

Transparent Data Encryption

Creating and opening the keystore Under the root container

1. From the Terminal session as oracle
2. Right click in the blue area
   1. Open Terminal Window
   2. Bash# su – oracle
   3. export ORACLE\_SID=orclcdb
   4. mkdir -p $ORACLE\_BASE/admin/orclcdb/wallet
3. In SQLDeveloper connect to the sys account (orclcdb\_sys)
4. Please make sure all steps through 6 are under the orclcdb (root) container
   1. Issue the following statement
   2. Create user c##km identified by password1;
   3. Grant syskm to c##km
   4. Create a connection string for c##km
      1. Host = localhost
      2. Service Name = orclcdb
5. In SQL Developer connect to the c##km connection string and issue the following statement

ADMINISTER KEY MANAGEMENT CREATE KEYSTORE '/u01/app/oracle/admin/orclcdb/wallet’ IDENTIFIED BY welcome1;

1. Open the keystore you created in the previous step by executing the following statement:

ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY welcome1;

https://s3.amazonaws.com/thinkific/file_uploads/154178/images/612/fbc/452/1551374467099.jpg

Setting master encryption key in software keystore

1. Create a master key for the password-based keystore still connected to system at the root container.

ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY welcome1 WITH BACKUP;

1. Create another SQL Developer connection string
   1. Name orclpdb1\_ckm
   2. Username: c##km
   3. Password: password1
   4. Host: localhost
   5. Service\_name orclpdb1
2. Connect to the orclpdb1\_ckm account under SQL Developer

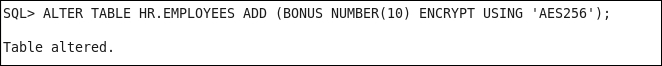
10.Issue the following commands in the orclpdb1 container as system.

* 1. ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY welcome1;
  2. ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY welcome1 WITH BACKUP;
  3. Select \* from v$encryption\_wallet;
     1. Make user the status is open and wallet\_type = password

Column encryption - adding new encrypted column to table

Connect to the hr account in SQL Developer or any account that can modify hr.employees.

1. Add a column (for example, bonus) to a table (for example, hr.employees), encrypted using the AES 256 algorithm.

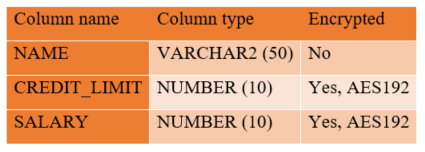


Adding the new encrypted column to the table

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

Connect to an account with can create a table(HR)

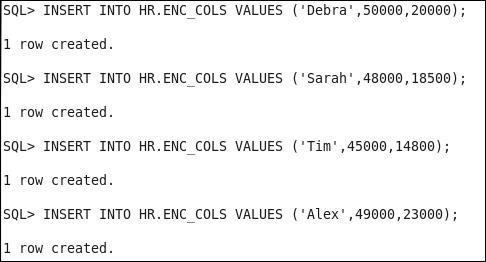
1. Create a new table (for example, table enc\_cols in schema hr) that has, for example, the following structure:





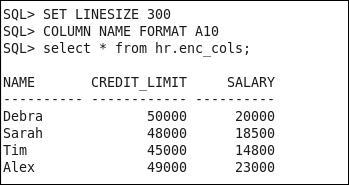
A syntax to create the table hr.enc\_cols

1. Connect to the database as a user who can insert and view data in the table (for example, hr user):
2. Insert several arbitrary values into the table HR.ENC\_COLS.



Test values

1. Verify that the user can view unencrypted values in all columns.



Using salt and MAC

1. Connect to HR
2. Encrypt two columns in an existing table, use hr.employees table and encrypt the salary and commission\_pct columns using the syntax below.
3. Alter table hr.employees modify (salary encrypt using ‘AES256’, commission\_pct encrypt using ‘AES256’ NO SALT);
4. Query the employees table and the salary and commission\_pct columns to view the data,

Encrypting tablespace

1. Connect to the database as a user who has a create tablespace privilege (for example, system)(Pluggable database)
2. Create encrypted tablespace (for example, TEST\_ENC) using AES192 encryption algorithm:

CREATE TABLESPACE TEST\_ENC DATAFILE

ENCRYPTION USING 'AES192' DEFAULT STORAGE (ENCRYPT);

1. Place a table in the newly created tablespace
   1. Create table hr.emp\_enc

tablespace test\_enc

as select \* from hr.employees;

1. Select \* from hr.emp\_enc;

## VPD Addendum Steps

1) Create the hr.emp\_vpd\_test table. Under hr@orclpdb1 or system@orclpdb1

Create table hr.emp\_vpd\_test

as select \* from hr.employees;

2) Create test data

Insert into hr.emp\_vpd\_test

Values (900,’maja’,’smith’,’maja@company.example.com’,’555-555-1212’,sysdate,’IT\_PROG’,40000,.5,100,60)

/

Insert into hr.emp\_vpd\_test

Values (901,’susan’,’jones’,’susan@company.example.com’,’555-555-1212’,sysdate,’IT\_PROG’,40000,.5,100,60)

/

Insert into hr.emp\_vpd\_test

Values (902,’joel’,’smith’,’joel@company.example.com’,’555-555-1212’,sysdate,’IT\_PROG’,40000,.5,100,60)

/

Insert into hr.emp\_vpd\_test

Values (903,’ernesto’,’lee’,’ernesto@company.example.com’,’555-555-1212’,sysdate,’IT\_PROG’,40000,.5,100,60)

/

Insert into hr.emp\_vpd\_test

Values (904,’emma’,’hansen’,’emma@company.example.com’,’555-555-1212’,sysdate,’IT\_PROG’,40000,.5,100,60)

Detective Controls

Creating a password profile

1. Connect to the database as a user who has create profile privilege:

       sqlplus un/pw@localhost:1521/orclpdb1

2. Create a password profile:

       create profile userprofile limit

       failed\_login\_attempts 4

       password\_lock\_time 2

       password\_life\_time 180;

3. Alter the user to use a newly created password profile:

       alter user hr profile userprofile;

4. Alter the default password profile:

       alter profile default limit

failed\_login\_attempts 4;

Creating password-authenticated users

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

       $ sqlplus un/pw@localhost:1521/orclpdb1

Or use SQLDeveloper connection string orclpdb1\_system

2. Create a password-authenticated user (for example, username: jessica, password: oracle\_1) as follows:

       SQL> create user jessica identified by oracle\_1;

3. Create a password-authenticated user with a more complex password:

       SQL> create user tom identified by "Qax7UnP!123\*";

4. Create a user that uses a specific password profile:

       SQL> create user steve identified by test1 profile

userprofile;

5. Create a user and force it to change password upon the first login:

       SQL> create user john identified by password1

password expire;

6. Create a user richard, whose default tablespace is users, temporary tablespace is temp, and who has their quota set to unlimited on the users tablespace:

       SQL> create user richard identified by oracle\_2 default

tablespace users temporary tablespace temp quota unlimited

on users;

Changing a user's password

1. Connect to the database as a user who has alter user privilege:

       $ sqlplus un/pw@localhost:1521/orclpdb1

Or use SQLDeveloper connection string orclpdb1\_system

2. Change the password for user jessica:

       SQL> password jessica;

3. Enter a new password (for example, oracle\_2) on a command line (note that typing will not be visible in the command line):

       New password:

4. Retype the new password (for example, oracle\_2) on the command line (note that typing will not be visible in the command line):

       Retype new password:

5. Connect to the database as any user (for example, tom, to change their own password):

       $ sqlplus tom/"Qax7UnP!123\*"

6. Change the password using the following code:

       SQL> password

7. Enter the old password (for example, Qax7UnP!123\*) on the command line (note that typing will not be visible on the command line):

       Old password:

8. Enter the new password (for example, oracle\_123) on the command line (note that typing will not be visible on the command line):

       New password:

9. Retype the new password (for example, oracle\_123) on the command line (note that typing will not be visible on the command line):

       Retype new password:

Creating a user with the same credentials on another database

1. Connect to the first database as a user who has a DBA role:

       $ sqlplus un/pw@localhost:1521/orclpdb1

2. Find a Data Definition Language (DDL) statement (ddl) that is used for user creation (for example, user jessica):

       SQL> select dbms\_metadata.get\_ddl('USER', 'JESSICA') from

dual;

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

       $ sqlplus system@orclpdb2

Or use SQLDeveloper connection string orclpdb1\_system

4. Create a user using the value you found in step 2:

SQL> create user "JESSICA" identified by values

'S:D82E6EF961F2EA7A878BCDDBC7E5C542BC148C4759D19A7

20A96BBF65658;H:F297A50FD538EF4AB119EB0278C9E72D;

C50B1E9C9AA52EC2';

Locking a user account

1. Connect to the database as a user who has alter user privilege:

$ sqlplus un/pw@localhost:1521/orclpdb1

Or use SQLDeveloper connection string orclpdb1\_syste

2. Lock the account of user steve:

SQL> alter user steve account lock;

3. Unlock the account of user steve:

SQL> alter user steve account unlock;

Expiring a user's password

1. Connect to the database as a user who has the alter user privilege:

$ sqlplus un/pw@localhost:1521/orclpdb1

2. Steve's password expires with the following command:

SQL> alter user steve password expire;

Creating and using proxy users

1. Connect to the database as a user who has a DBA role:

$ sqlplus un/pw@localhost:1521/orclpdb1

2. Create a proxy user named appserver:

SQL> create user appserver identified by oracle\_1;

3. Grant create session to the user appserver:

SQL> grant create session to appserver;

4. Alter the user to connect through the proxy user:

SQL> alter user steve grant connect through appserver;

5. Connect to the database through proxy user:

SQL> connect appserver[steve]

6. Enter a password for the appserver user (for example, oracle\_1):

Enter password:

7. To revoke connection through the proxy user, first connect to the database as a user who has altered user privilege:

$ sqlplus un/pw@localhost:1521/orclpdb1

8. Revoke connection through the proxy user appserver from user steve:

SQL> alter user steve revoke connect through appserver;

Creating and using database roles

1. Connect to the database as a user who has a dba role:

$ sqlplus un/pw@localhost:1521/orclpdb1

2. Create the role usr\_role:

SQL> create role usr\_role;

3. Grant system privilege to usr\_role:

SQL> grant create session to usr\_role;

4. Grant object privileges to usr\_role:

SQL> grant select, insert on hr.employees to usr\_role;

5. Create another role as follows:

SQL> create role mgr\_role;

6. Grant usr\_role to mgr\_role:

SQL> grant usr\_role to mgr\_role;

7. Grant system privileges to mgr\_role:

SQL> grant create table to mgr\_role;

8. Grant object privileges to mgr\_role:

SQL> grant update, delete on hr.employees to mgr\_role;

9. Grant usr\_role to user (steve):

SQL> grant usr\_role to steve;

10. Grant mgr\_role to user (tom):

SQL> grant mgr\_role to tom;

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

Database authentication

The instructions for database authentication are as follows:

1. Connect to the database as sysdba (or another user that can grant the sysbackup privilege):

sqlplus un/pw@localhost:1521/orclpdb1as sysdba

2. Grant the sysbackup privilege to user tom:

grant sysbackup to tom;

3. Verify that there is an entry in the password file that grants user tom the sysbackup administrative privilege. Select data from the v$pwfile\_users view:

select \* from v$pwfile\_users;

The following table is the result of the preceding command:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Username | sysdb | sysop | sysas | sysba | sysdg | syskm | con\_id |
| sys | TRUE | TRUE | FALSE | FALSE | FALSE | FALSE | 0 |
| sysdg | FALSE | FALSE | FALSE | FALSE | TRUE | FALSE | 0 |
| sysbackup | FALSE | FALSE | FALSE | TRUE | FALSE | FALSE | 0 |
| syskm | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE | 0 |
| tom | FALSE | FALSE | FALSE | TRUE | FALSE | FALSE | 0 |

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

Database authentication

The instructions for database authentication are as follows:

1. Connect to the database as sysdba (or another user that can grant the syskm privilege):

sqlplus / as sysdba

2. Grant the syskm privilege to user jessica:

grant syskm to jessica;

3. Connect user jessica to the database as syskm:

SQL> connect jessica/oracle\_1 as syskm

4. View privileges:

SQL> select \* from user\_tab\_privs;

SQL> select \* from session\_privs;

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

Database authentication

The instructions for database authentication are as follows:

1. Connect to the database as sysdba (or another user who can grant the sysdg privilege):

sqlplus / as sysdba

2. Grant SYSDG privilege to user steve:

SQL> grant sysdg to steve;

Security Considerations in Multitenant Environment

Creating a common user

1. Create a common user at the root level

Create user c##ernesto identified by fenago;

Grant dba to c##ernesto container=all;

Connect to the root container as a common user who has create user privilege granted commonly (for example, c##ernesto or system user):

       SQL> connect c##ernesto@orclpdb1

2. Create a common user (for example, c##maja):

       c##ernesto@orclcdb>

create user c##maja identified by oracle1 container=all;

Creating a local user

1. Connect to PDB (for example, orclpdb1) as a common user or local user who has create userprivilege in that PDB (for example, c##ernesto or system user):

       SQL> connect c##ernesto@orclpdb1

2. Create a local user (for example, steve):

       create user steve identified by fenago container=current;

Creating a common role

1. Connect to the root container as a common user who has create role privilege granted commonly (for example, c##ernesto or system user):

Sqlplus system/fenago@orclcdb

2. Create a common role (for example, c##role1):

SQL> create role c##role1 container=all;

Creating a local role

1. Connect to PDB (for example, orclpdb1) as a common or local user who has create role privilege in that PDB (for example, c##maja):

Sqlplus system/fenago@orclpdb1

2. Create a local role (for example, local\_role1):

create role local\_role1 container=current;

Granting privileges and roles commonly

1. You should connect to the root container as a common user who can grant these privileges and roles (for example, c##maja or system user):

Sqlplus system/fenago@orclcdb

2. Grant a privilege (for example, create session) to a common user (for example, c##john) commonly:

grant create session to c##john container=all;

3. Grant a privilege (for example, select any table) to a common role (for example, c##role1) commonly:

grant select any table to c##role1 container=all;

4. Grant a common role (for example, c##role1) to a common role (for example, c##role2) commonly:

grant c##role1 to c##role2 container=all;

5. Grant a common role (for example, c##role2) to a common user (for example, c##john) commonly:

grant c##role2 to c##john container=all;

Granting privileges and roles locally

1. You should connect to the container (root or pluggable database) in which you want to grant the privilege as a common or local user who can grant that privilege (for example, c##maja):

