Practice

Managing Users and User Security

Practice Target

In this practice, you will perform the common tasks involved in managing database users and user security.

Specifically, you will perform the following:

- Implement a simple example of managing database users and roles
  - Create user accounts and roles and assign privileges to them
  - Use synonyms to access privileged objects
  - Create a user and assign a privilege to it in a single command
  - Examine the difference between `SELECT` and `READ` privileges
  - Examine schema-only accounts
  - Examine password-protected roles
- Check on the user account tablespace quota
- Manage user password settings using user profiles

Practice Assumptions

This practice assumes that the virtual machine `srv1` is restored from its **non-CDB** snapshots and is up and running.

Note: In this practice, you will concentrate on the database user authentication. In a different practice lecture, you will examine other authentications methods: OS authentication and password file authentication.

Preparing for the Practice

This practice is implemented on the non-CDB database. In the following steps, you will restore `srv1` from the non-CDB database snapshot.

1. Make sure `srv1` is shutdown. If it is running, shut it down.

2. In Oracle VirtualBox manager, select `srv1` appliance

3. In the snapshots panel, click on the snapshot "**oradb non-CDB database**" > click on **Restore** button > the following message pops up > **unselect** the checkbox "Create a snapshot of the current machine state" > click on **Restore** button.

4. Startup `srv1`

5. Start a Putty session to `srv1` as `oracle`

6. Verify the non-CDB database is up and running.

```
sqlplus / as sysdba
SELECT CDB FROM V$DATABASE;
```

Managing Database Users and Roles

In the following steps, you will create new database users and perform some security functionality testing on them.

The `HR` account represents the application owner schema. You will create two other user accounts. The first account is a user which has the privileges to query and apply changes on the HR data. The second account has the ability to only read from the `HR` tables.

Creating User Accounts and Roles and Assigning Privileges to them

7. Login to the database as `sys`

```
conn / as sysdba
```

8. Create two database user accounts named as `HR_OFFICER` and `HR_REPORTER`

In our case, these users do not need quota in their default tablespaces because they do not need to create objects for themselves. They only want to accesses `HR` objects.

```
CREATE USER HR_OFFICER IDENTIFIED BY ABcd##1234 DEFAULT TABLESPACE USERS;

CREATE USER HR_REPORTER IDENTIFIED BY ABcd##1234 DEFAULT TABLESPACE USERS;
```

9. Create two roles for the new users.

```
CREATE ROLE HR_OFFICER_ROLE;
CREATE ROLE HR_REPORTER_ROLE;
```

10. Verify that the roles are created.

```
col ROLE for a18
SELECT ROLE, PASSWORD_REQUIRED, AUTHENTICATION_TYPE
FROM DBA_ROLES WHERE ROLE IN ('HR_OFFICER_ROLE', 'HR_REPORTER_ROLE');
```

11. Grant the required privileges to the roles.

Note: the double hyphens are used for writing a single-line remarks in SQL scripts. Sometimes, we use it for describing the code. You can also surround the remarks by '`/*`' and '`*/`' for multiple line remarks.

