Oracle 19c New Features
Lab Guide

Table of Contents

Course Practice Environment: Security Credentials

For product-specific credentials used in this course, see the following table:

Product-Specific Credentials		
Product/Application	**Username**	**Password**
VM (for on-premise DBs)		
`ORCL` and `PDB1`	Any user	`Welcome_1`
Other PDBs created in `ORCL`	Any user	`Welcome_1`
Practice 3-2: CDB keystore		`Welcome_1`
Practice 3-2: PDB keystore		*`Your_defined_password`* You can set `Welcome_1` or any other value. Then distinguish *password_CDB* and *password_PDB.*
Practice 3-4: steps i and j, Data Pump export and import encryption password		*`Your_defined_password`* You can set any value for *enc_password*. Then use the same value set at export (step i), at import (step j).
SSO wallet in practice 3-5 step 4 (d)		`Welcome1`

Lab Environment

In this practice, you will start oracle database which has been already created as follows:

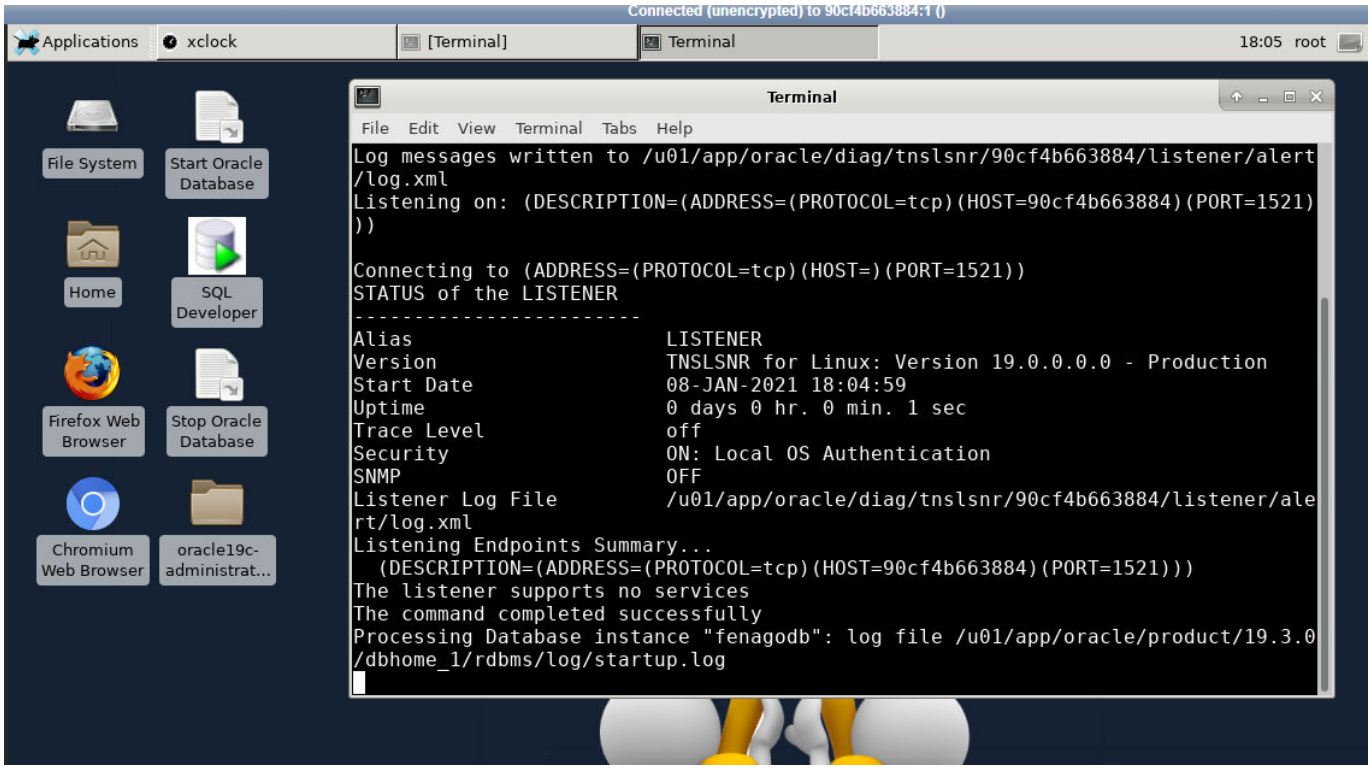
ORACLE_SID=fenagodb

ORACLE_PDB=fenagodb1

ORACLE_PWD=fenago

Tasks

1. Log in to your lab environment and double click **“Start Oracle Database”** shortcut to start database server.

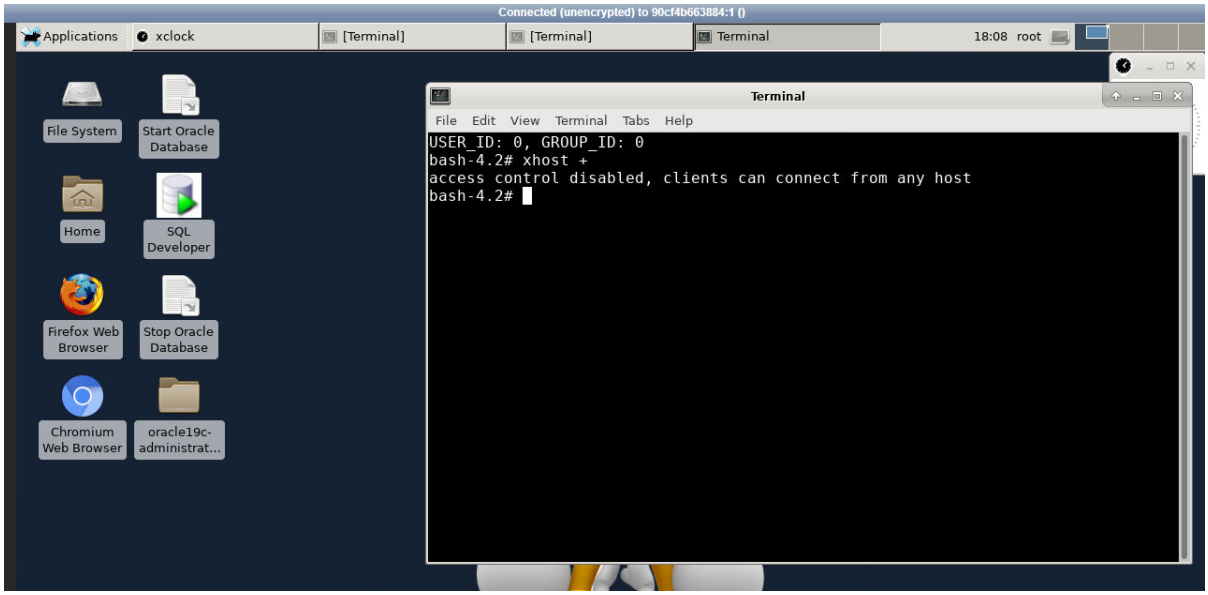


Overview

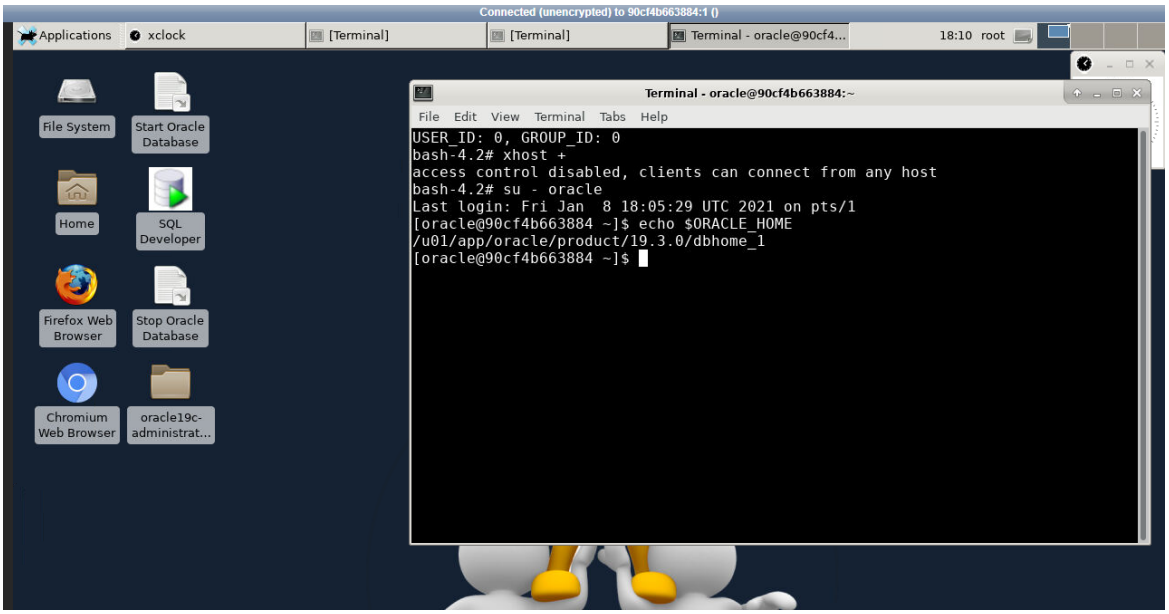
In this practice, you will switch to oracle user from terminal

Tasks

a. Open terminal and run “xhost +” command as root user:



b. Run and run “su - oracle” command in the terminal to switch to **oracle** user:



Practices for Lesson 1: Introduction

Practices for Lesson 1: Overview

Overview

In this practice, you will discover the system environment for the practices of this course.

Notes

- All practices are independent from one lesson to another.
- There are two solutions to clean up your CDB.

1. At the beginning of the practices of each lesson, you can execute the cleanup shell script from previous practices, then the first shell script of the current practice.
2. Alternatively, you can execute the `/home/oracle/labs/admin/cleanup_PDBs.sh` shell script. The shell script drops all PDBs that may have been created by any of the practices, and finally recreates `PDB1`.

Practice 1-1: Discovering the Practice Environment

Your system has one compute node (VM). The VM is dedicated to Oracle Database 19c, installed with a pre-created CDB and one pre-created PDB.

- The `ORCL` CDB with `PDB1` PDB
- Net service names for all PDBs created in practices are already logged in the `$ORACLE_HOME/network/admin/tnsnames.ora` file.

The configuration for the `ORCL` database on the VM corresponds to the preconfigured directories on an Oracle Cloud compute node associated with the pre-created `ORCL` database of a database deployment (or Database Cloud Service instance).

- CDB data files in `/u02/app/oracle/oradata/`
- CDB root data files in `/u02/app/oracle/oradata/<dbname>`
- PDB data files in `/u02/app/oracle/oradata/<dbname>/PDB1`
- Control files in `/u02/app/oracle/oradata/<dbname>` and `/u03/app/oracle/fast_recovery_area/<dbname>`
- All redo log files in `/u04/app/oracle/redo` (The preconfigured directory for redo logs of the unique CDB on Oracle Cloud is `/u04/app/oracle/redo`.)
- All backup files in `/u03/app/oracle/fast_recovery_area/<dbname>`
- Password and initialization parameter files in `$ORACLE_HOME/dbs`
- Diagnostics files in `/u01/app/oracle/diag/rdbms/<dbname_lower_case>/<dbname>/…`
- TDE keystore in `/u01/app/oracle/admin/<dbname>/tde_wallet`
- Net files in `$ORACLE_HOME/network/admin`

1. On the Welcome page of the Secure Global Desktop, in the Cloud Login Instructions section, you find important information:
   - The IP address of the VM
   - The text of the SSH private key that will allow you to connect to the VM. Be ready to copy this text into an SSH private key file that you are going to create. The text starts with `-----BEGIN RSA PRIVATE KEY-----` and ends with `-----END RSA PRIVATE KEY-----`. Both lines will be included in the SSH private key file.
2. In the left navigator of the Secure Global Desktop, click My Training Environments.
3. Select the Linux Desktop and click Start.
4. Enter the `oracle` password and change the password as requested. Then click OK to log in to the Linux Desktop.
5. Right-click Terminal and select Open to open a Linux session.
6. Edit the `/home/oracle/.ssh/config` file and update the first line `Host <IP address of DB instance from OPC>` to `Host IP_Address` with the IP address that you obtained in step 1. Save the file.

7. Open a text editor to create the file, `mykey` and copy the SSH private key text into the file as defined in step 1. Save the file.
8. Change the permissions of the `mykey` file.

```
$ chmod 600 mykey
$
```

9. Use the SSH private key file, `mykey`, to log in to the VM whose IP address has been retrieved in step 1.

```
$ ssh –i mykey oracle@IP_Address
…
[oracle@DBCS18-99699623-DB1 ~]
```

You are now connected to the VM. Observe that the UNIX prompt contains the Unix username and the database deployment name. The prompt is different for every student. Therefore, it is now displayed as `$` in all the following instructions.

10. Check the connections to `ORCL` CDB and verify the `PDB1` PDB open status.

```
$ env | grep ORA
ORACLE_UNQNAME=ORCL
ORACLE_SID=ORCL
ORACLE_HOSTNAME=my_vm
ORACLE_BASE=/u01/app/oracle
ORACLE_HOME=/u01/app/oracle/product/18.0.0/dbhome_1
$ sqlplus / AS SYSDBA

SQL> SELECT name, cdb FROM V$DATABASE;

NAME                      CDB
------------------------------------------------- --------
ORCL                      YES

SQL> SHOW PDBS

    CON_ID CON_NAME                          OPEN MODE  RESTRICTED
---------- ------------------------------_ ---------- ----------
         2 PDB$SEED                        READ ONLY  NO
         3 PDB1                            READ WRITE NO
SQL> EXIT
$
```

Note: The `ORACLE_HOME` value is for the time being 18.0.0 and may change to 18.1.0 over time.

11. Verify that the listener is running and listening to `ORCL` and `pdb1` services. The value *xxxxxxxxx* is assigned during the automated Database Cloud Service instance deployment before the course is started.

```
$ lsnrctl status
…
Service "ORCL.xxxxxxxxx.oraclecloud.internal" has 1 instance(s).
  Instance "ORCL", status READY, has 1 handler(s) for this
service
...
Service "pdb1.xxxxxxxxx.oraclecloud.internal" has 1 instance(s).
  Instance "ORCL", status READY, has 1 handler(s) for this
service...
The command completed successfully
$
```

Practices for Lesson 2:
Leveraging Multitenant Enhancements

Practices for Lesson 2: Overview

Overview

In these practices, you will discover how to monitor PDBs of different CDBs centrally from one CDB lead; create, manage, and restore carousel PDB snapshots; use the dynamic container map; handle lockdown profiles at different levels of the CDB; use static and dynamic lockdown profiles; and finally, switch over refreshable cloned PDBs.

Practice 2-1: Managing a CDB Fleet

Overview

In this practice, you learn how to create a CDB fleet with a CDB lead and its CDB members. You also learn how to monitor PDBs across all CDBs from the CDB lead.

Tasks

1. You use the Oracle By Example “Managing CDB Fleets” to see how to proceed. Launch a browser and click the bookmark from the Bookmarks toolbar of the browser. The URL is: file:////home/oracle/labs/OBE/managing_cdb_fleets/managing_cdb_fleets.html.

Practice 2-2: Managing and Using PDB Snapshots

Overview

In this practice, you enable PDBs for creating PDB snapshots and use the PDB snapshots created manually or automatically to create new PDBs, or flash back a PDB to a point in time.

Tasks

1. Before starting the practice, execute the `$HOME/labs/admin/glogin_2.sh` shell script. It sets formatting for all columns selected in queries and sets appropriate instance parameters for snapshot creation.

```
$ $HOME/labs/admin/glogin_2.sh
…
$
```

2. Create `PDB_1` in `ORCL` and open `PDB_1`.

```
$ mkdir -p /u02/app/oracle/oradata/ORCL/pdb_1
$ sqlplus / AS SYSDBA

SQL> CREATE PLUGGABLE DATABASE pdb_1
       ADMIN USER pdb_1_admin IDENTIFIED BY password
       ROLES=(CONNECT)
       CREATE_FILE_DEST='/u02/app/oracle/oradata/ORCL/pdb_1';
  2    3    4

Pluggable database created.

SQL> ALTER PLUGGABLE DATABASE pdb_1 OPEN;

Pluggable database altered.

SQL>
```

Q/ Are PDBs enabled for manual snapshot creation by default?

```
SQL> SELECT pdb_name, snapshot_mode FROM cdb_pdbs;

PDB_NAME  SNAPSH
--------- -------
PDB_1     MANUAL
PDB$SEED  MANUAL
PDB1      MANUAL

SQL>
```

A/ Yes, they are.

Q/ How many PDB snapshots can be created for a PDB?

```
SQL> SELECT property_name, property_value
     FROM   database_properties
     WHERE  property_name LIKE '%SNAP%';
  2    3
PROPERTY_NAME                                 PROPERTY_VALUE
--------------------------------------------- ----------------------------------------
MAX_PDB_SNAPSHOTS                             8

SQL>
```

A/ Eight, by default

Q/ Can you increase this value?

```
SQL> ALTER DATABASE SET MAX_PDB_SNAPSHOTS = 10;
ALTER DATABASE SET MAX_PDB_SNAPSHOTS = 10
*
ERROR at line 1:
ORA-65046: operation not allowed from outside a pluggable
database

SQL> CONNECT sys@PDB_1 AS SYSDBA
Enter password: password
Connected.
SQL> ALTER DATABASE SET MAX_PDB_SNAPSHOTS = 10;
ALTER DATABASE SET MAX_PDB_SNAPSHOTS = 10
*
ERROR at line 1:
ORA-65383: unable to set MAX_PDB_SNAPSHOTS property to greater
than 8

SQL>
```

A/ No. You can only decrease it. Decrease it to six.

```
SQL> ALTER DATABASE SET MAX_PDB_SNAPSHOTS = 6;

Database altered.

SQL>
```

Q/ Are there any PDB snapshots created for `PDB_1`?

```
SQL> SELECT con_name, snapshot_name, snapshot_scn,
            snapshot_time, full_snapshot_path
     FROM   cdb_pdb_snapshots;
  2    3
no rows selected

SQL>
```

A/ No. `PDB_1` is enabled for manual snapshot creation.

3. Create `pdb_2` in `ORCL`, enabled for automatic PDB snapshot creation every two minutes, and open the PDB.

```
SQL> CONNECT / AS SYSDBA
Connected.
SQL> HOST mkdir -p /u02/app/oracle/oradata/ORCL/pdb_2

SQL> CREATE PLUGGABLE DATABASE pdb_2
       ADMIN USER pdb_2_admin IDENTIFIED BY password
       ROLES=(CONNECT)
       CREATE_FILE_DEST='/u02/app/oracle/oradata/ORCL/pdb_2'
       SNAPSHOT MODE EVERY 2 MINUTES;
   2    3    4    5
Pluggable database created.

SQL> ALTER PLUGGABLE DATABASE pdb_2 OPEN;

Pluggable database altered.

SQL>
```

Q/ How do you check whether the new PDB is enabled for automatic snapshot creation?

```
SQL> SELECT pdb_name, snapshot_mode FROM cdb_pdbs;

PDB_NAME SNAPSH
-------- ------
PDB_1    MANUAL
PDB$SEED MANUAL
PDB1     MANUAL
PDB_2    AUTO

SQL>
```

A/ The CDB_PDBS view contains a new column, SNAPSHOT_MODE.

4. After two minutes, you can verify that PDB snapshots are automatically created for `pdb_2`.

```
SQL> SELECT con_name, snapshot_name, snapshot_scn,
            snapshot_time, full_snapshot_path
     FROM   cdb_pdb_snapshots;
  2    3
CON_NAME SNAPSHOT_NAME                SNAPSHOT_SCN SNAPSHOT_TIME
-------- ---------------------------- ------------ -------------
FULL_SNAPSHOT_PATH
--------------------------------------------------------------------------------
PDB_2    SNAP_1176155135_963393302      2822136    1513934110
/u02/app/oracle/oradata/snap_1176155135_2822136.pdb

SQL>
```

Q/ What type is the PDB snapshot?
A/ The PDB snapshot is an archive file (.pdb) containing the contents of the copy of the PDB at snapshot creation.

Observe that the SCN at which the PDB snapshot was created takes part of the PDB snapshot archive file name.

5. Create PDB snapshots.
   a. Create two manual PDB snapshots for `pdb_1` with different application data. First, execute `$HOME/labs/MULTI/app1.sql` that creates a tablespace, a user, and a table and inserts rows. Then create the `pdb1_first_snap` PDB snapshot.

```
SQL> CONNECT sys@PDB_1 AS SYSDBA
```

```
Enter password: password
Connected.
SQL> @$HOME/labs/MULTI/app1.sql
…
         C LABEL
---------- --------------------------------------------------------------------------------
         1 Label1
…
SQL>
```

```
SQL> ALTER PLUGGABLE DATABASE pdb_1 SNAPSHOT pdb1_first_snap;

Pluggable database altered.

SQL> SELECT con_name, snapshot_name, snapshot_scn,
            snapshot_time, full_snapshot_path
     FROM   cdb_pdb_snapshots;
  2    3
CON_NAME SNAPSHOT_NAME              SNAPSHOT_SCN SNAPSHOT_TIME
-------- -------------------------- ------------ ------------------------------
FULL_SNAPSHOT_PATH
--------------------------------------------------------------------------------
----------
PDB_1    PDB1_FIRST_SNAP                 2823699    1513934427
/u02/app/oracle/oradata/snap_779719242_2823699.pdb

SQL>
```

b. After applying, the application changes by executing `$HOME/labs/MULTI/app2.sql`, which inserts more rows. Then create the `pdb1_second_snap` PDB snapshot.
The errors related to the `HR` schema are expected to show that, in this second PDB snapshot, the `HR` schema will not be present.

```
SQL> @$HOME/labs/MULTI/app2.sql
…
SQL> INSERT INTO schema_app1.tab1 VALUES (2,'Label2');

1 row created.

SQL> CREATE TABLE schema_app1.tab2 (C number, description
VARCHAR2(40));

Table created.
```

```
SQL> INSERT INTO schema_app1.tab2 VALUES (1,'Description1');

1 row created.

SQL> COMMIT;

Commit complete.

SQL> SELECT * FROM schema_app1.tab2;

         C DESCRIPTION
---------- --------------------------------------------------------------------------------
         1 Description1

SQL> set echo off
DROP USER hr CASCADE
          *
ERROR at line 1:
ORA-01918: user 'HR' does not exist


SQL> SELECT * FROM HR.departments;
SELECT * FROM HR.departments
                *
ERROR at line 1:
ORA-00942: table or view does not exist

SQL> ALTER PLUGGABLE DATABASE pdb_1 SNAPSHOT pdb1_second_snap;

Pluggable database altered.

SQL> SELECT con_name, snapshot_name, snapshot_scn,
            snapshot_time, full_snapshot_path
     FROM   cdb_pdb_snapshots;
  2    3
CON_NAME SNAPSHOT_NAME              SNAPSHOT_SCN SNAPSHOT_TIME
---------- ------------------------------ ------------ ------------------------------
FULL_SNAPSHOT_PATH
--------------------------------------------------------------------------------
PDB_1    PDB1_FIRST_SNAP                 2823699    1513934427
/u02/app/oracle/oradata/snap_779719242_2823699.pdb
```

```
PDB_1     PDB1_SECOND_SNAP                2824976    1513934620
/u02/app/oracle/oradata/snap_779719242_2824976.pdb

SQL>
```

c. After creating the second snapshot, the application workload continues. Execute `$HOME/labs/MULTI/app3.sql`, which creates a new table and the `HR` schema.

```
SQL> @$HOME/labs/MULTI/app3.sql
SQL> CREATE TABLE schema_app1.tab3 (COL1 number, Description
VARCHAR2(10));

Table created.

SQL> INSERT INTO schema_app1.tab3 VALUES (1,'Desc1');

1 row created.

SQL> COMMIT;

Commit complete.

SQL> DROP USER hr CASCADE;
DROP USER hr CASCADE
          *
ERROR at line 1:
ORA-01918: user 'HR' does not exist

SQL> CREATE USER hr IDENTIFIED BY oracle_4U;

User created.

SQL> GRANT create session, create table, unlimited tablespace TO
hr;

Grant succeeded.

SQL> CREATE TABLE hr.departments
  2      ( department_id    NUMBER(4)
  3      , department_name  VARCHAR2(30)
  4          CONSTRAINT  dept_name_nn  NOT NULL
  5      , manager_id       NUMBER(6)
  6      , location_id      NUMBER(4)
```

```
  7      ) ;

Table created.

SQL> INSERT INTO hr.departments VALUES
  2          ( 10
  3          , 'Administration'
  4          , 200
  5          , 1700
  6          );

1 row created.

SQL>
SQL> INSERT INTO hr.departments VALUES
  2          ( 20
  3          , 'Marketing'
  4          , 201
  5          , 1800
  6          );

1 row created.

SQL>
SQL> INSERT INTO hr.departments VALUES
  2          ( 30
  3          , 'Purchasing'
  4          , 114
  5          , 1700
  6          );

1 row created.

SQL>
SQL> INSERT INTO hr.departments VALUES
  2          ( 40
  3          , 'Human Resources'
  4          , 203
  5          , 2400
  6          );

1 row created.
```

```
SQL>
SQL> INSERT INTO hr.departments VALUES
  2          ( 50
  3          , 'Shipping'
  4          , 121
  5          , 1500
  6          );

1 row created.

SQL>
SQL> INSERT INTO hr.departments VALUES
  2          ( 60
  3          , 'IT'
  4          , 103
  5          , 1400
  6          );

1 row created.

SQL>
SQL> INSERT INTO hr.departments VALUES
  2          ( 70
  3          , 'Public Relations'
  4          , 204
  5          , 2700
  6          );

1 row created.

SQL>
SQL> INSERT INTO hr.departments VALUES
  2          ( 80
  3          , 'Sales'
  4          , 145
  5          , 2500
  6          );

1 row created.

SQL>
```

```
SQL> INSERT INTO hr.departments VALUES
  2          ( 90
  3          , 'Executive'
  4          , 100
  5          , 1700
  6          );

1 row created.

SQL>
SQL> INSERT INTO hr.departments VALUES
  2          ( 100
  3          , 'Finance'
  4          , 108
  5          , 1700
  6          );

1 row created.

SQL>
SQL> INSERT INTO hr.departments VALUES
  2          ( 110
  3          , 'Accounting'
  4          , 205
  5          , 1700
  6          );

1 row created.

SQL>
SQL> INSERT INTO hr.departments VALUES
  2          ( 120
  3          , 'Treasury'
  4          , NULL
  5          , 1700
  6          );

1 row created.

SQL> INSERT INTO hr.departments VALUES
  2          ( 130
  3          , 'Corporate Tax'
```

```
  4           , NULL
  5           , 1700
  6           );

1 row created.

SQL>
SQL> INSERT INTO hr.departments VALUES
  2           ( 140
  3           , 'Control And Credit'
  4           , NULL
  5           , 1700
  6           );

1 row created.

SQL>
SQL> INSERT INTO hr.departments VALUES
  2           ( 150
  3           , 'Shareholder Services'
  4           , NULL
  5           , 1700
  6           );

1 row created.

SQL>
SQL> INSERT INTO hr.departments VALUES
  2           ( 160
  3           , 'Benefits'
  4           , NULL
  5           , 1700
  6           );

1 row created.

SQL>
SQL> INSERT INTO hr.departments VALUES
  2           ( 170
  3           , 'Manufacturing'
  4           , NULL
  5           , 1700
```

```
  6          );

1 row created.

SQL>
SQL> INSERT INTO hr.departments VALUES
  2          ( 180
  3          , 'Construction'
  4          , NULL
  5          , 1700
  6          );

1 row created.

SQL>
SQL> INSERT INTO hr.departments VALUES
  2          ( 270
  3          , 'Payroll'
  4          , NULL
  5          , 1700
  6          );

1 row created.

SQL>
SQL> COMMIT;

Commit complete.

SQL> SELECT * FROM HR.departments;

DEPARTMENT_ID DEPARTMENT_NAME       MANAGER_ID LOCATION_ID
------------- ------------------------------ ---------- -----------
           10 Administration              200        1700
           20 Marketing                   201        1800
           30 Purchasing                  114        1700
           40 Human Resources             203        2400
           50 Shipping                    121        1500
           60 IT                          103        1400
           70 Public Relations            204        2700
           80 Sales                       145        2500
           90 Executive                   100        1700
```

```
          100 Finance                    108        1700
          110 Accounting                 205        1700
          120 Treasury                              1700
          130 Corporate Tax                         1700
          140 Control And Credit                    1700
          150 Shareholder Services                  1700
          160 Benefits                              1700
          170 Manufacturing                         1700
          180 Construction                          1700
          270 Payroll                               1700

19 rows selected.

SQL>
```

d. Check whether PDB snapshots are still automatically created for `pdb_2`.

```
SQL> CONNECT sys@PDB_2 AS SYSDBA
Enter password: password
Connected.
SQL> SELECT con_name, snapshot_name, snapshot_scn,
            snapshot_time, full_snapshot_path
     FROM   cdb_pdb_snapshots;
  2    3
CON_NAME SNAPSHOT_NAME              SNAPSHOT_SCN SNAPSHOT_TIME
---------- ---------------------------------------- ------------ ---------------------------
FULL_SNAPSHOT_PATH
--------------------------------------------------------------------------------------------------------------------------------
----------
PDB_2    SNAP_1176155135_963393302      2822136    1513934110
/u02/app/oracle/oradata/snap_1176155135_2822136.pdb

PDB_2    SNAP_1176155135_963393421      2822706    1513934229
/u02/app/oracle/oradata/snap_1176155135_2822706.pdb

PDB_2    SNAP_1176155135_963393541      2823167    1513934349
/u02/app/oracle/oradata/snap_1176155135_2823167.pdb

PDB_2    SNAP_1176155135_963393661      2824087    1513934473
/u02/app/oracle/oradata/snap_1176155135_2824087.pdb

PDB_2    SNAP_1176155135_963393781      2824629    1513934596
/u02/app/oracle/oradata/snap_1176155135_2824629.pdb

PDB_2    SNAP_1176155135_963393901      2825586    1513934710
```

```
/u02/app/oracle/oradata/snap_1176155135_2825586.pdb

PDB_2    SNAP_1176155135_963394021      2827895    1513934827
/u02/app/oracle/oradata/snap_1176155135_2827895.pdb

7 rows selected.

SQL>
```

Q/ Why are there more PDB snapshots created than six after around 20 minutes?