SQL> connect c##ernesto@orclpdb1

2. Grant a privilege (for example, create synonym) to a common user (for example, c##john) locally:

grant create synonym to c##john container=current;

3. Grant a privilege (for example, create view) to a local user (for example, steve) locally:

grant create view to steve container=current;

4. Grant a privilege (for example, create table) to a common role (for example, c##role1) locally:

grant create table to c##role1 container=current;

5. Grant a privilege (for example, create procedure) to a local role (for example, local\_role1) locally:

grant create procedure to local\_role1

       container=current;

6. Grant a common role (for example, c##role2) to another common role (for example, c##role3) locally:

grant c##role2 to c##role3 container=current;

7. Grant a common role (for example, c##role3) to a local role (for example, local\_role1) locally:

grant c##role3 to local\_role1 container=current;

8. Grant a local role (for example, local\_role1) to a common role (for example, c##role4) locally:

grant local\_role1 to c##role4 container=current;

9. Grant a common role (for example, c##role4) to a common user (for example, c##john) locally:

grant c##role4 to c##john container=current;

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

1. Connect to the root container of orclpdb1 as user sys:

SQL> connect sys@orclpdb1 as sysdba

2. Unplug orclpdb1 by creating an XML metadata file:

SQL> alter pluggable database orclpdb1 unplug into

       '/u02/oradata/orclpdb1.xml';

3. Drop orclpdb1 and keep the datafiles:

SQL> drop pluggable database orclpdb1 keep datafiles;

4. Connect to the root container of cdb2 as user sys:

SQL> connect sys@cdb2 as sysdba

5. Create (plug) orclpdb1 to cdb2 by using the previously created metadata file:

SQL> create pluggable database orclpdb1 using '/u02/oradata/orclpdb1.xml'

       nocopy;

PL/SQL Security

Creating and using definer's rights procedures

1. Connect to the database as a user with the DBA role (for example, ernesto)

Sqlplus ernesto/fenago@orclpdb1

2. Create two users (procowner and procuser) and grant them appropriate privileges:

SQL> create user procowner identified by oracle1;

SQL> create user procuser identified by oracle2;

SQL> grant create session, create procedure to procowner;

SQL> grant create session to procuser;

3. Create a table called ernesto.tbl and grant users privileges on this table:

SQL> create table ernesto.tbl(a number, b varchar2(40));

SQL> insert into ernesto.tbl values(1, 'old\_value');

SQL> commit;

SQL> grant select on ernesto.tbl to procuser;

SQL> grant update on ernesto.tbl to procowner;

4. Connect as a user, procowner, create a procedure to update table ernesto.tbl, and grant execute on this procedure to user procuser:

SQL> connect procowner/oracle1

CREATE OR REPLACE PROCEDURE UpdateTbl (x IN number,

       y IN varchar2)

     AUTHID DEFINER

       AS

         BEGIN

          UPDATE ERNESTO.TBL

          SET b = y

           WHERE a = x;

       END;

     /

SQL> grant execute on UpdateTbl to procuser;

5. Connect as user procuser and try to directly update table ernesto.tbl:

SQL> connect procuser/oracle2

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. When the previous step fails, update table by using the UpdateTbl procedure:

SQL> EXEC procowner.UpdateTbl(1, 'new\_value');

PL/SQL procedure successfully completed.

7. Check whether the table is updated:

SQL> select \* from ernesto.tbl;

   A          B

---------- ----------------------------------------

   1          new\_value

Creating and using invoker's right procedures

1. Connect to the database as a user with the DBA role (for example, ernesto):

SQL> connect ernesto

2. Create two users (procuser1, procuser2) and grant them privileges:

SQL> create user procuser1 identified by oracle1;

SQL> create user procuser2 identified by oracle2;

SQL> grant create session to procuser1;

SQL> grant create session to procuser2;

3. Create the table table1 and grant select and update privileges on that table to procuser1 and only select privilege to procuser2:

SQL> create table table1(a number, b varchar2(30));

SQL> insert into ernesto.table1 values(1, 'old\_value');

SQL> commit;

SQL> grant select on ernesto.table1 to procuser1;

SQL> grant update on ernesto.table1 to procuser1;

SQL> grant select on ernesto.table1 to procuser2;

4. Create an invoker's rights procedure to update table1:

CREATE OR REPLACE PROCEDURE UpdateTable1 (x IN number,

       y IN varchar2)

  AUTHID CURRENT\_USER

    AS

      BEGIN

       UPDATE ERNESTO.TABLE1

       SET b = y

        WHERE a = x;

    END;

  /

5. Grant execute on that procedure to procuser1 and procuser2:

SQL> grant execute on ernesto.UpdateTable1 to procuser1;

SQL> grant execute on ernesto.UpdateTable1 to procuser2;

6. Connect as user procuser1 and execute the procedure UpdateTable1:

SQL> connect procuser1

SQL> EXEC ernesto.UpdateTable1(1, 'new\_value');

PL/SQL procedure successfully completed.

SQL> commit;

7. Check whether the table is updated:

SQL> select \* from ernesto.table1;

   A          B

---------- ----------------------------------------

   1          new\_value

8. Connect as the user procuser2 and try to execute the procedure UpdateTable1:

SQL> connect procuser2

SQL> EXEC ernesto.UpdateTable1(1, 'newer\_value');

BEGIN ernesto.UpdateTable1(1, 'new\_value'); END;

\*

ERROR at line 1:

ORA-01031: insufficient privileges

ORA-06512: at "ERNESTO.UPDATETABLE1", line 5

ORA-06512: at line 1

Using code-based access control

1. Connect to the database as a user with a DBA role (for example, ernesto), create proc\_user, and grant him the create session privilege:

SQL> create user proc\_user identified by oracle1;

SQL> grant create session to proc\_user;

2. Create table tbl1 and insert test data:

SQL> create table tbl1(a number, b varchar2(30));

SQL> insert into tbl1 values (1, 'old\_value');

SQL> commit;

3. Create the invoker's rights procedure UpdateTbl1 and grant execute on that procedure to proc\_user:

CREATE OR REPLACE PROCEDURE UpdateTbl1 (x IN number,

       y IN varchar2)

  AUTHID CURRENT\_USER

    AS

      BEGIN

       UPDATE ERNESTO.TBL1

       SET b = y

        WHERE a = x;

    END;

    /

SQL> grant execute on ernesto.UpdateTbl1 to proc\_user;

4. Create the role proc\_role and grant update on tbl1 to proc\_role:

SQL> create role proc\_role;

SQL> grant update on ernesto.tbl1 to proc\_role;

5. Grant proc\_role to the procedure UpdateTbl1:

SQL> grant proc\_role to procedure ernesto.UpdateTbl1;

6. Connect as a user proc\_user:

SQL> connect proc\_user

7. Try to directly update the table:

SQL> update ernesto.tbl1 set b = 'value1' where a = 1;

update ernesto.tbl1 set b = 'value1' where a = 1

             \*

ERROR at line 1:

ORA-00942: table or view does not exist

8. Execute the procedure UpdateTbl1:

SQL> execute ernesto.UpdateTbl1(1, 'new\_value');

PL/SQL procedure successfully completed.

9. Connect as the user ernesto and verify whether the table is updated:

SQL> connect ernesto

SQL> select \* from tbl1;

   A          B

---------- ----------------------------------------

   1          new\_value

Restricting access to program units by using accessible by

1. Connect as a user who has the create procedure privilege (for example, ernesto):

SQL> connect ernesto

2. Create the protected\_pkg package that is only accessible by public\_pkg:

CREATE OR REPLACE PACKAGE protected\_pkg

   ACCESSIBLE BY (public\_pkg)

IS

   PROCEDURE protected\_proc;

END;

/

CREATE OR REPLACE PACKAGE BODY protected\_pkg

IS

   PROCEDURE protected\_proc

   IS

   BEGIN

         DBMS\_OUTPUT.PUT\_LINE ('This is a Protected Procedure

             that can only be accessed from Public Package');

      END;

END;

/

3. Create the public\_pkg package:

CREATE OR REPLACE PACKAGE public\_pkg

IS

      PROCEDURE public\_proc;

END;

/

CREATE OR REPLACE PACKAGE BODY public\_pkg

IS

      PROCEDURE public\_proc

      IS

      BEGIN

         DBMS\_OUTPUT.PUT\_LINE ('This is Public Procedure from

             Public Package!');

         protected\_pkg.protected\_proc;

      END;

END;

/

4. Execute the public\_proc procedure from public\_pkg:

SQL> set serveroutput on

SQL> EXEC public\_pkg.public\_proc;

This is Public Procedure from Public Package!

This is a Protected Procedure that can only be accessed from

       Public Package

PL/SQL procedure successfully completed.

5. Try to directly execute protected\_proc from protected\_pkg and observe the error:

SQL> EXEC protected\_pkg.protected\_proc;

BEGIN protected\_pkg.protected\_proc; END;

      \*

ERROR at line 1:

ORA-06550: line 1, column 7:

PLS-00904: insufficient privilege to access object

       PROTECTED\_PKG

ORA-06550: line 1, column 7:

PL/SQL: Statement ignored

6. Try to create another package that accesses protected\_proc from protected\_pkg:

CREATE OR REPLACE PACKAGE other\_pkg

IS

   PROCEDURE other\_proc;

END;

/

CREATE OR REPLACE PACKAGE BODY other\_pkg

IS

   PROCEDURE other\_proc

   IS

   BEGIN

        DBMS\_OUTPUT.PUT\_LINE ('This is Other Procedure from

               Other Package!');

         protected\_pkg.protected\_proc;

   END;

END;

/

Warning: Package Body created with compilation errors.

7. Find the compilation errors, as follows:

SQL> show errors

Errors for PACKAGE BODY OTHER\_PKG:

LINE/COL ERROR

-------- --------------------------------------------------

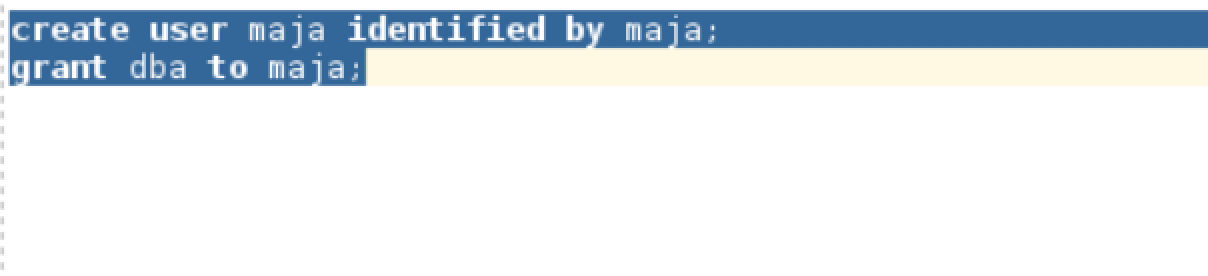
7/7         PL/SQL: Statement ignored

7/7         PLS-00904: insufficient privilege to access object

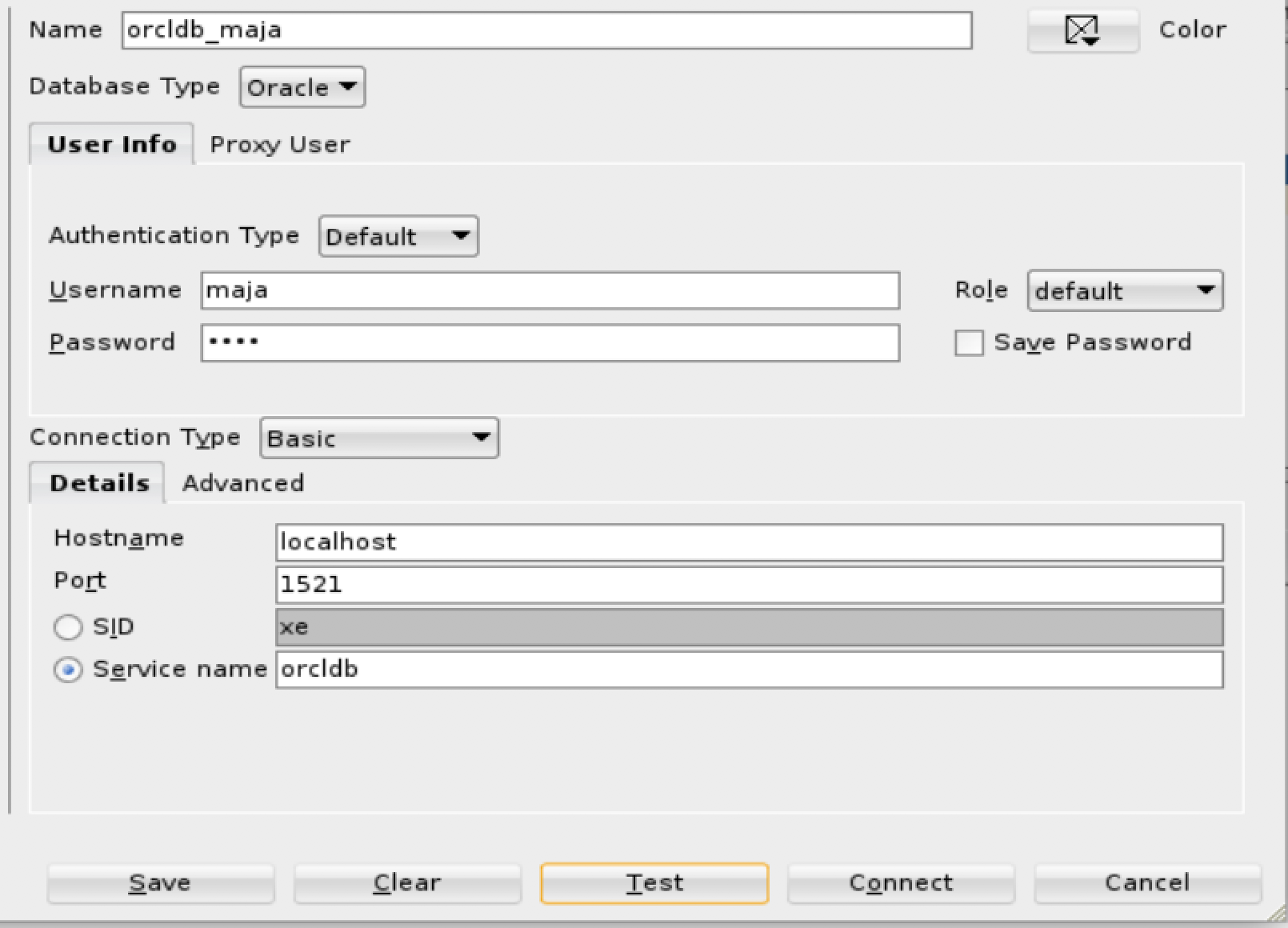
                   PROTECTED\_PKG

Virtual Private Database

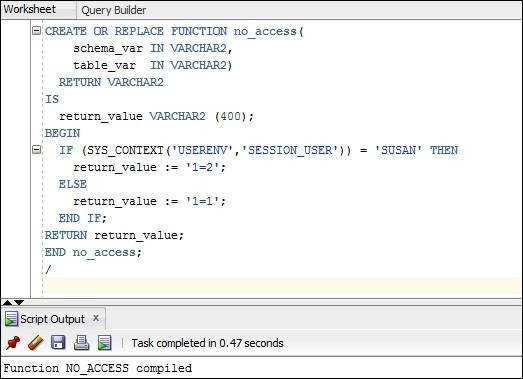
Creating different policy functions

1. Connect to the database as a user who has appropriate privileges (for example, user maja):

$ sqlplus maja/pw@localhost:1521/orclpdb1



2. Create a policy function that satisfies this condition: The user susan can't access data in a table (for example, hr.emp\_vpd\_test) and other users can access entire data in the table.