Note: Normally the object owner grants the privileges to the roles.

```
-- start with HR_OFFICER_ROLE
-- grant the required system privileges
GRANT CREATE SESSION, CREATE SYNONYM TO HR_OFFICER_ROLE;

-- grant the required object privileges to the role:
GRANT SELECT, INSERT, UPDATE ON HR.REGIONS TO HR_OFFICER_ROLE;
GRANT SELECT, INSERT, UPDATE ON HR.COUNTRIES TO HR_OFFICER_ROLE;
GRANT SELECT, INSERT, UPDATE ON HR.LOCATIONS TO HR_OFFICER_ROLE;
GRANT SELECT, INSERT, UPDATE ON HR.DEPARTMENTS TO HR_OFFICER_ROLE;
```

```
GRANT SELECT, INSERT, UPDATE ON HR.JOBS TO HR_OFFICER_ROLE;
GRANT SELECT, INSERT, UPDATE ON HR.EMPLOYEES TO HR_OFFICER_ROLE;
GRANT SELECT, INSERT, UPDATE ON HR.JOB_HISTORY TO HR_OFFICER_ROLE;

-- now for HR_REPORTER_ROLE
-- grant the required system privileges
GRANT CREATE SESSION, CREATE SYNONYM TO HR_REPORTER_ROLE;

-- grant the required object privileges to the role:
GRANT READ ON HR.REGIONS TO HR_REPORTER_ROLE;
GRANT READ ON HR.COUNTRIES TO HR_REPORTER_ROLE;
GRANT READ ON HR.LOCATIONS TO HR_REPORTER_ROLE;
GRANT READ ON HR.DEPARTMENTS TO HR_REPORTER_ROLE;
GRANT READ ON HR.JOBS TO HR_REPORTER_ROLE;
GRANT READ ON HR.EMPLOYEES TO HR_REPORTER_ROLE;
GRANT READ ON HR.JOB_HISTORY TO HR_REPORTER_ROLE;
```

12. Verify that the privileges are assigned to the roles.

```
col GRANTEE for a20
SELECT GRANTEE , PRIVILEGE FROM DBA_SYS_PRIVS WHERE GRANTEE IN
('HR_OFFICER_ROLE', 'HR_REPORTER_ROLE')
ORDER BY 1,2;

SELECT GRANTEE , PRIVILEGE FROM DBA_TAB_PRIVS WHERE GRANTEE IN
('HR_OFFICER_ROLE', 'HR_REPORTER_ROLE')
ORDER BY 1,2;
```

13. Grant the roles to their corresponding users.

```
GRANT HR_OFFICER_ROLE TO HR_OFFICER;
GRANT HR_REPORTER_ROLE TO HR_REPORTER;
```

14. Verify that the roles are granted to the users.

Observe that the same view is used for retrieving privileges granted to roles as well as for retrieving roles granted to users.

```
col GRANTED_ROLE for a20
SELECT GRANTEE , GRANTED_ROLE FROM DBA_ROLE_PRIVS WHERE GRANTEE IN
('HR_OFFICER', 'HR_REPORTER')
ORDER BY 1,2;
```

15. Login to the database as `HR_OFFICER` and verify that it can access the granted `HR` objects.

```
conn HR_OFFICER/ABcd##1234

SELECT COUNT(*) FROM HR.EMPLOYEES;
```

Using Synonyms to Access Privileged Objects

As it is shown in the preceding example, the user needs to access the object using the format `<schema name>.<object name>`. This method is impractical, especially when we have dozens of tables to be seen. One solution to this challenge is by creating synonyms.

16. As the created users, create synonyms for the granted objects with the same name as the original object name.

```
conn HR_OFFICER/ABcd##1234

CREATE SYNONYM REGIONS FOR HR.REGIONS;
CREATE SYNONYM COUNTRIES FOR HR.COUNTRIES;
CREATE SYNONYM LOCATIONS FOR HR.LOCATIONS;
CREATE SYNONYM DEPARTMENTS FOR HR.DEPARTMENTS;
CREATE SYNONYM JOBS FOR HR.JOBS;
CREATE SYNONYM EMPLOYEES FOR HR.EMPLOYEES;
CREATE SYNONYM JOB_HISTORY FOR HR.JOB_HISTORY;

conn HR_REPORTER/ABcd##1234

CREATE SYNONYM REGIONS FOR HR.REGIONS;
CREATE SYNONYM COUNTRIES FOR HR.COUNTRIES;
CREATE SYNONYM LOCATIONS FOR HR.LOCATIONS;
CREATE SYNONYM DEPARTMENTS FOR HR.DEPARTMENTS;
CREATE SYNONYM JOBS FOR HR.JOBS;
CREATE SYNONYM EMPLOYEES FOR HR.EMPLOYEES;
CREATE SYNONYM JOB_HISTORY FOR HR.JOB_HISTORY;
```