A/ When you decreased the value of the maximum PDB snapshots creatable, you were logged in to `PDB_1`. Therefore, `PDB_2` still uses the default value set at the CDB root level. You could set another value for `PDB_2`.

e. After creating a new user in `PDB_2`, create a manual PDB snapshot for `PDB_2` even if the PDB is enabled for automatic PDB snapshots.

```
SQL> CREATE USER test IDENTIFIED BY password;

User created.

SQL> ALTER PLUGGABLE DATABASE pdb_2
           SNAPSHOT pdb_2_manual_tbs_snap;
 2
Pluggable database altered.
```

```
SQL> SELECT con_name, snapshot_name, snapshot_scn,
            snapshot_time, full_snapshot_path
     FROM   cdb_pdb_snapshots;
  2    3
CON_NAME SNAPSHOT_NAME               SNAPSHOT_SCN SNAPSHOT_TIME
-------- --------------------------- ------------ ------------------------------
FULL_SNAPSHOT_PATH
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
PDB_2    SNAP_1176155135_963393302      2822136    1513934110
/u02/app/oracle/oradata/snap_1176155135_2822136.pdb

PDB_2    SNAP_1176155135_963393421      2822706    1513934229
/u02/app/oracle/oradata/snap_1176155135_2822706.pdb

PDB_2    SNAP_1176155135_963393541      2823167    1513934349
/u02/app/oracle/oradata/snap_1176155135_2823167.pdb
```

```
PDB_2    SNAP_1176155135_963393661      2824087    1513934473
/u02/app/oracle/oradata/snap_1176155135_2824087.pdb

PDB_2    SNAP_1176155135_963393781      2824629    1513934596
/u02/app/oracle/oradata/snap_1176155135_2824629.pdb

PDB_2    SNAP_1176155135_963393901      2825586    1513934710
/u02/app/oracle/oradata/snap_1176155135_2825586.pdb

PDB_2    SNAP_1176155135_963394021      2827895    1513934827
/u02/app/oracle/oradata/snap_1176155135_2827895.pdb

PDB_2    PDB_2_MANUAL_TBS_SNAP          2828361    1513934938
/u02/app/oracle/oradata/snap_1176155135_2828361.pdb


8 rows selected.

SQL>
```

Q/ After a certain time elapses (6 minutes), can you retrieve the PDB snapshots automatically created since the beginning of the practice?

```
SQL> SELECT con_name, snapshot_name, snapshot_scn,
            snapshot_time, full_snapshot_path
     FROM   cdb_pdb_snapshots;
  2    3
```

```
CON_NAME SNAPSHOT_NAME              SNAPSHOT_SCN SNAPSHOT_TIME
-------- -------------------------- ------------ ---------------------------
FULL_SNAPSHOT_PATH
--------------------------------------------------------------------------------
----------
PDB_2    SNAP_1176155135_963393421      2822706    1513934229
/u02/app/oracle/oradata/snap_1176155135_2822706.pdb

PDB_2    SNAP_1176155135_963393541      2823167    1513934349
/u02/app/oracle/oradata/snap_1176155135_2823167.pdb

PDB_2    SNAP_1176155135_963393661      2824087    1513934473
/u02/app/oracle/oradata/snap_1176155135_2824087.pdb
PDB_2    SNAP_1176155135_963393781      2824629    1513934596
/u02/app/oracle/oradata/snap_1176155135_2824629.pdb
```

```
PDB_2    SNAP_1176155135_963393901      2825586    1513934710
/u02/app/oracle/oradata/snap_1176155135_2825586.pdb

PDB_2    SNAP_1176155135_963394021      2827895    1513934827
/u02/app/oracle/oradata/snap_1176155135_2827895.pdb

PDB_2    PDB_2_MANUAL_TBS_SNAP          2828361    1513934938
/u02/app/oracle/oradata/snap_1176155135_2828361.pdb

PDB_2    SNAP_1176155135_963394186      2828785    1513934995
/u02/app/oracle/oradata/snap_1176155135_2828785.pdb

8 rows selected.

SQL>
```

A/ The oldest ones were dropped to let new ones be created. It works on a FIFO basis. You can, therefore, flash back to the time or SCN of the available PDB snapshots.

6. Disable PDB snapshot creation for `pdb_2`.

```
SQL> ALTER PLUGGABLE DATABASE pdb_2 SNAPSHOT MODE NONE;

Pluggable database altered.

SQL>
```

Q/ After two minutes elapse, are there still PDB snapshots automatically created for `PDB_2`?

```
SQL> SELECT con_name, snapshot_name, snapshot_scn,
            snapshot_time, full_snapshot_path
     FROM   cdb_pdb_snapshots;
  2    3
PDB_2    SNAP_1176155135_963393541      2823167    1513934349
/u02/app/oracle/oradata/snap_1176155135_2823167.pdb

PDB_2    SNAP_1176155135_963393661      2824087    1513934473
/u02/app/oracle/oradata/snap_1176155135_2824087.pdb

PDB_2    SNAP_1176155135_963393781      2824629    1513934596
/u02/app/oracle/oradata/snap_1176155135_2824629.pdb
```

```
PDB_2    SNAP_1176155135_963393901      2825586    1513934710
/u02/app/oracle/oradata/snap_1176155135_2825586.pdb

PDB_2    SNAP_1176155135_963394021      2827895    1513934827
/u02/app/oracle/oradata/snap_1176155135_2827895.pdb

PDB_2    PDB_2_MANUAL_TBS_SNAP          2828361    1513934938
/u02/app/oracle/oradata/snap_1176155135_2828361.pdb

PDB_2    SNAP_1176155135_963394186      2828785    1513934995
/u02/app/oracle/oradata/snap_1176155135_2828785.pdb

PDB_2    SNAP_1176155135_963394261      2829474    1513935070
/u02/app/oracle/oradata/snap_1176155135_2829474.pdb

8 rows selected.

SQL>
```

A/ There could be one or two, depending on the time you took to launch the disabling command between step 5.e and step 6. You can thus flash back to the time or SCN of the available PDB snapshots.

Q/ How can you check whether the mode is now set to `NONE`?

```
SQL> CONNECT / AS SYSDBA
Connected.
SQL> SELECT pdb_name, snapshot_mode FROM cdb_pdbs;

PDB_NAME SNAPSH
-------- -------
PDB_1    MANUAL
PDB$SEED MANUAL
PDB1     MANUAL
PDB_2    NONE

SQL> EXIT
$
```

A/ Use the `SNAPSHOT_MODE` column in the `CDB_PDBS`.

7. A user asks you to report information about the application in `PDB_1` at different times. Create PDBs from PDB snapshots.

a. A user asks you to report information about the application in `PDB_1` at `pdb1_first_snap` time. Create `pdb_1_from_first_snap` PDB from the `pdb1_first_snap` PDB snapshot.

```
$ mkdir -p /u02/app/oracle/oradata/ORCL/pdb_1_from_first_snap
$ sqlplus / AS SYSDBA

SQL> ALTER SESSION SET db_create_file_dest =
         '/u02/app/oracle/oradata/ORCL/pdb_1_from_first_snap';
 2
Session altered.

SQL> CREATE PLUGGABLE DATABASE pdb_1_from_first_snap
      FROM pdb_1 USING SNAPSHOT pdb1_first_snap;
 2
Pluggable database created.

SQL> ALTER PLUGGABLE DATABASE pdb_1_from_first_snap OPEN;

Pluggable database altered.

SQL>
```

```
SQL> CONNECT sys@pdb_1_from_first_snap AS SYSDBA
Enter password: password
Connected.
SQL> SELECT * FROM schema_app1.tab1;

         C LABEL
---------- --------------------------------------------------------------------------------
         1 Label1

SQL> SELECT * FROM schema_app1.tab2;
SELECT * FROM schema_app1.tab2
                          *
ERROR at line 1:
ORA-00942: table or view does not exist

SQL> SELECT * FROM schema_app1.tab3;
SELECT * FROM schema_app1.tab3
                          *
ERROR at line 1:
ORA-00942: table or view does not exist
```

```
SQL> SELECT * FROM hr.departments;
SELECT * FROM hr.departments
                         *
ERROR at line 1:
ORA-00942: table or view does not exist

SQL>
```

b. The user now asks you to report information about the application in `PDB_1` at `pdb1_second_snap` time. Create the `pdb_1_from_second_snap` PDB from `pdb1_second_snap` PDB snapshot.

```
SQL> host mkdir -p
/u02/app/oracle/oradata/ORCL/pdb_1_from_second_snap
SQL> CONNECT / AS SYSDBA
Connected.
SQL> ALTER SESSION SET db_create_file_dest =
'/u02/app/oracle/oradata/ORCL/pdb_1_from_second_snap';
 2
Session altered.

SQL> CREATE PLUGGABLE DATABASE pdb_1_from_second_snap
      FROM pdb_1 USING SNAPSHOT pdb1_second_snap;
 2
Pluggable database created.

SQL> ALTER PLUGGABLE DATABASE pdb_1_from_second_snap OPEN;

Pluggable database altered.

SQL> CONNECT sys@pdb_1_from_second_snap AS SYSDBA
Enter password: password
Connected.
SQL> SELECT * FROM schema_app1.tab1;

         C LABEL
---------- --------------------------------------------------------------------------------
         1 Label1
         2 Label2

SQL> SELECT * FROM schema_app1.tab2;

         C DESCRIPTION
---------- --------------------------------------------------------------------------------
```

```
         1 Description1

SQL> SELECT * FROM hr.departments;
SELECT * FROM hr.departments
                          *
ERROR at line 1:
ORA-00942: table or view does not exist

SQL>
SQL> SELECT * FROM schema_app1.tab3;
SELECT * FROM schema_app1.tab3
                          *
ERROR at line 1:
ORA-00942: table or view does not exist

SQL>
```

```
SQL> CONNECT / AS SYSDBA
Connected.
SQL> SHOW PDBS

    CON_ID CON_NAME                       OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         2 PDB$SEED                       READ ONLY  NO
         3 PDB_1                          READ WRITE NO
         4 PDB1                           READ WRITE NO
         5 PDB_2                          READ WRITE NO
         6 PDB_1_FROM_FIRST_SNAP          READ WRITE NO
         8 PDB_1_FROM_SECOND_SNAP         READ WRITE NO
SQL>
```

8. For further tests, execute `$HOME/labs/MULTI/app4.sql`. The script executes the following steps:
   - Creates the `pdb1_third_snap` PDB snapshot
   - Drops the `HR` schema
   - Creates the `pdb1_forth_snap` PDB snapshot

```
SQL> @$HOME/labs/MULTI/app4.sql
…
SQL>
```

9. You do not need the `pdb1_first_snap` and `pdb1_second_snap` PDB snapshots. You decide to drop them.

```
SQL> CONNECT sys@PDB_1 AS SYSDBA
Enter password: password
```

```
Connected.
SQL> ALTER PLUGGABLE DATABASE pdb_1
           DROP SNAPSHOT pdb1_first_snap;
 2
Pluggable database altered.

SQL> ALTER PLUGGABLE DATABASE pdb_1
           DROP SNAPSHOT pdb1_second_snap;
 2
Pluggable database altered.

SQL>
```

Q/ Do the PDBs created on the dropped PDB snapshots still exist?

```
SQL> CONNECT / AS SYSDBA
Connected.
SQL> SHOW PDBS

    CON_ID CON_NAME                       OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         2 PDB$SEED                       READ ONLY  NO
         3 PDB_1                          READ WRITE NO
         4 PDB1                           READ WRITE NO
         5 PDB_2                          READ WRITE NO
         6 PDB_1_FROM_FIRST_SNAP          READ WRITE NO
         8 PDB_1_FROM_SECOND_SNAP         READ WRITE NO
SQL>
```

```
SQL> SELECT snapshot_name, snapshot_scn,
            snapshot_time, full_snapshot_path
     FROM   cdb_pdb_snapshots
     WHERE  con_name = 'PDB_1';
  2    3    4
SNAPSHOT_NAME              SNAPSHOT_SCN SNAPSHOT_TIME
-------------------------- ------------ ---------------------------
FULL_SNAPSHOT_PATH
--------------------------------------------------------------------------------------------------------------------------------
----------
PDB1_FORTH_SNAP                 2835359    1513935844
/u02/app/oracle/oradata/snap_779719242_2835359.pdb

PDB1_THIRD_SNAP                 2834784    1513935784
```

```
/u02/app/oracle/oradata/snap_779719242_2834784.pdb

SQL>
```

A/ Yes. The PDBs created on dropped PDB snapshots are not implicitly dropped when the PDB snapshots are dropped. They are dropped with the DROP PLUGGABLE DATABASE command.

10. Imagine a situation where you are informed that a user has inadvertently dropped the `HR` schema in `pdb_1 between the time when pdb1_third_snap and pdb1_forth_snap were created`. (This is what `$HOME/labs/MULTI/app4.sql` executed behind the scenes.) Recover the situation when the `HR` schema still existed.
    a. Close `PDB_1`.

```
SQL> ALTER PLUGGABLE DATABASE pdb_1 CLOSE;

Pluggable database altered.

SQL>
```

    b. Create `PDB_1b` from the `pdb1_third_snap` snapshot created before the user's error.

```
SQL> HOST mkdir /u02/app/oracle/oradata/ORCL/pdb_1b

SQL> ALTER SESSION SET
db_create_file_dest='/u02/app/oracle/oradata/ORCL/pdb_1b';

Session altered.

SQL> CREATE PLUGGABLE DATABASE pdb_1b FROM pdb_1
             USING SNAPSHOT pdb1_third_snap;
  2    3
Pluggable database created.

SQL> ALTER PLUGGABLE DATABASE pdb_1b OPEN;

Pluggable database altered.

SQL>
```

    c. Drop `PDB_1`.

```
SQL> DROP PLUGGABLE DATABASE pdb_1 INCLUDING DATAFILES;

Pluggable database dropped.

SQL>
```

d. Rename PDB_1b to PDB_1.

```
SQL> ALTER SESSION SET CONTAINER = pdb_1b;

Session altered.

SQL> ALTER PLUGGABLE DATABASE CLOSE IMMEDIATE;

Pluggable database altered.

SQL> ALTER PLUGGABLE DATABASE OPEN RESTRICTED;

Pluggable database altered.

SQL> ALTER PLUGGABLE DATABASE RENAME GLOBAL_NAME TO pdb_1;

Pluggable database altered.

SQL> ALTER PLUGGABLE DATABASE CLOSE IMMEDIATE;

Pluggable database altered.

SQL>
```

e. Open PDB_1.

```
SQL> ALTER PLUGGABLE DATABASE OPEN;

Pluggable database altered.

SQL>
```

f. Verify that the HR schema now exists in PDB_1.

```
SQL> CONNECT sys@PDB_1 AS SYSDBA
Enter password: password
Connected.
SQL> SELECT * FROM hr.departments;

 DEPARTMENT_ID DEPARTMENT_NAME       MANAGER_ID LOCATION_ID
---------------- ------------------------------ ---------- -----------
            10 Administration             200        1700
            20 Marketing                  201        1800
            30 Purchasing                 114        1700
            40 Human Resources            203        2400
            50 Shipping                   121        1500
            60 IT                         103        1400
```

```
            70 Public Relations                 204        2700
            80 Sales                            145        2500
            90 Executive                        100        1700
           100 Finance                          108        1700
           110 Accounting                       205        1700
           120 Treasury                                    1700
           130 Corporate Tax                               1700
           140 Control And Credit                          1700
           150 Shareholder Services                        1700
           160 Benefits                                    1700
           170 Manufacturing                               1700
           180 Construction                                1700
           270 Payroll                                     1700

19 rows selected.

SQL>
```

g. Create a new snapshot for a potential further recovery.

```
SQL> ALTER PLUGGABLE DATABASE pdb_1 SNAPSHOT pdb1_fifth_snap;

Pluggable database altered.

SQL> EXIT
$
```

11. Execute the `$HOME/labs/MULTI/cleanup_snapshots.sh` shell script. It drops all PDB snapshots, and the `pdb_1 and pdb_2 PDBs`.

```
$ $HOME/labs/MULTI/cleanup_snapshots.sh
…
$
```

Practice 2-3: Using a Dynamic Container Map

Overview

In this practice, you will use dynamic container maps. In Oracle Database 12c, a container map enables a session connected to an application root to issue SQL statements that are routed to the appropriate PDB, depending on the value of a predicate used in the SQL statement. Container maps enable the partitioning of data at the application PDB level when the data is not physically partitioned at the table level.

In Oracle Database 19c, a container map can be dynamically updated when new PDBs are created or dropped.

Tasks

1. Before starting the practice, execute the `$HOME/labs/MULTI/Dynamic_container_map.sh` shell script. It creates the `HR_ROOT` application root, installs `HR_APP` in the application root, then creates the `ACCOUNTING`, `RESEARCH, and SALES` application PDBs. The application `HR_APP` contains two `SCOTT` tables, `DEPT` and `EMP`. There is a container map table created and set at application root `HR_ROOT` level that maps department 10 to the `ACCOUNTING` PDB, department 20 to the `RESEARCH` PDB, and department 30 to the `SALES` PDB.

```
$ $HOME/labs/MULTI/Dynamic_container_map.sh
…
$
```

2. Connect to `HR_ROOT` as `SYS` and verify the contents of the `SCOTT.DEPT` and `SCOTT.EMP` tables.

```
$ sqlplus sys@HR_ROOT AS SYSDBA

Enter password: password

SQL> SELECT deptno, dname, loc FROM scott.dept;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        30 SALES          CHICAGO
        20 RESEARCH       DALLAS

SQL> SELECT empno, ename, deptno FROM scott.emp;

     EMPNO ENAME          DEPTNO
---------- ---------- ----------
      7369 SMITH              20
```

```
      7566 JONES               20
      7788 SCOTT               20
      7876 ADAMS               20
      7902 FORD                20
      7782 CLARK               10
      7839 KING                10
      7934 MILLER              10
      7499 ALLEN               30
      7521 WARD                30
      7654 MARTIN              30
      7698 BLAKE               30
      7844 TURNER              30
      7900 JAMES               30

14 rows selected.

SQL>
```

3. Find the application PDBs associated to `HR_ROOT`.

```
SQL> SHOW PDBS

    CON_ID CON_NAME                       OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
                                 6 HR_ROOT   READ WRITE NO
                                 7 SALES     READ WRITE NO
         8 ACCOUNTING                      READ WRITE NO
        12 RESEARCH                        READ WRITE NO
SQL>
```

4. Find the container map table and the description of the container map table.

```
SQL> SELECT owner, table_name FROM dba_tables
     WHERE  container_map_object='YES';
 2
OWNER           TABLE_NAME
--------------- ----------------------------------------------------
SCOTT           MAPTABLE

SQL> SELECT dbms_metadata.get_ddl('TABLE','MAPTABLE','SCOTT')
     FROM dual;
 2
DBMS_METADATA.GET_DDL('TABLE','MAPTABLE','SCOTT')
--------------------------------------------------------------------------------

  CREATE TABLE "SCOTT"."MAPTABLE"
```

```
   (    "DEPTNO" NUMBER,
        "NAME" VARCHAR2(30)
   ) PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
  STORAGE(
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
DEFAULT)
  TABLESPACE "SYSTEM"
  PARTITION BY LIST ("DEPTNO")
 (PARTITION "ACCOUNTING" VALUES (10) SEGMENT CREATION DEFERRED
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
 NOCOMPRESS LOGGING
  STORAGE(
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
DEFAULT)
  TABLESPACE "SYSTEM" ,
 PARTITION "RESEARCH" VALUES (20) SEGMENT CREATION DEFERRED
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
 NOCOMPRESS LOGGING
  STORAGE(
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
DEFAULT)
  TABLESPACE "SYSTEM" ,
 PARTITION "SALES" VALUES (30) SEGMENT CREATION DEFERRED
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
 NOCOMPRESS LOGGING
  STORAGE(
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
DEFAULT)
  TABLESPACE "SYSTEM" )

SQL>
```

5. Verify that the query for department 20 is routed to the appropriate PDB.

```
SQL> SET LINESIZE 180
SQL> SELECT deptno, dname, loc FROM scott.dept
     WHERE deptno = 20;
 2
    DEPTNO DNAME          LOC
---------- -------------- -------------
        20 RESEARCH       DALLAS

SQL> SELECT * FROM dbms_xplan.display_cursor();

PLAN_TABLE_OUTPUT
```

```
-------------------------------------------------------------------------------------------------------------
SQL_ID  fp3p8g8cxr4jm, child number 0
-------------------------------------
SELECT deptno, dname, loc FROM scott.dept      WHERE deptno = 20

Plan hash value: 1918422367

-------------------------------------------------------------------------------------------------------------
| Id  | Operation             | Name | Rows  | Bytes | Cost (%CPU)| Time      | Pstart| Pstop |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |      |       |       |     2 (100)|           |       |       |
|   1 |  PARTITION LIST SINGLE|      |   700 | 21000 |     2 (100)| 00:00:01 |     2 |     2 |
|   2 |   CONTAINERS FULL     | DEPT |   700 | 21000 |     2 (100)| 00:00:01 |       |       |
-------------------------------------------------------------------------------------------------------------

16 rows selected.

SQL>
```

The output of the execution plan is truncated to display the relevant columns only.

6. Create a new PDB for departments 60, 70, 80, 90, and 100.

```
SQL> CREATE PLUGGABLE DATABASE devt
             ADMIN USER admin IDENTIFIED BY password
  CREATE_FILE_DEST='/u02/app/oracle/oradata/ORCL/hr_root/devt'
  CONTAINER_MAP UPDATE
            (ADD PARTITION devt VALUES (60, 70, 80, 90,100));
 2    3    4    5
Pluggable database created.

SQL>
```

7. Open the PDB.

```
SQL> ALTER PLUGGABLE DATABASE devt OPEN;

Pluggable database altered.

SQL>
```

8. Synchronize the application `HR_APP` in `DEVT`.

```
SQL> CONNECT sys@DEVT AS SYSDBA
Enter password: password
Connected.
SQL> ALTER PLUGGABLE DATABASE APPLICATION hr_app SYNC;

Pluggable database altered.

SQL>
```

9. Insert rows into `SCOTT.DEPT` and `SCOTT.EMP`.

```
SQL> INSERT INTO scott.DEPT VALUES (60,'DEVT','FRANCE');

1 row created.

SQL> INSERT INTO scott.DEPT VALUES (70,'DEVT','GERMANY');

1 row created.

SQL> INSERT INTO scott.DEPT VALUES (40,'Admin','Poland');
INSERT INTO scott.DEPT VALUES (40,'Admin','Poland')
                  *
ERROR at line 1:
ORA-65391: violation of container map partitions

SQL> INSERT INTO scott.EMP VALUES
(7499,'ALLEN','Dev',7698,TO_date('20-2-1981','dd-mm-
yyyy'),1600,300,60);

1 row created.

SQL> INSERT INTO scott.EMP VALUES
(7521,'WARD','Dev',7698,TO_date('22-2-1981','dd-mm-
yyyy'),1250,500,70);

1 row created.

SQL> INSERT INTO scott.EMP VALUES
(7654,'MARTIN','Admin',7698,TO_date('28-9-1981','dd-mm-
yyyy'),1250,1400,40);
INSERT INTO scott.EMP VALUES
(7654,'MARTIN','Admin',7698,TO_date('28-9-1981','dd-mm-
yyyy'),1250,1400,40)
*
ERROR at line 1:
ORA-02291: integrity constraint (SCOTT.FK_DEPTNO) violated -
parent
key not found

SQL> COMMIT;

Commit complete.

SQL>
```

Q/ Why do the third and last inserts fail?

A/ Department 40 does not refer to any PDB whose value is defined in the partitions of the LIST partitioned map table SCOTT.MAPTABLE.

10. Verify that department 60 and 70 correspond to values defined in partitions of the LIST partitioned map table SCOTT.MAPTABLE.

```
SQL> CONNECT sys@HR_ROOT AS SYSDBA
Enter password: password
Connected.
SQL> SELECT partition_name, high_value
     FROM   dba_tab_partitions
     WHERE  table_name='MAPTABLE' ORDER BY 1;
  2    3
PARTITION_NAME HIGH_VALUE
------------------------------ --------------------------------------------------------------------------------
ACCOUNTING     10
DEVT           60, 70, 80, 90, 100
RESEARCH       20
SALES          30

SQL>
```

Q/ Are the partitions automatically created in the map table?

A/ Yes. The creation of the PDB DEVT for departments 60, 70, 80, 90, and 100 defined a new partition in the LIST partitioned map table SCOTT.MAPTABLE.

11. You want to maintain departments 60 and 70 in the DEVT PDB but move the rest of the values defined in DEVT into another PDB, ADMINISTRATION.

```
SQL> CREATE PLUGGABLE DATABASE administration ADMIN USER admin
            IDENTIFIED BY password
CREATE_FILE_DEST =
         '/u02/app/oracle/oradata/ORCL/hr_root/administration'
  CONTAINER_MAP UPDATE
       (SPLIT PARTITION devt
        INTO (PARTITION devt values (60, 70),
              PARTITION administration));
 2   3   4   5   6   7   8
Pluggable database created.

SQL>
```

12. Open the PDB.

```
SQL> ALTER PLUGGABLE DATABASE administration OPEN;

Pluggable database altered.

SQL>
```

13. Synchronize the application HR_APP in ADMINISTRATION.

```
SQL> CONNECT sys@ADMINISTRATION AS SYSDBA
Enter password: password
Connected.
SQL> ALTER PLUGGABLE DATABASE APPLICATION hr_app SYNC;

Pluggable database altered.

SQL>
```

14. Verify that the department 60 and 70 correspond to values defined for the DEVT PDB and 80, 90, and 100 for the ADMINISTRATION PDB, and that a new partition is created in the LIST partitioned map table SCOTT.MAPTABLE.

```
SQL> CONNECT sys@HR_ROOT AS SYSDBA
Enter password: password
Connected.
SQL> SELECT partition_name, high_value
     FROM   dba_tab_partitions
     WHERE  table_name='MAPTABLE' ORDER BY 1;
  2    3
PARTITION_NAME HIGH_VALUE
------------------------------ --------------------------------------------------------------------------------
ACCOUNTING     10
ADMINISTRATION 80, 90, 100
DEVT           60, 70
RESEARCH       20
SALES          30

SQL> EXIT
$
```

15. Execute the $HOME/labs/MULTI/cleanup_container_map.sh shell script. It drops the DEVT and ADMINISTRATION PDBs.

```
$ $HOME/labs/MULTI/cleanup_container_map.sh
…
$
```

Practice 2-4: Using Static and Dynamic Lockdown Profiles

Overview

In this practice, you will observe how lockdown profiles in the CDB root, in application roots, and application PDBs behave regarding inheritance. You will create and alter lockdown profiles and set the profiles at different PDB levels in a CDB. You will also discover the behavior of static and dynamic lockdown profiles created from base lockdown profiles.