3. Create an application context that has the emp\_id attribute and the value is emp\_id (from the hr.emp\_vpd\_test) of the connected user or if the connected user is not employee.

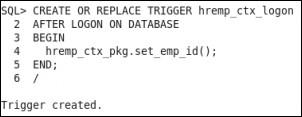
Create an application context

Sql>create context hremp\_ctx using hremp\_ctx\_pkg;

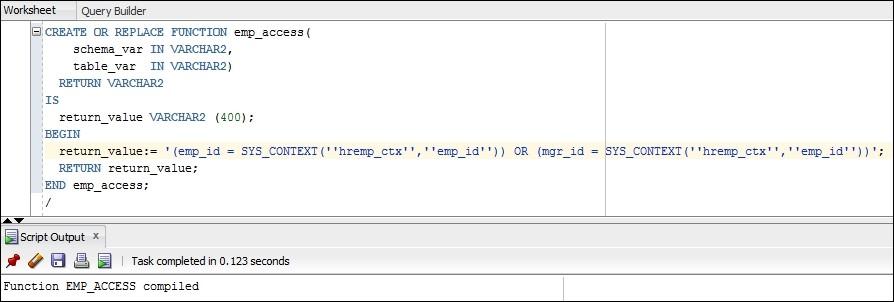
Create a PL/SQL package



Create a logon trigger



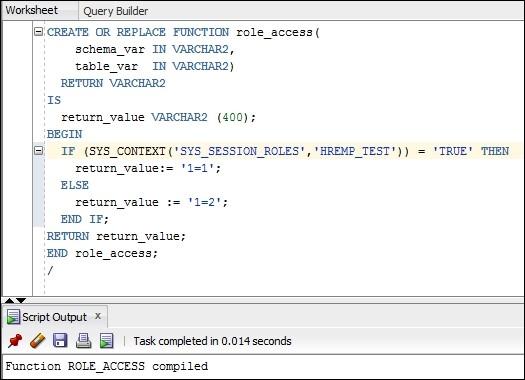
4. Create a policy function (for example, emp\_access) that satisfies this condition: a "regular" employee can access only his or her data in a table (for example, hr.emp\_vpd\_test) and manager users can access his or her data in the table and data for employees he or she directly manages.



The emp\_access policy function

5. Create a role (for example, HREMP\_TEST).

6. Create a policy function that satisfies this condition: Only users who have the HREMP\_TEST role can view data in a table (for example, hr.emp\_vpd\_test).



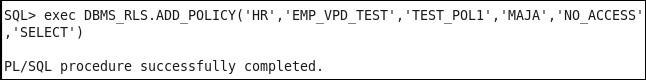
The role\_access policy function

Creating Oracle Virtual Private Database row-level policies

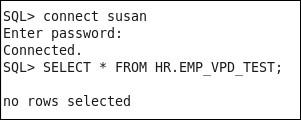
1. Connect to the database as a user who has appropriate privileges (for example, the user maja):

$ sqlplus maja

2. Create a VPD policy (for example, test\_pol1) that protects the hr.emp\_vpd\_test table in the following way: it restricts SELECT operation based on a policy function (for example, no\_access).



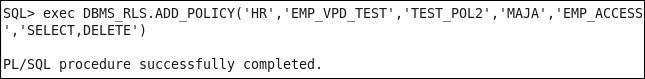
3. To test VPD policy created in the previous step, connect as the user susan to the database (keep in mind that she has the SELECT ANY TABLE privilege) and try to access data in the table hr.emp\_vpd\_test.



Susan can't access data

4. Connect to the database as a user who can create a VPD policy (for example, user maja). Create a VPD policy (for example, test\_pol2) that additionally protects the hr.emp\_vpd\_test table in the following way: it restricts the SELECT and DELETE operations based on a policy function (for example, emp\_access).

SQL> connect maja

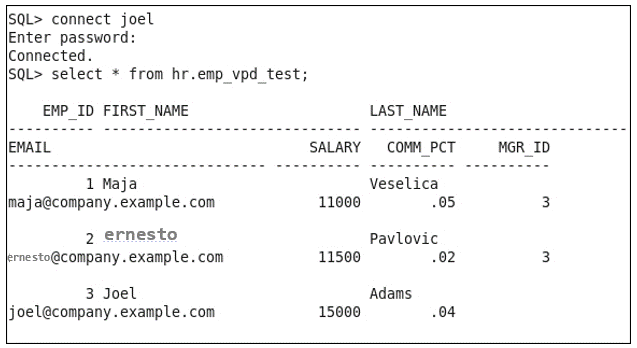


The VPD policy TEST\_POL2

5. Connect to the database as the user joel and execute the following query:

       SELECT \* FROM HR.EMP\_VPD\_TEST;

The result will show 3 rows, because joel can view his data and data for his direct employees (policy function emp\_access).

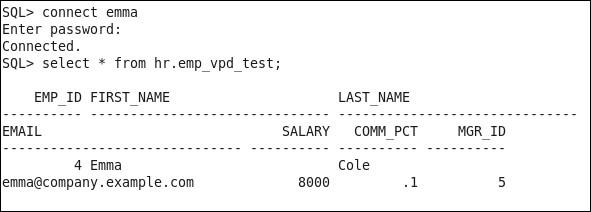


Joel can view his data and data for his direct employees

6. Connect to the database as the user emma and execute the following query:

SELECT \* FROM HR.EMP\_VPD\_TEST;

The result will show only 1 row, because emma is a "regular" employee, so she can view only her own data (policy function emp\_access).



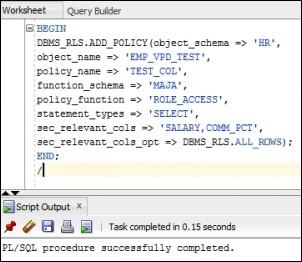
Emma can only view her own data

Creating column-level policies

1. Connect to the database as a user who has appropriate privileges (for example, the user maja):

$ sqlplus maja

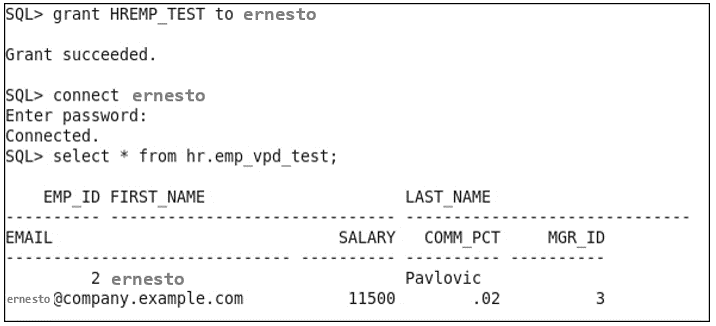
2. Create a VPD policy (for example, test\_col) that protects the hr.emp\_vpd\_test table in the following way: it defines that salary and comm\_pct are sensitive columns and a user can access them only if he or she has the HREMP\_TEST role (the role\_access policy function).



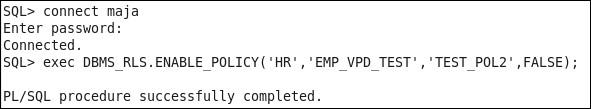
3. Grant the role HREMP\_TEST to user ernesto:

SQL> grant HREMP\_TEST to ernesto;

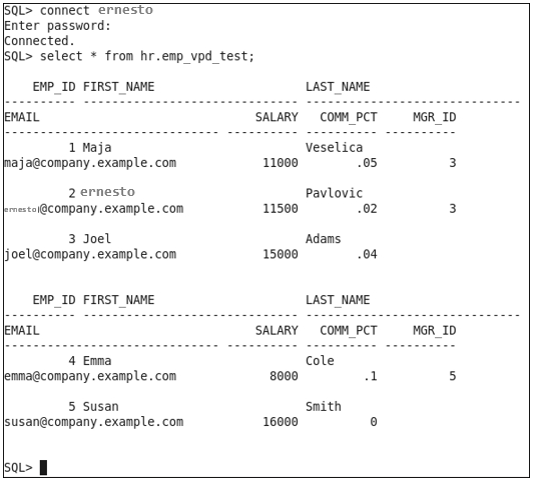
4. Connect to the database as the user ernesto and view data in the table hr.emp\_vpd\_test.



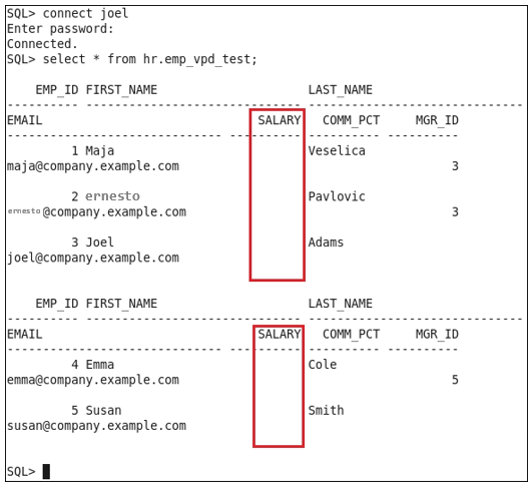
5. Connect to the database as the user maja and disable the VPD policy TEST\_POL2.



6. Repeat step 4.



7. Connect to the database as the user joel and execute the same statement as in the previous step.



Creating a driving context

1. Connect to the database as a user who has appropriate privileges (for example, the user maja):

$ sqlplus maja

2. Create a driving context (for example, driver\_ctx):

SQL> CREATE CONTEXT driver\_ctx using driver\_ctx\_pkg;

3. Set the driving context:

SQL> CREATE OR REPLACE PACKAGE driver\_ctx\_pkg IS

          PROCEDURE set\_driver (p\_group varchar2);

        END;

        /

SQL> CREATE OR REPLACE PACKAGE BODY driver\_ctx\_pkg IS

          PROCEDURE set\_driver (p\_group varchar2)

          IS

          BEGIN

           DBMS\_SESSION.SET\_CONTEXT('driver\_ctx','ACTIVE',p\_group);

         END;

      END;

/

Creating policy groups

1. Connect to the database as a user who has appropriate privileges (for example, the user maja):

$ sqlplus maja

2. Create the first policy group (for example, pol\_grp\_A):

SQL> BEGIN

DBMS\_RLS.CREATE\_POLICY\_GROUP(

object\_schema => 'HR',

object\_name => 'EMP\_VPD\_TEST',

policy\_group => 'pol\_grp\_A');

END;

/

3. Create the second policy group (for example, pol\_grp\_B):

SQL> BEGIN

DBMS\_RLS.CREATE\_POLICY\_GROUP(

object\_schema => 'HR',

object\_name => 'EMP\_VPD\_TEST',

policy\_group => 'pol\_grp\_B');

END;

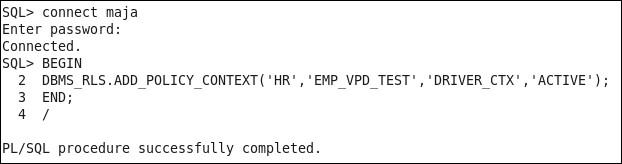
/

Setting context as a driving context

1. Connect to the database as a user who has appropriate privileges (for example, the user maja):

$ sqlplus maja

2. Make an existing application context a driving context.

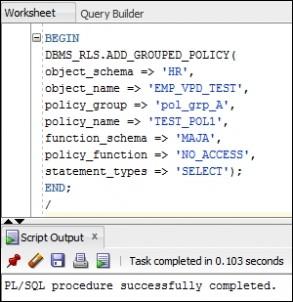


Adding policy to a group

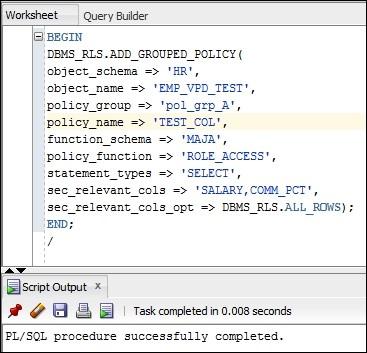
1. Connect to the database as a user who has appropriate privileges (for example, the user maja):

$ sqlplus maja

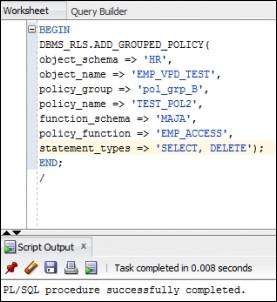
2. Add TEST\_POL1 to policy group pol\_grp\_A.



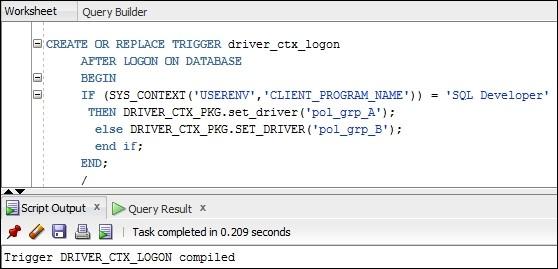
3. Add TEST\_COL to policy group pol\_grp\_A.