17. Verify that each user can access the HR tables.

```
conn HR_OFFICER/ABcd##1234
SELECT COUNT(*) FROM EMPLOYEES;

conn HR_REPORTER/ABcd##1234
SELECT COUNT(*) FROM EMPLOYEES;
```

18. Verify that the `HR_OFFICER` can update `EMPLOYEES` table.

```
conn HR_OFFICER/ABcd##1234
UPDATE EMPLOYEES SET SALARY=SALARY*1 WHERE EMPLOYEE_ID=100;
COMMIT;
```

19. Verify that the `HR_REPORTER` cannot delete from `EMPLOYEES` table.

```
conn HR_REPORTER/ABcd##1234
DELETE EMPLOYEES WHERE EMPLOYEE_ID=100;
```

20. Verify that `HR_REPORTER` cannot update `EMPLOYEES` table.

```
UPDATE EMPLOYEES SET SALARY=SALARY*1 WHERE EMPLOYEE_ID=100;
```

Creating a User and Assigning a Privilege to it in a Single Command

You created two users, two roles, granted privileges to the roles, and tested the privileges functionalities. Let see how we can create a new user and assign a role to it in the same command.

21. Run the following code to

```
conn system/ABcd##1234
GRANT HR_REPORTER_ROLE TO HR_REPORTER2 IDENTIFIED BY ABcd##1234;
```

22. Verify that the new user is granted the `HR_REPORTER_ROLE` privileges.

```
conn HR_REPORTER2/ABcd##1234

SELECT COUNT(*) FROM HR.EMPLOYEES;
```

Examining the Difference between SELECT and READ Privileges

Let's now examine the difference between the `SELECT` and `READ` object privileges.

23. Login as `HR_REPORTER` and try locking the rows in the `EMPLOYEES` table.

The user is unable to lock the rows in the `EMPLOYEES` table because it is assigned the `READ` privilege, not the `SELECT` privilege.

```
conn HR_REPORTER/ABcd##1234

SELECT EMPLOYEE_ID FROM EMPLOYEES FOR UPDATE;
```

24. Login as `HR_OFFICER` and try locking the rows in the `EMPLOYEES` table.

The lock succeeds because the user has `SELECT` privilege on the `EMPLOYEES` table.

```
conn HR_OFFICER/ABcd##1234

SELECT EMPLOYEE_ID FROM EMPLOYEES FOR UPDATE;
-- rollback is to release the lock
ROLLBAC;
```

Examining Schema-only Accounts

Let's convert the HR account to a schema-only account and verify this action does not affect the role functionality.

25. Verify that the HR user is not granted an administrative privilege.

The passwords of administrative accounts are saved externally in the password file.

```
conn / as sysdba

SELECT USERNAME, SYSDBA, SYSOPER, SYSASM, SYSBACKUP, SYSDG, SYSKM
FROM V$PWFILE_USERS WHERE USERNAME = 'HR';
```

26. Convert the HR account into a schema-only account.

```
ALTER USER HR NO AUTHENTICATION;
```

27. Verify that the HR cannot login to the database.

```
conn hr/ABcd##1234
```

28. Login as HR_OFFICER and verify that the account still has access to the HR objects after converting the HR account to schema-only account.

The UPDATE statement succeeds. In real life, if we have a system where all the application objects are contained in a single schema, it is recommended to consider converting the application account into a schema-only account.

```
conn HR_OFFICER/ABcd##1234

UPDATE EMPLOYEES SET SALARY=SALARY*1 WHERE EMPLOYEE_ID=100;
COMMIT;
```

Examining Password-Protected Roles

Let's examine now the password-protected roles.

Note: Password protected roles are normally used to enable the roles from within specific application so that the user cannot enable it from other interfaces like SQL*Plus.

Note: roles could be authenticated by authentication methods different than passwords.