Tasks

1. Before starting the practice, execute the `$HOME/labs/MULTI/HR_ROOT.sh` shell script. It creates the `HR_ROOT` application root, installs `HR_APP` in the application root, and creates the `OPERATIONS` and `SALES` application PDBs.

```
$ $HOME/labs/MULTI/HR_ROOT.sh
…
$
```

2. Create lockdown profiles at different levels in the CDB.
   a. Create two lockdown profiles in the CDB root: `CDB_prof1` and `CDB_prof2`.

```
$ sqlplus / AS SYSDBA

SQL> CREATE LOCKDOWN PROFILE CDB_prof1;

Lockdown Profile created.

SQL> ALTER LOCKDOWN PROFILE CDB_prof1
           DISABLE STATEMENT = ('alter system');
 2
Lockdown Profile altered.

SQL> ALTER LOCKDOWN PROFILE CDB_prof1
           ENABLE STATEMENT = ('alter system')
           CLAUSE = ('set');
  2    3
Lockdown Profile altered.

SQL>
```

```
SQL> CREATE LOCKDOWN PROFILE CDB_prof2;

Lockdown Profile created.

SQL> ALTER LOCKDOWN PROFILE CDB_prof2
```

```
           DISABLE STATEMENT = ('alter pluggable database');
 2
Lockdown Profile altered.

SQL>
```

b. Verify the existence of lockdown profiles.

```
SQL> SELECT profile_name, rule, clause, status
     FROM   cdb_lockdown_profiles;
 2
PROFILE_NAME   RULE                          CLAUSE          STATUS
-------------- ----------------------------- --------------- --------
CDB_PROF1      ALTER SYSTEM                                  DISABLE
CDB_PROF1      ALTER SYSTEM                  SET             ENABLE
CDB_PROF2      ALTER PLUGGABLE DATABASE                      DISABLE
PRIVATE_DBAAS                                                EMPTY
PUBLIC_DBAAS                                                 EMPTY
SAAS                                                         EMPTY

6 rows selected.

SQL>
```

c. Log in to the `HR_ROOT` application root `as HR_LOCK_MGR` to create the `app_root_prof` lockdown profile in `HR_ROOT`.

```
SQL> CONNECT hr_lock_mgr@hr_root
Enter password: password
Connected.
SQL> CREATE LOCKDOWN PROFILE app_root_prof;

Lockdown Profile created.

SQL> ALTER LOCKDOWN PROFILE app_root_prof
           DISABLE STATEMENT = ('alter system');
  2
Lockdown Profile altered.

SQL> ALTER LOCKDOWN PROFILE app_root_prof
           ENABLE STATEMENT = ('alter system')
           CLAUSE = ('flush shared_pool');
  2    3
Lockdown Profile altered.

SQL>
```

d. Verify the existence of the lockdown profile in the application root.

```
SQL> SELECT profile_name, rule, clause, status
     FROM   cdb_lockdown_profiles;
 2
PROFILE_NAME   RULE                CLAUSE                STATUS
-------------- ------------------- --------------------- --------
APP_ROOT_PROF  ALTER SYSTEM                              DISABLE
APP_ROOT_PROF  ALTER SYSTEM        FLUSH SHARED_POOL     ENABLE

SQL>
```

e. Create the `sales_prof` lockdown profile for the `SALES` application PDB in the application root.

```
SQL> CREATE LOCKDOWN PROFILE sales_prof;

Lockdown Profile created.

SQL> ALTER LOCKDOWN PROFILE sales_prof
           DISABLE STATEMENT = ('alter pluggable database');
  2
Lockdown Profile altered.

SQL> SELECT profile_name, rule, clause, status
     FROM  cdb_lockdown_profiles;
  2
PROFILE_NAME   RULE                       CLAUSE
STATUS
-------------- -- ------------------------------------------------------- ---------------------------------------------
--------
APP_ROOT_PROF ALTER SYSTEM
DISABLE

APP_ROOT_PROF  ALTER SYSTEM              FLUSH SHARED_POOL
ENABLE

SALES_PROF     ALTER PLUGGABLE DATABASE
DISABLE


SQL>
```

3. Observe profile lockdown inheritance when the CDB root lockdown profile is set.

a. Set the `CDB_prof1` lockdown profile in the CDB root.

```
SQL> CONNECT / AS SYSDBA
Connected.
SQL> SHOW PARAMETER pdb_lockdown
```

```
NAME                                 TYPE         VALUE
------------------------------------ ------------ ------------------------------
pdb_lockdown                         string

SQL> ALTER SYSTEM SET pdb_lockdown = CDB_prof1 SCOPE = both;

System altered.

SQL> SHOW PARAMETER pdb_lockdown

NAME                                 TYPE         VALUE
------------------------------------ ------------ ------------------------------
pdb_lockdown                         string       CDB_PROF1
SQL>
```

b. Observe how the `PDB1` regular PDB inherits the restrictions of the CDB root lockdown profile.

```
SQL> CONNECT sys@PDB1 as sysdba
Enter password: password
Connected.
SQL> SHOW PARAMETER pdb_lockdown

NAME                                 TYPE         VALUE
------------------------------------ ------------ ------------------------------
pdb_lockdown                         string       CDB_PROF1
SQL> ALTER SYSTEM SET ddl_lock_timeout=30 scope=both;

System altered.

SQL> ALTER SYSTEM flush shared_pool;
ALTER SYSTEM flush shared_pool
*
ERROR at line 1:
ORA-01031: insufficient privileges

SQL> ALTER SYSTEM CHECKPOINT;
ALTER SYSTEM CHECKPOINT
*
ERROR at line 1:
ORA-01031: insufficient privileges

SQL>
```

c. Observe how the HR_ROOT application root inherits the restrictions of the CDB root lockdown profile.

```
SQL> CONNECT sys@hr_root AS SYSDBA
Enter password: password
Connected.
SQL> SHOW PARAMETER pdb_lockdown

NAME                                 TYPE         VALUE
------------------------------------ ------------ ------------------------------
pdb_lockdown                         string       CDB_PROF1
SQL> ALTER SYSTEM SET ddl_lock_timeout=30 scope=both;

System altered.

SQL> ALTER SYSTEM flush shared_pool;
ALTER SYSTEM flush shared_pool
*
ERROR at line 1:
ORA-01031: insufficient privileges


SQL> ALTER SYSTEM CHECKPOINT;
ALTER SYSTEM CHECKPOINT
*
ERROR at line 1:
ORA-01031: insufficient privileges

SQL>
```

d. Observe how the SALES application PDB inherits the restrictions of the CDB root lockdown profile.

```
SQL> CONNECT sys@sales AS SYSDBA
Enter password: password
Connected.
SQL> SHOW PARAMETER pdb_lockdown

NAME                                 TYPE         VALUE
------------------------------------ ------------ ------------------------------
pdb_lockdown                         string       CDB_PROF1
SQL> ALTER SYSTEM SET ddl_lock_timeout=30 scope=both;

System altered.
```

```
SQL> ALTER SYSTEM flush shared_pool;
ALTER SYSTEM flush shared_pool
*
ERROR at line 1:
ORA-01031: insufficient privileges


SQL> ALTER SYSTEM CHECKPOINT;
ALTER SYSTEM CHECKPOINT
*
ERROR at line 1:
ORA-01031: insufficient privileges

SQL>
```

4. Observe profile lockdown inheritance when the application root lockdown profile is set in the application root.
   a. Set the `app_root_prof` lockdown profile in the application root.

```
SQL> CONNECT hr_lock_mgr@hr_root
Enter password: password
Connected.
SQL> SHOW PARAMETER pdb_lockdown

NAME                                 TYPE         VALUE
------------------------------------ ------------ ------------------------------
pdb_lockdown                         string       CDB_PROF1
SQL> ALTER SYSTEM SET pdb_lockdown = app_root_prof
           SCOPE = both;
  2
System altered.

SQL> SHOW PARAMETER pdb_lockdown

NAME                                 TYPE         VALUE
------------------------------------ ------------ ------------------------------
pdb_lockdown                         string       APP_ROOT_PROF
SQL>
```

   b. Observe how the `HR_ROOT` application root inherits the restrictions of the CDB root lockdown profile.

```
SQL> CONNECT system@hr_root
Enter password: password
Connected.
SQL> SHOW PARAMETER pdb_lockdown
```

```
NAME                                 TYPE         VALUE
------------------------------------ ------------ ------------------------------
pdb_lockdown                         string       APP_ROOT_PROF
SQL> ALTER SYSTEM SET ddl_lock_timeout=30 scope=both;

System altered.

SQL> ALTER SYSTEM flush shared_pool;
ALTER SYSTEM flush shared_pool
*
ERROR at line 1:
ORA-01031: insufficient privileges


SQL> ALTER SYSTEM CHECKPOINT;
ALTER SYSTEM CHECKPOINT
*
ERROR at line 1:
ORA-01031: insufficient privileges


SQL>
```

Observe that `app_root_prof` affects all application PDBs in the application container, but the application root still inherits the rules of its nearest ancestor, `CDB_prof1`.

c. Observe how the `SALES` application PDB inherits the restrictions of the application root lockdown profile.

```
SQL> CONNECT sys@sales AS SYSDBA
Enter password: password
Connected.
SQL> ALTER SYSTEM SET ddl_lock_timeout=30 scope=both;
ALTER SYSTEM SET ddl_lock_timeout=30 scope=both
*
ERROR at line 1:
ORA-01031: insufficient privileges


SQL> ALTER SYSTEM flush shared_pool;
ALTER SYSTEM flush shared_pool
*
```

```
ERROR at line 1:
ORA-01031: insufficient privileges


SQL> ALTER SYSTEM CHECKPOINT;
ALTER SYSTEM CHECKPOINT
*
ERROR at line 1:
ORA-01031: insufficient privileges


SQL>
```

The SALES application PDB inherits the restrictions of the application root lockdown profile in addition to those set in its nearest ancestor, CDB_prof1.

5. Observe profile lockdown inheritance when the application root lockdown profile is set in the application PDB.
   a. Set the sales_prof lockdown profile in the SALES application PDB.

```
SQL> CONNECT hr_lock_mgr@sales
Enter password: password
Connected.
SQL> SHOW PARAMETER pdb_lockdown

NAME                                 TYPE         VALUE
------------------------------------ ------------ ------------------------------
pdb_lockdown                         string       APP_ROOT_PROF
SQL> ALTER SYSTEM SET pdb_lockdown = '' SCOPE = both;
ALTER SYSTEM SET pdb_lockdown = '' SCOPE = both
*
ERROR at line 1:
ORA-01031: insufficient privileges


SQL>
```

   b. Temporarily unset app_root_prof so as to be able to set the sales_prof lockdown profile in the SALES application PDB. Then reset app_root_prof.

```
SQL> CONNECT hr_lock_mgr@hr_root
Enter password: password
Connected.
SQL> ALTER SYSTEM SET pdb_lockdown = '' SCOPE = both;
```

```
System altered.

SQL>
```

c. Set the `sales_prof` lockdown profile in the `SALES` application PDB.

```
SQL> CONNECT hr_lock_mgr@sales
Enter password: password
Connected.
SQL> ALTER SYSTEM SET pdb_lockdown = sales_prof SCOPE = both;

System altered.

SQL> SHOW PARAMETER pdb_lockdown

NAME                                 TYPE         VALUE
------------------------------------ ------------ ------------------------------
pdb_lockdown                         string       SALES_PROF
SQL>
```

d. Reset `app_root_prof` in the application root.

```
SQL> CONNECT hr_lock_mgr@hr_root
Enter password: password
Connected.
SQL> ALTER SYSTEM SET pdb_lockdown = app_root_prof
                  SCOPE = both;
 2
System altered.

SQL>
```

e. Observe how the `SALES` application PDB still inherits the restrictions of the application root lockdown profile and the CDB root lockdown profile.

```
SQL> CONNECT sys@sales AS SYSDBA
Enter password: password
Connected.
SQL> ALTER PLUGGABLE DATABASE sales CLOSE;
ALTER PLUGGABLE DATABASE sales CLOSE
*
ERROR at line 1:
ORA-01031: insufficient privileges


SQL> ALTER SYSTEM SET ddl_lock_timeout=30 scope=both;
```

```
System altered.

SQL> ALTER SYSTEM flush shared_pool;
ALTER SYSTEM flush shared_pool
*
ERROR at line 1:
ORA-01031: insufficient privileges


SQL> ALTER SYSTEM CHECKPOINT;
ALTER SYSTEM CHECKPOINT
*
ERROR at line 1:
ORA-01031: insufficient privileges


SQL>
```

Because a lockdown profile is set in the application PDB, the application PDB inherits the restrictions of the application PDB lockdown profile and those of the CDB root lockdown profile and not from the application root lockdown profile.

6. Drop lockdown profiles.
   a. Drop the `CDB_prof1` and `CDB_prof2` lockdown profiles in the CDB root.

```
SQL> CONNECT / AS SYSDBA
Connected.
SQL> ALTER SYSTEM SET pdb_lockdown = '' SCOPE = both;

System altered.

SQL> DROP LOCKDOWN PROFILE CDB_prof1;

Lockdown Profile dropped.

SQL> DROP LOCKDOWN PROFILE CDB_prof2;

Lockdown Profile dropped.

SQL>
```

b. Drop the `sales_prof` lockdown profile in the `SALES` application PDB.

```
SQL> CONNECT sys@sales AS SYSDBA
Enter password: password
Connected.
SQL> ALTER SYSTEM SET pdb_lockdown = '' SCOPE = both;

System altered.

SQL> CONNECT hr_lock_mgr@hr_root
Enter password: password
Connected.
SQL> DROP LOCKDOWN PROFILE sales_prof;

Lockdown Profile dropped.

SQL>
```

c. Update the existing `app_root_prof` to remove `ALTER SYSTEM SET` restriction.

```
SQL> ALTER LOCKDOWN PROFILE app_root_prof
           ENABLE STATEMENT = ('alter system')
           CLAUSE = ('set');
  2    3
Lockdown Profile altered.

SQL>
```

7. In the second part of this practice, you will create static and dynamic lockdown profiles. You create and set, respectively, a static lockdown profile in the `SALES` application PDB from a basic application root lockdown profile, and a dynamic lockdown profile in the `OPERATIONS` application PDB from the same basic application root lockdown profile.

a. Log in to the application root to create the static application root lockdown profile `static_sales_prof` from the `app_root_prof` lockdown profile for the `SALES` application PDB.

```
SQL> CREATE LOCKDOWN PROFILE static_sales_prof
                             FROM app_root_prof;
 2
Lockdown Profile created.

SQL> ALTER LOCKDOWN PROFILE static_sales_prof
           DISABLE STATEMENT = ('alter pluggable database');
  2
Lockdown Profile altered.
```

```
SQL> SELECT profile_name, rule, clause, status
     FROM   cdb_lockdown_profiles;
 2
PROFILE_NAME        RULE                          CLAUSE
------------------------------------ ------------------------------------------------------ ----------------------------------------
STATUS
-------------------
APP_ROOT_PROF       ALTER SYSTEM
DISABLE

APP_ROOT_PROF       ALTER SYSTEM              FLUSH SHARED_POOL
ENABLE

APP_ROOT_PROF       ALTER SYSTEM              SET
ENABLE

STATIC_SALES_PROF ALTER PLUGGABLE DATABASE
DISABLE

STATIC_SALES_PROF ALTER SYSTEM
DISABLE

STATIC_SALES_PROF  ALTER SYSTEM               FLUSH SHARED_POOL
ENABLE

STATIC_SALES_PROF  ALTER SYSTEM               SET
ENABLE


7 rows selected.

SQL>
```

Q/ What do you observe about `static_sales_prof` in the `CDB_LOCKDOWN_PROFILES` view?

A/ The rules from the basic `app_root_prof` lockdown profile are all copied into the `static_sales_prof` lockdown profile.

b. Log in to the `SALES` application PDB to set the static `static_sales_prof` lockdown profile.

```
SQL> CONNECT hr_lock_mgr@sales
Enter password: password
```

```
Connected.
SQL> ALTER SYSTEM SET pdb_lockdown = '' SCOPE = both;

System altered.

SQL> ALTER SYSTEM SET pdb_lockdown = static_sales_prof
                      SCOPE = both;
 2
System altered.

SQL> SHOW PARAMETER pdb_lockdown

NAME                                 TYPE         VALUE
------------------------------------ ------------ ------------------------------
pdb_lockdown                         string       STATIC_SALES_PROF
SQL>
```

c. Before creating the dynamic lockdown profile in the `OPERATIONS` application PDB, log in to the application PDB to verify that you can still create a partitioned table because the restriction rule of the dynamic lockdown profile will be the partitioning feature.

```
SQL> CONNECT hr_lock_mgr@operations
Enter  password:  password
Connected.
SQL> SHOW PARAMETER pdb_lockdown

NAME                                 TYPE         VALUE
------------------------------------ ------------ ------------------------------
pdb_lockdown                         string       APP_ROOT_PROF
SQL>
SQL> CREATE TABLE sales ( SALESMAN_ID  NUMBER(5),
        SALESMAN_NAME VARCHAR2(30), SALES_STATE VARCHAR2(20))
       PARTITION BY LIST (SALES_STATE) AUTOMATIC
        (PARTITION P_CAL VALUES ('CALIFORNIA'));
  2    3    4
Table created.

SQL> DROP TABLE sales;

Table dropped.

SQL>
```

d. Log in to the application root to create the dynamic application root lockdown profile `dynamic_op_prof` from `app_root_prof` for the `OPERATIONS` application PDB.

```
SQL> CONNECT hr_lock_mgr@hr_root
Enter password: password
Connected.
SQL> CREATE LOCKDOWN PROFILE dynamic_op_prof
             INCLUDING app_root_prof;
 2
Lockdown Profile created.

SQL> ALTER LOCKDOWN PROFILE dynamic_op_prof
           DISABLE OPTION = ('PARTITIONING');
   2
Lockdown Profile altered.

SQL> SELECT profile_name, rule, clause, status
     FROM cdb_lockdown_profiles;
 2
PROFILE_NAME        RULE                          CLAUSE
------------------------------------ ------------------------------------------------------ ----------------------------------------
STATUS
----------
APP_ROOT_PROF       ALTER SYSTEM
DISABLE

APP_ROOT_PROF       ALTER SYSTEM                  FLUSH SHARED_POOL
ENABLE

APP_ROOT_PROF       ALTER SYSTEM                  SET
ENABLE

DYNAMIC_OP_PROF     PARTITIONING
DISABLE

STATIC_SALES_PROF ALTER PLUGGABLE DATABASE
DISABLE

STATIC_SALES_PROF ALTER SYSTEM
DISABLE

STATIC_SALES_PROF  ALTER SYSTEM                   FLUSH SHARED_POOL
ENABLE
```

```
STATIC_SALES_PROF  ALTER SYSTEM              SET
ENABLE


8 rows selected.

SQL>
```

Q/ What do you observe about `dynamic_op_prof` in the `CDB_LOCKDOWN_PROFILES` view?
A/ The rules from the basic `app_root_prof` lockdown profile are not copied into the `dynamic_op_prof` lockdown profile. You will check whether they are in effect.

e. Log in to the `OPERATIONS` application PDB to set the dynamic `dynamic_op_prof` lockdown profile.

```
SQL> CONNECT hr_lock_mgr@operations
Enter  password:  password
Connected.
SQL> ALTER SYSTEM SET pdb_lockdown = '' SCOPE = both;

System altered.

SQL> ALTER SYSTEM SET pdb_lockdown = dynamic_op_prof
                      SCOPE = both;
  2
System altered.

SQL> SHOW PARAMETER pdb_lockdown

NAME                                 TYPE         VALUE
------------------------------------ ------------ ------------------------------
pdb_lockdown                         string       DYNAMIC_OP_PROF
SQL>
```

8. Test the behavior of static and dynamic lockdown profiles set in the `SALES` and `OPERATIONS` application PDBs, respectively.

```
SQL> CONNECT sys@sales AS SYSDBA
Enter password: password
Connected.
SQL> ALTER SYSTEM SET ddl_lock_timeout=30 scope=both;

System altered.
```

```
SQL> ALTER SYSTEM flush shared_pool;

System altered.

SQL> ALTER SYSTEM CHECKPOINT;
ALTER SYSTEM CHECKPOINT
*
ERROR at line 1:
ORA-01031: insufficient privileges


SQL> ALTER PLUGGABLE DATABASE CLOSE;
ALTER PLUGGABLE DATABASE CLOSE
*
ERROR at line 1:
ORA-01031: insufficient privileges

SQL> CREATE TABLE sales
       ( SALESMAN_ID  NUMBER(5), SALESMAN_NAME VARCHAR2(30),
         SALES_STATE VARCHAR2(20))
        PARTITION BY LIST (SALES_STATE) AUTOMATIC
         (PARTITION P_CAL VALUES ('CALIFORNIA'));
   2    3    4    5
Table created.

SQL>
```

The first, second, and third commands respect rules inherited from `app_root_prof`.
The fourth command respects rules inherited from `static_sales_prof`.
There is no restriction in `static_sales_prof` for the fifth command to execute.

9. Test in the `OPERATIONS` application PDB if the restriction rules of the dynamic `dynamic_op_prof` lockdown profile using the values from the existing basic `app_root_prof` lockdown profile apply.

```
SQL> CONNECT sys@operations AS SYSDBA
Enter password: password
Connected.
SQL> ALTER SYSTEM SET ddl_lock_timeout=30 scope=both;

System altered.
```

```
SQL> ALTER SYSTEM flush shared_pool;

System altered.

SQL> ALTER SYSTEM CHECKPOINT;
ALTER SYSTEM CHECKPOINT
*
ERROR at line 1:
ORA-01031: insufficient privileges


SQL> ALTER PLUGGABLE DATABASE CLOSE;

Pluggable database altered.

SQL> ALTER PLUGGABLE DATABASE OPEN;

Pluggable database altered.

SQL> CREATE TABLE sales
       ( SALESMAN_ID NUMBER(5), SALESMAN_NAME VARCHAR2(30),
         SALES_STATE VARCHAR2(20))
       PARTITION BY LIST (SALES_STATE) AUTOMATIC
       (PARTITION P_CAL VALUES ('CALIFORNIA'));
  2    3    4    5
CREATE TABLE sales
*
ERROR at line 1:
ORA-00439: feature not enabled: Partitioning


SQL>
```

The first, second, and third commands respect rules inherited from `app_root_prof`.
There is no restriction in `dynamic_op_prof` for the fourth command to execute.
The fifth command respects rules inherited from `dynamic_op_prof`.

10. Test how any subsequent changes to the existing basic `app_root_prof` lockdown profile impact the derived `static_sales_prof` and `dynamic_op_prof` lockdown profiles. Add a rule to `app_root_prof`.

```
SQL> CONNECT hr_lock_mgr@hr_root
Enter password: password
```

```
Connected.
SQL> ALTER LOCKDOWN PROFILE App_root_prof
         ENABLE STATEMENT = ('alter system')
         CLAUSE = ('checkpoint');
  2    3
Lockdown Profile altered.

SQL> SELECT profile_name, rule, clause, status
     FROM   cdb_lockdown_profiles;
  2
PROFILE_NAME          RULE                          CLAUSE
--------------------------------------------- ---------------------------------------------------------- ----------------------------------------
--
STATUS
-------------------
APP_ROOT_PROF         ALTER SYSTEM
DISABLE

APP_ROOT_PROF         ALTER SYSTEM              FLUSH SHARED_POOL
ENABLE

APP_ROOT_PROF         ALTER SYSTEM              SET
ENABLE

APP_ROOT_PROF         ALTER SYSTEM              CHECKPOINT
ENABLE

DYNAMIC_OP_PROF       PARTITIONING
DISABLE

STATIC_SALES_PROF     ALTER PLUGGABLE DATABASE
DISABLE

STATIC_SALES_PROF     ALTER SYSTEM
DISABLE

STATIC_SALES_PROF     ALTER SYSTEM              FLUSH SHARED_POOL
ENABLE

STATIC_SALES_PROF     ALTER SYSTEM              SET
ENABLE
```

```
9 rows selected.

SQL>
```

11. Check whether the rule added to the existing basic `app_root_prof` lockdown profile does not impact the derived `static_sales_prof`.

```
SQL> CONNECT sys@sales AS SYSDBA
Enter password: password
Connected.
SQL> ALTER SYSTEM CHECKPOINT;
ALTER SYSTEM CHECKPOINT
*
ERROR at line 1:
ORA-01031: insufficient privileges


SQL>
```

12. Check whether the rule added to the existing basic `app_root_prof` lockdown profile impacts the derived `dynamic_op_prof`.

```
SQL> CONNECT sys@operations AS SYSDBA
Enter password: password
Connected.
SQL> ALTER SYSTEM CHECKPOINT;

System altered.

SQL> CREATE TABLE sales
       ( SALESMAN_ID NUMBER(5), SALESMAN_NAME VARCHAR2(30),
         SALES_STATE VARCHAR2(20))
       PARTITION BY LIST (SALES_STATE) AUTOMATIC
        (PARTITION P_CAL VALUES ('CALIFORNIA'));
  2    3    4    5
CREATE TABLE sales
*
ERROR at line 1:
ORA-00439: feature not enabled: Partitioning


SQL> EXIT
$
```

13. Execute the `$HOME/labs/MULTI/cleanup_profiles.sh` shell script. It drops the `HR_ROOT` application root PDB and its associated application PDBs—`SALES` and `OPERATIONS`—and therefore, all lockdown profiles created in these PDBs.

```
$ $HOME/labs/MULTI/cleanup_profiles.sh
…
$
```

Practice 2-5: Switching Over Refreshable Cloned PDBs

Overview

In this practice, you reverse the roles of a PDB and its refreshable cloned PDB. The refreshable cloned PDB can be made the primary PDB while the primary PDB becomes the refreshable cloned PDB.

Tasks

1. Before starting the practice, execute the `$HOME/labs/admin/glogin_2b.sh` shell script. It sets formatting for all columns selected in queries.

```
$ $HOME/labs/admin/glogin_2b.sh
…
$
```

2. Configure the primary CDB and PDB. The session in `ORCL` is maintained and called *`Primary` Session*.
   a. Execute the `$HOME/labs/MULTI/create_PDB1.sql` SQL script. It re-creates `PDB1` using Oracle Managed Files (OMF) and drops `PDB_REFRESHED` if it exists. Then grant the `SYSOPER` privilege required to create refreshable cloned PDBs. Then create the database link to connect to the future refreshable PDB in `ORCL`.

```
$ sqlplus / AS SYSDBA

SQL> @$HOME/labs/MULTI/create_PDB1.sql
…
SQL>
```

   b. Create a common user in `ORCL` and grant the common user the `SYSOPER` system privilege and others.

```
SQL> DROP USER c##u1 CASCADE;
DROP USER c##u1 CASCADE
          *
ERROR at line 1:
ORA-01918: user 'C##U1' does not exist

SQL> CREATE USER c##u1 IDENTIFIED BY password;

User created.

SQL> GRANT create session, resource, create any table,
      unlimited tablespace, sysoper, create pluggable database
      TO c##u1 CONTAINER = ALL;
 2    3
Grant succeeded.

SQL>
```

c. Create in ORCL a public fixed-user database link relying on the common user created in the previous step.

```
SQL> DROP PUBLIC DATABASE LINK link_ORCL;
DROP PUBLIC DATABASE LINK link_ORCL
                          *
ERROR at line 1:
ORA-02024: database link not found


SQL> CREATE PUBLIC DATABASE LINK link_ORCL
            CONNECT TO c##u1 IDENTIFIED BY password
            USING 'ORCL';
  2    3
Database link created.

SQL>
```

d. In PDB1, execute the $HOME/labs/MULTI/create_userswitch_tab.sql SQL script that creates the USERSWITCH.BIGTAB table with 10000 rows.

```
SQL> CONNECT system@PDB1
Enter password: password
Connected.
SQL> SET sqlprompt "SQL_prim> "
SQL_prim> @$HOME/labs/MULTI/create_userswitch_tab.sql
…
SQL_prim>
```

e. Count the rows and display the data in the rows.

```
SQL_prim> SELECT count(*) FROM userswitch.bigtab;

  COUNT(*)
----------
     10000

SQL_prim> SELECT DISTINCT label FROM userswitch.bigtab;

LABEL
--------------------------------------------------------------------------------
DATA FROM PDB1

SQL_prim>
```

3. Open another terminal window called *Switch Session* where you log in to ORCL and create the refreshable cloned PDB PDB_REFRESHED. Create PDB_REFRESHED automatically refreshed from PDB1 every two minutes.

```
$ mkdir -p /u02/app/oracle/oradata/ORCL/pdb_refreshed
$ sqlplus / AS SYSDBA

SQL> SET sqlprompt "SQL_switch> "
SQL_switch> CREATE PLUGGABLE DATABASE pdb_refreshed
               FROM pdb1@link_ORCL
             CREATE_FILE_DEST =
               '/u02/app/oracle/oradata/ORCL/pdb_refreshed'
               REFRESH MODE every 2 minutes;
  2    3    4    5    6
Pluggable database created.

SQL_switch>
```

   a. Check the status of the refreshable PDB.

```
SQL_switch> SELECT pdb_name, status FROM cdb_pdbs;

PDB_NAME        STATUS
--------------- ----------
PDB1            NORMAL
PDB$SEED        NORMAL
PDB_REFRESHED   REFRESHING

SQL_switch>
```

4. In *Primary Session*, start and commit a transaction in the production PDB1.

```
SQL_prim> INSERT INTO userswitch.bigtab
                    VALUES ('New DATA FROM PDB1');
 2
1 row created.

SQL_prim> COMMIT;

Commit complete.

SQL_prim> SELECT DISTINCT label FROM userswitch.bigtab;

LABEL
--------------------------------------------------------------------------------
DATA  FROM  PDB1
New DATA FROM PDB1

SQL_prim>
```

5. In *Switch Session*, wait two minutes before you can observe that the data in PDB_REFRESHED is refreshed.

```
SQL_switch> ALTER SESSION SET container = pdb_refreshed;

Session altered.

SQL_switch> ALTER PLUGGABLE DATABASE OPEN READ ONLY;

Pluggable database altered.

SQL_switch> SELECT DISTINCT label FROM userswitch.bigtab;

LABEL
--------------------------------------------------------------------------------
DATA  FROM  PDB1
New DATA FROM PDB1

SQL_switch> SELECT count(*) FROM userswitch.bigtab;

  COUNT(*)
----------
     10001

SQL_switch>
```

6. Switch over the refreshable cloned PDB.
   a. In *Primary Session*, switch the primary PDB1 PDB over the refreshable clone PDB_REFRESHED PDB.

```
SQL_prim> CONNECT sys@ORCL AS SYSDBA
Enter password: password
Connected.
SQL_prim> ALTER SESSION SET CONTAINER = pdb1;

Session altered.

SQL_prim> ALTER PLUGGABLE DATABASE
             REFRESH MODE EVERY 2 MINUTES
             FROM pdb_refreshed@link_ORCL SWITCHOVER;
  2   3
Pluggable database altered.

SQL_prim>
```

b. Verify the status of the refreshable PDB1.

```
SQL_prim> ALTER SESSION SET CONTAINER = CDB$ROOT;

Session altered.

SQL_prim> SELECT pdb_name, status FROM cdb_pdbs;

PDB_NAME        STATUS
--------------- ------------
PDB$SEED        NORMAL
PDB_REFRESHED   NORMAL
PDB1            REFRESHING

SQL_prim>
```

c. In *Switch Session*, start and commit a transaction in the primary PDB_REFRESHED.

```
SQL_switch> ALTER SESSION SET CONTAINER = pdb_refreshed;

Session altered.

SQL_switch> INSERT INTO userswitch.bigtab
                  VALUES ('New DATA FROM pdb_refreshed');
  2
1 row created.

SQL_switch> COMMIT;

Commit complete.

SQL_switch> SELECT DISTINCT label FROM userswitch.bigtab;

LABEL
----------------------------------------
New DATA FROM pdb_refreshed
DATA FROM PDB1
New DATA FROM PDB1

SQL_switch>
```

d. Back in *Primary Session*, wait two minutes before you can observe that the data in PDB1 is refreshed.

```
SQL_prim> ALTER SESSION SET CONTAINER = pdb1;

Session altered.

SQL_prim> ALTER PLUGGABLE DATABASE OPEN READ ONLY;
```

```
Pluggable database altered.

SQL_prim> SELECT DISTINCT label FROM userswitch.bigtab;

LABEL
----------------------------------------
New DATA FROM pdb_refreshed
DATA FROM PDB1
New DATA FROM PDB1

SQL_prim> SELECT count(*) FROM userswitch.bigtab;

  COUNT(*)
----------
     10002

SQL_prim>
```

e. Do not forget to close the refreshable PDB to be refreshed.

```
SQL_prim> ALTER PLUGGABLE DATABASE CLOSE;

Pluggable database altered.

SQL_prim>
```

7. Re-switch. You now observe that you can switch over back and forth.

a. Before you switch the primary `PDB_REFRESHED` PDB over the refreshable cloned `PDB1`, update data in the `USERSWITCH.BIGTAB` table in the primary PDB. Proceed in the *`Switch` Session*.

```
SQL_switch> UPDATE userswitch.bigtab
                  SET label = 'DATA updated IN pdb_refreshed';
  2
10002 rows updated.

SQL_switch> COMMIT;

Commit complete.

SQL_switch> SELECT DISTINCT label FROM userswitch.bigtab;

LABEL
----------------------------------------
DATA updated IN pdb_refreshed

SQL_switch>
```

b. In the *Primary Session*, wait two minutes before you can observe that the data in `PDB1` is refreshed.

```
SQL_prim> ALTER PLUGGABLE DATABASE OPEN READ ONLY;

Pluggable database altered.

SQL_prim> SELECT DISTINCT label FROM userswitch.bigtab;

LABEL
----------------------------------------
DATA updated IN pdb_refreshed

SQL_prim> EXIT
$
```

c. From the *Switch Session*, switch the primary `PDB_REFRESHED` PDB over the refreshable cloned `PDB1` PDB.

```
SQL_switch> ALTER SESSION SET CONTAINER = pdb_refreshed;

Session altered.

SQL_switch> ALTER PLUGGABLE DATABASE
                REFRESH MODE EVERY 2 MINUTES
                FROM pdb1@link_ORCL SWITCHOVER;
 2   3
Pluggable database altered.

SQL_switch>
```

d. Verify the status of the refreshable `PDB_REFRESHED`.

```
SQL_switch> ALTER SESSION SET CONTAINER = CDB$ROOT;

Session altered.

SQL_switch> SELECT pdb_name, status FROM cdb_pdbs;

PDB_NAME        STATUS
--------------- ----------
PDB$SEED        NORMAL
PDB_REFRESHED REFRESHING
PDB1            NORMAL

SQL_switch> EXIT
$
```

8. Execute the `$HOME/labs/MULTI/cleanup_PDB_REFRESHED.sh` shell script to drop `PDB_REFRESHED` in `ORCL`.

```
$ $HOME/labs/MULTI/cleanup_PDB_REFRESHED.sh
…
$
```

Practices for Lesson 3:
Managing Security

Practices for Lesson 3: Overview

Overview

In these practices, you will discover how to create schema-only accounts, manage PDB keystores, create user-defined TDE master keys, export and import fixed-user database links without the database links password, and finally encrypt sensitive data in Database Replay files.