4. Add TEST\_POL2 to policy group pol\_grp\_B.



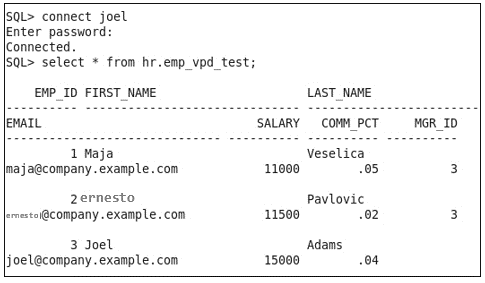
5. Create a logon trigger.



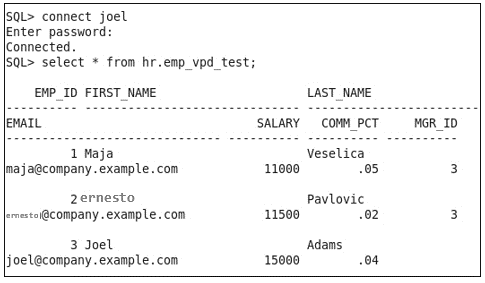
6. Connect to the database as the user joel using SQL\*Plus and execute the SELECT statement, as shown:



7. View data in the table hr.emp\_vpd\_test.



8. Connect to the database as the user susan using SQL\*Plus and view data in the table hr.emp\_vpd\_test:

SQL> connect susan

9. Connect as the user emma using SQL Developer and view data in the table hr.emp\_vpd\_test.



Exempting users from VPD policies

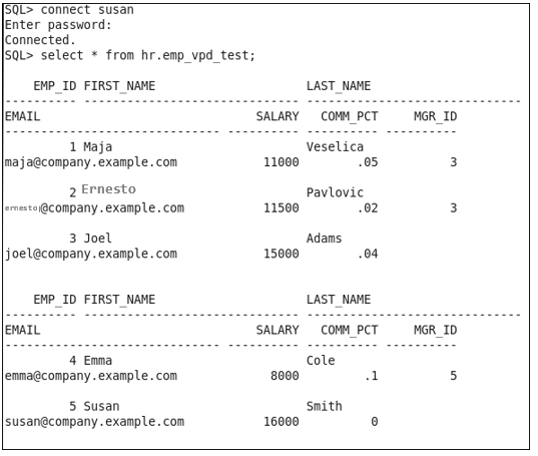
1. Connect to the database as SYS user:

$ sqlplus / as sysdba

2. Grant the EXEMPT ACCESS POLICY privilege to an existing user (for example, susan):

SQL> grant EXEMPT ACCESS POLICY to susan;

3. Connect to the database as the user susan and verify that now she can access data in the hr.emp\_vpd\_test table.



Data Redaction

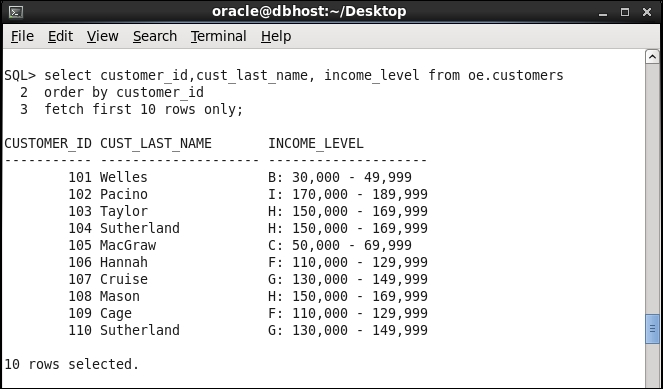
Creating a redaction policy when using full redaction

1. Connect to the database as a user who has the SELECT privilege on the OE.CUSTOMERS table or the SELECT ANY TABLE privilege (for example, the oe user):

$ sqlplus oe

2. Verify that the user (for example, the user oe) can view data by executing the following query:

select customer\_id, cust\_last\_name, income\_level from       oe.customers order by customer\_id fetch first 10 rows       only;



Data in the clear text format (before redaction) in the OE.CUSTOMERS table

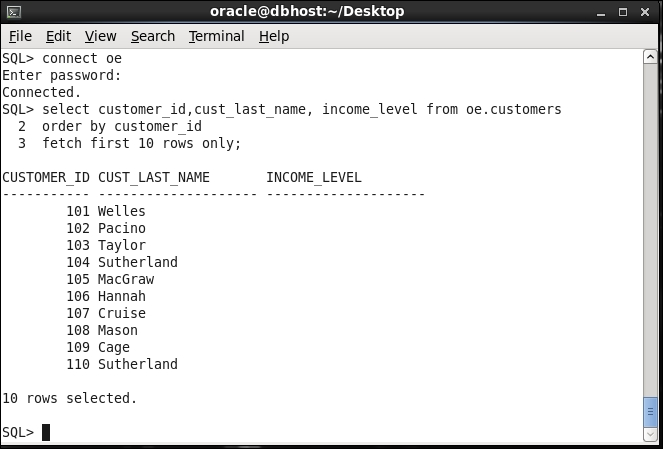
3. Connect to the database as a user who can create the user secmgr (who will be responsible for managing redaction policies) and grant him appropriate privileges (for example, SYS):

SQL> create user secmgr identified by oracle;   
SQL> grant create session to secmgr;   
SQL> grant execute on dbms\_redact to secmgr;

4. Connect to the database as the secmgr user:  
SQL> connect secmgr/oracle

5. Create the redaction policy CUST\_POL in such a manner that data in the column income\_level (the table oe.customers) is redacted using full redaction:

SQL> begin   2  dbms\_redact.add\_policy   3  (object\_schema => 'OE',   4  object\_name => 'CUSTOMERS',   5  policy\_name => 'CUST\_POL',   6  column\_name => 'INCOME\_LEVEL',   7  function\_type => DBMS\_REDACT.FULL, 8 expression => '1=1'); 9 end; 10 /   
PL/SQL procedure successfully completed.  
  
  
6. Connect to the database as the same user as in step 1 (for example, oe) and execute the same query as in step 2.



After applying the redaction policy

Creating a redaction policy when using partial redaction

 1. Log in to database as a user who has a DBA role (for instance, ernesto):

$ sqlplus ernesto/oracle

2. Create a test table and insert some data in it:

SQL> create table tbl (a number);  
  
SQL> insert into tbl values (123456);  
  
SQL> insert into tbl values (234567);  
  
SQL> insert into tbl values (345678);  
  
SQL> commit;

3. Create role (that is going to be used in redaction policy) and user usr1 as the first test user:

SQL> create role myrole;   
  
SQL> create user usr1 identified by oracle1;   
  
SQL> grant create session to usr1;

4. Grant the select privilege and role to usr1:

SQL> grant select on ernesto.tbl to usr1;   
  
SQL> grant myrole to usr1;

5. Create the second test user and grant him create session and select privilege, but don't grant him the role myrole:

SQL> create user usr2 identified by oracle2;   
  
SQL> grant create session to usr2;   
  
SQL> grant select on ernesto.tbl to usr2;

6. Create redaction policy to redact column a of the type Number using partial redaction (first four digits will be redacted and won't be seen at all). This redaction policy will be applied only to users that don't have role myrole and don't have the EXEMPT REDACTION POLICY privilege:

SQL> BEGIN  
 2 DBMS\_REDACT.ADD\_POLICY(  
  3  object\_schema          => 'ernesto',  
 4 object\_name => 'tbl',  
 5 column\_name => 'a',  
 6 column\_description => 'Sensitive column A',  
 7 policy\_name => 'a\_tbl\_partial',  
 8 policy\_description => 'Redact column A of tbl',  
 9 function\_type => DBMS\_REDACT.PARTIAL,  
 10 function\_parameters => '0,1,4',  
 11 expression => 'SYS\_CONTEXT( ''SYS\_SESSION\_ROLES'', ''MYROLE'') = ''FALSE'''); 12 END; 13 /

7. Connect to database as the user usr1 and select from the table tbl in the schema ernesto:

SQL> connect usr1/oracle1   
  
SQL> select a from ernesto.tbl;   
 A ---------- 123456 234567 345678

8. Now, connect to database as the user usr2 and again select from the table tbl in the schema ernesto:

SQL> connect usr2/oracle2   
  
usr2@ORA12CR1> select a from ernesto.tbl;   
 A ---------- 56 67 78

9. Log in to database as a user who has a DBA role (for instance, ernesto):

$ sqlplus ernesto/oracle

10. Create the test table to store credit cards data and insert some data in it:

SQL> create table customers (name varchar2(20 CHAR), credit\_card varchar2(20 CHAR));   
  
SQL> insert into customers values ('tom', '3455647456589132');   
  
SQL> insert into customers values ('steve',        '3734982321225691');   
  
SQL> insert into customers values ('john', '3472586894975806');   
  
SQL> commit;

11. Grant select privilege on table customers in the schema ernesto to usr1:

SQL> grant select on ernesto.customers to usr1;

12. Create a redaction policy to redact column credit\_card of type Varchar2 using partial redaction (first 12 values will be redacted with #sign). This redaction policy will be applied to all users, except those who have the EXEMPT REDACTION POLICY privilege (see the Exempting users from data redaction policies recipe):

SQL> BEGIN  
 2 DBMS\_REDACT.ADD\_POLICY(  
  3  object\_schema          => 'ernesto',  
 4 object\_name => 'customers',  
 5 column\_name => 'credit\_card',  
 6 column\_description => 'Credit Card numbers',  
 7 policy\_name => 'CCN\_POLICY',  
 8 policy\_description => 'Redact column credit\_card of table customers', 9 function\_type => DBMS\_REDACT.PARTIAL, 10 function\_parameters => 'VVVVVVVVVVVVVVVV, VVVVVVVVVVVVVVVV, #, 1, 12', 11 expression => '1=1'); 12 END; 13 /

13. Connect to database as the user usr1 and select from the table customers in the schema ernesto:

SQL> connect usr1/oracle1   
  
SQL> select \* from ernesto.customers;   
  
NAME                   CREDIT\_CARD -------------------- -------------------- tom                  ############9132 steve                ############5691 john ############5806

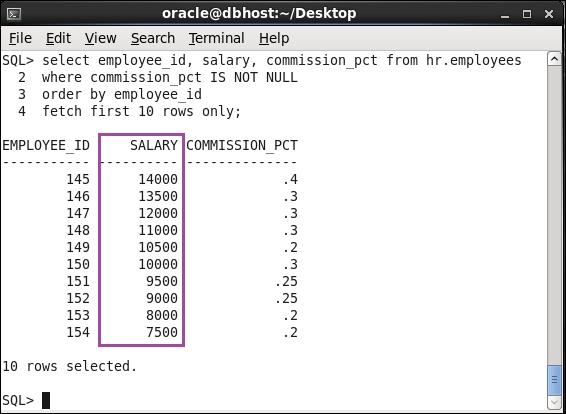
Creating a redaction policy when using random redaction

1. Connect to the database as a user who has the SELECT privilege on the HR.EMPLOYEES table or the SELECT ANY TABLE privilege (for example, hr user):

$ sqlplus hr

2. Verify that the user (for example, hr user) can view data by executing the following query:

select employee\_id, salary, commission\_pct from        hr.employees where commission\_pct IS NOT NULL order by        employee\_id fetch first 10 rows only;



Data in the clear text format in the HR.EMPOYEES table

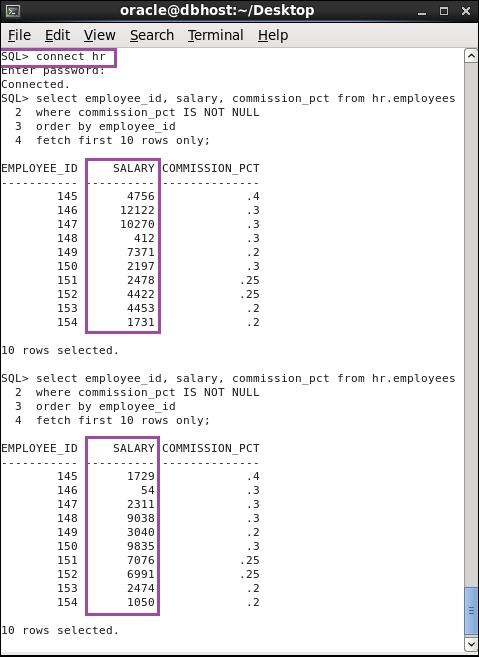
3. Connect to the database as the secmgr user:

SQL> connect secmgr/oracle

4. Create the redaction policy EMP\_POL in such a way that data in column salary (the table hr.employees) is redacted using random redaction only when user in step 1 (for example, hr) tries to view it. If you don't use the hr user, modify line 8 to reflect that change:

SQL> begin   
 2 dbms\_redact.add\_policy  
 3 (object\_schema => 'HR',  
 4 object\_name => 'EMPLOYEES',  
 5 policy\_name => 'EMP\_POL',  
 6 column\_name => 'SALARY',  
 7 function\_type => DBMS\_REDACT.RANDOM,  
 8 expression => 'SYS\_CONTEXT(''USERENV'', ''SESSION\_USER'') = ''HR''');  
 9 end;  
 10 /   
  
PL/SQL procedure successfully completed.

5.Connect to the database as the same user as in step 1 (for example, hr) and execute the same query, as in step 2, twice.