29. Check the value of the parameter SQL92_SECURITY

The parameter is set to TRUE. This means if we want to grant the UPDATE or DELETE object privileges to a user, we must grant the SELECT privilege as well to the user.

```
conn / as sysdba
show parameter SQL92_SECURITY
```

30. Create the following password protected role and assign the UPDATE privilege on EMPLOYEES to it.

```
CREATE ROLE EMP_UPDATE_R IDENTIFIED BY ABcd##1234;
GRANT SELECT, UPDATE ON HR.EMPLOYEES TO EMP_UPDATE_R;
```

31. Grant the role to the HR_REPORTER

```
GRANT EMP_UPDATE_R TO HR_REPORTER;
```

32. Exempt the new role from the user default roles.

```
ALTER USER HR_REPORTER DEFAULT ROLE ALL EXCEPT EMP_UPDATE_R;
```

33. Login as `HR_REPORTER` and try updating the `EMPLOYEES` table.

The user cannot update the `EMPLOYEES` table because the `EMP_UPDATE_R` role is not enabled.

```
conn HR_REPORTER/ABcd##1234
UPDATE EMPLOYEES SET SALARY=SALARY*1 WHERE EMPLOYEE_ID=100;
```

34. Enable the `EMP_UPDATE_R` role then try updating the `EMPLOYEES` table.

In PL/SQL, we would use the `DBMS_SESSION.SET_ROLE` instead.

```
SET ROLE EMP_UPDATE_R IDENTIFIED BY ABcd##1234 ;
UPDATE EMPLOYEES SET SALARY=SALARY*1 WHERE EMPLOYEE_ID=100;
COMMIT;
```

Cleanup

35. Drop the created users and roles.

```
conn / as sysdba
DROP USER HR_OFFICER CASCADE;
DROP USER HR_REPORTER CASCADE;
DROP USER HR_REPORTER2 CASCADE;
DROP ROLE HR_REPORTER_ROLE;
DROP ROLE HR_OFFICER_ROLE;
DROP ROLE EMP_UPDATE_R;
```

Managing User Tablespace Quotas

In the following steps, you will examine how the user tablespace quotas are managed by the database.

36. Create a tablespace with a datafile of size 10M.

```
conn / as sysdba

CREATE TABLESPACE SAMPLE_TBS DATAFILE
'/u01/app/oracle/oradata/ORADB/sampletbs.dbf' SIZE 50M AUTOEXTEND OFF;
```

37. Create a user and set its quota on the tablespace to 5M.

```
CREATE USER USER1 IDENTIFIED BY ABcd##1234
DEFAULT TABLESPACE SAMPLE_TBS
QUOTA 5M ON SAMPLE_TBS;

GRANT CREATE SESSION, CREATE TABLE TO USER1 ;
```

38. As the new user, create a testing table and run a loop code to insert endless data into it.

The code fails and returns the following error:

```
ORA-01536: space quota exceeded for tablespace 'SAMPLE_TBS'
```

```
conn user1/ABcd##1234
-- the table will be created in the SAMPLE_TBS because this tablespace is
--  the default tablespace of the current user
CREATE TABLE TEST ( PID NUMBER, PNAME VARCHAR2(20));

BEGIN
 FOR I IN 1..1000000 LOOP
  INSERT INTO TEST VALUES ( I, DBMS_RANDOM.STRING('x',20));
END LOOP;
END;
/
```

39. Grant unlimited space for the user on the tablespace.

```
conn / as sysdba
ALTER USER USER1 QUOTA UNLIMITED ON SAMPLE_TBS ;
```

40. Re-run the same loading test.

The code fails and returns the following error because the tablespace size reached to its maximum size:

```
ORA-01653: unable to extend table USER1.TEST by 128 in tablespace SAMPLE_TBS
```

```
conn user1/ABcd##1234

BEGIN
 FOR I IN 1..10000000 LOOP
  INSERT INTO TEST VALUES ( I, DBMS_RANDOM.STRING('x',20));
END LOOP;
END;
/
```