Practice 3-1: Creating Schema-Only Accounts

Overview

In this practice, you will create a user with no authentication. Application designers may want to create a user account that contains the application data dictionary, but should not be allowed to log in. This can be used to enforce data access through the application, separation of duties at the application level, and other security mechanisms.

Tasks

1. Before starting the practice, execute the `$HOME/labs/admin/glogin_3.sh` shell script. It sets formatting for all columns selected in queries.

```
$ $HOME/labs/admin/glogin_3.sh
…
$
```

2. Log in to `PDB1 as SYSTEM` to create the `SCHEMA_NOAUTH` schema-only account. If `PDB1` does not exist, execute the `$HOME/labs/admin/PDB1.sh` shell script.

```
$ sqlplus system@PDB1

Enter password: password

SQL> CREATE USER schema_noauth NO AUTHENTICATION;

User created.

SQL>
```

3. Grant `CREATE TABLE` and `UNLIMITED TABLESPACE` to `SCHEMA_NOAUTH`.

```
SQL> GRANT create table, unlimited tablespace
           TO schema_noauth;
 2
Grant succeeded.

SQL>
```

4. Verify that a schema-only account with sufficient privileges can create objects in the schema but cannot connect under the schema-only account directly.

```
SQL> CREATE TABLE schema_noauth.tab1 (c NUMBER);

Table created.

SQL>
```

```
SQL> CONNECT schema_noauth@PDB1
Enter password:
```

```
ERROR:
ORA-01005: null password given; logon denied

Warning: You are no longer connected to ORACLE.
SQL>
```

The schema-only account does not have any authentication, neither basic by password, nor strong by global authentication and, therefore, cannot connect.

5. List the schema-only accounts.

```
SQL> CONNECT system@PDB1
Enter password: password
Connected.
SQL> SELECT username, authentication_type FROM dba_users
     ORDER BY 2;
  2
USERNAME                       AUTHENTI
------------------------------ --------
SCHEMA_NOAUTH                  NONE
OJVMSYS                        NONE
DVSYS                          NONE
LBACSYS                        NONE
DVF                            NONE
AUDSYS                         NONE
REMOTE_SCHEDULER_AGENT         PASSWORD
GSMUSER                        PASSWORD
…
ORDSYS                         PASSWORD
SYS                            PASSWORD
SYSTEM                         PASSWORD

39 rows selected.

SQL>
```

You can observe that some Oracle-supplied schema accounts are also schema-only accounts like Database Vault `DVSYS` and `DVF`, Oracle Label Security `LBACSYS` and Unified Auditing `AUDSYS`.

Q/ Can you change the authentication type of `SCHEMA_NOAUTH`?

```
SQL> ALTER USER schema_noauth IDENTIFIED BY password;

User altered.

SQL>
```

A/ Yes.

6. Verify that the user can now log in to the database.

```
SQL> CONNECT schema_noauth@PDB1
Enter password: password
ERROR:
ORA-01045: user SCHEMA_NOAUTH lacks CREATE SESSION privilege;
logon denied


Warning: You are no longer connected to ORACLE.
SQL>
```

7. Grant the user the `CREATE SESSION` privilege.

```
SQL> CONNECT system@PDB1
Enter password: password
Connected.
SQL> GRANT create session TO schema_noauth;

Grant succeeded.

SQL> CONNECT schema_noauth@PDB1
Enter password: password
Connected.
SQL>
```

8. Drop the `SCHEMA_NOAUTH` schema-only account.

```
SQL> CONNECT system@PDB1
Enter password: password
Connected.
SQL> DROP USER schema_noauth CASCADE;

User dropped.

SQL> EXIT
$
```

Practice 3-2: Managing PDB Keystores

Overview

In this practice, you will learn about the keystore isolated and united modes, and the reason for introducing these modes, and set up isolated mode PDB keystores.

Tasks

1. Execute the `$HOME/labs/admin/PDB1.sh` shell script to recreate `PDB1`.

```
$ $HOME/labs/admin/PDB1.sh
…
$
```

2. You plan to isolate PDB keystores. To use isolated mode, the `WALLET_ROOT` initialization parameter must be set. Configure the root directory location of the CDB keystore and the keystore type.
   a. Configure the wallet root location and restart the database instance.

```
$ sqlplus / AS SYSDBA

SQL> SHOW PARAMETER wallet

NAME                                 TYPE      VALUE
------------------------------------ --------- ------------------------------
wallet_root                          string

SQL> ALTER SYSTEM SET
       wallet_root = '/u01/app/oracle/admin/ORCL/tde_keystore'
       SCOPE=SPFILE;
  2    3
System altered.

SQL>
```

```
SQL> SHUTDOWN IMMEDIATE
Database closed.
Database dismounted.
ORACLE instance shut down.
SQL> STARTUP
ORACLE instance started.

Total System Global Area 1426060816 bytes
Fixed Size                  8893968 bytes
Variable Size             570425344 bytes
```

```
Database Buffers          838860800 bytes
Redo Buffers                7880704 bytes
Database mounted.
Database opened.
SQL> ALTER PLUGGABLE DATABASE all OPEN;

Pluggable database altered.

SQL> SHOW PARAMETER wallet_root

NAME            TYPE    VALUE
--------------- ------- ------------------------------------------------------------------------------
wallet_root     string  /u01/app/oracle/admin/ORCL/tde_keystore
SQL>
```

b. Configure the default keystore type for any future isolated PDB. In this case, the isolated keystores will be stored as OS files.

```
SQL> ALTER SYSTEM SET tde_configuration =
                      'KEYSTORE_CONFIGURATION=FILE'
                       SCOPE=BOTH;
  2   3
System altered.

SQL>
```

`KEYSTORE_CONFIGURATION` can take the values `FILE`, `OKV`, `HSM`, `FILE|OKV`, `FILE|HSM`, `OKV|FILE`, and `HSM|FILE` to configure the type of isolated keystore for each PDB.

```
SQL> SELECT wrl_parameter, status, keystore_mode
     FROM   v$encryption_wallet;
  2
WRL_PARAMETER                                      STATUS
KEYSTORE
-------------------------------------------------- ------------------------------ ----------
----------
/u01/app/oracle/admin/ORCL/tde_keystore/tde/ NOT_AVAILABLE  NONE
                                                   NOT_AVAILABLE
UNITED
                                                   NOT_AVAILABLE
UNITED

SQL>
```

c. Create the CDB root keystore with its own password.

```
SQL> ADMINISTER KEY MANAGEMENT CREATE KEYSTORE IDENTIFIED BY
password_CDB;

keystore altered.

SQL>
```

Observe that you do not have to provide the path of the keystore. It is already defined by using the `WALLET_ROOT` initialization parameter replacing the `ENCRYPTION_WALLET_LOCATION` parameter in the `$ORACLE_HOME/network/admin/sqlnet.ora` file. After you create the keystore, it is closed by default.

```
SQL> SELECT wrl_parameter, status, keystore_mode
     FROM   v$encryption_wallet;
  2
WRL_PARAMETER                                    STATUS
KEYSTORE
-------------------------------------------------- ------------------------------ ----------
----------
/u01/app/oracle/admin/ORCL/tde_keystore/tde/ CLOSED          NONE
                                                 CLOSED
UNITED
                                                 CLOSED
UNITED

SQL> !ls -l /u01/app/oracle/admin/ORCL/tde_keystore/tde/*
-rw------- 1 oracle oinstall 2555 Nov 24 02:50
/u01/app/oracle/admin/ORCL/tde_keystore/tde/ewallet.p12

SQL>
```

d. Open the CDB root keystore.

```
SQL> ADMINISTER KEY MANAGEMENT
        SET KEYSTORE OPEN IDENTIFIED BY password_CDB
        CONTAINER=ALL;
 2    3
keystore altered.

SQL> SELECT con_id, wrl_parameter, status, keystore_mode
     FROM v$encryption_wallet;
 2
    CON_ID WRL_PARAMETER
```

```
---------- ------------------------------------------------------------------------------------------------
STATUS              KEYSTORE
---------- ------------------------------------------------------------------------------------------------
         1 /u01/app/oracle/admin/ORCL/tde_keystore/tde/
OPEN_NO_MASTER_KEY NONE

         2
CLOSED              UNITED

         3
OPEN_NO_MASTER_KEY UNITED

SQL>
```

The CDB root keystore is opened without a master key set.

e. Set the master key in the CDB root keystore.

```
SQL> ADMINISTER KEY MANAGEMENT
       SET KEY IDENTIFIED BY password_CDB WITH BACKUP
       CONTAINER=ALL;
  2    3
keystore altered.

SQL>
```

An ORA-46665 error message may appear because some PDBs did not open the keystore in their container. Therefore, the TDE master encryption key for those PDBs cannot be activated until they are opened. Nevertheless, the CDB root keystore is opened with a TDE master encryption key set.

```
SQL> SELECT con_id, wrl_parameter, status, wallet_type,
            keystore_mode
     FROM   v$encryption_wallet;
 2   3
    CON_ID WRL_PARAMETER
---------- ------------------------------------------------------------------------------------------------
STATUS              WALLET_TYPE          KEYSTORE
---------- ---------- ---------- ------------------------------------------ ------------------
         1 /u01/app/oracle/admin/ORCL/tde_keystore/tde/
OPEN                PASSWORD             NONE

         2
CLOSED              UNKNOWN              UNITED
```

```
         3
OPEN                PASSWORD              UNITED


SQL>
```

The new `KEYSTORE_MODE` column displays `NONE for the CDB root`. This value is seen when `V$ENCRYPTION_WALLET` is queried from the CDB root, or when the system is a non-CDB. In the latter case, the keystore mode does not apply.

The new `KEYSTORE_MODE` column displays `UNITED for the PDBs`. This is the default mode.

3. For further tasks in the practice, set the key in `PDB1`.

```
SQL> CONNECT sys@PDB1 AS SYSDBA
Enter password: password
Connected.
SQL> ADMINISTER KEY MANAGEMENT
       SET KEY IDENTIFIED BY password_CDB WITH BACKUP;
  2
keystore altered.

SQL>
```

4. Create a PDB to run in isolated mode.
   a. Create the new `PDB_ISOLATED` PDB and open it. By default, the `PDB_ISOLATED` PDB uses the default mode of the CDB, which is the united mode.

```
SQL> CONNECT / AS SYSDBA
Connected.
SQL> !mkdir /u02/app/oracle/oradata/ORCL/PDB_ISOLATED

SQL> CREATE PLUGGABLE DATABASE pdb_isolated
            ADMIN USER pdb2_admin IDENTIFIED BY password
            ROLES=(CONNECT)
CREATE_FILE_DEST='/u02/app/oracle/oradata/ORCL/PDB_ISOLATED';
  2    3
Pluggable database created.

SQL> ALTER PLUGGABLE DATABASE pdb_isolated OPEN;

Pluggable database altered.

SQL>
```

b. Isolate the PDB keystore for the newly created PDB.

```
SQL> CONNECT sys@pdb_isolated AS SYSDBA
Enter password: password
Connected.
SQL> ALTER SYSTEM SET tde_configuration =
           'KEYSTORE_CONFIGURATION=FILE' SCOPE=BOTH;
  2
System altered.

SQL> SELECT wrl_parameter, status, wallet_type, keystore_mode
     FROM v$encryption_wallet;
  2
WRL_PARAMETER                                          STATUS
------------------------------------------------------------------------------------------------------ ----------
WALLET_TYPE          KEYSTORE
-------------------- --------
/u01/app/oracle/admin/ORCL/tde_keystore/5EB2 NOT_AVAILABLE
7B9A049874D8E0532133960A3FA3/tde/
UNKNOWN              ISOLATED


SQL>
```

The new KEYSTORE_MODE column displays ISOLATED. This value is seen when V$ENCRYPTION_WALLET is queried from the PDB. The PDB is configured to use its own keystore and to open the keystore using its own isolated keystore password.

c. Create the security officer in the PDB responsible for PDB keystore management.

```
SQL> CREATE USER sec_tde IDENTIFIED BY password;

User created.

SQL> GRANT create session, administer key management,
           select any dictionary, create any table TO sec_tde;
 2
Grant succeeded.

SQL>
```

d. Now create the PDB keystore with its own password, open the PDB keystore, and set the PDB TDE master encryption key.

e. Create the PDB keystore with its own password.

```
SQL> CONNECT sec_tde@pdb_isolated
Enter password: password
Connected.
SQL> ADMINISTER KEY MANAGEMENT
          CREATE KEYSTORE IDENTIFIED BY password_PDB;
 2
keystore altered.

SQL>
```

f. Open the PDB keystore.

```
SQL> ADMINISTER KEY MANAGEMENT
          SET KEYSTORE OPEN IDENTIFIED BY password_PDB;
 2
keystore altered.

SQL>
```

g. Set the PDB TDE master encryption key.

```
SQL> ADMINISTER KEY MANAGEMENT
          SET KEY IDENTIFIED BY password_PDB WITH BACKUP;
 2
keystore altered.

SQL>
```

h. Verify that the PDB holds its own key.

```
SQL> SELECT wrl_parameter, status, wallet_type, keystore_mode
     FROM   v$encryption_wallet;
  2
WRL_PARAMETER                                    STATUS
------------------------------------------------------------------------------------------ ----------
WALLET_TYPE          KEYSTORE
-------------------- --------
/u01/app/oracle/admin/ORCL/tde_keystore/5EB2 OPEN
7B9A049874D8E0532133960A3FA3/tde/
PASSWORD             ISOLATED

SQL>
```

i. Check the existence of the `ewallet.p12` file in the subdirectory created under the `WALLET_ROOT` directory.

```
SQL> !ls -l
/u01/app/oracle/admin/ORCL/tde_keystore/5EB27B9A049874D8E0532133
960A3FA3/tde/
total 4
-rw------- 1 oracle oinstall 2555 Nov 24 04:10 ewallet.p12
-rw------- 1 oracle oinstall 2555 Jan 5 15:57
ewallet_2018010515572043.p12

SQL>
```

j. Create an encrypted table.

```
SQL> CREATE TABLE system.test (c NUMBER ENCRYPT);

Table created.

SQL>
```

5. Convert a united mode PDB to run in isolated mode.

a. Find an existing PDB that holds its TDE master encryption key in the CDB root keystore, working in united mode.

```
SQL> CONNECT / AS SYSDBA
Connected.
SQL> SELECT pdb_name, keystore_mode
     FROM   v$encryption_wallet w, cdb_pdbs p
     WHERE  w.con_id = p.pdb_id;
 2    3
PDB_NAME        KEYSTORE
--------------------------------- --------------------
PDB1            UNITED
PDB$SEED        UNITED
PDB_ISOLATED    ISOLATED

SQL>
```

```
SQL> CONNECT sys@PDB1 AS SYSDBA
Enter password: password
Connected.
SQL> SELECT wrl_parameter, status, wallet_type, keystore_mode
     FROM   v$encryption_wallet;
  2
WRL_PARAMETER                                    STATUS
------------------------------------------------------------------------------------------ ----------
```

```
WALLET_TYPE           KEYSTORE
--------------------- --------------------

                                          OPEN
PASSWORD              UNITED


SQL>
```

Observe that `WRL_PARAMETER` in the `V$ENCRYPTION_WALLET` view displays a null value. There is no value because the PDB TDE master encryption keys for united PDBs are stored in the CDB root keystore.
The new `KEYSTORE_MODE` column displays `UNITED`. This value is seen when `V$ENCRYPTION_WALLET` is queried from the PDB. The PDB is configured to use the CDB root keystore to store its PDB TDE master encryption key.

If you wanted the PDB keystore to be of another keystore type than the default value set at the CDB root level, then you would set the `TDE_CONFIGURATION` parameter to another value.

b. Create and open the PDB keystore with its own password.

```
SQL> ADMINISTER KEY MANAGEMENT ISOLATE KEYSTORE
                               IDENTIFIED BY password_PDB
               FROM ROOT KEYSTORE IDENTIFIED BY password_CDB
               WITH BACKUP;
  2    3    4
keystore altered.

SQL>
```

All the previously active (historical) TDE master encryption keys associated with the PDB are copied to the isolated keystore.

c. Find the directory where the PDB keystore is created.

```
SQL> SELECT wrl_parameter, status, wallet_type, keystore_mode
     FROM   v$encryption_wallet;
  2
WRL_PARAMETER                                    STATUS
------------------------------------------------------------------------------------------ ----------------------
WALLET_TYPE           KEYSTORE
--------------------- --------------------
/u01/app/oracle/admin/ORCL/tde_keystore/6413 OPEN
979102FD7535E053BA05C40AAAAD/tde/
```

```
PASSWORD             ISOLATED

SQL>
```

d. Check the existence of the `ewallet.p12` file in the subdirectory created under the `WALLET_ROOT` directory.

```
SQL> !ls -l
/u01/app/oracle/admin/ORCL/tde_keystore/6413979102FD7535E053BA05
C40AAAAD/tde
total 4
-rw------- 1 oracle oinstall 2059 Nov 27 02:07 ewallet.p12

SQL>
```

e. Create an encrypted table.

```
SQL> CREATE TABLE system.test (c NUMBER ENCRYPT);

Table created.

SQL>
```

6. Convert an isolated mode PDB to run in united mode.

a. Find an existing PDB that runs in isolated mode.

```
SQL> CONNECT / AS SYSDBA
Connected.
SQL> SELECT pdb_name, keystore_mode
     FROM   v$encryption_wallet w, cdb_pdbs p
     WHERE  w.con_id = p.pdb_id;
  2    3
PDB_NAME        KEYSTORE
------------------------------ --------------------
PDB1            ISOLATED
PDB$SEED        UNITED
PDB_ISOLATED    ISOLATED

SQL>
```

b. Log in to `PDB_ISOLATED` as `SEC_TDE` to convert `PDB_ISOLATED` to run in united mode.

```
SQL> CONNECT sec_tde@PDB_ISOLATED
Enter password: password
Connected.
SQL> ADMINISTER KEY MANAGEMENT UNITE KEYSTORE
                                IDENTIFIED BY password_PDB
             WITH ROOT KEYSTORE IDENTIFIED BY password_CDB
             WITH BACKUP;
```

```
  2  ADMINISTER KEY MANAGEMENT UNITE KEYSTORE IDENTIFIED BY
password_PDB
*
ERROR at line 1:
ORA-01031: insufficient privileges

SQL>
```

Observe that even if the SEC_TDE local user has the ADMINISTER KEY MANAGEMENT privilege, it is not sufficient to convert the PDB keystore to united mode.

c. Create a common user in the CDB root and grant the user the ADMINISTER KEY MANAGEMENT privilege commonly.

```
SQL> CONNECT / AS SYSDBA
Connected.
SQL> CREATE USER c##sec_off IDENTIFIED BY password
                            CONTAINER=ALL;
 2
User created.

SQL> GRANT create session, administer key management,
           select any dictionary, create any table
           TO c##sec_off CONTAINER=ALL;
 2    3
Grant succeeded.

SQL>
```

d. Log in to PDB_ISOLATED as the common user to convert the PDB to run in united mode.

```
SQL> CONNECT c##sec_off@PDB_ISOLATED
Enter password: password
Connected.
SQL> ADMINISTER KEY MANAGEMENT UNITE KEYSTORE
                                IDENTIFIED BY password_CDB
          WITH ROOT KEYSTORE IDENTIFIED BY password_CDB
          WITH BACKUP;
  2   3   4
 ADMINISTER KEY MANAGEMENT UNITE KEYSTORE
*
ERROR at line 1:
ORA-28357: password required to open the wallet

SQL>
```

Q/ Which keystore password does the error refer to?

A/ It refers to the PDB isolated keystore.

```
SQL> ADMINISTER KEY MANAGEMENT UNITE KEYSTORE
                                IDENTIFIED BY password_PDB
             WITH ROOT KEYSTORE IDENTIFIED BY password_CDB
             WITH BACKUP;
 2   3   4
keystore altered.

SQL>
```

e. Verify that the PDB is now running in united mode.

```
SQL> SELECT wrl_parameter, status, wallet_type, keystore_mode
     FROM   v$encryption_wallet;
 2
WRL_PARAMETER         STATUS      WALLET_TYPE           KEYSTORE
--------------------------------------------- ---------------------- ---------------------------------------------- ----------

                      OPEN        PASSWORD              UNITED

SQL>
```

All the previously active (historical) TDE master encryption keys associated with the PDB are copied to the keystore of the CDB root.

f. Create an encrypted table.

```
SQL> CREATE TABLE system.test2 (c NUMBER ENCRYPT);

Table created.

SQL> EXIT
$
```

7. Execute the `$HOME/labs/admin/cleanup_PDBs.sh` shell script to drop `PDB1` and `PDB_ISOLATED in ORCL`.
   If you used other passwords for the keystores of `PDB1 and PDB_ISOLATED in ORCL`, `update the passwords in the shell script and SQL scripts.`

```
$ $HOME/labs/admin/cleanup_PDBs.sh
…
$
```

Practice 3-3: Creating User-Defined TDE Master Encryption Keys

Overview

In this practice, you will set your own PDB TDE master encryption key and create and later use your own PDB TDE master encryption key.

Tasks

1. Re-create `PDB1` and create `PDB2` by executing the `$HOME/labs/SEC/PDBs.sh` shell script.

```
$ $HOME/labs/SEC/PDBs.sh
…
$
```

2. Set your own TDE master encryption key.
   a. Log in to the CDB root and verify that `PDB1` PDB exists and is open.

```
$ sqlplus / AS SYSDBA

SQL> SHOW PDBS

    CON_ID CON_NAME                       OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         2 PDB$SEED                       READ ONLY  NO
         5 PDB1                           READ WRITE NO
         7 PDB2                           READ WRITE NO
SQL>
```

   b. Check whether the CDB root keystore exists and is opened with a TDE master encryption key set.

```
SQL> SELECT wrl_parameter, status, keystore_mode
     FROM   v$encryption_wallet
     WHERE  con_id = 1;
 2    3
WRL_PARAMETER                                    STATUS
------------------------------------------------------------------------------------------ ----------
KEYSTORE
----------
/u01/app/oracle/admin/ORCL/tde_keystore/tde/ OPEN
NONE

SQL>
```

c. Log in to PDB1 and verify that the PDB keystore is not created yet.

```
SQL> CONNECT sys@PDB1 AS SYSDBA
Enter password: password
Connected.
```

```
SQL> SELECT wrl_parameter, status, keystore_mode
     FROM   v$encryption_wallet;
 2
WRL_PARAMETER         STATUS                              KEYSTORE
--------------------- ----------------------------------- ----------

                      CLOSED                               UNITED

SQL>SQL>
```

d. Create the security officer who will define and activate a TDE master encryption for PDB1 by using a defined raw binary value.

```
SQL> CREATE USER sec_tde IDENTIFIED BY password;

User created.

SQL> GRANT create session, administer key management,
           unlimited tablespace, create any table,
           select any dictionary TO sec_tde;
  2    3
Grant succeeded.

SQL>
```

e. Connect as the security officer and create the PDB keystore.

```
SQL> CONNECT sec_tde@PDB1
Enter password: password
Connected.
SQL> ADMINISTER KEY MANAGEMENT CREATE KEYSTORE
                IDENTIFIED BY password_PDB;
 2
keystore altered.

SQL>
```

f. Check whether the PDB keystore is created.

```
SQL> SELECT wrl_parameter, status, keystore_mode
     FROM   v$encryption_wallet;
 2
WRL_PARAMETER                                          STATUS
------------------------------------------------------------------------------------------ ----------
KEYSTORE
----------
```

```
/u01/app/oracle/admin/ORCL/tde_keystore/624 CLOSED
B9740A1760A7E053CE4CC40A6F4F/tde/
ISOLATED

SQL>
```

g. Open the keystore of the PDB.

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
                IDENTIFIED BY password_PDB;
  2
keystore altered.

SQL>
```

h. Define and activate your own TDE master encryption key for PDB1.

```
SQL> ADMINISTER KEY MANAGEMENT CREATE KEY '17BA048707B402493C'
                IDENTIFIED BY password_PDB WITH BACKUP;
ADMINISTER KEY MANAGEMENT CREATE KEY '17BA048707B402493C'
*
ERROR at line 1:
ORA-46685: invalid master key identifier or master key value

SQL>
```

```
SQL> !oerr ora 46685
46685, 00000, "invalid master key identifier or master key
value"
// *Cause: The master key identifier or master key value was
invalid.
//          To be valid, the values supplied for both the master
key identifier and the master key value must be hex-encoded and
must have the following properties:
//          - For a master key value to be valid, its length
must be a valid key length for the specified encryption
algorithm.
//          - For a master key identifier to be valid, it must
have a length of 16 bytes.
//          - For a master key identifier to be valid, it must
have a non-zero value.
// *Action: Specify a correct value for the master key
identifier or master key in the master key string.

SQL>
```

i. Define a proper value.

```
SQL> ADMINISTER KEY MANAGEMENT CREATE KEY
'102030405060708011121314151617 18:3D432109DF88967A541967062A6F4E
460E892318E307F017BA048707B402493C'
      IDENTIFIED BY password_PDB WITH BACKUP;
 2    3
keystore altered.