After applying redaction policy

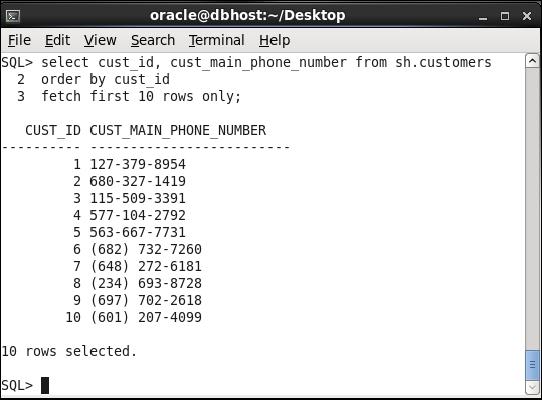
Creating a redaction policy when using regular expression redaction

1. Connect to the database as a user who has the SELECT privilege on the SH.CUSTOMERS table or the SELECT ANY TABLE privilege (for example, the sh user):

$ sqlplus sh

2. Verify that the user (for example, the user sh) can view data by executing the following query:

select cust\_id, cust\_main\_phone\_number from sh.customers order by cust\_id fetch first 10 rows only;



Data in the clear text format (before redaction) in the SH.CUSTOMERS table

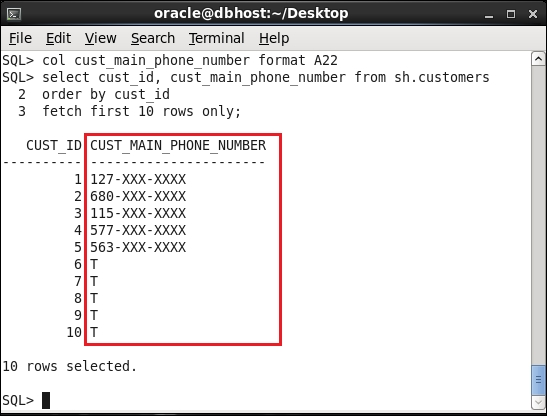
3. Connect to the database as the secmgr user:

SQL> connect secmgr/oracle

4. Create the redaction policy SHORT\_POL in such a manner that data in the column cust\_main\_phone\_number (the table sh.customers) is redacted using regular expression redaction:

SQL> begin   
 2 dbms\_redact.add\_policy  
 3 (object\_schema => 'SH',  
 4 object\_name => 'CUSTOMERS',  
 5 policy\_name => 'SHORT\_POL',  
 6 column\_name => 'CUST\_MAIN\_PHONE\_NUMBER',  
 7 function\_type => DBMS\_REDACT.REGEXP,  
 8 expression => '1=1',  
 9 regexp\_pattern => DBMS\_REDACT.RE\_PATTERN\_US\_PHONE,  
 10 regexp\_replace\_string => DBMS\_REDACT. RE\_REDACT\_US\_PHONE\_L7,  
 11 regexp\_position => DBMS\_REDACT.RE\_BEGINNING,  
 12 regexp\_occurrence => DBMS\_REDACT.RE\_FIRST);  
 13 end;  
 14 /   
  
PL/SQL procedure successfully completed.

5. Connect to the database as the same user as in step 1 (for example, sh) and execute the same query as in step 2.



After applying the redaction policy

Changing the function parameters for a specified column

1. Connect to the database as the secmgr user and alter the policy EMP\_POL:

$ sqlplus secmgr  
SQL> BEGIN  
 2 DBMS\_REDACT.ALTER\_POLICY(  
  3  object\_schema   => 'ernesto',  
 4 object\_name => 'tbl',  
 5 policy\_name => 'a\_tbl\_partial',  
 6 action => DBMS\_REDACT.MODIFY\_COLUMN,  
 7 column\_name => 'a',  
 8 function\_type => DBMS\_REDACT.PARTIAL,  
 9 function\_parameters => '9,1,4');  
 10 END;  
 11 /

2. Connect as the user usr2 to the database and view data in column A in the ernesto.tb1 table:

SQL> connect usr2/oracle2  
Connected.   
  
SQL> select a from ernesto.tbl;   
 A ---------- 999956 999967 999978

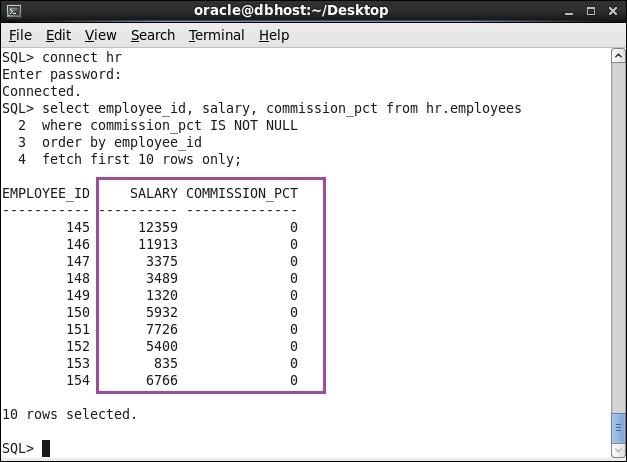
Add a column to the redaction policy

1. Connect to the database as the secmgr user and alter the EMP\_POL policy:

$ sqlplus secmgr  
SQL> BEGIN  
 2 DBMS\_REDACT.ALTER\_POLICY(  
 3 object\_schema => 'HR',  
 4 object\_name => 'EMPLOYEES',  
 5 policy\_name => 'EMP\_POL',  
 6 action => DBMS\_REDACT.ADD\_COLUMN,  
 7 column\_name => 'COMMISSION\_PCT',  
 8 function\_type => DBMS\_REDACT.FULL);  
 9 END;  
 10 /   
  
PL/SQL procedure successfully completed.

2. Connect the user hr to the database and execute the following query:

select employee\_id, salary, commission\_pct from hr.employees        where commission\_pct IS NOT NULL order by employee\_id fetch        first 10 rows only;



Two columns are redacted

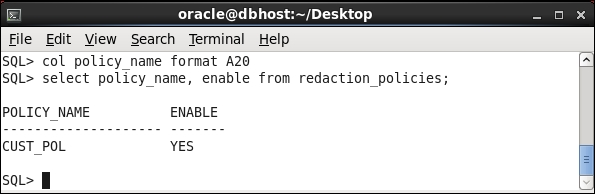
Enabling, disabling, and dropping redaction policy

1. Connect to the database as a user who has an execute privilege on dbms\_redact package and select\_catalog\_role role (for example, secmgr user):

$ sqlplus secmgr

2. Find out which redaction policies exist in the database by querying the redaction\_policies view:

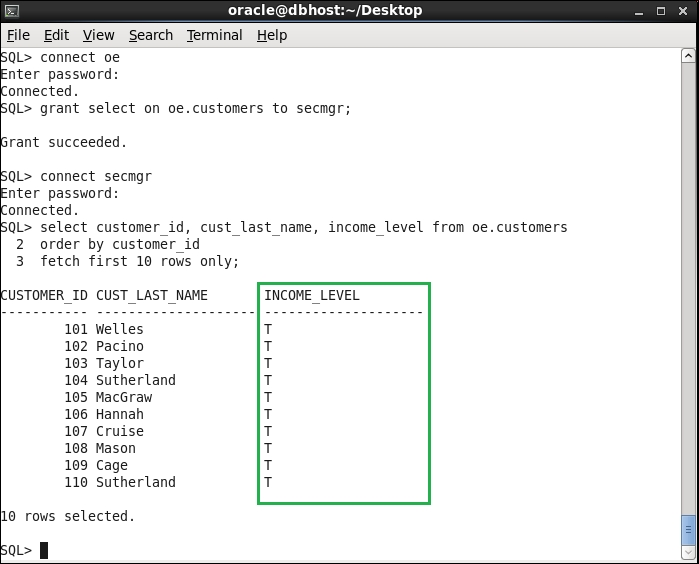
SQL> col policy\_name format A20  
select policy\_name, enable from redaction\_policies;



Finding defined redaction policies

3. Connect to the database as the oe user and grant the SELECT privilege on OE.CUSTOMERS to the secmgr user. Connect to the database as the secmgr user. Verify that the secmgr user can't see original data in the column INCOME\_LEVEL:

SQL> connect oe   
  
SQL> grant select on oe.customers to secmgr;   
  
SQL> connect secmgr   
  
SQL> select customer\_id, cust\_last\_name, income\_level from oe.customers 2 order by customer\_id 3 fetch first 10 rows only;



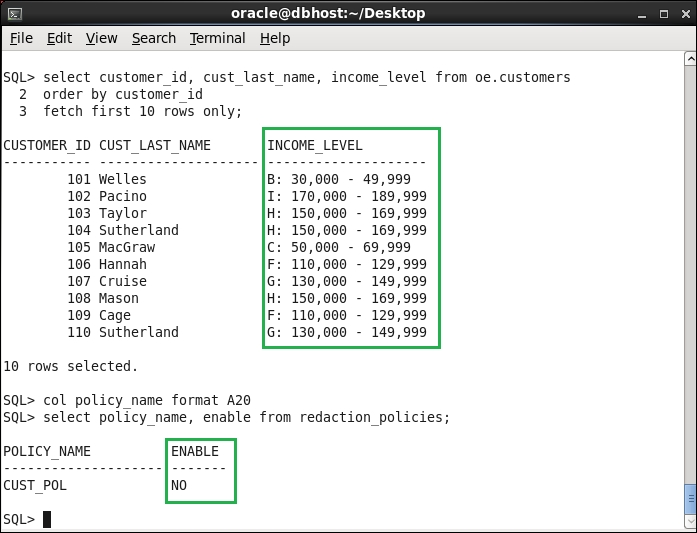
Redacted data is displayed even to the user who created the policy

4. Disable the redaction policy CUST\_POL (as the secmgr user):

SQL> begin   
 2 dbms\_redact.disable\_policy  
 3 (object\_schema => 'OE',  
 4 object\_name => 'CUSTOMERS',  
 5 policy\_name => 'CUST\_POL');  
 6 end;  
 7 /   
  
PL/SQL procedure successfully completed.

5. Verify that now the secmgr user can view original data in the column INCOME\_LEVEL and query the redaction\_policies view by executing the following statements:

* select customer\_id, cust\_last\_name, income\_level from oe.customers order by customer\_id fetch first 10 rows only;
* col policy\_name format A20
* select policy\_name, enable from redaction\_policies;



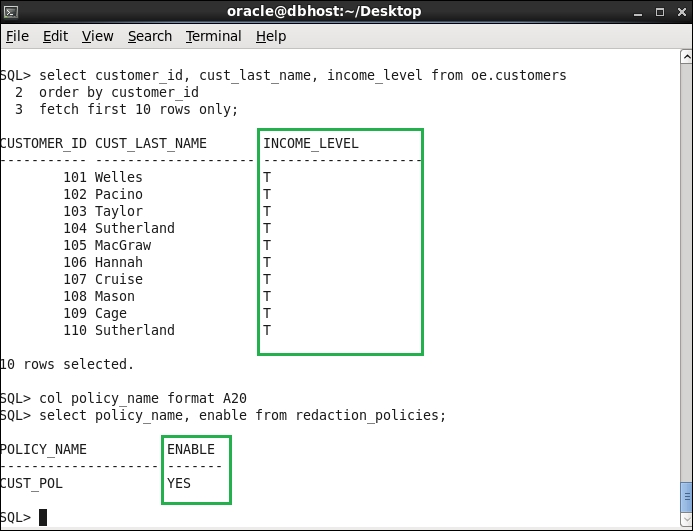
secmgr can view unmasked data in the column income\_level, because the cust\_pol policy is disabled

* Enable the redaction policy CUST\_POL:

SQL> begin   
 2 dbms\_redact.enable\_policy  
 3 (object\_schema => 'OE',  
 4 object\_name => 'CUSTOMERS',  
 5 policy\_name => 'CUST\_POL');  
 6 end;  
 7 /   
  
PL/SQL procedure successfully completed.

6. Verify that redaction is working properly by executing the following statements:

* select customer\_id, cust\_last\_name, income\_level from oe.customers order by customer\_id fetch first 10 rows only;
* col policy\_name format A20
* select policy\_name, enable from redaction\_policies;



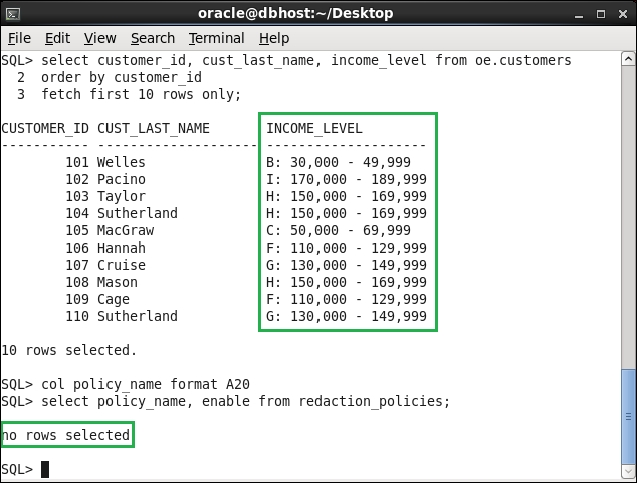
Redacted data is displayed to the secmgr user because the cust\_pol redaction policy is enabled

* Drop the redaction policy CUST\_POL:

SQL> begin   
 2 dbms\_redact.drop\_policy  
 3 (object\_schema => 'OE',  
 4 object\_name => 'CUSTOMERS',  
 5 policy\_name => 'CUST\_POL');  
 6 end;  
 7 /   
  
PL/SQL procedure successfully completed.

7. Verify that the redaction policy CUST\_POL doesn't exist in the database by executing the following statements:

* select customer\_id, cust\_last\_name, income\_level from oe.customers order by customer\_id fetch first 10 rows only;
* col policy\_name format A20
* select policy\_name, enable from redaction\_policies;



The redaction policy cust\_pol doesn't exist anymore

Exempting users from data redaction policies

1. Connect to the database as a user who has a DBA role (for example, user ernesto):

$ sqlplus ernesto/oracle  
  
  
2. Create a new user (for example, vipuser) and grant him the create session privilege and select privilege on table customers in schema ernesto:

SQL> create user vipuser identified by oracle;   
  
SQL> grant create session to vipuser;   
  
SQL> grant select on ernesto.customers to vipuser;

3. Connect as a newly created user and try to select from the ernesto.customers table:

SQL> connect vipuser/oracle   
  
SQL> select \* from ernesto.customers;   
  
NAME CREDIT\_CARD ---------------- ------------------------------ tom              ############9132 steve ############5691 john ############5806

4. Connect again as the user ernesto, and grant the EXEMPT REDACTION POLICY privilege to the vipuser user:

SQL> connect ernesto/oracle   
  
SQL> grant exempt redaction policy to vipuser;

5. As the user vipuser, now try to select from the table ernesto.customers:

SQL> connect vipuser/oracle   
  
SQL> select \* from ernesto.customers;   
 NAME CREDIT\_CARD --------------- ------------------------------ tom              3455647456589132 steve 3734982321225691 john 3472586894975806

Transparent Sensitive Data Protection

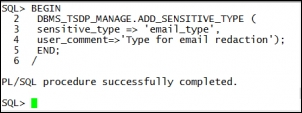
Creating a sensitive type

1. Connect to the database (for example, orclpdb1) as a user who has appropriate privileges (for example, c##ernesto):

$ sqlplus c##ernesto@orclpdb1

2. Create a sensitive type (for example, email\_type):

BEGIN   
 DBMS\_TSDP\_MANAGE.ADD\_SENSITIVE\_TYPE (   
 sensitive\_type => '<your\_type>',   
 user\_comment=> '<description>');  
 END;  
 /



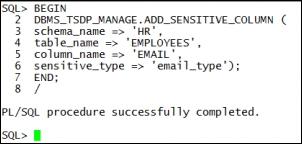
Creating a sensitive type

Determining sensitive columns

1. Connect to the database (for example, orclpdb1) as a user who has appropriate privileges (for example, c##ernesto user):

$ sqlplus c##ernesto@orclpdb1

Adding sensitive column email\_address to email\_type sensitive type

3. Associate another sensitive column (for example, schema HR, table EMPLOYEES, column EMAIL) with the same sensitive data type (for example, email\_type).  