41. Expand the tablespace datafile size to 15M.

```
conn / as sysdba
ALTER DATABASE DATAFILE '/u01/app/oracle/ORADB/sampletbs.dbf' RESIZE 15M;
```

42. Try the loading loop again.

It should succeed this time.

```
conn user1/ABcd##1234

BEGIN
 FOR I IN 1..10000000 LOOP
  INSERT INTO TEST VALUES ( I, DBMS_RANDOM.STRING('x',20));
END LOOP;
END;
/
```

Cleanup

43. Drop the created tablespace and user

```
conn / as sysdba

DROP USER USER1 CASCADE;
DROP TABLESPACE SAMPLE_TBS INCLUDING CONTENTS AND DATAFILES;
```

Note: All the concepts and the steps implemented in the preceding steps in this practice apply in CDB databases at the PDB level.

Managing User Password Settings using User Profiles

In the following steps, you will examine the password management settings in user profile.

44. Submit the following query to retrieve list of the user profiles available in the database.

```
conn / as sysdba

SELECT DISTINCT PROFILE FROM DBA_PROFILES;
```

45. Display the password limits in the `DEFAULT` profile.

By default, these limits apply to the newly created database users, unless otherwise is specified in the `CREATE USER` statement.

```
col RESOURCE_NAME for A35
col LIMIT for A30

SELECT RESOURCE_NAME, LIMIT
FROM DBA_PROFILES WHERE RESOURCE_TYPE='PASSWORD' AND PROFILE='DEFAULT';
```

46. Display the password limits of the `ORA_STIG_PROFILE` profile. Compare between its limits and the `DEFAULT` profile limits.

`ORA_STIG_PROFILE` profile has more restricted limits than the `DEFAULT` profile.

```
SELECT RESOURCE_NAME, LIMIT
FROM DBA_PROFILES WHERE RESOURCE_TYPE='PASSWORD' AND PROFILE='ORA_STIG_PROFILE';
```

47. Create a user and assign the `ORA_STIG_PROFILE` profile to it.

The command fails because its password doesn't comply with the password verification complexity function of the `ORA_STIG_PROFILE` profile.

```
CREATE USER USER1 IDENTIFIED BY ABcd##1234 PROFILE ORA_STIG_PROFILE;
```

48. Create the user with a longer password. Grant `CREATE SESSION` to it.

```
CREATE USER USER1 IDENTIFIED BY ABcd##012346789 PROFILE ORA_STIG_PROFILE;
GRANT CREATE SESSION TO USER1;
```

49. Try connecting to the database using the new user and a wrong password for four times.

After the fourth failed attempt, the account is locked. This happened because the `FAILED_LOGIN_ATTEMPTS` is set to 3 in the profile.

```
conn user1/test
```

50. Try connecting to the database using the same account and its correct password.

Although the password is correct, the login fails because the account is locked.

```
conn USER1/ABcd##012346789
```

51. Login as `SYS` and unlock the account.

```
conn / as sysdba
ALTER USER USER1 ACCOUNT UNLOCK;
```

52. Try connecting to the database using the same account and its correct password.

The connection is successful.

```
conn USER1/ABcd##012346789
```

Cleanup

53. Drop the created user

```
conn / as sysdba
-- no need for the CASCADE option because the user owns no object
DROP USER USER1;
```

Summary

Regarding creating users and their security settings, in this practice, we learnt the following:

- Privileges can be assigned to roles and then the roles can be assigned to users.
- If we want to grant a user the ability to query a table without locking its rows, it is better to grant it the `READ` privileges (rather than the `SELECT` privilege).
- Schema-only accounts can be used to host the application objects without allowing any user to login to the database using the application owner account.
- Roles can be protected by passwords. Password-protected roles must be removed from the user default roles.
- We can set a quota for a user from its tablespace. The user cannot consume size from the tablespace more than the quota assigned to it.
- We can apply the password policy settings on database users using user profiles