SQL>
```

Q/ You could have received the following error message:

```
ORA-46658: keystore not open in the container
```

What would have been the reason of the failure?

A/ The PDB keystore must be opened. The keystore can be temporarily opened with the FORCE KEYSTORE clause. In this case, you would have used the following command:

```
SQL> ADMINISTER KEY MANAGEMENT SET KEY '17BA048707B402493C'
      FORCE KEYSTORE IDENTIFIED BY password_PDB WITH BACKUP;
```

j. Verify that the key is set in the PDB keystore.

```
SQL> SELECT key_id, activating_pdbname FROM v$encryption_keys;

KEY_ID
------------------------------------------------------------------------------------------------------------------------
ACTIVATING_PDBNAME
--------------------------------------------------
ARAgMEBQYHCAERITFBUWFxgAAAAAAAAAAAAAAAAAAAAAAAAAAA

SQL>
```

Q/ Is the key activated in the PDB?

A/ It seems not. The ACTIVATING_PDBNAME column does not show any value.

k. Test if you can create a table with an encrypted column.

```
SQL> CREATE TABLE test (c NUMBER ENCRYPT);
CREATE TABLE test (c NUMBER ENCRYPT)
*
ERROR at line 1:
ORA-28361: master key not yet set

SQL>
```

This confirms our assumption. By creating the key with the `ADMINISTER KEY MANAGEMENT CREATE KEY` command, you can activate the key whenever you want.

l. Use the `key_id` found in the `v$encryption_keys` view in Step 2-j.

```
SQL> ADMINISTER KEY MANAGEMENT USE KEY
       'ARAgMEBQYHCAERITFBUWFxgAAAAAAAAAAAAAAAAAAAAAAAAAAAAA'
       FORCE KEYSTORE IDENTIFIED BY password_PDB WITH BACKUP;
 2     3
keystore altered.

SQL>
```

m. Test if you can create a table with an encrypted column.

```
SQL> CREATE TABLE test (c NUMBER ENCRYPT);

Table created.

SQL> DESC test
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 C                                                  NUMBER ENCRYPT

SQL>
```

n. Verify that the key is now set in the PDB keystore.

```
SQL> SELECT key_id, activating_pdbname FROM v$encryption_keys;

KEY_ID
------------------------------------------------------------------------------------------------------------------------
ACTIVATING_PDBNAME
------------------------------------------------
ARAgMEBQYHCAERITFBUWFxgAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
PDB1

SQL>
```

3. Define and use your own TDE master encryption key in one single statement.

a. Log in to `PDB2` and verify that the PDB keystore does not exist yet.

```
SQL> CONNECT sys@PDB2 AS SYSDBA
Enter password: password
Connected.
SQL> SELECT wrl_parameter, status, keystore_mode
     FROM   v$encryption_wallet;
 2
```

```
WRL_PARAMETER                                                    STATUS
---------------------------------------------------------------- ------------------------------------
KEYSTORE
--------------------

                                                                 CLOSED
UNITED

SQL>
```

b. Connect as the security officer and create the PDB keystore.

```
SQL> CREATE USER sec_tde IDENTIFIED BY password;

User created.

SQL> GRANT create session, ADMINISTER KEY MANAGEMENT,
           select any dictionary, unlimited tablespace,
           create any table
     TO sec_tde;
  2    3    4
Grant succeeded.

SQL>
```

```
SQL> CONNECT sec_tde@PDB2
Enter password: password
Connected.
SQL> ADMINISTER KEY MANAGEMENT CREATE KEYSTORE
                    IDENTIFIED BY password_PDB;
 2
keystore altered.

SQL>
```

c. Check whether the PDB keystore is created.

```
SQL> SELECT wrl_parameter, status, keystore_mode
     FROM   v$encryption_wallet;
  2
WRL_PARAMETER                                    STATUS
------------------------------------------------ ------------------------------------
KEYSTORE
--------------------
/u01/app/oracle/admin/ORCL/tde_keystore/6249 CLOSED
3C4D0CFD420EE053CE4CC40A86D4/tde/
```

```
ISOLATED

SQL>
```

d. Define and use your own TDE master encryption key for `PDB2`.

```
SQL> ADMINISTER KEY MANAGEMENT SET KEY
'102030405060708011121314151617 18:3D432109DF88967A541967062A6F4E
460E892318E307F017BA048707B402493C'
      FORCE KEYSTORE IDENTIFIED BY password_PDB WITH BACKUP;
 2   3
keystore altered.

SQL>
```

e. Verify that the key is set in the PDB keystore.

```
SQL> SELECT key_id, activating_pdbname FROM v$encryption_keys;

KEY_ID
------------------------------------------------------------------------------------------------------------------------
ACTIVATING_PDBNAME
------------------------------------------------
ARAgMEBQYHCAERITFBUWFxgAAAAAAAAAAAAAAAAAAAAAAAAAAAA
PDB2

SQL>
```

Observe that the key is activated. Compare with the result of Step 2-j.

f. Test if you can create a table with an encrypted column.

```
SQL> CREATE TABLE test (c NUMBER ENCRYPT);

Table created.

SQL> DESC test
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 C                                                  NUMBER ENCRYPT

SQL> EXIT
$
```

4. Execute the `$HOME/labs/admin/cleanup_PDBs.sh` shell script to drop `PDB1 and PDB2 in ORCL`.

```
$ $HOME/labs/admin/cleanup_PDBs.sh
…
$
```

Practice 3-4: Exporting and Importing Fixed-User Database Links

Overview

In this practice, you will prevent an intruder from de-obfuscating the obfuscated password of fixed-user database links by not exporting the password, but the metadata only. After exporting database links with invalid password values, you must reset the passwords after reimporting them.

Tasks

1. Execute the `$HOME/labs/SEC/PDBs_TEST_TAB.sh` shell script to re-create `PDB1` and `PDB2`, and create `SYSTEM.TEST` in `PDB2`.

```
$ $HOME/labs/SEC/PDBs_TEST_TAB.sh
…
$
```

2. Export and import database links **with obfuscated passwords**.
   a. Log in to the `PDB1` PDB and create a fixed-user database link that points to `PDB2`. (This could be another CDB if there was another one on the VM.)

```
$ sqlplus system@PDB1

Enter password: password

SQL> CREATE DATABASE LINK test CONNECT TO system
                      IDENTIFIED BY password
                      USING 'PDB2';
  2    3
Database link created.

SQL>
```

   b. Test that the database link allows you to read the `SYSTEM.TEST` table in `PDB2`.

```
SQL> SELECT * FROM system.test@test;

         C
----------
         1

SQL>
```

c. Check the enforcement status of credentials encryption in the dictionary.

```
SQL> SELECT enforcement FROM dictionary_credentials_encrypt;

ENFORCEM
--------
DISABLED

SQL>
```

d. Before exporting the database link, create a logical directory for the export dump file.

```
SQL> CREATE OR REPLACE DIRECTORY dp_dump AS '/tmp';

Directory created.

SQL> EXIT
$
```

e. Export the fixed-user database link.

```
$ rm /tmp/dp_test.dmp
rm: cannot remove `/tmp/dp_test.dmp': No such file or directory
$ expdp system@PDB1 directory=dp_dump dumpfile=dp_test
Password: password
…
Processing object type SCHEMA_EXPORT/DB_LINK
Processing object type SCHEMA_EXPORT/TABLE/TABLE
Processing object type SCHEMA_EXPORT/TABLE/COMMENT
Processing object type SCHEMA_EXPORT/TABLE/INDEX/INDEX
Processing object type SCHEMA_EXPORT/TABLE/CONSTRAINT/CONSTRAINT
Master table "SYSTEM"."SYS_EXPORT_SCHEMA_01" successfully
loaded/unloaded
******************************************************************
Dump file set for SYSTEM.SYS_EXPORT_SCHEMA_01 is:
  /tmp/dp_test.dmp
Job "SYSTEM"."SYS_EXPORT_SCHEMA_01" successfully completed at
Tue Nov 28 09:12:10 2017 elapsed 0 00:00:54

$
```

The obfuscated password of the exported database link is not protected in the dump file.

f. Drop the fixed-user database link.

```
$ sqlplus system@PDB1

Enter password: password

SQL> DROP DATABASE LINK test;

Database link dropped.

SQL> EXIT
$
```

g. Import the fixed-user database link.

```
$ impdp system@PDB1 directory=dp_dump dumpfile=dp_test
Password: password
…
Processing object type SCHEMA_EXPORT/DB_LINK
Job "SYSTEM"."SYS_IMPORT_FULL_01" successfully completed at Tue
Nov 28 09:14:50 2017 elapsed 0 00:00:05
$
```

h. Test that the database link works to read the `TEST` table in `PDB1` of `cdb1`.

```
$ sqlplus system@PDB1

Enter password: password

SQL> SELECT * FROM system.test@test;

         C
----------
         1

SQL> EXIT
$
```

i. To avoid any security risk, execute the export operation with the recommended Data Pump parameter that encrypts the data in the dump file. By default, all data including metadata is encrypted in the export dump file.

```
$ rm /tmp/dp_test.dmp
$ expdp system@PDB1 directory=dp_dump dumpfile=dp_test
ENCRYPTION_PWD_PROMPT=YES
Enter password: password
…
Encryption Password: enc_password
…
Processing object type SCHEMA_EXPORT/DB_LINK
```

```
Processing object type SCHEMA_EXPORT/TABLE/TABLE
Processing object type SCHEMA_EXPORT/TABLE/COMMENT
Processing object type SCHEMA_EXPORT/TABLE/INDEX/INDEX
Processing object type SCHEMA_EXPORT/TABLE/CONSTRAINT/CONSTRAINT
Master table "SYSTEM"."SYS_EXPORT_SCHEMA_01" successfully
loaded/unloaded
******************************************************************
Dump file set for SYSTEM.SYS_EXPORT_SCHEMA_01 is:
  /tmp/dp_test.dmp
Job "SYSTEM"."SYS_EXPORT_SCHEMA_01" successfully completed at
Tue Nov 28 09:18:52 2017 elapsed 0 00:00:52
$
```

j. After dropping the database link, import it back.

```
$ sqlplus system@PDB1

Enter password: password

SQL> DROP DATABASE LINK test;

Database link dropped.

SQL> EXIT
$
```

```
$ impdp system@PDB1 directory=dp_dump dumpfile=dp_test
ENCRYPTION_PWD_PROMPT=YES
Password: password
…
Encryption Password: enc_password
…
Processing object type SCHEMA_EXPORT/DB_LINK
Job "SYSTEM"."SYS_IMPORT_FULL_01" successfully completed at Tue
Nov 28 09:21:25 2017 elapsed 0 00:00:09

$
```

Remark: Then during the import operation, whether the keystore is opened or closed affects whether or not an encryption password must be provided. If the keystore is opened during the export operation and you provided an encryption password, then you do not need to provide the password during the import operation. If the keystore is closed during the export operation, then you must provide the password during the import operation.

k. Test that the database link works to read the `TEST` table in `PDB1`.

```
$ sqlplus system@PDB1

Enter password: password

SQL> SELECT * FROM system.test@test;

         C
----------
         1

SQL>
```

If you want to protect only the passwords of database links because there is no other sensitive data in the data to be exported, then use the method explained in the next task.

3. Enable credentials encryption.

a. Enable the credentials encryption to allow the fixed-user database links to be exported with an invalid password value in the dump file during export. Log in to the PDB and enable the enforcement of credentials encryption in the dictionary.

```
SQL> ALTER DATABASE dictionary encrypt credentials;
ALTER DATABASE dictionary encrypt credentials
*
ERROR at line 1:
ORA-28447: insufficient privilege to execute ALTER DATABASE
DICTIONARY statement

SQL> CONNECT sys@PDB1 AS SYSDBA
Enter password: password
Connected.
SQL> ALTER DATABASE dictionary encrypt credentials;
ALTER DATABASE dictionary encrypt credentials
*
ERROR at line 1:
ORA-28447: insufficient privilege to execute ALTER DATABASE
DICTIONARY statement

SQL>
```

b. Create a user with the SYSKM privilege.

```
SQL> CREATE USER enccreds IDENTIFIED BY password;

User created.

SQL> GRANT create session, syskm TO enccreds;

Grant succeeded.

SQL>
```

c. Connect as the SYSKM privileged user in PDB1 and enable the enforcement of credentials encryption in the dictionary.

```
SQL> CONNECT enccreds@PDB1 AS SYSKM
Enter password: password
Connected.
SQL> ALTER DATABASE dictionary encrypt credentials;

Database dictionary altered.

SQL> SELECT enforcement FROM dictionary_credentials_encrypt;

ENFORCEM
--------
ENABLED

SQL> EXIT
$
```

4. Export and import database links **with invalid passwords**.

a. Export the fixed-user database link.

```
$ rm /tmp/dp_test.dmp
$ expdp system@PDB1 directory=dp_dump dumpfile=dp_test
Enter password: password
…
Processing object type SCHEMA_EXPORT/DB_LINK
ORA-39395: Warning: object SYSTEM.TEST requires password reset
after import

Processing object type SCHEMA_EXPORT/TABLE/TABLE
Processing object type SCHEMA_EXPORT/TABLE/COMMENT
Processing object type SCHEMA_EXPORT/TABLE/INDEX/INDEX
Processing object type SCHEMA_EXPORT/TABLE/CONSTRAINT/CONSTRAINT
```

```
Master table "SYSTEM"."SYS_EXPORT_SCHEMA_01" successfully
loaded/unloaded
******************************************************
Dump file set for SYSTEM.SYS_EXPORT_SCHEMA_01 is:
  /tmp/dp_test.dmp
Job "SYSTEM"."SYS_EXPORT_SCHEMA_01" completed with 1 error(s) at
Tue Nov 28 09:50:14 2017 elapsed 0 00:00:44

$
```

The database link password is stored as an invalid value in the dump file. This is the reason for the warning message that tells you that after importing the database link, you must reset the password of the database link.

b. Drop the fixed-user database link.

```
$ sqlplus system@PDB1

Enter password: password

SQL> DROP DATABASE LINK test;

Database link dropped.

SQL> EXIT
$
```

c. Import the fixed-user database link. The database link password is still stored as an invalid value in the dump file. This is the reason for the warning message.

```
$ impdp system@PDB1 directory=dp_dump dumpfile=dp_test
Password: password
…
Processing object type SCHEMA_EXPORT/DB_LINK
ORA-39395: Warning: object SYSTEM.TEST requires password reset
after import

Job "SYSTEM"."SYS_IMPORT_FULL_01" completed with 1 error(s) at
Tue Nov 28 09:53:08 2017 elapsed 0 00:00:05

$
```

d. Test that the database link works to read the `TEST` table in `PDB1` of `cdb1`. If the `SYSTEM` user is not granted the `ALTER DATABASE LINK` system privilege, ask the user with the `SYSDBA` privilege to grant the privilege to `SYSTEM`.

```
$ sqlplus system@PDB1

Enter password: password

SQL> SELECT * FROM system.test@test;
SELECT * FROM system.test@test
                         *
ERROR at line 1:
ORA-28449: cannot use an invalidated database link


SQL> ALTER DATABASE LINK test CONNECT TO system
                                IDENTIFIED BY password;
ALTER DATABASE LINK test CONNECT TO system IDENTIFIED BY
password
*
ERROR at line 1:
ORA-01031: insufficient privileges


SQL> CONNECT sys@PDB1 AS SYSDBA
Enter password: password
Connected.
SQL> GRANT alter database link TO system;

Granted succeeded.

SQL> CONNECT system@PDB1
Enter password: password
Connected.
SQL> ALTER DATABASE LINK test CONNECT TO system
                                IDENTIFIED BY password;
Database link altered.

SQL> SELECT * FROM system.test@test;

         C
----------
         1
```

```
SQL> EXIT
$
```

5. Execute the `$HOME/labs/SEC/cleanup_DBlinks.sh` shell script to drop `the database link created in PDB1 and then the $HOME/labs/admin/cleanup_PDBs.sh` shell script.

```
$ $HOME/labs/SEC/cleanup_DBlinks.sh
…
$ $HOME/labs/admin/cleanup_PDBs.sh
…
$
```

Practice 3-5: Encrypting Sensitive Data in Database Replay Files

Overview

In this practice, you will protect sensitive data when the data is captured by Database Replay in capture files. You will encrypt the data captured by Database Replay in capture files.

Tasks

1. Before starting the practice, execute the `$HOME/labs/SEC/DBReplay.sh` shell script. The script re-creates `PDB1` and sets the master key set in `PDB1`.

```
$ $HOME/labs/SEC/DBReplay.sh
…
$
```

2. Set the password for the Database Replay user in the keystore.
   a. Log in to `ORCL` and verify the master key set.

```
$ sqlplus / AS SYSDBA

SQL> SELECT wrl_parameter, status, keystore_mode, wallet_type
     FROM   v$encryption_wallet WHERE con_id = 1;
  2
WRL_PARAMETER                                          STATUS
------------------------------------------------------------------------------------------------------------------------ ------------------------------
KEYSTORE WALLET_TYPE
-------- --------------------
/u01/app/oracle/admin/ORCL/tde_keystore/tde/ OPEN
NONE     PASSWORD


SQL>
```

   b. Set the password for the `oracle.rat.database_replay.encryption` user in the keystore.

```
SQL> ADMINISTER KEY MANAGEMENT ADD SECRET 'replay_password'
        FOR CLIENT 'oracle.rat.database_replay.encryption'
        IDENTIFIED BY password WITH BACKUP;
  2    3
keystore altered.

SQL>
```

3. Capture workload data by using Database Replay.
   a. The Database Replay capture creates files in a directory. Create the logical directory for the capture files.

```
SQL> CONNECT system
Enter password: password
Connected.
SQL> HOST mkdir /home/oracle/replay

SQL> CREATE OR REPLACE DIRECTORY oltp
                  AS '/home/oracle/replay';
 2
Directory created.

SQL>
```

   b. Start capturing data with the Database Replay procedure. Specify the encryption algorithm used to encrypt the data in the workload capture files.

```
SQL> exec DBMS_WORKLOAD_CAPTURE.START_CAPTURE (name =>
'OLTP_peak', dir => 'OLTP', ENCRYPTION => 'AES256')

PL/SQL> procedure successfully completed.

SQL>
```

   c. During the capture, in another terminal window session called *Workload Session*, execute the workload on the CDB and PDBs.

```
$ $HOME/labs/SEC/workload.sh
…
$
```

   d. When you consider that the workload is sufficient for replay testing, stop the capture.

```
SQL> exec DBMS_WORKLOAD_CAPTURE.FINISH_CAPTURE ()

PL/SQL> procedure successfully completed.

SQL>
```

   The data in capture files is encrypted with an encryption key stored in the header of each file. The encryption key was generated with the encryption key mapped to the `oracle.rat.database_replay.encryption` user retrieved from the keystore.

4. Process the capture files and replay the workload.
   a. As in the normal whole process of Database Replay, after capturing the workload into files, you process the capture files.

```
SQL> exec DBMS_WORKLOAD_REPLAY.PROCESS_CAPTURE (capture_dir =>
'OLTP')

PL/SQL> procedure successfully completed.

SQL>
```

   b. Then you initialize and prepare the replay.

```
SQL> exec DBMS_WORKLOAD_REPLAY.INITIALIZE_REPLAY(replay_name =>
'R', replay_dir => 'OLTP')

PL/SQL> procedure successfully completed.

SQL> exec DBMS_WORKLOAD_REPLAY.PREPARE_REPLAY ()

PL/SQL> procedure successfully completed.

SQL>
```

   During the last three steps, the keystore must be open. The password for the Database Replay user `oracle.rat.database_replay.encryption` is retrieved and verified in the keystore.

   c. You are ready to start workload clients to replay the captured workload with `wrc` clients. In *Workload Session*, if the workload is not finished, interrupt the `$HOME/labs/SEC/workload.sh` shell script and start the `wrc` process.

```
$ wrc REPLAYDIR=/home/oracle/replay USERID=system
…
Password: password
(wrc_main_7555.trc) ORA-15526: failed to validate user password
for capture
$
```

   Observe that the client fails with an error message.

   The client user password needs to be validated against the `oracle.rat.database_replay.encryption` user password stored in the keystore. On the client side, the `oracle.rat.database_replay.encryption` client password is set in an SSO wallet.

d. Create the SSO wallet to store the `oracle.rat.database_replay.encryption` client password that must match the password set in step 2-b. First, create the directory for the `cwallet.sso` file. Then create the `cwallet.sso` file.

```
$ mkdir /tmp/replay_encrypt_cwallet
$ mkstore -wrl /tmp/replay_encrypt_cwallet -create
…
Enter password: cwallet_password
Enter password again: cwallet_password
$
```

e. Add the credentials for the `oracle.rat.database_replay.encryption` user.

```
$ mkstore -wrl /tmp/replay_encrypt_cwallet -createEntry
'oracle.rat.database_replay.encryption' "replay_password"
 …
Enter wallet password: cwallet_password
$
```

f. Start the workload clients to replay the captured workload with `wrc` clients.

```
$ wrc REPLAYDIR=/home/oracle/replay USERID=system
WALLETDIR=/tmp/replay_encrypt_cwallet
…
Password: password
Wait for the replay to start (17:20:04)
```

The password required is the `SYSTEM` user password.

g. The `wrc` client is waiting for DB Replay to start in the database. Back to the initial session in the CDB, execute the `START_REPLAY` procedure.

```
SQL> exec DBMS_WORKLOAD_REPLAY.START_REPLAY ()

PL/SQL> procedure successfully completed.

SQL> EXIT
$
```

h. As soon as the DB Replay procedure is started in the database, the clients start replaying and finally completes.

```
Replay client 1 started (17:22:33)
Replay client 1 finished (17:24:24)
$
```

5. Execute the `$HOME/labs/SEC/cleanup_DBReplay.sh` shell script to drop `the DB Replay capture files.`

```
$ $HOME/labs/SEC/cleanup_DBReplay.sh
…
$
```

Practices for Lesson 4: Using RMAN Enhancements

Practices for Lesson 4: Overview

Overview

In these practices, you recover a plugged non-CDB and PDB using preplugin backups, duplicate a PDB into an existing CDB without instantiating a new instance, and finally duplicate an on-premises CDB for migration to cloud.

Practice 4-1: Recovering a Plugged Non-CDB Using Preplugin Backups

Overview

In this practice, you recover a plugged non-CDB using preplugin backups. Preplugin backups are all backups of a non-CDB taken before unplugging/plugging operations, usable after plugging into a new CDB.

Tasks

1. You use the Oracle By Example “Recovering Plugged Non-CDB Using Preplugin Backups” to see how to proceed. Launch a browser and click the bookmark from the Bookmarks toolbar of the browser. The URL is `file:////home/oracle/labs/OBE/recovering_plugged_noncdb_using_preplugin_backups/recovering_plugged_noncdb_using_preplugin_backups.html`

Practice 4-2: Recovering a Plugged PDB Using Preplugin Backups

Overview

In this practice, you recover a plugged PDB using preplugin backups.

Tasks

1. Before starting the practice, execute the `$HOME/labs/admin/glogin_4.sh` shell script. It sets formatting for all columns selected in queries.

```
$ $HOME/labs/admin/glogin_4.sh
…
$
```

2. Re-create `PDB1` for the practice by executing the `$HOME/labs/RMAN/PDB1_backup.sh` shell script. The shell script also sets the PDB key in the keystore (because PDBs use encryption on cloud VMs) and creates the `HR` schema in `PDB1`. In the case you run out of space, clean up the Fast Recovery Area with commands such as:

```
rman target /
DELETE OBSOLETE;
DELETE ARCHIVELOG ALL;
ALTER SYSTEM SET DB_RECOVERY_FILE_DEST_SIZE=30G;
EXIT
```

```
$ $HOME/labs/RMAN/PDB1_backup.sh
…
$
```

3. Prepare the PDB before unplugging to use preplugin backups.
   a. In your current terminal window (called *Session1*), display the content of the `HR.DEPARTMENTS` table in `PDB1`.

```
$ sqlplus system@PDB1

Enter password: password

SQL> SELECT * FROM hr.departments;

DEPARTMENT_ID DEPARTMENT_NAME      MANAGER_ID LOCATION_ID
------------- -------------------- ---------- -----------
           10 Administration              200        1700
           20 Marketing                   201        1800
           30 Purchasing                  114        1700
           40 Human Resources             203        2400
           50 Shipping                    121        1500
           60 IT                          103        1400
```

```
           70 Public Relations            204        2700
           80 Sales                       145        2500
           90 Executive                   100        1700
          100 Finance                     108        1700
          110 Accounting                  205        1700
          120 Treasury                               1700
          130 Corporate Tax                          1700
          140 Control And Credit                     1700
          150 Shareholder Services                   1700
          160 Benefits                               1700
          170 Manufacturing                          1700
          180 Construction                           1700
          190 Contracting                            1700
          200 Operations                             1700
          210 IT Support                             1700
          220 NOC                                    1700
          230 IT Helpdesk                            1700
          240 Government Sales                       1700
          250 Retail Sales                           1700
          260 Recruiting                             1700
          270 Payroll                                1700

27 rows selected.

SQL>
```

b. In another terminal window (called *Session2*), launch `rman` to create backups for `PDB1` of `ORCL` before unplugging. It is recommended to back up all the archivelogs before the unplug operation so that they can be restored from preplugin backups when a recovery might be required in the destination CDB.

```
$ rman target /

connected to target database: ORCL (DBID=1488874136)

RMAN> BACKUP PLUGGABLE DATABASE pdb1 PLUS ARCHIVELOG;

Starting backup at 30-NOV-17
current log archived
using target database control file instead of recovery catalog
allocated channel: ORA_DISK_1
channel ORA_DISK_1: SID=43 device type=DISK
skipping archived logs of thread 1 from sequence 121 to 175;
already backed up
channel ORA_DISK_1: starting archived log backup set
channel ORA_DISK_1: specifying archived log(s) in backup set
```

```
input archived log thread=1 sequence=176 RECID=176
STAMP=961428612
input archived log thread=1 sequence=177 RECID=177
STAMP=961428715
channel ORA_DISK_1: starting piece 1 at 30-NOV-17
channel ORA_DISK_1: finished piece 1 at 30-NOV-17
piece
handle=/u03/app/oracle/fast_recovery_area/ORCL/backupset/2017_11
_30/o1_mf_annnn_TAG20171130T153156_f2093d7p_.bkp
tag=TAG20171130T153156 comment=NONE
channel ORA_DISK_1: backup set complete, elapsed time: 00:00:01
Finished backup at 30-NOV-17

Starting backup at 30-NOV-17
using channel ORA_DISK_1
channel ORA_DISK_1: starting full datafile backup set
channel ORA_DISK_1: specifying datafile(s) in backup set
input datafile file number=00110
name=/u02/app/oracle/oradata/ORCL/pdb1/ORCL/5F35EC627C3606EEE053
0241C40A1875/datafile/o1_mf_sysaux_f208zk5x_.dbf
input datafile file number=00109
name=/u02/app/oracle/oradata/ORCL/pdb1/ORCL/5F35EC627C3606EEE053
0241C40A1875/datafile/o1_mf_system_f208zk5t_.dbf
input datafile file number=00111
name=/u02/app/oracle/oradata/ORCL/pdb1/ORCL/5F35EC627C3606EEE053
0241C40A1875/datafile/o1_mf_undotbs1_f208zk60_.dbf
channel ORA_DISK_1: starting piece 1 at 30-NOV-17
channel ORA_DISK_1: finished piece 1 at 30-NOV-17
piece
handle=/u03/app/oracle/fast_recovery_area/ORCL/5F35EC627C3606EEE
0530241C40A1875/backupset/2017_11_30/o1_mf_nnndf_TAG20171130T153
157_f2093ghl_.bkp tag=TAG20171130T153157 comment=NONE
channel ORA_DISK_1: backup set complete, elapsed time: 00:00:15
Finished backup at 30-NOV-17

Starting backup at 30-NOV-17
current log archived
using channel ORA_DISK_1
channel ORA_DISK_1: starting archived log backup set
channel ORA_DISK_1: specifying archived log(s) in backup set
input archived log thread=1 sequence=178 RECID=178
STAMP=961428733
channel ORA_DISK_1: starting piece 1 at 30-NOV-17
channel ORA_DISK_1: finished piece 1 at 30-NOV-17
piece
handle=/u03/app/oracle/fast_recovery_area/ORCL/backupset/2017_11
```

```
_30/o1_mf_annnn_TAG20171130T153214_f2093y4m_.bkp
tag=TAG20171130T153214 comment=NONE
channel ORA_DISK_1: backup set complete, elapsed time: 00:00:01
Finished backup at 30-NOV-17

Starting Control File and SPFILE Autobackup at 30-NOV-17
piece
handle=/u03/app/oracle/fast_recovery_area/ORCL/autobackup/2017_1
1_30/o1_mf_s_961428735_f2093zo7_.bkp comment=NONE
Finished Control File and SPFILE Autobackup at 30-NOV-17

RMAN>
```

c. Back in *Session1*, export RMAN backup information that belongs to `PDB1` to its dictionary before unplugging so that preplugin backups can be used in the target CDB even if no new backup is performed after plugging. The metadata is transported along with the PDB during the migration.

```
SQL> exec DBMS_PDB.EXPORTRMANBACKUP('PDB1')

PL/SQL procedure successfully completed.

SQL>
```

4. Unplug and plug the PDB.

a. Unplug `PDB1` from `ORCL and export the PDB key.` (This is required on cloud VMs because the PDBs use encryption.)

