges
ORA-06512: at "SYS.DBMS_SESSION", line 122
ORA-06512: at line 1
```

## *Using an application context*

- In this, you'll see one possible usage (in SQL) of the application contexts.
- Create a new user (for example, **sofia**).
- Make sure that his or her e-mail in the format **user@company.example.com** is unique.
- Grant him or her privileges: **create session** and **select** on **sh.customers** table.

# *Using an application context*

- New user

```
SQL> create user sofia identified by Q14be7NP;  
User created.  
  
SQL> grant create session to sofia;  
Grant succeeded.  
  
SQL> grant select on sh.customers to sofia;  
Grant succeeded.
```

# *Using an application context*

- Insert data about him or her into the **sh.customers** table.

```
SQL> insert into sh.customers values (80000,'Sofia','Smith','F',1979,'Married','Albert Embankment 19','SE1 7HD','London',11111,'England','1111',52790,'1111111','12',30,'Sofia@company.example.com',10000,1,1,sysdate,sysdate,'T');

1 row created.

SQL> commit;
```