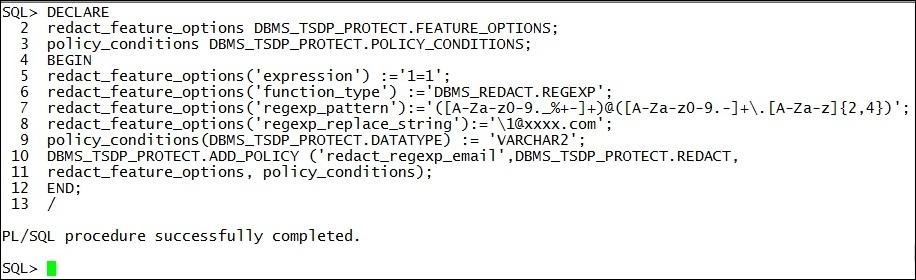
Adding sensitive column email to sensitive type email\_type

Creating transparent sensitive data protection policy

1. Connect to the database (for example, orclpdb1) as a user who has appropriate privileges (for example, c##ernesto user):

$ sqlplus c##ernesto@orclpdb1

2. Create TSDP policy using Data Redaction.



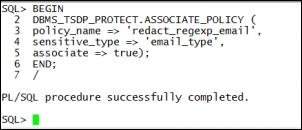
TSDP policy using Oracle Data Redaction

Associating transparent sensitive data protection policy with sensitive type

1. Connect to the database as a user (for example, orclpdb1) who has appropriate privileges (for example, c##ernesto user):

$ sqlplus c##ernesto@orclpdb1

2. Associate TSDP policy with sensitive type:



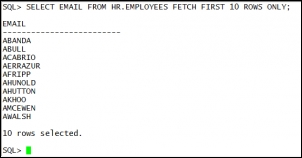
Enabling, disabling, and dropping policy

1. Connect to the database (for example, orclpdb1) as a user who has the SELECT privilege on the HR.EMPLOYEES table and the CHALLENGEERNESTO.T1 table or the SELECT ANY TABLE privilege (for example, maja).

$ sqlplus maja@orclpdb1

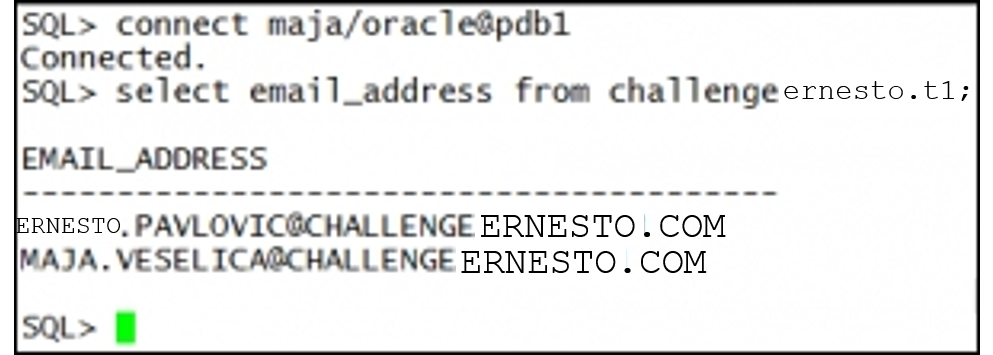
2.View sensitive data by executing the following two queries:

SELECT EMAIL FROM HR.EMPLOYEES FETCH FIRST 10 ROWS ONLY;



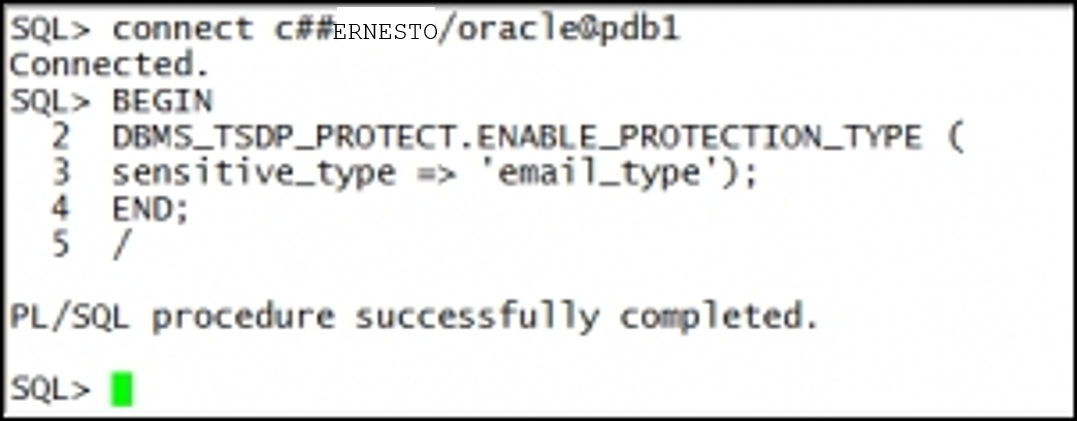
Before enabling the policy

SELECT EMAIL\_ADDRESS FROM CHALLENGEERNESTO.T1;

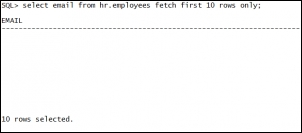


Before enabling the policy

3. Connect to the database (for example, orclpdb1) as a user who can manage TSDP policies (for example, c##ernesto). Enable the TSDP policy:

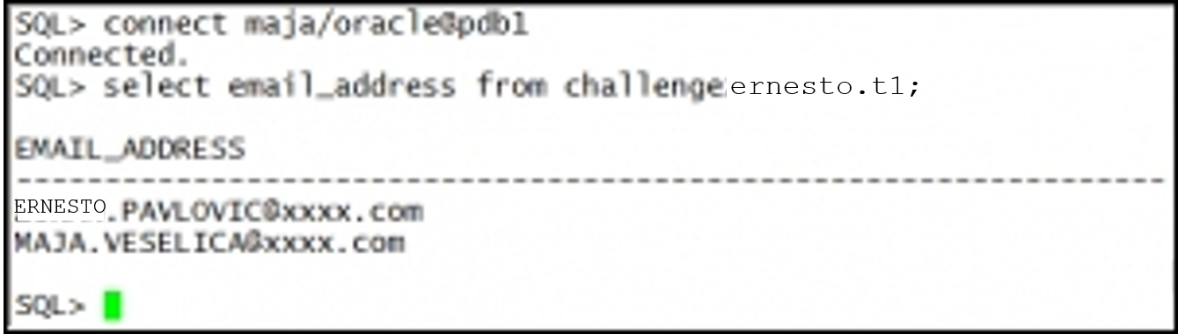


4. Repeat step 2 as user maja.



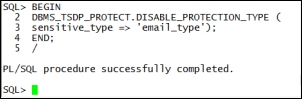
Sensitive data is protected

5.Result of the second query is shown:

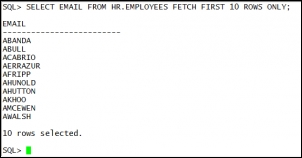


After enabling the policy

6. Connect to the database (for example, orclpdb1) as a user who can manage TSDP policies (for example, c##ernesto). Disable the TSDP policy.

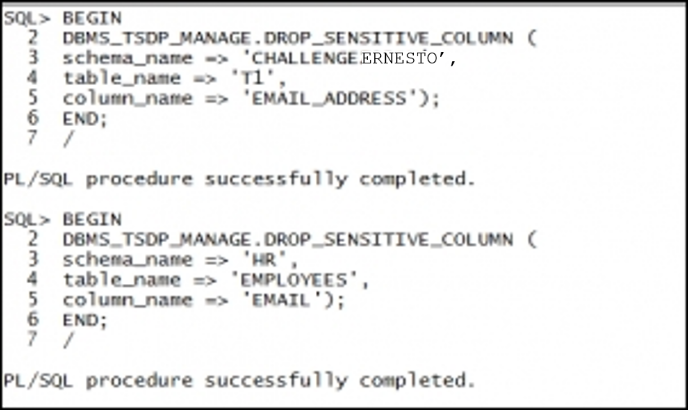


7. Repeat step 2 as user maja.

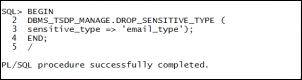


After the policy was disabled

9. Connect to the database (for example, orclpdb1) as a user who can manage TSDP policies (for example, c##ernesto). Drop both sensitive columns.



10. Drop the sensitive type.



11. Drop the TSDP policy.



Altering transparent sensitive data protection policy

1. Connect to the database (for example, orclpdb1) as a user who can manage TSDP policies (for example, c##ernesto):

$ sqlplus c##ernesto@orclpdb1

2. If the policy is enabled, disable it for all columns (for instructions how to disable the TSDP policy, see recipe Enabling, disabling, and dropping policy).

3. Connect to the database (for example, orclpdb1) as a user who can view sensitive data (for example, maja). Execute the following queries:

SELECT EMAIL FROM HR.EMPLOYEES FETCH FIRST 10 ROWS ONLY;



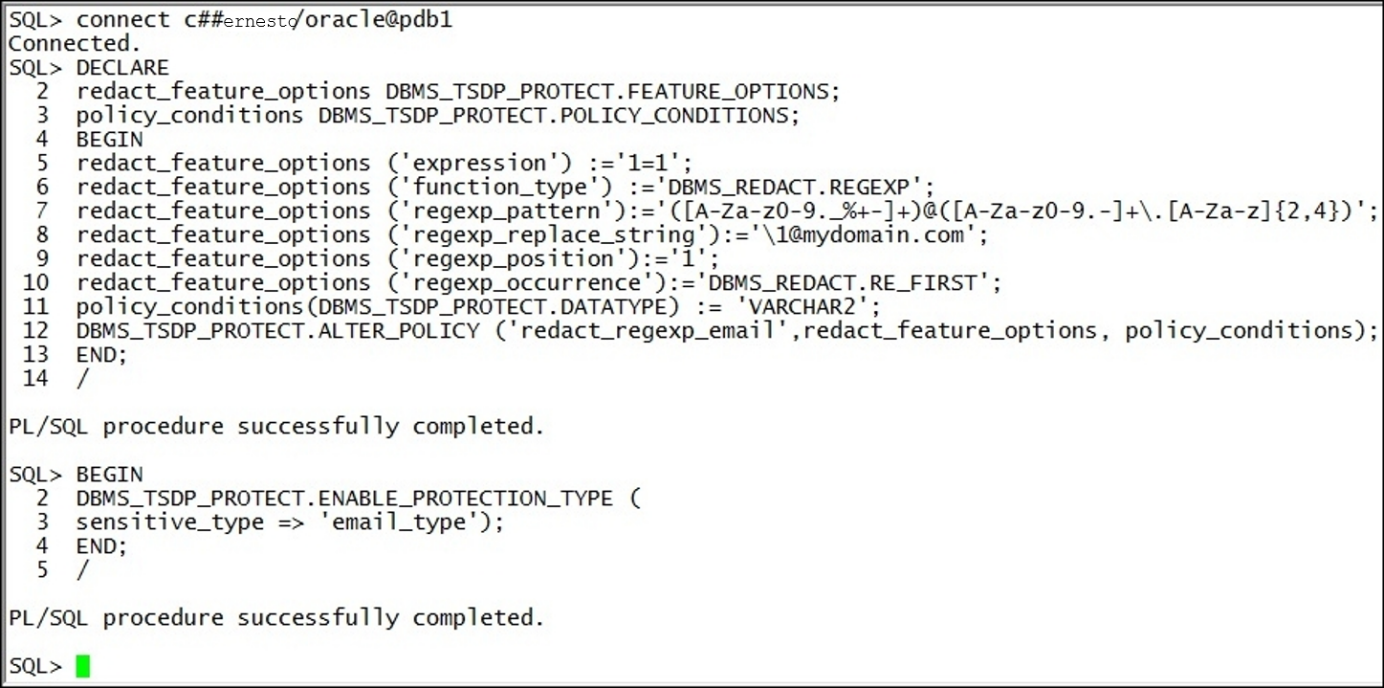
Before altering and enabling the policy

SELECT EMAIL\_ADDRESS FROM CHALLENGEERNESTO.T1;



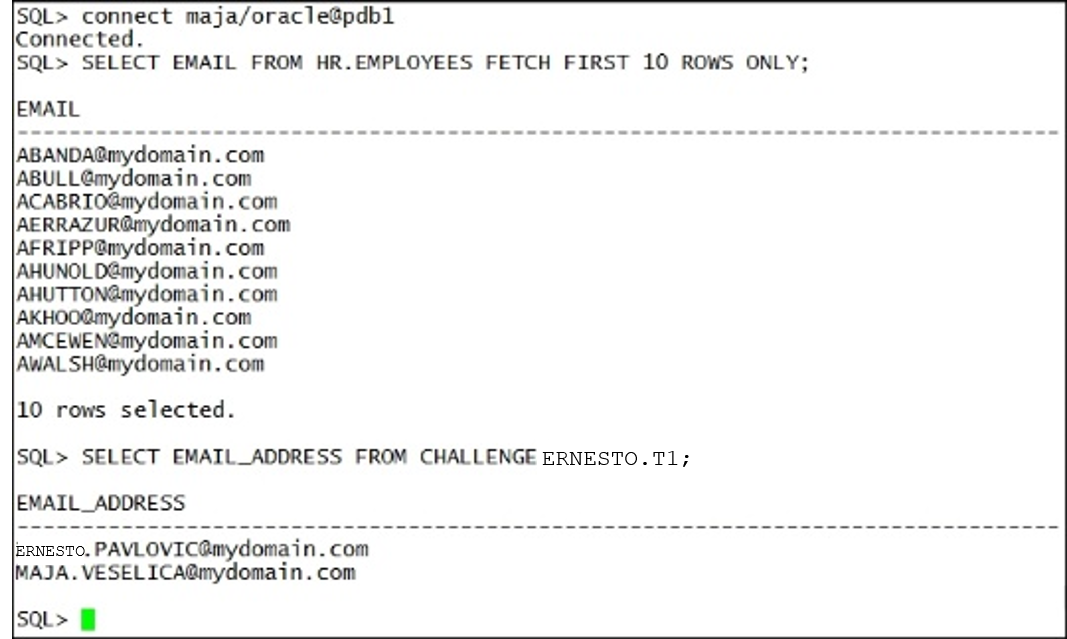
Before altering and enabling the policy

4. Connect to the database (for example, orclpdb1) as a user who can manage TSDP policies (for example, c##ernesto). Alter the TSDP policy and enable it.