```
SQL> CONNECT / AS SYSDBA
Connected.
SQL> ALTER PLUGGABLE DATABASE pdb1 CLOSE;

Pluggable database altered.

SQL> HOST rm /tmp/pdb1.xml
rm: Cannot remove '/tmp/pdb1.xml': No such file or directory

SQL> ALTER PLUGGABLE DATABASE pdb1
           UNPLUG INTO '/tmp/pdb1.xml'
           ENCRYPT USING "unplug_password";
 2   3
Pluggable database altered.

SQL>
```

b. Archive the current redo log and quit the session.

```
SQL> ALTER SYSTEM SWITCH LOGFILE;

System altered.

SQL> SELECT name, next_change#, sequence#
     FROM   v$archived_log ORDER BY 3;

NAME
--------------------------------------------------------------------------------
----------
NEXT_CHANGE#  SEQUENCE#
------------ ----------
…
/u03/app/oracle/fast_recovery_area/ORCL/archivelog/2017_11_30/o1
_mf_1_178_f2093xs8_.arc
     3500964        178

/u03/app/oracle/fast_recovery_area/ORCL/archivelog/2017_11_30/o1
_mf_1_179_f209rf05_.arc
     3502788        179

179 rows selected.

SQL>
```

c. Plug PDB1 as ORCL_PDB1 into ORCL and open ORCL_PDB1. (The target CDB could be another CDB. There is no other CDB on the VM.)

```
SQL> HOST mkdir /u02/app/oracle/oradata/ORCL/orcl_pdb1

SQL> CREATE PLUGGABLE DATABASE orcl_pdb1 AS CLONE
           USING '/tmp/pdb1.xml'
           KEYSTORE IDENTIFIED BY CDB_keystore_password
            DECRYPT USING "unplug_password"
  CREATE_FILE_DEST = '/u02/app/oracle/oradata/ORCL/orcl_pdb1';
  2    3    4    5
Pluggable database created.

SQL> ALTER PLUGGABLE DATABASE orcl_pdb1 OPEN;

Pluggable database altered.

SQL>
```

d. In *Session2*, check whether the preplugin backups for `ORCL_PDB1` are cataloged in `ORCL`.

```
RMAN> SET PREPLUGIN CONTAINER=orcl_pdb1;

executing command: SET PREPLUGIN CONTAINER

RMAN>
```

```
RMAN> LIST PREPLUGIN BACKUP;

List of Backup Sets
===================
BS Key  Size       Device Type Elapsed Time Completion Time
------- ---------- ----------- ------------ ---------------
3       768.00K    SBT_TAPE    00:00:01     27-NOV-17
        BP Key: 3   Status: AVAILABLE Compressed: YES Tag:
TAG20171127T080017
        Handle: 05skjnki_1_1   Media:
storage.oraclecorp.com/v1/Storage-jpizana/dbaaslrg

  List of Archived Logs in backup set 3
  Thrd Seq     Low SCN    Low Time  Next SCN   Next Time
  ---- ------- ---------- --------- ---------- ---------
  1    3       2510936    27-NOV-17 2512455    27-NOV-17
  1    4       2512455    27-NOV-17 2612473    27-NOV-17
…
BS Key  Size       Device Type Elapsed Time Completion Time
------- ---------- ----------- ------------ ---------------
201     28.50K     DISK        00:00:00     30-NOV-17
        BP Key: 210   Status: AVAILABLE  Compressed: NO  Tag:
TAG20171130T153214
        Piece Name:
/u03/app/oracle/fast_recovery_area/ORCL/backupset/2017_11_30/o1_
mf_annnn_TAG20171130T153214_f2093y4m_.bkp

  List of Archived Logs in backup set 201
  Thrd Seq     Low SCN    Low Time  Next SCN   Next Time
  ---- ------- ---------- --------- ---------- ---------
  1    178     3500932    30-NOV-17 3500964    30-NOV-17

RMAN>
```

e. Check whether the preplugin archive log files for `ORCL_PDB1` are cataloged in `ORCL`.

```
RMAN> LIST PREPLUGIN ARCHIVELOG ALL;

List of Archived Log Copies for database with db_unique_name
ORCL
=====================================================================
Key     Thrd Seq     S Low Time
------- ---- ------- - --------------------
121     1    121     A 29-NOV-17
        Name:
/u03/app/oracle/fast_recovery_area/ORCL/archivelog/2017_11_29/o1
_mf_1_121_f1xpddp4_.arc

…

178     1    178     A 30-NOV-17
        Name:
/u03/app/oracle/fast_recovery_area/ORCL/archivelog/2017_11_30/o1
_mf_1_178_f2093xs8_.arc


RMAN>
```

Q/ Has the last redo log file archived been cataloged?

A/ No, because it was created after the PDB was unplugged.

f. Catalog the last archived log file missing, generated after the unplug operation.

```
RMAN> CATALOG PREPLUGIN ARCHIVELOG
'/u03/app/oracle/fast_recovery_area/ORCL/archivelog/2017_11_30/o
1_mf_1_179_f209rf05_.arc';

cataloged archived log
archived log file
name=/u03/app/oracle/fast_recovery_area/ORCL/archivelog/2017_11_
30/o1_mf_1_179_f209rf05_.arc RECID=179 STAMP=0

RMAN>
```

g. Verify that the cataloged preplugin backups and archive log files are available on disk.

```
RMAN> CROSSCHECK PREPLUGIN BACKUP;

allocated channel: ORA_SBT_TAPE_1
channel ORA_SBT_TAPE_1: SID=38 device type=SBT_TAPE
```

```
channel ORA_SBT_TAPE_1: Oracle Database Backup Service Library
VER=3.17.9.5
…
backup piece
handle=/u03/app/oracle/fast_recovery_area/ORCL/5F35EC627C3606EEE
0530241C40A1875/backupset/2017_11_30/o1_mf_nnndf_TAG20171130T153
157_f2093ghl_.bkp RECID=209 STAMP=961428718
crosschecked backup piece: found to be 'AVAILABLE'
backup piece
handle=/u03/app/oracle/fast_recovery_area/ORCL/backupset/2017_11
_30/o1_mf_annnn_TAG20171130T153214_f2093y4m_.bkp RECID=210
STAMP=961428734
Crosschecked 94 objects



RMAN>
```

```
RMAN> CROSSCHECK PREPLUGIN ARCHIVELOG ALL;


released channel: ORA_SBT_TAPE_1
released channel: ORA_SBT_TAPE_2
released channel: ORA_SBT_TAPE_3
released channel: ORA_SBT_TAPE_4
released channel: ORA_SBT_TAPE_5
released channel: ORA_DISK_1
allocated channel: ORA_DISK_1
channel ORA_DISK_1: SID=38 device type=DISK
validation succeeded for archived log
archived log file
name=/u03/app/oracle/fast_recovery_area/ORCL/archivelog/2017_11_
29/o1_mf_1_122_f1xr4mvc_.arc RECID=122 STAMP=961345811
…
validation succeeded for archived log
archived log file
name=/u03/app/oracle/fast_recovery_area/ORCL/archivelog/2017_1
1_30/o1_mf_1_178_f2093xs8_.arc RECID=178 STAMP=961428733
validation succeeded for archived log
archived log file
name=/u03/app/oracle/fast_recovery_area/ORCL/archivelog/2017_11_
30/o1_mf_1_179_f209rf05_.arc RECID=179 STAMP=961413125
Crosschecked 59 objects


RMAN>
```

5. In *Session1*, a data file is unintentionally removed from `ORCL_PDB1`.
   a. Remove the system data file of `ORCL_PDB1`.

```
SQL> CONNECT sys@ORCL_PDB1 AS SYSDBA
Enter password: password
Connected.
SQL> SELECT name FROM v$datafile;

NAME
--------------------------------------------------------------------------------
/u02/app/oracle/oradata/ORCL/orcl_pdb1/ORCL/5F35FE374FC20D36E053
0241C40AA7AB/datafile/o1_mf_system_f20bbbnk_.dbf

/u02/app/oracle/oradata/ORCL/orcl_pdb1/ORCL/5F35FE374FC20D36E053
0241C40AA7AB/datafile/o1_mf_sysaux_f20bbbns_.dbf

/u02/app/oracle/oradata/ORCL/orcl_pdb1/ORCL/5F35FE374FC20D36E053
0241C40AA7AB/datafile/o1_mf_undotbs1_f20bbbnz_.dbf


SQL> HOST rm
/u02/app/oracle/oradata/ORCL/orcl_pdb1/ORCL/5F35FE374FC20D36E053
0241C40AA7AB/datafile/o1_mf_system_f20bbbnk_.dbf

SQL>
```

   b. Try to connect to the PDB. You may be able to connect but then get an error, or maybe not. In the latter case, you would get the "`You are no longer connected to ORACLE.`" error message.

```
SQL> CONNECT system@ORCL_PDB1
Enter password: password
ERROR:
ORA-02002: error while writing to audit trail
ORA-01116: error in opening database file 112
ORA-01110: data file 112:
'/u02/app/oracle/oradata/ORCL/orcl_pdb1/ORCL/5F35FE374FC20D36E05
3024
1C40AA7AB/datafile/o1_mf_system_f20bbbnk_.dbf'
ORA-27041: unable to open file
Linux-x86_64 Error: 2: No such file or directory
Additional information: 3

Error accessing package DBMS_APPLICATION_INFO
Connected.
SQL>
```

c. Close ORCL_PDB1 if it is not already closed.

```
SQL> CONNECT / AS SYSDBA
Connected.
SQL> SHOW PDBS

    CON_ID CON_NAME                       OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         2 PDB$SEED                       READ ONLY  NO
         4 PDB1                           MOUNTED
         5 ORCL_PDB1                      READ WRITE NO
SQL> ALTER PLUGGABLE DATABASE orcl_pdb1 CLOSE;

Pluggable database altered.

SQL> SHOW PDBS

    CON_ID CON_NAME                       OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         2 PDB$SEED                       READ ONLY  NO
         4 PDB1                           MOUNTED
         5 ORCL_PDB1                      MOUNTED
SQL>
```

6. Recover the PDB from preplugin backups.
   a. To recover the situation, because there is no new backup completed after the plug-in operation, you use the preplugin backups. The data files are restored from backups taken before the PDB was plugged in.
   In *Session2*, restore and recover ORCL_PDB1 from the preplugin backups. Because you are working on a cloud VM, encryption is set and, therefore, RMAN requires decryption from backups.

```
RMAN> SET DECRYPTION IDENTIFIED BY CDB_keystore_password;

executing command: SET decryption

RMAN>
```

   b. Run a normal recovery after preplugin recovery in order to synchronize the PDB with the CDB. If the channel used is different, ignore this potential discrepancy.

```
RMAN> RUN
       { RESTORE PLUGGABLE DATABASE orcl_pdb1 FROM PREPLUGIN;
         RECOVER PLUGGABLE DATABASE orcl_pdb1 FROM PREPLUGIN;
       }
 2> 3> 4>
```

```
Starting restore at 30-NOV-17
using channel ORA_SBT_TAPE_1
using channel ORA_SBT_TAPE_2
using channel ORA_SBT_TAPE_3
using channel ORA_SBT_TAPE_4
using channel ORA_SBT_TAPE_5
using channel ORA_DISK_1

channel ORA_DISK_1: starting datafile backup set restore
channel ORA_DISK_1: specifying datafile(s) to restore from
backup set
channel ORA_DISK_1: restoring datafile 00112 to
/u02/app/oracle/oradata/ORCL/orcl_pdb1/ORCL/5F35FE374FC20D36E053
0241C40AA7AB/datafile/o1_mf_system_f20bbbnk_.dbf
channel ORA_DISK_1: restoring datafile 00113 to
/u02/app/oracle/oradata/ORCL/orcl_pdb1/ORCL/5F35FE374FC20D36E053
0241C40AA7AB/datafile/o1_mf_sysaux_f20bbbns_.dbf
channel ORA_DISK_1: restoring datafile 00114 to
/u02/app/oracle/oradata/ORCL/orcl_pdb1/ORCL/5F35FE374FC20D36E053
0241C40AA7AB/datafile/o1_mf_undotbs1_f20bbbnz_.dbf
channel ORA_DISK_1: reading from backup piece
/u03/app/oracle/fast_recovery_area/ORCL/5F35EC627C3606EEE0530241
C40A1875/backupset/2017_11_30/o1_mf_nnndf_TAG20171130T153157_f20
93ghl_.bkp
channel ORA_DISK_1: piece
handle=/u03/app/oracle/fast_recovery_area/ORCL/5F35EC627C3606EEE
0530241C40A1875/backupset/2017_11_30/o1_mf_nnndf_TAG20171130T153
157_f2093ghl_.bkp tag=TAG20171130T153157
channel ORA_DISK_1: restored backup piece 1
channel ORA_DISK_1: restore complete, elapsed time: 00:00:15
Finished restore at 30-NOV-17

Starting recover at 30-NOV-17
using channel ORA_SBT_TAPE_1
using channel ORA_SBT_TAPE_2
using channel ORA_SBT_TAPE_3
using channel ORA_SBT_TAPE_4
using channel ORA_SBT_TAPE_5
using channel ORA_DISK_1

starting media recovery

archived log for thread 1 with sequence 178 is already on disk
as file
/u03/app/oracle/fast_recovery_area/ORCL/archivelog/2017_11_30/o1
_mf_1_178_f2093xs8_.arc
```

```
archived log for thread 1 with sequence 179 is already on disk
as file
/u03/app/oracle/fast_recovery_area/ORCL/archivelog/2017_11_30/o1
_mf_1_179_f209rf05_.arc
media recovery complete, elapsed time: 00:00:00
Finished recover at 30-NOV-17


RMAN>
```

c. In *Session1*, open `ORCL_PDB1`.

```
SQL> ALTER PLUGGABLE DATABASE orcl_pdb1 OPEN;
ALTER PLUGGABLE DATABASE ORCL_PDB1 open
*
ERROR at line 1:
ORA-01122: database file 114 failed verification check
ORA-01110: data file 114:
'/u02/app/oracle/oradata/ORCL/orcl_pdb1/ORCL/5F35FE374FC20D36E05
3024
1C40AA7AB/datafile/o1_mf_undotbs1_f20bbbnz_.dbf'
ORA-01204: file number is 111 rather than 114 - wrong file



SQL>
```

d. Because the PDB is in LOCAL UNDO mode, you have to recover the UNDO tablespace. In *Session2*, perform the recovery operation for the PDB.

```
RMAN> RECOVER PLUGGABLE DATABASE orcl_pdb1;

Starting recover at 30-NOV-17
using channel ORA_SBT_TAPE_1
using channel ORA_SBT_TAPE_2
using channel ORA_SBT_TAPE_3
using channel ORA_SBT_TAPE_4
using channel ORA_SBT_TAPE_5
using channel ORA_DISK_1

starting media recovery
media recovery complete, elapsed time: 00:00:01

Finished recover at 30-NOV-17

RMAN> EXIT
$
```

e. Back in *Session1*, open `ORCL_PDB1`.

```
SQL> ALTER PLUGGABLE DATABASE orcl_pdb1 OPEN;

Pluggable database altered.

SQL>
```

f. Verify the content of the `HR.DEPARTMENTS` table in `ORCL_PDB1`.

```
SQL> CONNECT hr@ORCL_PDB1
Enter password: password
Connected.
SQL> SELECT * FROM hr.departments;

DEPARTMENT_ID DEPARTMENT_NAME      MANAGER_ID LOCATION_ID
------------- -------------------- ---------- -----------
           10 Administration              200        1700
           20 Marketing                   201        1800
           30 Purchasing                  114        1700
           40 Human Resources             203        2400
           50 Shipping                    121        1500
           60 IT                          103        1400
           70 Public Relations            204        2700
           80 Sales                       145        2500
           90 Executive                   100        1700
          100 Finance                     108        1700
          110 Accounting                  205        1700
          120 Treasury                               1700
          130 Corporate Tax                          1700
          140 Control And Credit                     1700
          150 Shareholder Services                   1700
          160 Benefits                               1700
          170 Manufacturing                          1700
          180 Construction                           1700
          190 Contracting                            1700
          200 Operations                             1700
          210 IT Support                             1700
          220 NOC                                    1700
          230 IT Helpdesk                            1700
          240 Government Sales                       1700
          250 Retail Sales                           1700
          260 Recruiting                             1700
          270 Payroll                                1700

27 rows selected.
```

```
SQL> EXIT
$
```

7. Execute the `$HOME/labs/RMAN/cleanup_preplugin_PDBs.sh` shell script to clean up preplugin backups and drop the PDBs.

```
$ $HOME/labs/RMAN/cleanup_preplugin_PDBs.sh
…
$
```

Practice 4-3: Duplicating a PDB into an Existing CDB

Overview

In this practice, you clone a PDB into an existing open read/write CDB by using the RMAN `DUPLICATE` command without the use of a fresh auxiliary instance.

Tasks

1. You use the Oracle By Example “Duplicating Active PDBs” to see how to proceed. Launch a browser and click the bookmark from the Bookmarks toolbar of the browser. The URL is `file:////home/oracle/labs/OBE/duplicating_active_pdbs/duplicating_active_pdbs.html`

Practice 4-4: Duplicating an On-Premises CDB for Cloud

Overview

In this practice, you duplicate an on-premises CDB with encrypted tablespaces to be compatible on cloud. Duplication as encrypted or decrypted aids with the transition from on-premises databases to cloud and vice versa.

Tasks

1. You use the Oracle By Example “Duplicating CDB as Encrypted” to see how to proceed. Launch a browser and click the bookmark from the Bookmarks toolbar of the browser. The URL is
   `file:////home/oracle/labs/OBE/duplicating_cdb_as_encrypted/duplicating_cdb_as_encrypted.html`

Practices for Lesson 5:
UsingGeneral Database Enhancements

Practices for Lesson 5: Overview

Overview

In these practices, you use private temporary tables, use Data Pump import with a new option to load corrupted files, convert partitioned tables online, and use batched DDL.

Practice 5-1: Managing Private Temporary Tables

Overview

In this practice, you create and use private temporary tables in your session.

Tasks

1. Before starting the practice, execute the `$HOME/labs/admin/glogin_5.sh` shell script. It sets formatting for all columns selected in queries.

```
$ $HOME/labs/admin/glogin_5.sh
…
$
```

2. Re-create `PDB1` for the practice by executing the `$HOME/labs/DB/create_PDB1_HR.sh` shell script. The shell script also creates the `HR` schema in `PDB1`.

```
$ $HOME/labs/DB/create_PDB1_HR.sh
…
$
```

3. Create a private temporary table (PTT).
   a. Log in to `PDB1` and start a session as `HR`. This is *Session1*.

```
$ sqlplus hr@PDB1

Enter password: password

SQL>
```

   b. Create a simple table with some data values. The `TEST` table is available only for transactions committed besides the PTT.

```
SQL> CREATE TABLE test (x NUMBER, y VARCHAR2(10));

Table created.

SQL> INSERT INTO test VALUES (1,'A');

1 row created.

SQL> COMMIT;

Commit complete.

SQL>
```

c. Create a PTT.

```
SQL> CREATE PRIVATE TEMPORARY TABLE mine
            (x NUMBER, y VARCHAR2(10));
CREATE PRIVATE TEMPORARY TABLE mine
*
ERROR at line 1:
ORA-00903: invalid table name

SQL>
```

Q/ Why does the attempt fail?

A/ Check the prefix defined by default.

```
SQL> CONNECT / AS SYSDBA
Connected.
SQL> SHOW PARAMETER PRIVATE_TEMP_TABLE_PREFIX

NAME                                 TYPE         VALUE

------------------------------------ ------------ -----------------------------------
private_temp_table_prefix            string       ORA$PTT_

SQL>
```

d. Create a PTT with the appropriate prefix.

```
SQL> CONNECT hr@PDB1
Enter password: password
Connected.
SQL>
SQL> CREATE PRIVATE TEMPORARY TABLE ora$ptt_mine
            (x NUMBER, y VARCHAR2(10));
 2
Table created.

SQL>
```

e. Insert rows in the PTT.

```
SQL> INSERT INTO ora$ptt_mine VALUES (1,'Work1');

1 row created.

SQL>
```

f. Display data from the PTT.

```
SQL> SELECT * FROM ora$ptt_mine;

         X Y
---------- ----------
         1 Work1

SQL>
```

4. Identify PTTs between sessions.

a. In *Session1*, describe the `ORA$PTT_mine` PTT.

```
SQL> DESC hr.ORA$PTT_mine
 Name                          Null?    Type
 ----------------------------------------- -------- ----------------------------

 X                                       NUMBER
 Y                                       VARCHAR2(10)

SQL>
```

b. Find all information related to the PTT.

```
SQL> SELECT sid, serial#, table_name, tablespace_name,
            duration
     FROM user_private_temp_tables;
  2    3
 SID SERIAL# TABLE_NAME    TABLESPACE_NAME DURATION
---------- ---------- ------------------------------ ---------- ------------------------
 271    58468 ORA$PTT_MINE  TEMP            TRANSACTION

SQL>
```

Observe that the PTT is of `TRANSACTION` type. This is the default duration type. This means that the PTT is automatically dropped at the end of the transaction in which the PTT has been created.

c. Log in to `PDB1` in another terminal window and start a session as `HR`. This is *Session2*. Verify that the PTT created by *Session1* is not visible to *Session2*.

```
$ sqlplus hr@PDB1

Enter password: password

SQL> DESC hr.ORA$PTT_mine
ERROR:
```

```
ORA-04043: object hr.ORA$PTT_mine does not exist

SQL> EXIT
$
```

5. Use PTTs.
   a. In *Session1*, insert a row into `TEST` and roll back the transaction.

```
SQL> INSERT INTO test VALUES (2,'B');

1 row created.

SQL> ROLLBACK;

Rollback complete.

SQL>
```

Q/ What happens to the PTT?

```
SQL> SELECT sid, serial#, table_name, tablespace_name,
            duration
     FROM user_private_temp_tables;
 2   3
no rows selected

SQL>
```

A/ Observe that the `TRANSACTION` duration type PTT is automatically dropped with the `ROLLBACK` statement. A `COMMIT` would have also dropped the PTT.

   b. Create a new PTT of `SESSION` duration type that will last until your session ends.

```
SQL> CREATE PRIVATE TEMPORARY TABLE ora$ptt_mine2
            (x NUMBER, y VARCHAR2(10))
            ON COMMIT PRESERVE DEFINITION;
  2   3
Table created.

SQL>
```

   c. Find all information related to the PTT.

```
SQL> SELECT sid, serial#, table_name, tablespace_name,
            duration
     FROM user_private_temp_tables;
  2   3
```

```
 SID SERIAL# TABLE_NAME     TABLESPACE_NAME DURATION
---- ------- ------------- --------------- ---------
 271   58468 ORA$PTT_MINE2 TEMP            SESSION

SQL>
```

d. Insert rows in the PTT.

```
SQL> INSERT INTO ora$ptt_mine2 VALUES (2,'Work2');

1 row created.

SQL>
```

e. Display data from the PTT.

```
SQL> SELECT * FROM ora$ptt_mine2;

         X Y
---------- ----------
         2 Work2

SQL>
```

f. Insert a row into `TEST` and commit the transaction.

```
SQL> INSERT INTO test VALUES (3,'C');

1 row created.

SQL> COMMIT;

Commit complete.

SQL>
```

Q/ What happens to the PTT?

```
SQL> SELECT sid, serial#, table_name, tablespace_name,
            duration
     FROM user_private_temp_tables;
  2    3
 SID SERIAL# TABLE_NAME     TABLESPACE_NAME DURATION
---- ------- ------------- --------------- ---------
 271   58468 ORA$PTT_MINE2 TEMP            SESSION

SQL>
```

A/ The PTT still exists because it will be dropped at the end of the session.

g. Display data from the PTT.

```
SQL> SELECT * FROM ora$ptt_mine2;

         X Y
---------- ----------
         2 Work2

SQL>
```

h. Quit *Session1.*

```
SQL> EXIT
$
```

i. Verify that the PTT is automatically dropped.

```
$ sqlplus hr@PDB1

Enter password: password

SQL> SELECT sid, serial#, table_name, tablespace_name,
            duration
     FROM user_private_temp_tables;
 2   3
no rows selected

SQL>
```

6. Use PTTs with save points.

a. Create the `EMP` table with `EMPLOYEES` rows.

```
SQL> DROP TABLE emp PURGE;
DROP TABLE emp PURGE
           *
ERROR at line 1:
ORA-00942: table or view does not exist

SQL> CREATE TABLE emp AS SELECT * FROM employees;

Table created.

SQL>
```

b. Create a PTT with the 107 rows of EMP.

```
SQL> CREATE PRIVATE TEMPORARY TABLE ora$ptt_emp
            AS SELECT * FROM emp;
 2
Table created.

SQL>
```

c. Insert another set of rows.

```
SQL> INSERT INTO ora$ptt_emp SELECT * FROM emp;

107 rows created.

SQL>
```

d. Create a first save point.

```
SQL> SAVEPOINT point1;

Savepoint created.

SQL>
```

e. Count the number of rows in the PTT.

```
SQL> SELECT count(*) FROM ora$ptt_emp;

  COUNT(*)
----------
       214

SQL>
```

f. Find all information related to the PTT.

```
SQL> SELECT sid, serial#, table_name, tablespace_name,
            duration, num_rows
     FROM user_private_temp_tables;
 2   3
 SID SERIAL# TABLE_NAME    TABLESPACE_NAME DURATION
NUM_ROWS
---------- ---------- ------------------------------ ------------------------------ -------------------------- ----------
-
 271   28457 ORA$PTT_EMP   TEMP            TRANSACTION
107

SQL>
```

The number of rows corresponds to the number of rows in the PTT at the time of the PTT's creation.

g. Insert another set of rows.

```
SQL> INSERT INTO ora$ptt_emp SELECT * FROM emp;

107 rows created.

SQL>
```

h. Count the number of rows in the PTT.

```
SQL> SELECT count(*) FROM ora$ptt_emp;

  COUNT(*)
----------
       321

SQL>
```

i. Create the second save point.

```
SQL> SAVEPOINT point2;

Savepoint created.

SQL>
```

j. Insert another set of rows.

```
SQL> INSERT INTO ora$ptt_emp SELECT * FROM emp;

107 rows created.

SQL>
```

k. Count the number of rows in the PTT.

```
SQL> SELECT count(*) FROM ora$ptt_emp;

  COUNT(*)
----------
       428

SQL>
```

l. You discover that you should keep only the first set of rows inserted in step 3. Roll back to `point1`.

```
SQL> ROLLBACK TO point1;
```

```
Rollback complete.

SQL>
```

Q/ What would have happened if you had used the `ROLLBACK` command?

A/ The PTT would have been dropped.

m. Count the number of rows in the PTT.

```
SQL> SELECT count(*) FROM ora$ptt_emp;

  COUNT(*)
----------
       214

SQL>
```

n. You can insert temporary rows into `EMP`.

```
SQL> INSERT INTO emp SELECT * FROM ORA$PTT_emp;

214 rows created.

SQL>
```

o. Commit and count the number of rows in `EMP`.

```
SQL> COMMIT;

Commit complete.

SQL> SELECT count(*) FROM emp;

  COUNT(*)
----------
       321

SQL>
```

Q/ Does the PTT exist?

```
SQL> SELECT sid, serial#, table_name, tablespace_name,
            duration, num_rows
     FROM user_private_temp_tables;
 2   3
```

```
no rows selected

SQL> EXIT
$
```

A/ No. The `COMMIT` statement has dropped the `TRANSACTION` duration PTT.

Practice 5-2: Using Data Pump Import with CONTINUE_LOAD_ON_FORMAT_ERROR

Overview

In this practice, you use the Data Pump import option to load data while detecting a data stream error. Oracle Database 19c Data Pump can resume loading at the next granule boundary instead of aborting a Data Pump import operation.

Tasks

1. Detect rows with errors and continue importing.
   a. Before importing data with Data Pump import, execute the `$HOME/labs/DB/SCOTT.sh` shell script. The script creates the `SCOTT` schema in `PDB1`, then creates a logical directory where the dump file resides, and grants the appropriate privilege on the directory to `SCOTT`. This is completed in a terminal window that you will call *Session_Sqlplus.*

```
$ $HOME/labs/DB/SCOTT.sh
…
$
```

   b. In another terminal window that you call *Session_Import*, import as `SYSTEM` into `PDB1` corrupted data by using a corrupted dump file that loads rows into the `SCOTT.L_FACT_PAGE` table.

```
$ impdp system@PDB1 FULL=y DUMPFILE=exp_corrupt1.dmp DIRECTORY=dp_dir
TABLE_EXISTS_ACTION=replace REMAP_TABLESPACE=TBS_6:SYSTEM
Password: password
…
Master table "SYSTEM"."SYS_IMPORT_FULL_01" successfully
loaded/unloaded
import done in AL32UTF8 character set and AL16UTF16 NCHAR
character set
export done in WE8DEC character set and AL16UTF16 NCHAR
character set
Warning: possible data loss in character set conversions
Starting "SYSTEM"."SYS_IMPORT_FULL_01": system/********@PDB1
FULL=y dumpfile=exp_corrupt1.dmp DIRECTORY=dp_dir
TABLE_EXISTS_ACTION=replace REMAP_TABLESPACE=TBS_6:SYSTEM
Processing object type TABLE_EXPORT/TABLE/TABLE
Processing object type TABLE_EXPORT/TABLE/TABLE_DATA
ORA-02374: conversion error loading table "SCOTT"."L_FACT_PAGE"
ORA-26093: input data column size (63) exceeds the maximum input
size (40)
ORA-02372: data for row: REFER_PAGE_NAME :
0X'5B5D0B4C6974657261747572652F02C102018002C102018009'
```

```
ORA-31693: Table data object "SCOTT"."L_FACT_PAGE" failed to
load/unload and is being skipped due to error:
ORA-02354: error in exporting/importing data
ORA-39840: A data load operation has detected data stream format
error 3000.