Alter the TSDP policy

5. View sensitive data as the user maja (repeat step 3).



After altering TSDP policy

Privilege Analysis

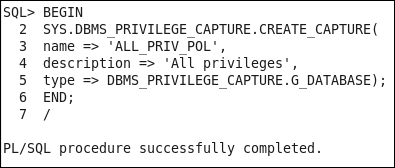
Creating database analysis policy

1. Connect to the database as system or a user who has appropriate privilege:

$ sqlplus system

2. Create a privilege analysis policy that captures all the used privileges in the database:

SQL> BEGIN SYS.DBMS\_PRIVILEGE\_CAPTURE.CREATE\_CAPTURE( name => '<policy\_name>', description => '<your\_desc>', type => DBMS\_PRIVILEGE\_CAPTURE.G\_DATABASE); END; /



Database (unconditional) analysis policy

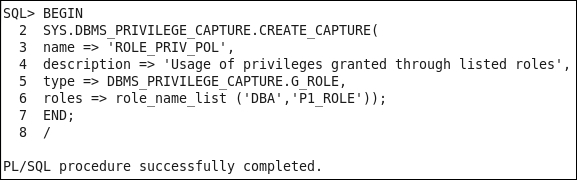
Creating role analysis policy

1. Connect to the database as system or a user who has appropriate privileges:

$ sqlplus system

2. Create a privilege analysis policy that captures all the used privileges granted through roles DBA and P1\_ROLE:

SQL> BEGIN SYS.DBMS\_PRIVILEGE\_CAPTURE.CREATE\_CAPTURE( name => '<policy\_name>', description => '<your\_desc>', type => DBMS\_PRIVILEGE\_CAPTURE.G\_ROLE, roles => role\_name\_list (<'role1',...,'role10'>)); END; /



The role analysis policy

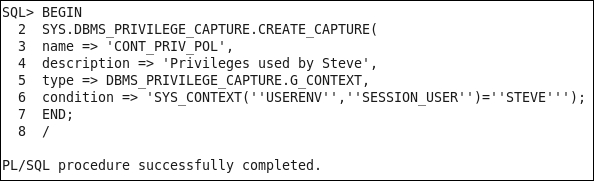
Creating context analysis policy

1. Connect to the database as system or a user who has appropriate privileges:

$ sqlplus system

2. Create a privilege analysis policy that captures all the used (and unused) privileges by Steve:

SQL> BEGIN SYS.DBMS\_PRIVILEGE\_CAPTURE.CREATE\_CAPTURE( name => '<policy\_name>', description => '<your\_desc>', type => DBMS\_PRIVILEGE\_CAPTURE.G\_CONTEXT, condition => '<your\_condition>'); END; /



The context analysis policy

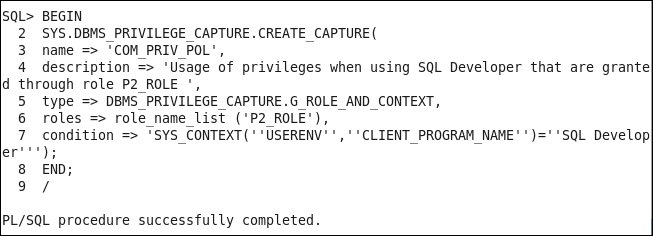
Creating combined analysis policy

1. Connect to the database as system or a user who has appropriate privileges:

$ sqlplus system

2. Create a privilege analysis policy that captures the usage of privileges, when using SQL Developer, which are granted through the role P2\_ROLE:

SQL> BEGIN SYS.DBMS\_PRIVILEGE\_CAPTURE.CREATE\_CAPTURE( name => '<policy\_name>', description => '<your\_desc>', type => DBMS\_PRIVILEGE\_CAPTURE.G\_ROLE\_AND\_CONTEXT, roles => role\_name\_list (<'role1',...,'role10'>), condition => '<your\_condition>'); END; /



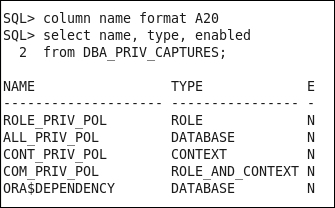
The combined analysis policy

Starting and stopping privilege analysis

1. Connect to the database as system or a user who has appropriate privileges:

$ sqlplus system

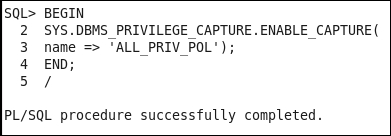
2. List all existing privilege analysis policies by querying DBA\_PRIV\_CAPTURES.



Finding all defined policies

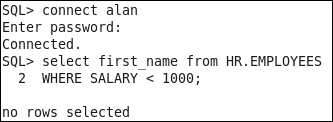
3. Enable a privilege analysis (for example, ALL\_PRIV\_POL, which you created in the first recipe in this chapter):

SQL> BEGIN SYS.DBMS\_PRIVILEGE\_CAPTURE.ENABLE\_CAPTURE( name => '<policy\_name>'); END; /



Start capturing all privileges

4. Connect to the database as the user alan and view the first names of employees who have salary less than 1000:



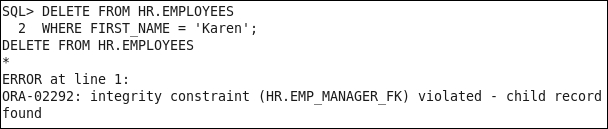
the first test of select privilege

5. Find first names of employees who earn less than 3 000.

https://s3.amazonaws.com/thinkific/file_uploads/154178/images/dd8/d6e/3dc/1550780592562.jpg

The second test of select privilege

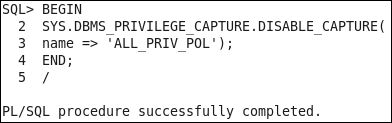
6. Try to delete all employees whose first name is Karen.



The test of delete privilege: integrity constraint violation

7. Connect to the database as system or a user who has appropriate privileges. Stop collecting data about privileges:

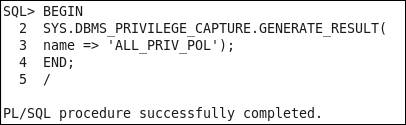
SQL> connect systemSQL> BEGIN SYS.DBMS\_PRIVILEGE\_CAPTURE.DISABLE\_CAPTURE( name => '<policy\_name>'); END; /



Stop capturing

8. Generate the result:

SQL> BEGIN SYS.DBMS\_PRIVILEGE\_CAPTURE.GENERATE\_RESULT( name => '<policy\_name>'); END; /



Generating the report

Reporting on used system privileges

1. Connect to the database as system or a user who has appropriate privileges:

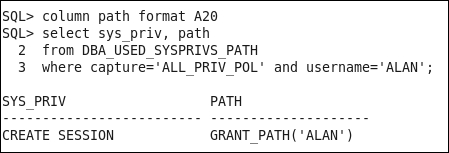
$ sqlplus system

2. View system privileges that the user ALAN used:



The used system privileges

3. View grant path for the used system privileges generated by ALL\_PRIV\_POL for the user ALAN:



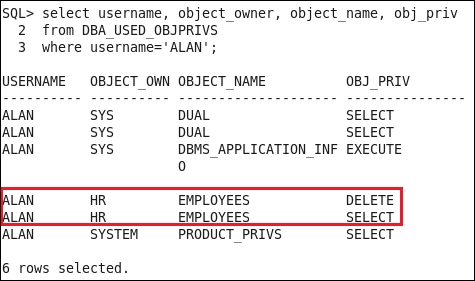
The Grant path

Reporting on used object privileges

1. Connect to the database as system or a user who has appropriate privileges:

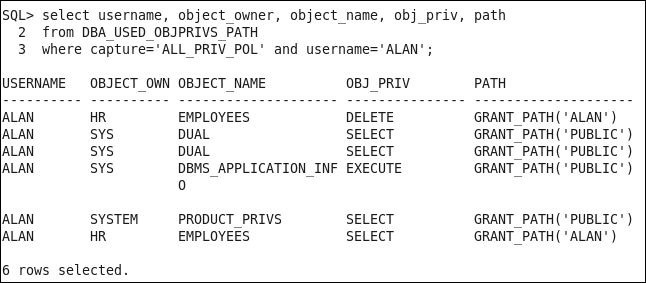
$ sqlplus system

2. View which object privileges the user Alan has used while database policy ALL\_PRIV\_POL has been active.



 The used object privileges

3. View grant path by querying DBA\_USED\_OBJPRIVS\_PATH:



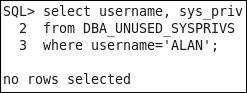
Object privileges grant path

Reporting on unused system privileges

1. Connect to the database as system or a user who has appropriate privileges:

$ sqlplus system

2. View that the user Alan has used all system privileges that have been granted to him (there are no unused system privileges):

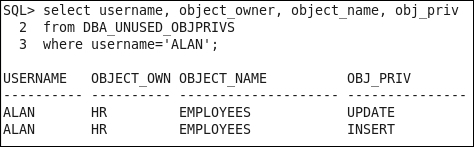


The unused system privileges for the user Alan during the database policy ALL\_PRIV\_POL capture interval

Reporting on unused object privilege

1. Connect to the database as system or a user who has appropriate privileges:

$ sqlplus system

2. View which object privileges the user Alan has used during the database policy capture interval:  


The unused object privileges

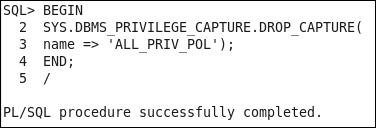
Dropping the analysis

1. Connect to the database as system or a user who has appropriate privileges:

$ sqlplus system

2. Drop a privilege analysis policy (for example, ALL\_PRIV\_POL,

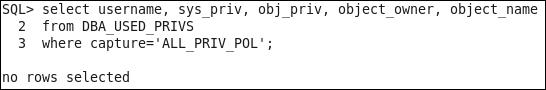
SQL> BEGIN SYS.DBMS\_PRIVILEGE\_CAPTURE.DROP\_CAPTURE( name => '<policy\_name>'); END; /



Drop policy

3. Verify that all the records about the used and unused privileges, which have been gathered according to the policy, are also dropped:

SQL> SELECT username, sys\_priv, obj\_priv, object\_owner, object\_name FROM DBA\_USED\_PRIVS WHERE capture='<policy\_name>';

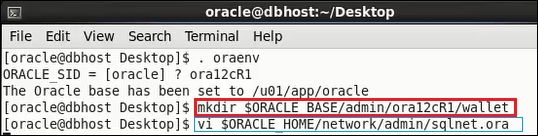


Records doesn't exist anymore

Transparent Data Encryption

Configuring keystore location in sqlnet.ora

1. Create a directory, to hold a keystore, that is accessible to the owner of Oracle software (for
2. Create the following directory
   1. mkdir /u01/app/oracle/admin/orclpdb1/wallet



Creating and opening the keystore Under the root container

1. In SQLDeveloper connect to the sys account
   1. Issue the following statement
   2. Grant administer key management to system;
2. In SQL Developer connect to the system account issue the following statement

ADMINISTER KEY MANAGEMENT CREATE KEYSTORE '/u01/app/oracle/admin/orclpdb1/wallet’ IDENTIFIED BY welcome1;

1. Open the keystore you created in the previous step by executing the following statement:

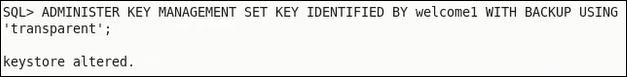
ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY welcome1;

https://s3.amazonaws.com/thinkific/file_uploads/154178/images/612/fbc/452/1551374467099.jpg

Setting master encryption key in software keystore

1. Create a master key for the password-based keystore still connected to system at the root container.

ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY welcome1 WITH BACKUP USING 'transparent';



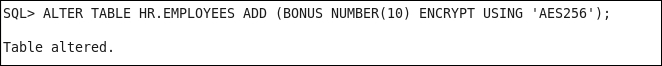
Connect to the orclpdb1\_system account under SQL Developer

1. Issue the following commands in the orclpdb1 container as system.
   1. ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY welcome1;
   2. ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY welcome1 WITH BACKUP USING 'transparent';
   3. Select \* from v$encryption\_wallet;
      1. Make user the status is open and wallet\_type = password

Column encryption - adding new encrypted column to table

Connect to the hr account in SQL Developer or any account that can modify hr.employees.