Job "SYSTEM"."SYS_IMPORT_FULL_01" completed with 1 error(s) at
Fri Dec 1 13:34:35 2017 elapsed 0 00:00:05

$
```

c. In *Session_Sqlplus*, check if the `SCOTT.L_FACT_PAGE` table is partially loaded.

```
$ sqlplus scott@PDB1

Enter password: password

SQL> SELECT * FROM scott.l_fact_page;

no rows selected

SQL>
```

d. In *Session_Import*, re-import the data into the `SCOTT.L_FACT_PAGE` table, skipping the corrupted records. Use the `CONTINUE_LOAD_ON_FORMAT_ERROR` value of the `DATA_OPTIONS` parameter.

```
$ impdp system@PDB1 FULL=y DUMPFILE=exp_corrupt1.dmp DIRECTORY=dp_dir
TABLE_EXISTS_ACTION=replace REMAP_TABLESPACE=TBS_6:SYSTEM
DATA_OPTIONS=CONTINUE_LOAD_ON_FORMAT_ERROR
Password: password
…
Master table "SYSTEM"."SYS_IMPORT_FULL_01" successfully
loaded/unloaded
import done in AL32UTF8 character set and AL16UTF16 NCHAR
character set
export done in WE8DEC character set and AL16UTF16 NCHAR
character set
Warning: possible data loss in character set conversions
Starting "SYSTEM"."SYS_IMPORT_FULL_01": system/********@PDB1
FULL=y DUMPFILE=exp_corrupt1.dmp DIRECTORY=dp_dir
TABLE_EXISTS_ACTION=replace REMAP_TABLESPACE=TBS_6:SYSTEM
DATA_OPTIONS=CONTINUE_LOAD_ON_FORMAT_ERROR
Processing object type TABLE_EXPORT/TABLE/TABLE
Processing object type TABLE_EXPORT/TABLE/TABLE_DATA
ORA-02374: conversion error loading table "SCOTT"."L_FACT_PAGE"
```

```
ORA-26093: input data column size (63) exceeds the maximum input
size (40)
ORA-02372: data for row: REFER_PAGE_NAME :
0X'5B5D0B4C6974657261747572652F02C102018002C102018009'


ORA-02374: conversion error loading table "SCOTT"."L_FACT_PAGE"
ORA-39840: A data load operation has detected data stream format
error 3000.
ORA-02372: data for row: REFER_PAGE_NAME :
0X'5B5D0B4C6974657261747572652F02C102018002C102018009'


ORA-39840: A data load operation has detected data stream format
error 3000.
. . imported "SCOTT"."L_FACT_PAGE"                       18.06
MB   91538 out of 200000 rows
Job "SYSTEM"."SYS_IMPORT_FULL_01" successfully completed at Fri
Dec 1 13:38:58 2017 elapsed 0 00:00:03
$
```

Before Oracle Database 19c, a detected corruption would cause the table load to fail. With this new parameter, the Data Pump skips over the bad row and then starts loading at a known good spot in the data stream. This feature helps you in cases when the Data Pump dump file is corrupt and you have no way to recreate the dump file with the correct data. This allows you to recover at least some of the data.

e. In *Session_Sqlplus*, check that the `SCOTT.L_FACT_PAGE` table is partially loaded with 91538 rows as the import log mentions.

```
SQL> SELECT COUNT(*) FROM scott.l_fact_page;

  COUNT(*)
----------
     91538

SQL>
```

f. Drop the table before completing another import.

```
SQL> DROP TABLE scott.l_fact_page;

Table dropped.

SQL>
```

g. In *Session_Import*, import as `SYSTEM` into `PDB1` by using another corrupted dump file that loads rows into the `SCOTT.L_FACT_PAGE` table.

```
$ impdp system@PDB1 FULL=y DUMPFILE=exp_corrupt2.dmp DIRECTORY=dp_dir
TABLE_EXISTS_ACTION=replace REMAP_TABLESPACE=TBS_6:SYSTEM
Password: password
…
ORA-39002: invalid operation
ORA-31694: master table "SYSTEM"."SYS_IMPORT_FULL_01" failed to
load/unload
ORA-02354: error in exporting/importing data
ORA-02357: header in file /home/oracle/labs/DB/exp_corrupt2.dmp
may not contain correct information


$
```

h. In *Session_Sqlplus*, check if the `SCOTT.L_FACT_PAGE` table is not loaded at all, or created.

```
SQL> SELECT COUNT(*) FROM scott.l_fact_page;
SELECT COUNT(*) FROM scott.l_fact_page
                        *
ERROR at line 1:
ORA-00942: table or view does not exist



SQL>
```

Q/ Why does the import fail to load rows?

A/ The error does not rely on corruption in rows but in the header of the dump file.

Q/ Will the `CONTINUE_LOAD_ON_FORMAT_ERROR` value of the `DATA_OPTIONS` parameter be sufficient to load data?

i. In *Session_Import*, re-import the data into the `SCOTT.L_FACT_PAGE` table, skipping the corrupted records. Use the `CONTINUE_LOAD_ON_FORMAT_ERROR` value of the `DATA_OPTIONS` parameter.

```
$ impdp system@PDB1 FULL=y DUMPFILE=exp_corrupt2.dmp DIRECTORY=dp_dir
TABLE_EXISTS_ACTION=replace REMAP_TABLESPACE=TBS_6:SYSTEM
DATA_OPTIONS=CONTINUE_LOAD_ON_FORMAT_ERROR
Password: password
…
ORA-39002: invalid operation
```

```
ORA-31694: master table "SYSTEM"."SYS_IMPORT_FULL_01" failed to
load/unload
ORA-02354: error in exporting/importing data
ORA-02357: header in file /home/oracle/labs/DB/exp_corrupt2.dmp
may not contain correct information


$
```

A/ The CONTINUE_LOAD_ON_FORMAT_ERROR value of the DATA_OPTIONS parameter is not sufficient to load data because the error resides in the header of the dump file, which is not readable. A corrupt header for a data pump dump file will prevent Data Pump from doing anything. The CONTINUE_LOAD_ON_FORMAT_ERROR parameter does not change the behavior. This parameter only applies when corruptions are found in the data stream for table data.

j. In *Session_Sqlplus*, check that the SCOTT.L_FACT_PAGE table is not created.

```
SQL> SELECT COUNT(*) FROM scott.l_fact_page;
SELECT COUNT(*) FROM scott.l_fact_page
                        *
ERROR at line 1:
ORA-00942: table or view does not exist
```

2. Import using the DBMS_DATAPUMP API.

a. Log in as SYSTEM in PDB1.

```
SQL> CONNECT system@PDB1
Enter password: password
Connected.
SQL> SET SERVEROUTPUT ON
SQL>
```

b. Use API without the new DBMS_DATAPUMP.KU$_DATAOPT_CONT_LD_ON_FMT_ERR argument.

```
SQL> DECLARE
       dp_handle NUMBER;
       job_status VARCHAR2(30);
       dir_name varchar2(30);
     BEGIN
       dir_name := 'DP_DIR';
       dp_handle := dbms_datapump.open (operation => 'IMPORT',
job_mode => 'FULL');
       dbms_datapump.add_file (handle => dp_handle, filename  =>
'exp_corrupt1.dmp', directory => dir_name, filetype  => 1);
```

```
      dbms_datapump.add_file (handle => dp_handle, filename =>
'EXP.LOG', directory => dir_name, filetype  => 3);
     dbms_datapump.set_parallel (dp_handle, 1);
     dbms_datapump.set_parameter (handle => dp_handle, name =>
'TABLE_EXISTS_ACTION', value => 'REPLACE');
     dbms_datapump.metadata_remap (handle => dp_handle, name =>
'REMAP_TABLESPACE', old_value => 'TBS_6', value => 'SYSTEM');
     dbms_datapump.start_job (dp_handle);
     dbms_datapump.wait_for_job (handle => dp_handle, job_state
=> job_status);
     dbms_output.put_line ('DataPump Export - '|| to_char
(sysdate, 'DD/MM/YYYY HH24:MI:SS')|| ' Status '||job_status);
  dbms_datapump.detach (handle => dp_handle);
END;
/
DataPump Export - 01/12/2017 14:12:36 Status COMPLETED

PL/SQL procedure successfully completed.

SQL>
```

c. Check if the `SCOTT.L_FACT_PAGE` table is partially loaded.

```
SQL> SELECT COUNT(*) FROM scott.l_fact_page;

  COUNT(*)
----------
         0

SQL>
```

d. Reuse API with the new `DBMS_DATAPUMP.KU$_DATAOPT_CONT_LD_ON_FMT_ERR` argument.

```
SQL> DECLARE
  dp_handle NUMBER;
  job_status VARCHAR2(30);
  dir_name varchar2(30);
BEGIN
  dir_name := 'DP_DIR';
  dp_handle := dbms_datapump.open (operation => 'IMPORT',
job_mode => 'FULL');
  dbms_datapump.add_file (handle => dp_handle, filename  =>
'exp_corrupt1.dmp', directory => dir_name, filetype =>
DBMS_DATAPUMP.KU$_FILE_TYPE_DUMP_FILE);
  dbms_datapump.add_file (handle => dp_handle, filename  =>
'EXP.LOG', directory => dir_name, filetype =>
DBMS_DATAPUMP.KU$_FILE_TYPE_LOG_FILE);
```

```
  dbms_datapump.set_parallel (dp_handle, 1);
  dbms_datapump.set_parameter (handle => dp_handle, name =>
'TABLE_EXISTS_ACTION', value => 'REPLACE');
  dbms_datapump.metadata_remap (handle => dp_handle, name =>
'REMAP_TABLESPACE', old_value => 'TBS_6', value => 'SYSTEM');
  dbms_datapump.set_parameter (handle => dp_handle, name   =>
'DATA_OPTIONS', value =>
DBMS_DATAPUMP.KU$_DATAOPT_CONT_LD_ON_FMT_ERR);
  dbms_datapump.start_job (dp_handle);
  dbms_datapump.wait_for_job (handle => dp_handle, job_state =>
job_status);
  dbms_output.put_line ('DataPump Export - '|| to_char (sysdate,
'DD/MM/YYYY HH24:MI:SS')|| ' Status '||job_status);
  dbms_datapump.detach (handle => dp_handle);
end;
/
DataPump Export - 01/12/2017 14:14:53 Status COMPLETED

PL/SQL procedure successfully completed.

SQL>
```

e. Check that the SCOTT.L_FACT_PAGE table is partially loaded.

```
SQL> SELECT COUNT(*) FROM scott.l_fact_page;

  COUNT(*)
----------
     91538

SQL> EXIT
$
```

3. Execute the $HOME/labs/DB/cleanup_DP.sh shell script to drop the SCOTT.L_FACT_PAGE table and the dump file.

```
$ $HOME/labs/DB/cleanup_DP.sh
…
$
```

Practice 5-3: Converting HASH Partitioned Tables to RANGE Partitioned Tables Online

Overview

In this practice, you convert a HASH partitioned table to a RANGE partitioned table online.

Tasks

1. Before starting the practice, execute the $HOME/labs/DB/part_hash.sh shell script. The SQL script creates the SH and HR users, and grants the users the DBA role and READ and WRITE privileges on the DP_PDB_ORCL logical directory.

```
$ $HOME/labs/DB/part_hash.sh
…
$
```

2. Log in to the PDB1 PDB (*Session1*) and create an HR.EMP HASH partitioned table.

```
$ sqlplus system@PDB1

Enter password: password

SQL> DROP TABLE hr.emp CASCADE CONSTRAINTS PURGE;
DROP TABLE hr.emp CASCADE CONSTRAINTS PURGE
              *
ERROR at line 1:
ORA-00942: table or view does not exist


SQL> CREATE TABLE hr.emp (empno NUMBER NOT NULL,
                          ename VARCHAR2(10), job VARCHAR2(9),
                          mgr NUMBER(4), sal NUMBER(7,2))
       PARTITION BY HASH (sal) PARTITIONS 4;
  2    3    4
Table created.

SQL>
```

3. Insert rows into the HR.EMP table. Execute the $HOME/labs/DB/insert1.sql SQL script.

```
SQL> @$HOME/labs/DB/insert1.sql
…
SQL>
```

4. Create three indexes on the partitioned table:
   - Local `I1_SAL` index on the `SAL` column
   - Global unique `I2_EMPNO` index on the `EMPNO` column
   - Global `I3_MGR` index on the `MGR` column

```
SQL> CREATE INDEX hr.i1_sal ON hr.emp (sal) LOCAL;

Index created.

SQL> CREATE UNIQUE INDEX hr.i2_empno ON hr.emp (empno);

Index created.

SQL> CREATE INDEX hr.i3_mgr ON hr.emp (mgr);

Index created.

SQL>
```

5. Display the partitioning method of the table.

```
SQL> SELECT owner, table_name, partitioning_type AS type,
            autolist, interval, interval_subpartition,
            autolist_subpartition
     FROMdba_part_tables  WHERE  owner  =  'HR';
 2    3    4
OWNER  TABLE_NAME TYPE  AUTOLIST INTERVAL         INTERVAL_SUB
-------------- ---------------------- ------------ ------------------- ---------------------------------- ------------------------------
AUTOLIST_SUB
------------------------------
HR     EMP         HASH  NO
NO



SQL>
```

6. Display the partitions of the table.

```
SQL> SELECT composite, partition_name, high_value
     FROM   dba_tab_partitions
     WHERE  table_name = 'EMP' and table_owner='HR';
 2    3
COM PARTITION_NAME HIGH_VALUE
----- ------------------------------ --------------------------------
NO SYS_P228
NO  SYS_P229
```

```
NO SYS_P230
NO  SYS_P231

SQL>
```

7. Display the list of indexes and whether they are partitioned or not.

```
SQL> SELECT index_name, partitioned
     FROM   dba_indexes
     WHERE  index_name LIKE 'I%' and owner='HR';
 2    3
INDEX_NAME PAR
---------- -------
I1_SAL     YES
I3_MGR     NO
I2_EMPNO   NO

SQL>
```

8. Display the type of partitioning of the partitioned indexes.

```
SQL> SELECT index_name, locality, partitioning_type AS type,
            autolist, interval, interval_subpartition,
            autolist_subpartition
     FROM   dba_part_indexes
     WHERE  owner='HR';
 2    3    4    5
INDEX_NAME LOCALI TYPE  AUTOLIST INTERVAL         INTERVAL_SUB
---------- ------ ----- -------- ---------------- ------------
AUTOLIST_SUB
------------
I1_SAL     LOCAL  HASH  NO
NO


SQL>
```

9. Display the partitions of the index.

```
SQL> SELECT index_name,partition_name, high_value
     FROM   dba_ind_partitions
     WHERE  index_name IN ('I1_SAL','I2_EMPNO','I3_MGR')
     AND    index_owner='HR';
 2    3    4
INDEX_NAME PARTITION_NAME HIGH_VALUE
---------- -------------- ----------
I1_SAL     SYS_P228
```

```
I1_SAL      SYS_P229
I1_SAL      SYS_P230
I1_SAL      SYS_P231

SQL>
```

10. In another terminal window, log in to PDB1 as HR and update the employees' salary *(Session2).*

```
$ sqlplus hr@PDB1

Enter password: password

SQL> UPDATE hr.emp SET mgr=208 WHERE empno=100;

1 row updated.

SQL>
```

11. In *Session1*, attempt to convert the HASH partitioned table to a RANGE partitioned table.

```
SQL> ALTER TABLE hr.emp MODIFY
       PARTITION BY RANGE (empno) INTERVAL (100)
         (PARTITION p1 VALUES LESS THAN (200),
          PARTITION p2 VALUES LESS THAN (500)
         )
       UPDATE INDEXES
        (hr.i1_sal LOCAL, hr.i2_empno GLOBAL PARTITION BY RANGE
(empno)
        (PARTITION ip1 VALUES LESS THAN (MAXVALUE)));
ALTER TABLE hr.emp MODIFY
              *
ERROR at line 1:
ORA-00054: resource busy and acquire with NOWAIT specified or
timeout expired

SQL>
```

Because the ONLINE keyword is not added to the statement, the operation cannot get the exclusive lock.

12. Re-execute the operation with the ONLINE keyword.

```
SQL> ALTER TABLE hr.emp MODIFY
       PARTITION BY RANGE (empno) INTERVAL (100)
         (PARTITION p1 VALUES LESS THAN (200),
          PARTITION p2 VALUES LESS THAN (500)
         )
```

```
        ONLINE
        UPDATE INDEXES
         (hr.i1_sal LOCAL,
          hr.i2_empno GLOBAL PARTITION BY RANGE (empno)
         (PARTITION ip1 VALUES LESS THAN (MAXVALUE)));
```

The statement waits for the `UPDATE` in *Session2* to complete.

13. In *Session2*, commit the `UPDATE` statement.

```
SQL> COMMIT;

Commit complete.

SQL> EXIT
$
```

14. In *Session1*, the `ALTER TABLE` statement completes.

```
Table altered.

SQL>
```

15. Display the new partitioning method of the table.

```
SQL> SELECT owner, table_name, partitioning_type AS type,
             autolist, interval, interval_subpartition,
             autolist_subpartition
     FROM   dba_part_tables WHERE owner = 'HR';
 2    3    4
OWNER  TABLE_NAME TYPE  AUTOLIST INTERVAL         INTERVAL_SUB
-------------- ------------------------ ------------ -------------------- ---------------------------------------- ----------------------------------------
AUTOLIST_SUB
--------------------------------
HR     EMP        RANGE NO       100
NO


SQL>
```

16. Display the new partitions of the table.

```
SQL> SELECT composite, partition_name, high_value
     FROM   dba_tab_partitions
     WHERE  table_name = 'EMP' and table_owner='HR';
 2    3
COM PARTITION_NAME HIGH_VALUE
```

```
----- ------------------------------ ------------------------------
NO   P1              200
NO   P2              500

SQL>
```

17. Display the list of indexes and whether they are partitioned or not.

```
SQL> SELECT index_name, partitioned
     FROM   dba_indexes
     WHERE  index_name LIKE 'I%' and owner='HR';
 2    3
INDEX_NAME PAR
---------- -------
I1_SAL     YES
I2_EMPNO   YES
I3_MGR     NO

SQL>
```

18. Display the partitions of the indexes.

```
SQL> SELECT index_name,partition_name, high_value
     FROM   dba_ind_partitions
     WHERE  index_name IN ('I1_SAL','I2_EMPNO','I3_MGR')
     AND    index_owner='HR';
 2    3    4
INDEX_NAME PARTITION_NAME HIGH_VALUE
---------- ------------------------------ --------------------------------
I1_SAL     P1             200
I1_SAL     P2             500
I2_EMPNO   IP1            MAXVALUE

SQL>
```

19. Display the type of partitioning of the partitioned indexes.

```
SQL> SELECT index_name, locality, partitioning_type AS type,
            autolist, interval, interval_subpartition,
            autolist_subpartition
     FROM   dba_part_indexes
     WHERE  owner='HR';
 2    3    4    5
INDEX_NAME LOCALI TYPE  AUTOLIST INTERVAL         INTERVAL_SUB
---------- ------ ----- -------- ---------------- ------------
AUTOLIST_SUB
------------
```

```
I2_EMPNO    GLOBAL RANGE NO
NO

I1_SAL      LOCAL  RANGE NO        100
NO


SQL> EXIT
$
```

20. Execute the `$HOME/labs/DB/cleanup_hash.sh` shell script to drop the `HR.EMP` partitioned table and indexes.

```
$ $HOME/labs/DB/cleanup_hash.sh
…
$
```

Practice 5-4: Converting LIST Partitioned Tables to LIST AUTOMATIC Partitioned Tables Online

Overview

In this practice, you convert a LIST partitioned table, partitioned on two keys, to a LIST AUTOMATIC partitioned table, online.

Tasks

1. Create the SH.SALES_BY_REGION_AND_CHANNEL LIST partitioned table. The table is partitioned by LIST on two keys, STATE and CHANNEL.

```
$ sqlplus system@PDB1

Enter password: password

SQL> DROP TABLE sh.sales_by_region_and_channel CASCADE
CONSTRAINTS PURGE;
DROP TABLE sh.sales_by_region_and_channel CASCADE CONSTRAINTS
PURGE
              *
ERROR at line 1:
ORA-00942: table or view does not exist


SQL> CREATE TABLE sh.sales_by_region_and_channel
       (deptno number, deptname varchar2(20),
        quarterly_sales number(10, 2),
        state varchar2(2), channel varchar2(1))
      PARTITION BY LIST (state, channel)
       (PARTITION q1_northwest_direct VALUES (('OR', 'D'),
('WA','D')),
        PARTITION q1_northwest_indirect VALUES (('OR', 'I'),
('WA','I')),
        PARTITION q1_southwest_direct VALUES (('AZ', 'D'),
('UT','D'), ('NM','D')),
        PARTITION q1_ca_direct VALUES ('CA','D'),
        PARTITION rest VALUES (DEFAULT));
 2   3   4   5   6   7   8   9
Table created.

SQL>
```

2. Insert rows into the `SH.SALES_BY_REGION_AND_CHANNEL` table. Execute the `$HOME/labs/DB/insert2.sql` SQL script.

```
SQL> @$HOME/labs/DB/insert2.sql
…
SQL>
```

3. Display the partitioning method of the table.

```
SQL> SELECT owner, table_name, partitioning_type AS type,
            autolist, interval, interval_subpartition,
            autolist_subpartition
     FROMdba_part_tables  WHERE  owner  =  'SH';
 2    3    4
OWNER  TABLE_NAME                    TYPE  AUTOLIST INTERVAL
-------------- ---------------------------------------------------------- ----------- ---------- ------------------------------
-
INTERVAL_SUB AUTOLIST_SUB
------------------------------ ------------------------------
SH     SALES_BY_REGION_AND_CHANNEL LIST  NO
             NO

SQL>
```

4. Display the partitions of the table and the high values in each partition.

```
SQL> SELECT partition_name, high_value
     FROM   dba_tab_partitions
     WHERE  table_owner = 'SH';
 2    3
PARTITION_NAME         HIGH_VALUE
------------------------------------------------ ------------------------------------------------------------------------------------------------
-
Q1_CA_DIRECT           ( 'CA', 'D' )
Q1_NORTHWEST_DIRECT    ( 'OR', 'D' ), ( 'WA', 'D' )
Q1_NORTHWEST_INDIRECT  ( 'OR', 'I' ), ( 'WA', 'I' )
Q1_SOUTHWEST_DIRECT    ( 'AZ', 'D' ), ( 'UT', 'D' ), ( 'NM',
'D' )
REST                   DEFAULT

SQL>
```

5. In *Session2*, log in to `PDB1` as `SH` and increase the quarterly sales.

```
$ sqlplus sh@PDB1

Enter password: password
```

```
SQL> UPDATE sh.sales_by_region_and_channel
     SET quarterly_sales=quarterly_sales*10;
 2
7 rows updated.

SQL>
```

6. In *Session1*, attempt to convert the `LIST` partitioned table on two keys to a `LIST AUTOMATIC` partitioned table on one key.

```
SQL> ALTER TABLE sh.sales_by_region_and_channel MODIFY
        PARTITION BY LIST (state) AUTOMATIC
         (PARTITION northwest VALUES ('OR', 'WA'),
          PARTITION southwest VALUES ('AZ', 'UT', 'NM'),
          PARTITION california VALUES ('CA'),
       PARTITION rest VALUES (DEFAULT))
       ONLINE;
ALTER TABLE sh.sales_by_region_and_channel MODIFY
               *
ERROR at line 1:
ORA-14851: DEFAULT [sub]partition cannot be specified for
AUTOLIST [sub]partitioned objects.

SQL>
```

7. Re-execute the operation without the `DEFAULT` partition.

```
SQL> ALTER TABLE sh.sales_by_region_and_channel MODIFY
        PARTITION BY LIST (state) AUTOMATIC
         (PARTITION northwest VALUES ('OR', 'WA'),
          PARTITION southwest VALUES ('AZ', 'UT', 'NM'),
          PARTITION california VALUES ('CA'))
       ONLINE UPDATE INDEXES;
```

The statement waits for the `UPDATE` in *Session2* to complete.

8. In *Session2*, commit the `UPDATE` statement.

```
SQL> COMMIT;

Commit complete.

SQL> EXIT
$
```

9. In *Session1*, the `ALTER TABLE` statement completes.

```
Table altered.

SQL>
```

10. Display the new partitioning method of the table.

```
SQL> SELECT owner, table_name, partitioning_type AS type,
            autolist, interval, interval_subpartition,
            autolist_subpartition
     FROMdba_part_tables  WHERE  owner  =  'SH';
 2    3    4
OWNER  TABLE_NAME                    TYPE  AUTOLIST INTERVAL
-------------- _------------------------------------------------------ ------------ -------------------- ---------------------------------
-
INTERVAL_SUB AUTOLIST_SUB
------------------------------ ------------------------------
SH     SALES_BY_REGION_AND_CHANNEL LIST  YES
             NO


SQL>
```

11. Display the partitions of the table and the high values in each partition.

```
SQL> SELECT partition_name, high_value FROM dba_tab_partitions
     WHERE  table_name = 'SALES_BY_REGION_AND_CHANNEL';
  2
PARTITION_NAME         HIGH_VALUE
---------------------------------------------- ----------------------------------------------------------------------------------------------------
CALIFORNIA             'CA'
NORTHWEST              'OR', 'WA'
SOUTHWEST              'AZ', 'UT', 'NM'
SYS_P248               'FL'
SYS_P249               'TX'
SYS_P250               ''

6 rows selected.

SQL> EXIT
$
```

12. Drop the `SH.SALES_BY_REGION_AND_CHANNEL` table. Execute the `$HOME/labs/DB/cleanup_list.sh` shell script.

```
$ $HOME/labs/DB/cleanup_list.sh
…
$
```

Practice 5-5: Converting `LIST AUTOMATIC` Partitioned Tables to Subpartitioned Tables Online

Overview

In this practice, you convert a `LIST AUTOMATIC` partitioned table to a `LIST AUTOMATIC` partitioned table with `SUBPARTITIONING`, online.

Tasks

1. Create the `HR.T LIST AUTOMATIC` partitioned table. The table is partitioned by `LIST` on one key, `C1`.

```
$ sqlplus system@PDB1

Enter password: password

SQL> DROP TABLE hr.t CASCADE CONSTRAINTS PURGE;
DROP TABLE hr.t CASCADE CONSTRAINTS PURGE
              *
ERROR at line 1:
ORA-00942: table or view does not exist

SQL> CREATE TABLE hr.t (c1 number, c2 number)
      PARTITION BY LIST (c1) AUTOMATIC
      ( PARTITION p1 values (1),
        PARTITION p2 values (2),
        PARTITION p3 values (3));
 2    3    4    5
Table created.

SQL>
```

2. Insert rows into the `HR.T` table. Execute the `$HOME/labs/DB/insert3.sql` SQL script.

```
SQL> @$HOME/labs/DB/insert3.sql
…
SQL>
```

3. Display the partitioning method of the table.

```
SQL> SELECT owner, table_name, partitioning_type AS type,
            autolist, subpartitioning_type
     FROM   dba_part_tables WHERE table_name = 'T';
 2    3
OWNER  TABLE_NAME TYPE  AUTOLIST SUBPARTIT
-------------- ------------------------ ------------ -------------------- ----------------------
```

```
HR      T           LIST  YES       NONE

SQL>
```

4. Display the partitions of the table and the high values in each partition.

```
SQL> SELECT partition_name, high_value FROM dba_tab_partitions
     WHERE table_name = 'T';
  2
PARTITION_NAME HIGH_VALUE
------------------------------ --------------------------------
P1             1
P2             2
P3             3

SQL>
```

5. In *Session2*, log in to `PDB1` as `HR` and increase the values in the `C2` column.

```
$ sqlplus hr@PDB1

Enter password: password

SQL> UPDATE hr.t SET c2=c2*10;

11 rows updated.

SQL>
```

6. In *Session1*, convert the `LIST AUTOMATIC` partitioned table to a `LIST AUTOMATIC` subpartitioned table.

```
SQL> ALTER TABLE hr.t MODIFY PARTITION BY LIST (c1) AUTOMATIC
       SUBPARTITION BY list (c2)
        SUBPARTITION TEMPLATE
       (  SUBPARTITION sp1 VALUES (1),
          SUBPARTITION sp2 VALUES (2),
          SUBPARTITION sp3 VALUES (3),
          SUBPARTITION sp_unknown VALUES (DEFAULT)
       )
        ( PARTITION p1 VALUES (1),
          PARTITION p2 VALUES (2),
          PARTITION p3 VALUES (3)
        )
      ONLINE;
```

The statement waits for the `UPDATE` in the second session to complete.

7. In *Session2*, commit the `UPDATE` statement.

```
SQL> COMMIT;

Commit complete.

SQL> EXIT
$
```

8. Back in the conversion session, the `ALTER TABLE` statement completes.

```
Table altered.

SQL>
```

9. Insert rows into `HR.T`. Execute the `$HOME/labs/DB/insert4.sql` SQL script.