1. Add a column (for example, bonus) to a table (for example, hr.employees), encrypted using the AES 256 algorithm.



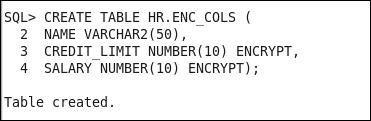
Adding the new encrypted column to the table

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

Connect to an account with can create a table(HR)

1. Create a new table (for example, table enc\_cols in schema hr) that has, for example, the following structure:

|  |  |  |
| --- | --- | --- |
| Column name | Column type | Encrypted |
| NAME | VARCHAR2 (50) | No |
| CREDIT\_LIMIT | NUMBER (10) | Yes, AES192 |
| SALARY | NUMBER (10) | Yes, AES192 |

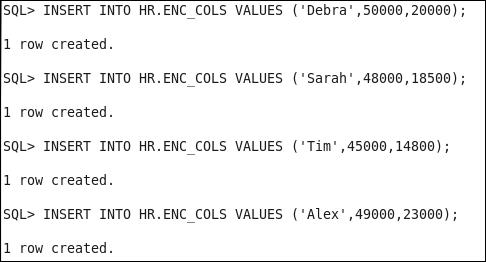


A syntax to create the table hr.enc\_cols

1. Connect to the database as a user who can insert and view data in the table (for example, hr user):

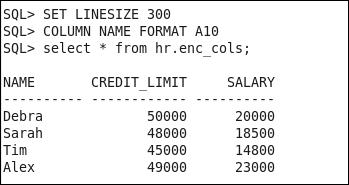
SQL> connect hr

1. Insert several arbitrary values into the table HR.ENC\_COLS.



Test values

1. Verify that the user can view unencrypted values in all columns.

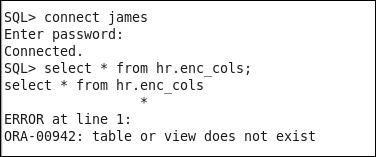


Encryption is transparent

1. Connect to the database as a user who can't view data in the table (for example, james) and try to view data in all columns:

SQL> connect james

SQL> select \* from hr.enc\_cols;



User who doesn't have "view" privilege(s) won't see encrypted values

Using salt and MAC

1. Connect to HR
2. Encrypt two columns in an existing table, use hr.employees table and encrypt the salary and commission\_pct columns using the syntax below.
3. Alter table hr.employees modify (salary encrypt using ‘AES256’, commission\_pct encrypt using ‘AES256’ NO SALT);
4. Query the employees table and the salary and commission\_pct columns to view the data,

Encrypting tablespace

1. Connect to the database as a user who has a create tablespace privilege (for example, system)(Pluggable database)
2. Create encrypted tablespace (for example, TEST\_ENC) using AES192 encryption algorithm:

CREATE TABLESPACE TEST\_ENC DATAFILE '/u02/oradata/ORCLPDB1/orclpdb1/testenc01.dbf' SIZE 20M ENCRYPTION USING 'AES192' DEFAULT STORAGE (ENCRYPT);

1. Place a table in the newly created tablespace
   1. Create table hr.emp\_enc

tablespace test\_enc

as select \* from hr.employees;

1. Select \* from hr.emp\_enc;

Rekeying

1. Connect to the database as a user who has administer key privilege or SYSKM privilege (for example, system in the pluggable database)
2. To rekey a table (for example, the oe.customer) using a different encryption algorithm (for example, AES128), execute the following statement:
3. Alter table hr.emp\_enc rekey using ‘AES128’;
4. Select \* from hr.emp\_enc

Backup and Recovery

1. Connect to the RMAN as user who has the sysbackup privilege:

$ rman target '"ernesto@orcl as sysbackup"'

1. Configure encryption on a database level:

RMAN> CONFIGURE ENCRYPTION FOR DATABASE ON;

1. Backup a tablespace example in transparent mode:

RMAN> BACKUP TABLESPACE EXAMPLE tag 'tran\_mode';

1. Enable dual mode encryption and backup tablespace example in dual mode:

RMAN> SET ENCRYPTION ON IDENTIFIED BY "password\_1";RMAN> BACKUP TABLESPACE EXAMPLE tag 'dual\_mode';

1. Enable password mode and backup tablespace example in password mode:

RMAN> SET ENCRYPTION ON IDENTIFIED BY "password\_2" ONLY; RMAN> BACKUP TABLESPACE EXAMPLE tag 'pass\_mode';

**Enabling Unified Auditing mode**

The process of enabling unified auditing is depicted in the next figure:

1. In our case, there is only one database instance. Connect to the instance as sysoper and shut it down. Also, stop the listener:

$ sqlplus / as sysdba  
SQL> shutdown immediate

SQL> exit

$ lsnrctl stop

1. Relink Oracle binaries with the uniaud\_on option:

$ cd $ORACLE\_HOME/rdbms/lib   
$ make -f ins\_rdbms.mk uniaud\_on ioracle

1. Start the listener and the database instance:

$ lsnrctl start

$ sqlplus / as sysdba

SQL> startup

To verify that unified auditing is enabled, issue the following SQL statement:

SQL> SELECT PARAMETER, VALUE   
 2 from v$option  
 3 where PARAMETER = 'Unified Auditing';

You should see that value for Unified Auditing parameter is true:

Configuring whether loss of audit data is acceptable

1. Connect to the database as user who has the audit\_admin role
2. SQL> connect system

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

1. Connect to the database as a user who has the dba role
2. $ sqlplus system/fenago@orclpdb1
3. Create the user jack and grant him the create session privilege and the audit\_admin role.

SQL> create user jack identified by pQ3s7a4w2;   
SQL> grant create session, audit\_admin to jack;

1. Create the user jill and grant her the create session privilege and the audit\_viewer role.

SQL> create user jill identified by t1m5\_R2f3;   
SQL> grant create session, audit\_viewer to jill;

Auditing RMAN operations

$ rman target /

Backup the EXAMPLE tablespace and view information about backups:

RMAN> backup database;   
RMAN> list backup;   
RMAN> exit

sqlplus system/fenago  
  
SQL> **EXEC SYS.DBMS\_AUDIT\_MGMT.FLUSH\_UNIFIED\_AUDIT\_TRAIL;**  
SQL> select dbusername, rman\_operation from unified\_audit\_trail where rman\_operation is not null;

Auditing Data Pump operations

1. Connect to the database as a user who has the audit\_admin role (for example, jack):

$ sqlplus hr/fenago@orclpdb1

1. Create an audit policy to audit Data Pump export operations:

SQL>

**CREATE AUDIT POLICY DP\_POLICY ACTIONS  
COMPONENT=datapump export;**

1. Enable the audit policy:

SQL> **AUDIT POLICY DP\_POLICY;**

1. Export the table hr.departments:

$ **expdp hr@orclpdb1 dumpfile=test tables=hr.departments DIRECTORY=data\_pump\_dir**

1. Verify that the export operation was successfully audited:

SQL> connect hr  
SQL> select DP\_TEXT\_PARAMETERS1,DP\_BOOLEAN\_PARAMETERS1  
from unified\_audit\_trail  
where audit\_type='Datapump' and dbusername='HR';

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

1. Connect to the database as a user who has the audit\_admin role (for example, jack):

$ sqlplus system/fenago@orclpdb1

1. Create audit policy my\_policy1:

**CREATE AUDIT POLICY MY\_POLICY1  
PRIVILEGES SELECT ANY TABLE  
ACTIONS CREATE TABLE, DROP TABLE;**

1. Create the audit policy role\_con\_policy:

**CREATE AUDIT POLICY ROLE\_CON\_POLICY  
ROLES HR\_ROLE  
WHEN 'SYS\_CONTEXT(''USERENV'',''HOST'')=”localhost”   
EVALUATE PER SESSION;**

1. Create the audit policy hr\_policy:

**CREATE AUDIT POLICY HR\_POLICY   
ACTIONS SELECT,INSERT,UPDATE,DELETE ON HR.DEPARTMENTS;**

1. Create the audit policy oe\_policy:

**CREATE AUDIT POLICY OE\_POLICY   
ACTIONS ALL ON OE.ORDERS;**

Enabling audit policy

1. Connect to the database as a user who has audit\_admin role (for example, jack)

SQL> connect system

1. Enable audit policy oe\_policy in such way that it applies only to user JOHN
   1. If user john does not exist create him under orclpdb1 and grant select on oe.orders table

SQL> audit policy OE\_POLICY BY JOHN;

1. Enable audit policy hr\_policy to capture only successful events.

SQL> AUDIT POLICY HR\_POLICY WHENEVER SUCCESSFUL;

1. Enable policy my\_policy1 to audit unsuccessful events for all users except maja and ernesto.

audit policy my\_policy1 EXCEPT MAJA, ERNESTO WHENEVER NOT SUCCESSFUL;

1. Enable audit policy role\_con\_policy using default options.

audit policy role\_con\_policy;

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

1. Connect to the database as a user who has the audit\_admin role (for example, jack):

$ connect system

1. Find which unified audit policies are defined (exist in the database):
2. select distinct policy\_name  
   from audit\_unified\_policies;
   1. desc audit\_unified\_policies
3. View which unified audit policies are enabled:
   1. select \* from audit\_unified\_enabled\_policies;
4. Connect to the database as the user john:
   1. Sqlplus john/pw@orclpdb1
5. Execute several statements on the tables HR.EMPLOYEES, HR.DEPARTMENTS, and OE.ORDERS:
6. As John
   1. create table t(a number(10))
   2. select count(\*) from oe.orders;
   3. select first\_name from hr.employees;
   4. drop table t;
   5. connect system/fenago@orclpdb1
   6. create table hr.my\_table(b varchar2(10));
   7. connect john
   8. drop table hr.my\_table;
7. View audit records:
8. Use SQLDeveloper Connection string orclpdb1\_system

select event\_timestamp, action\_name,unified\_audit\_policies, sql\_text from unified\_audit\_trail ORDER BY EVENT\_TIMESTAMP DESC;

**Auditing application contexts**

1. Connect to the database as a user who has the audit\_admin role (for example, jack):

$ sqlplus jack

1. Configure application context auditing:

AUDIT CONTEXT NAMESPACE USERENV  
ATTRIBUTES SESSION\_USER, SERVICE\_NAME;   
  
Audit succeeded.   
  
AUDIT CONTEXT NAMESPACE USERENV ATTRIBUTES HOST;

1. select \* from audit\_unified\_contexts;
2. View audit records:
   1. SELECT APPLICATION\_CONTEXTS FROM UNIFIED\_AUDIT\_TRAIL  
      WHERE APPLICATION\_CONTEXTS IS NOT NULL;

**Purging audit trail**

1. Connect to the database as a user who has the audit\_admin rol
2. View number of audit records in the unified audit trail before the cleanup:
3. select count (\*) from unified\_audit\_trail;
4. To perform the manual cleanup, execute:
5. exec DBMS\_AUDIT\_MGMT.CLEAN\_AUDIT\_TRAIL(  
   AUDIT\_TRAIL\_TYPE => DBMS\_AUDIT\_MGMT.AUDIT\_TRAIL\_UNIFIED)
6. Create a purge job  
   exec **DBMS\_AUDIT\_MGMT.CREATE\_PURGE\_JOB  
   (AUDIT\_TRAIL\_TYPE => DBMS\_AUDIT\_MGMT.AUDIT\_TRAIL\_UNIFIED,  
   AUDIT\_TRAIL\_PURGE\_INTERVAL => 24,  
   AUDIT\_TRAIL\_PURGE\_NAME => 'My\_Job',  
   USE\_LAST\_ARCH\_TIMESTAMP => TRUE)**
7. View number of audit records in the unified audit trail after the cleanup:

SQL> select count (\*) from unified\_audit\_trail;

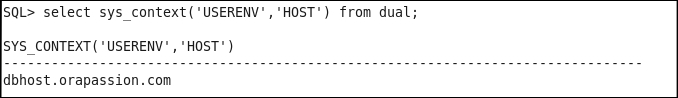
Disabling and dropping audit policies

1. Connect to the database as a user who has the audit\_admin role
2. Verify that the policy is enabled:
3. SELECT POLICY\_NAME, ENABLED\_OPT, USER\_NAME,  
   SUCCESS, FAILURE  
   FROM AUDIT\_UNIFIED\_ENABLED\_POLICIES;
4. Disable the policy oe\_policy:
   1. NOAUDIT policy oe\_policy BY JOHN;
5. Verify that oe\_policy is disabled:
   1. select \* from AUDIT\_UNIFIED\_ENABLED\_POLICIES;
6. Drop the policy oe\_policy:
7. drop audit policy oe\_policy;

Appendix – Application Contexts

Exploring and using built-in contexts

1. Connect to the database as a user who has appropriate privileges
2. Find the name of host machine from which the client has connected to the database.



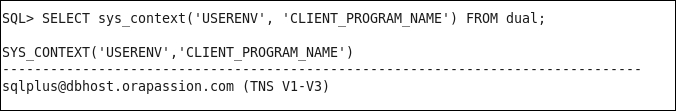
The name of the client host machine

1. Find the name of the user who logged on to the database.



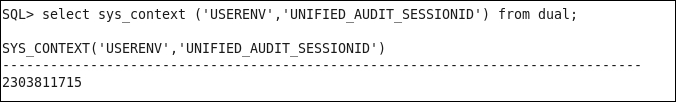
The name of the session user

1. Find the name of the program used for the database session.



The name of the client program

1. Find unified audit session ID.



A unified audit session ID

Creating an application context

1. Connect to the database as a user who has appropriate privileges (for example, user maja).

$ sqlplus maja

1. Create a local application context (for example, sh\_client).

Note

The PL/SQL package that will be used to set application context attributes doesn't have to exist at this time, but you have to specify its name.

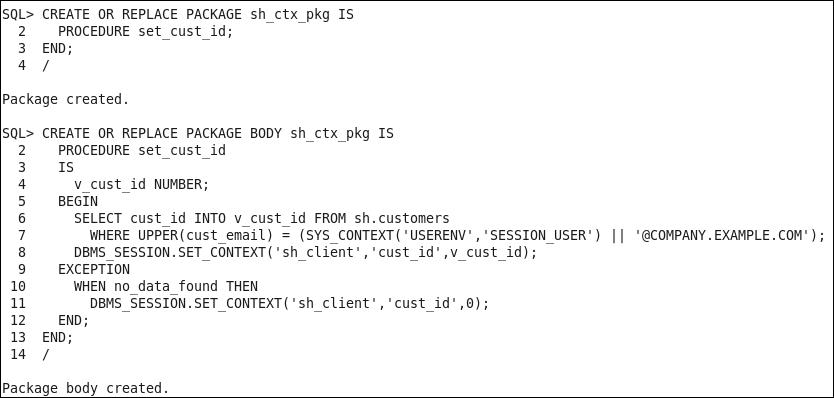
SQL> CREATE CONTEXT <context\_name> USING <PL/SQL\_package\_name>;

https://s3.amazonaws.com/thinkific/file_uploads/154178/images/c07/80f/e6f/1551401564647.jpg

Creating an application context

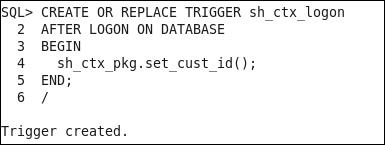
Setting application context attributes

1. Connect to the database as a user who has appropriate
2. Create the PL/SQL package that will set the cust\_id attribute with the value, which is equal to the value of the cust\_id column when the following statement is evaluated: UPPER(cust\_email) = (SYS\_CONTEXT('USERENV', 'SESSION\_USER') || '@COMPANY.EXAMPLE.COM'). In case session user is not a customer, set the value for cust\_id attribute in the application context to 0.



Creating a PL/SQL package

1. Create a logon trigger that calls the sh\_ctx\_pkg.set\_cust\_id procedure.



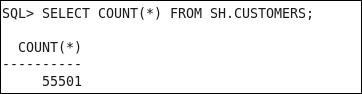
A logon trigger

Using an application context

1. Connect to the database as a newly created user (for example, user sofia):

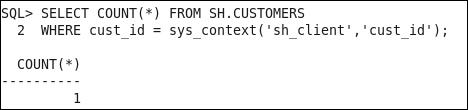
$ sqlplus sofia

1. Verify that the user (for example, sofia) can access all data in the sh.customers table.



The entire data in sh.customers

1. Verify that when executing the following statement, he or she (for example, sofia) can view only his or her data.



Only data about newly created user