```
SQL> @$HOME/labs/DB/insert4.sql
…
SQL>
```

10. Display the new subpartitioning method of the table.

```
SQL> SELECT owner, table_name, partitioning_type AS type,
            autolist, subpartitioning_type
     FROM   dba_part_tables WHERE table_name = 'T';
 2    3
OWNER TABLE_NAME TYPE  AUTOLIST SUBPARTIT
----- ---------- ----- -------- ---------
HR    T          LIST  YES      LIST

SQL>
```

11. Display the partitions of the table and the high values in each partition.

```
SQL> SELECT partition_name, subpartition_name, high_value
     FROM   dba_tab_subpartitions
     WHERE  table_name = 'T';
 2    3
PARTITION_NAME SUBPARTITION_ HIGH_VALUE
-------------- ------------- ----------
P1             P1_SP1        1
P1             P1_SP2        2
P1             P1_SP3        3
P1             P1_SP_UNKNOWN DEFAULT
P2             P2_SP1        1
P2             P2_SP2        2
P2             P2_SP3        3
```

```
P2              P2_SP_UNKNOWN DEFAULT
P3              P3_SP1        1
P3              P3_SP2        2
P3              P3_SP3        3
P3              P3_SP_UNKNOWN DEFAULT

12 rows selected.

SQL>
```

12. Display values from the subpartitions.

```
SQL> SELECT * FROM hr.t SUBPARTITION (P1_SP2);

        C1          C2
---------- ----------
         1           2

SQL> SELECT * FROM hr.t SUBPARTITION (P2_SP3);

        C1          C2
---------- ----------
         2           3

SQL> SELECT * FROM hr.t SUBPARTITION (P2_SP_UNKNOWN);

        C1          C2
---------- ----------
         2          10
         2          20
         2          20
         2          30

SQL> SELECT * FROM hr.t SUBPARTITION (P3_SP_UNKNOWN);

        C1          C2
---------- ----------
         3          10
         3          20
         3          20
         3           4
         3           5

SQL> EXIT
$
```

13. Drop the `HR.T` table. Execute the `$HOME/labs/DB/cleanup_list_auto.sh` shell script.

```
$ $HOME/labs/DB/cleanup_list_auto.sh
…
$
```

Practice 5-6: Merging Partitions of a Partitioned Table Online

Overview

In this practice, you merge partitions of a RANGE partitioned table online.

Tasks

1. Before starting the practice, execute the $HOME/labs/DB/part_merge.sh shell script. The shell script creates a directory for the future import and the RANGE partitioned table SH.SALES, imports rows with a dump file, and creates the SH.I2_PROMO_ID on the PROMO_ID column of the SH.SALES table. Errors will occur during the import operation. Ignore them.

```
$ $HOME/labs/DB/part_merge.sh
…
$
```

2. Log in to PDB1 again as SYSTEM and display the partitioning method of the table.

```
$ sqlplus system@PDB1

Enter password: password

SQL> SELECT owner, table_name, partitioning_type AS type,
            autolist, interval
     FROM   dba_part_tables WHERE  owner='SH';
 2
OWNER TABLE_NAME TYPE  AUTOLIST INTERVAL
----- ---------- ----- -------- ------------------------------
SH    SALES      RANGE NO

SQL>
```

3. Display the partitions of the table and the high values in each partition.

```
SQL> COL high_value FORMAT A44
SQL> SELECT partition_name, high_value FROM dba_tab_partitions
     WHERE  table_name = 'SALES' and table_owner='SH';
 2
PARTITION_NAME HIGH_VALUE
-------------- ----------------------------------------------------------------------------------------------------
SALES_Q1_1998 TO_DATE(' 1998-04-01 00:00:00', 'SYYYY-MM-DD
               HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q2_1998 TO_DATE(' 1998-07-01 00:00:00', 'SYYYY-MM-DD
               HH24:MI:SS', 'NLS_CALENDAR=GREGORIA
```

```
SALES_Q3_1998 TO_DATE(' 1998-10-01 00:00:00', 'SYYYY-MM-DD
               HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q4_1998 TO_DATE(' 1999-01-01 00:00:00', 'SYYYY-MM-DD
               HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q1_1999 TO_DATE(' 1999-04-01 00:00:00', 'SYYYY-MM-DD
               HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q2_1999 TO_DATE(' 1999-07-01 00:00:00', 'SYYYY-MM-DD
               HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q3_1999 TO_DATE(' 1999-10-01 00:00:00', 'SYYYY-MM-DD
               HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q4_1999 TO_DATE(' 2000-01-01 00:00:00', 'SYYYY-MM-DD
               HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q1_2000 TO_DATE(' 2000-04-01 00:00:00', 'SYYYY-MM-DD
               HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q2_2000 TO_DATE(' 2000-07-01 00:00:00', 'SYYYY-MM-DD
               HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q3_2000 TO_DATE(' 2000-10-01 00:00:00', 'SYYYY-MM-DD
               HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q4_2000 TO_DATE(' 2001-01-01 00:00:00', 'SYYYY-MM-DD
               HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q1_2001 TO_DATE(' 2001-04-01 00:00:00', 'SYYYY-MM-DD
               HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q2_2001 TO_DATE(' 2001-07-01 00:00:00', 'SYYYY-MM-DD
               HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q3_2001 TO_DATE(' 2001-10-01 00:00:00', 'SYYYY-MM-DD
               HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q4_2001 TO_DATE(' 2002-01-01 00:00:00', 'SYYYY-MM-DD
               HH24:MI:SS', 'NLS_CALENDAR=GREGORIA
```

```
SALES_1995     TO_DATE(' 1996-01-01 00:00:00', 'SYYYY-MM-DD
                HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_1996     TO_DATE(' 1997-01-01 00:00:00', 'SYYYY-MM-DD
                HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_H1_1997 TO_DATE(' 1997-07-01 00:00:00', 'SYYYY-MM-DD
                HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_H2_1997 TO_DATE(' 1998-01-01 00:00:00', 'SYYYY-MM-DD
                HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q1_2002 TO_DATE(' 2002-04-01 00:00:00', 'SYYYY-MM-DD
                HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q1_2003 TO_DATE(' 2003-04-01 00:00:00', 'SYYYY-MM-DD
                HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q2_2002 TO_DATE(' 2002-07-01 00:00:00', 'SYYYY-MM-DD
                HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q2_2003 TO_DATE(' 2003-07-01 00:00:00', 'SYYYY-MM-DD
                HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q3_2002 TO_DATE(' 2002-10-01 00:00:00', 'SYYYY-MM-DD
                HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q3_2003 TO_DATE(' 2003-10-01 00:00:00', 'SYYYY-MM-DD
                HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q4_2002 TO_DATE(' 2003-01-01 00:00:00', 'SYYYY-MM-DD
                HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q4_2003 TO_DATE(' 2004-01-01 00:00:00', 'SYYYY-MM-DD
                HH24:MI:SS', 'NLS_CALENDAR=GREGORIA


28 rows selected.

SQL>
```

4. Display the indexes of the table and check whether the indexes are partitioned or not.

```
SQL> SELECT index_name, index_type, partitioned
     FROM  dba_indexes
     WHERE table_name='SALES';
 2
INDEX_NAME                     INDEX_TYPE                     PAR
------------------------------ ------------------------------ -------
SALES_PROD_BIX                 BITMAP                         YES
SALES_CUST_BIX                 BITMAP                         YES
SALES_TIME_BIX                 BITMAP                         YES
SALES_CHANNEL_BIX              BITMAP                         YES
I2_PROMO_ID                    NORMAL                         NO

SQL>
```

5. Display the type of partitioning of the partitioned indexes.

```
SQL> SELECT index_name, locality, partitioning_type AS type,
            autolist, interval
     FROM   dba_part_indexes WHERE owner='SH';
  2    3
INDEX_NAME                     LOCALI TYPE  AUTOLIST INTERVAL
------------------------------ ------ ------ -------- --------------------
SALES_CHANNEL_BIX              LOCAL  RANGE  NO
SALES_CUST_BIX                 LOCAL  RANGE  NO
SALES_PROD_BIX                 LOCAL  RANGE  NO
SALES_TIME_BIX                 LOCAL  RANGE NO

SQL>
```

6. In *Session2*, log in to `PDB1` as `SH` and increase the quantity sold in the `SALES_Q1_2000` partition.

```
$ sqlplus sh@PDB1

Enter password: password

SQL> UPDATE sh.sales PARTITION (sales_q1_2000)
            SET QUANTITY_SOLD=QUANTITY_SOLD*10;
 2
62197 rows updated.

SQL>
```

7. In *Session1*, merge the partition of 2000 year to a single partition. This operation should be possible concurrently with the DML operation.

```
SQL> ALTER TABLE sh.sales
           MERGE PARTITIONS sales_q1_2000, sales_q2_2000,
```

```
                              sales_q3_2000,sales_q4_2000
            INTO PARTITION sales_2000
            COMPRESS UPDATE INDEXES ONLINE;
 2    3    4    5
```

The statement waits for the `UPDATE` in the second session to complete.

8. In the `UPDATE` session, commit the `UPDATE` statement.

```
SQL> COMMIT;

Commit complete.

SQL> EXIT
$
```

9. In the merge session, the `ALTER TABLE` statement completes.

```
Table altered.

SQL>
```

10. Verify that the four partitions are merged into one single partition. The query of step 5 reported 28 partitions. The current query reports 25 partitions.

```
SQL> SELECT partition_name, high_value
     FROM dba_tab_partitions
     WHERE  table_name = 'SALES' AND table_owner = 'SH';
  2    3
PARTITION_NAME HIGH_VALUE
------------------------------ --------------------------------------------------------------------------------
SALES_1995     TO_DATE(' 1996-01-01 00:00:00', 'SYYYY-MM-DD
                HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_1996     TO_DATE(' 1997-01-01 00:00:00', 'SYYYY-MM-DD
                HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_2000     TO_DATE(' 2001-01-01 00:00:00', 'SYYYY-MM-DD
                HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_H1_1997 TO_DATE(' 1997-07-01 00:00:00', 'SYYYY-MM-DD
                HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_H2_1997 TO_DATE(' 1998-01-01 00:00:00', 'SYYYY-MM-DD
                HH24:MI:SS', 'NLS_CALENDAR=GREGORIA
```

```
SALES_Q1_1998 TO_DATE(' 1998-04-01 00:00:00', 'SYYYY-MM-DD
               HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q1_1999 TO_DATE(' 1999-04-01 00:00:00', 'SYYYY-MM-DD
               HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q1_2001 TO_DATE(' 2001-04-01 00:00:00', 'SYYYY-MM-DD
               HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q1_2002 TO_DATE(' 2002-04-01 00:00:00', 'SYYYY-MM-DD
               HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q1_2003 TO_DATE(' 2003-04-01 00:00:00', 'SYYYY-MM-DD
               HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q2_1998 TO_DATE(' 1998-07-01 00:00:00', 'SYYYY-MM-DD
               HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q2_1999 TO_DATE(' 1999-07-01 00:00:00', 'SYYYY-MM-DD
               HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q2_2001 TO_DATE(' 2001-07-01 00:00:00', 'SYYYY-MM-DD
               HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q2_2002 TO_DATE(' 2002-07-01 00:00:00', 'SYYYY-MM-DD
               HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q2_2003 TO_DATE(' 2003-07-01 00:00:00', 'SYYYY-MM-DD
               HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q3_1998 TO_DATE(' 1998-10-01 00:00:00', 'SYYYY-MM-DD
               HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q3_1999 TO_DATE(' 1999-10-01 00:00:00', 'SYYYY-MM-DD
               HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q3_2001 TO_DATE(' 2001-10-01 00:00:00', 'SYYYY-MM-DD
               HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q3_2002 TO_DATE(' 2002-10-01 00:00:00', 'SYYYY-MM-DD
               HH24:MI:SS', 'NLS_CALENDAR=GREGORIA
```

```
SALES_Q3_2003 TO_DATE(' 2003-10-01 00:00:00', 'SYYYY-MM-DD
              HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q4_1998 TO_DATE(' 1999-01-01 00:00:00', 'SYYYY-MM-DD
              HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q4_1999 TO_DATE(' 2000-01-01 00:00:00', 'SYYYY-MM-DD
              HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q4_2001 TO_DATE(' 2002-01-01 00:00:00', 'SYYYY-MM-DD
              HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q4_2002 TO_DATE(' 2003-01-01 00:00:00', 'SYYYY-MM-DD
              HH24:MI:SS', 'NLS_CALENDAR=GREGORIA

SALES_Q4_2003 TO_DATE(' 2004-01-01 00:00:00', 'SYYYY-MM-DD
              HH24:MI:SS', 'NLS_CALENDAR=GREGORIA


25 rows selected.

SQL> EXIT
$
```

11. Drop the `SH.SALES` table. Execute the `$HOME/labs/DB/cleanup_merge.sh` shell script.

```
$ $HOME/labs/DB/cleanup_merge.sh
…
$
```

Practice 5-7: Using Batched DDL

Overview

In this practice, you identify differences between two table definitions and generate a single `ALTER TABLE` statement that can be executed for one of the two tables to make it look like the other table.

Tasks

1. Before starting the practice, execute the `$HOME/labs/DB/DDL.sh` shell script. The SQL script recreates `PDB1` and the `SCOTT` schema with two tables, `TEST1` and `TEST2`.

```
$ $HOME/labs/DB/DDL.sh
…
$
```

2. Use the existing `DBMS_METADATA.SET_TRANSFORM_PARAM` procedure to compare the two tables without the new `BATCHED_ALTER_DDL` name of the transform parameter.

```
$ sqlplus SCOTT@PDB1

Enter password: password

SQL>
SQL> SET SERVEROUTPUT ON
```

```
DECLARE
h           NUMBER;    -- Open handle
och         NUMBER;    -- openc handle
th          NUMBER;    -- transform handle
clob1       CLOB;      -- object1 sxml doc
clob2       CLOB;      -- object2 sxml doc
difdoc      CLOB;      -- difference doc
altxml      CLOB;      -- alter xmldoc
altddl      CLOB;      -- alter ddl doc

BEGIN
-- Fetch SXML metadata for the first doc
h    := dbms_metadata.open('TABLE');
dbms_metadata.set_filter(h,'NAME', 'TEST1');
th   := dbms_metadata.add_transform(h,'SXML');
clob1 := dbms_metadata.fetch_clob(h);
dbms_metadata.close(h);

-- Fetch SXML metadata for the second doc
h    := dbms_metadata.open('TABLE');
dbms_metadata.set_filter(h, 'NAME', 'TEST2');
```

```
th    := dbms_metadata.add_transform(h, 'SXML');
clob2 := dbms_metadata.fetch_clob(h);
dbms_metadata.close(h);

-- Compare the two objects and generate an SXML difference document
och := dbms_metadata_diff.openc('TABLE');
dbms_metadata_diff.add_document(och, clob1);
dbms_metadata_diff.add_document(och, clob2);
difdoc :=  dbms_metadata_diff.fetch_clob(och);
dbms_metadata_diff.close(och);

-- Using the difdoc get the ALTER xmldoc
h := dbms_metadata.openw('TABLE');
th := dbms_metadata.add_transform(h,'ALTERXML');
dbms_metadata.set_parse_item(h,'XPATH');
dbms_metadata.set_parse_item(h,'ALTERABLE');
dbms_metadata.set_parse_item(h,'CLAUSE_TYPE');
dbms_metadata.set_parse_item(h,'NAME');
dbms_metadata.set_parse_item(h,'COLUMN_ATTRIBUTE');
dbms_metadata.set_parse_item(h,'CONSTRAINT_TYPE');
dbms_metadata.set_parse_item(h,'CONSTRAINT_STATE');
dbms_metadata.set_parse_item(h,'PARTITION_TYPE');

-- Get the alter xmldoc
dbms_lob.createtemporary(altxml, TRUE);
dbms_metadata.convert (h, difdoc, altxml);
dbms_metadata.close(h);

-- convert the alter xmldoc to ddl
h := dbms_metadata.openw('TABLE');
th := dbms_metadata.add_transform(h,'ALTERDDL');
dbms_lob.createtemporary(altddl, TRUE);
dbms_metadata.convert (h, altxml, altddl);

-- display the ddl resulting from this compare
dbms_output.put_line(altddl);

END;
/
```

```
SQL> DECLARE
  2  h           NUMBER;    -- Open handle
  3  och         NUMBER;    -- openc handle
  4  th          NUMBER;    -- transform handle
  5  clob1       CLOB;      -- object1 sxml doc
```

```
  6  clob2       CLOB;     -- object2 sxml doc
  7  difdoc      CLOB;     -- difference doc
  8  altxml      CLOB;     -- alter xmldoc
  9  altddl      CLOB;     -- alter ddl doc
 10
 11  BEGIN
 12  -- Fetch SXML metadata for the first doc
 13  h    := dbms_metadata.open('TABLE');
 14  dbms_metadata.set_filter(h,'NAME', 'TEST1');
 15  th   := dbms_metadata.add_transform(h,'SXML');
 16  clob1 := dbms_metadata.fetch_clob(h);
 17  dbms_metadata.close(h);
 18
 19  -- Fetch SXML metadata for the second doc
 20  h    := dbms_metadata.open('TABLE');
 21  dbms_metadata.set_filter(h, 'NAME', 'TEST2');
 22  th    := dbms_metadata.add_transform(h, 'SXML');
 23  clob2 := dbms_metadata.fetch_clob(h);
 24  dbms_metadata.close(h);
 25
 26  -- Compare the two objects and generate an SXML difference
document
 27  och := dbms_metadata_diff.openc('TABLE');
 28  dbms_metadata_diff.add_document(och, clob1);
 29  dbms_metadata_diff.add_document(och, clob2);
 30  difdoc :=  dbms_metadata_diff.fetch_clob(och);
 31  dbms_metadata_diff.close(och);
 32
 33  -- Using the difdoc get the ALTER xmldoc
 34  h := dbms_metadata.openw('TABLE');
 35  th := dbms_metadata.add_transform(h,'ALTERXML');
 36  dbms_metadata.set_parse_item(h,'XPATH');
 37  dbms_metadata.set_parse_item(h,'ALTERABLE');
 38  dbms_metadata.set_parse_item(h,'CLAUSE_TYPE');
 39  dbms_metadata.set_parse_item(h,'NAME');
 40  dbms_metadata.set_parse_item(h,'COLUMN_ATTRIBUTE');
 41  dbms_metadata.set_parse_item(h,'CONSTRAINT_TYPE');
 42  dbms_metadata.set_parse_item(h,'CONSTRAINT_STATE');
 43  dbms_metadata.set_parse_item(h,'PARTITION_TYPE');
 44
 45  -- Get the alter xmldoc
 46  dbms_lob.createtemporary(altxml, TRUE);
 47  dbms_metadata.convert (h, difdoc, altxml);
```

```
 48  dbms_metadata.close(h);
 49
 50  -- convert the alter xmldoc to ddl
 51  h := dbms_metadata.openw('TABLE');
 52  th := dbms_metadata.add_transform(h,'ALTERDDL');
 53  dbms_lob.createtemporary(altddl, TRUE);
 54  dbms_metadata.convert (h, altxml, altddl);
 55
 56  -- display the ddl resulting from this compare
 57  dbms_output.put_line(altddl);
 58
 59  END;
 60  /
ALTER TABLE "SCOTT"."TEST1" ADD ("T" VARCHAR2(30))
  ALTER TABLE "SCOTT"."TEST1" MODIFY ("Y" VARCHAR2(40))
  ALTER TABLE "SCOTT"."TEST1" RENAME TO "TEST2"

PL/SQL procedure successfully completed.

SQL>
```

Observe that there are three `ALTER TABLE` commands for the three differences detected. The reason is that the three clauses required to make the first `SCOTT.TEST1` table look like the second `SCOTT.TEST2` table—`ADD`, `MODIFY`, and `RENAME`—cannot be combined within a single `ALTER TABLE` statement. In this case, Oracle Database 12c would anyway display the same result.

3. Re-create the first `SCOTT.TEST2` table and compare.

```
SQL> DROP TABLE scott.test2;

Table dropped.

SQL> CREATE TABLE scott.test2 (x NUMBER);

Table created.

SQL>
```

4. Use the same `DBMS_METADATA.SET_TRANSFORM_PARAM` procedure to compare the two tables without the new `BATCHED_ALTER_DDL` name of the transform parameter.

```
DECLARE
h           NUMBER;   -- Open handle
och         NUMBER;   -- openc handle
th          NUMBER;   -- transform handle
clob1       CLOB;     -- object1 sxml doc
clob2       CLOB;     -- object2 sxml doc
difdoc      CLOB;     -- difference doc
altxml      CLOB;     -- alter xmldoc
altddl      CLOB;     -- alter ddl doc

BEGIN
-- Fetch SXML metadata for the first doc
h    := dbms_metadata.open('TABLE');
dbms_metadata.set_filter(h,'NAME', 'TEST1');
th   := dbms_metadata.add_transform(h,'SXML');
clob1 := dbms_metadata.fetch_clob(h);
dbms_metadata.close(h);

-- Fetch SXML metadata for the second doc
h    := dbms_metadata.open('TABLE');
dbms_metadata.set_filter(h, 'NAME', 'TEST2');
th    := dbms_metadata.add_transform(h, 'SXML');
clob2 := dbms_metadata.fetch_clob(h);
dbms_metadata.close(h);

-- Compare the two objects and generate an SXML difference document
och := dbms_metadata_diff.openc('TABLE');
dbms_metadata_diff.add_document(och, clob1);
dbms_metadata_diff.add_document(och, clob2);
difdoc :=  dbms_metadata_diff.fetch_clob(och);
dbms_metadata_diff.close(och);

-- Using the difdoc get the ALTER xmldoc
h := dbms_metadata.openw('TABLE');
th := dbms_metadata.add_transform(h,'ALTERXML');
dbms_metadata.set_parse_item(h,'XPATH');
dbms_metadata.set_parse_item(h,'ALTERABLE');
dbms_metadata.set_parse_item(h,'CLAUSE_TYPE');
dbms_metadata.set_parse_item(h,'NAME');
dbms_metadata.set_parse_item(h,'COLUMN_ATTRIBUTE');
dbms_metadata.set_parse_item(h,'CONSTRAINT_TYPE');
dbms_metadata.set_parse_item(h,'CONSTRAINT_STATE');
dbms_metadata.set_parse_item(h,'PARTITION_TYPE');
```

```
-- Get the alter xmldoc
dbms_lob.createtemporary(altxml, TRUE);
dbms_metadata.convert (h, difdoc, altxml);
dbms_metadata.close(h);

-- convert the alter xmldoc to ddl
h := dbms_metadata.openw('TABLE');
th := dbms_metadata.add_transform(h,'ALTERDDL');
dbms_lob.createtemporary(altddl, TRUE);
dbms_metadata.convert (h, altxml, altddl);

-- display the ddl resulting from this compare
dbms_output.put_line(altddl);

END;
/
```

```
SQL> DECLARE
  2  h           NUMBER;    -- Open handle
  3  och         NUMBER;    -- openc handle
  4  th          NUMBER;    -- transform handle
  5  clob1       CLOB;      -- object1 sxml doc
  6  clob2       CLOB;      -- object2 sxml doc
  7  difdoc      CLOB;      -- difference doc
  8  altxml      CLOB;      -- alter xmldoc
  9  altddl      CLOB;      -- alter ddl doc
 10
 11  BEGIN
 12  -- Fetch SXML metadata for the first doc
 13  h    := dbms_metadata.open('TABLE');
 14  dbms_metadata.set_filter(h,'NAME', 'TEST1');
 15  th   := dbms_metadata.add_transform(h,'SXML');
 16  clob1 := dbms_metadata.fetch_clob(h);
 17  dbms_metadata.close(h);
 18
 19  -- Fetch SXML metadata for the second doc
 20  h    := dbms_metadata.open('TABLE');
 21  dbms_metadata.set_filter(h, 'NAME', 'TEST2');
 22  th    := dbms_metadata.add_transform(h, 'SXML');
 23  clob2 := dbms_metadata.fetch_clob(h);
 24  dbms_metadata.close(h);
 25
```

```
 26  -- Compare the two objects and generate an SXML difference
document
 27  och := dbms_metadata_diff.openc('TABLE');
 28  dbms_metadata_diff.add_document(och, clob1);
 29  dbms_metadata_diff.add_document(och, clob2);
 30  difdoc :=  dbms_metadata_diff.fetch_clob(och);
 31  dbms_metadata_diff.close(och);
 32
 33  -- Using the difdoc get the ALTER xmldoc
h := dbms_metadata.openw('TABLE');
 34   35  th := dbms_metadata.add_transform(h,'ALTERXML');
 36  dbms_metadata.set_parse_item(h,'XPATH');
 37  dbms_metadata.set_parse_item(h,'ALTERABLE');
 38  dbms_metadata.set_parse_item(h,'CLAUSE_TYPE');
 39  dbms_metadata.set_parse_item(h,'NAME');
 40  dbms_metadata.set_parse_item(h,'COLUMN_ATTRIBUTE');
 41  dbms_metadata.set_parse_item(h,'CONSTRAINT_TYPE');
 42  dbms_metadata.set_parse_item(h,'CONSTRAINT_STATE');
 43  dbms_metadata.set_parse_item(h,'PARTITION_TYPE');
 44
 45  -- Get the alter xmldoc
 46  dbms_lob.createtemporary(altxml, TRUE);
 47  dbms_metadata.convert (h, difdoc, altxml);
 48 dbms_metadata.close(h);
 49
 50 -- convert the alter xmldoc to ddl
 51 h := dbms_metadata.openw('TABLE');
 52 th := dbms_metadata.add_transform(h,'ALTERDDL');
 53 dbms_lob.createtemporary(altddl, TRUE);
 54 dbms_metadata.convert (h, altxml, altddl);
 55
 56 -- display the ddl resulting from this compare
 57   dbms_output.put_line(altddl);
 58 END;
 59 /
 ALTER TABLE "SCOTT"."TEST1" DROP ("Y")
  ALTER TABLE "SCOTT"."TEST1" DROP ("Z")
  ALTER TABLE "SCOTT"."TEST1" RENAME TO "TEST2"

PL/SQL procedure successfully completed.

SQL>
```

Observe that there is now one single `ALTER TABLE` command for the first two differences detected. The reason is that these two differences correspond to the same `DROP` clause required to make the first `SCOTT.TEST1` table look like the second `SCOTT.TEST2` table.

In this case, Oracle Database 12c would display two `ALTER TABLE` commands for the first two differences detected:

```
ALTER TABLE "SCOTT"."TEST1" DROP ("Y")
  ALTER TABLE "SCOTT"."TEST1" DROP ("Z")
```

5. Use the same `DBMS_METADATA.SET_TRANSFORM_PARAM` procedure to compare the two tables with the new `BATCHED_ALTER_DDL` name of the transform parameter.

```
DECLARE
h           NUMBER;   -- Open handle
och         NUMBER;   -- openc handle
th          NUMBER;   -- transform handle
clob1       CLOB;     -- object1 sxml doc
clob2       CLOB;     -- object2 sxml doc
difdoc      CLOB;     -- difference doc
altxml      CLOB;     -- alter xmldoc
altddl      CLOB;     -- alter ddl doc

BEGIN
-- Fetch SXML metadata for the first doc
h    := dbms_metadata.open('TABLE');
dbms_metadata.set_filter(h,'NAME', 'TEST1');
th   := dbms_metadata.add_transform(h,'SXML');
clob1 := dbms_metadata.fetch_clob(h);
dbms_metadata.close(h);

-- Fetch SXML metadata for the second doc
h    := dbms_metadata.open('TABLE');
dbms_metadata.set_filter(h, 'NAME',  'TEST2');
th    := dbms_metadata.add_transform(h, 'SXML');
clob2 := dbms_metadata.fetch_clob(h);
dbms_metadata.close(h);

-- Compare the two objects and generate an SXML difference document
och := dbms_metadata_diff.openc('TABLE');
dbms_metadata_diff.add_document(och, clob1);
dbms_metadata_diff.add_document(och, clob2);
difdoc :=  dbms_metadata_diff.fetch_clob(och);
dbms_metadata_diff.close(och);

-- Using the difdoc get the ALTER xmldoc
h := dbms_metadata.openw('TABLE');
```

```
th := dbms_metadata.add_transform(h,'ALTERXML');
dbms_metadata.set_parse_item(h,'XPATH');
dbms_metadata.set_parse_item(h,'ALTERABLE');
dbms_metadata.set_parse_item(h,'CLAUSE_TYPE');
dbms_metadata.set_parse_item(h,'NAME');
dbms_metadata.set_parse_item(h,'COLUMN_ATTRIBUTE');
dbms_metadata.set_parse_item(h,'CONSTRAINT_TYPE');
dbms_metadata.set_parse_item(h,'CONSTRAINT_STATE');
dbms_metadata.set_parse_item(h,'PARTITION_TYPE');

-- Get the alter xmldoc
dbms_metadata.set_transform_param(th, 'BATCH_ALTER_DDL', true);
dbms_lob.createtemporary(altxml, TRUE);
dbms_metadata.convert (h, difdoc, altxml);
dbms_metadata.close(h);

-- convert the alter xmldoc to ddl
h := dbms_metadata.openw('TABLE');
th := dbms_metadata.add_transform(h,'ALTERDDL');
dbms_lob.createtemporary(altddl, TRUE);
dbms_metadata.convert (h, altxml, altddl);

-- display the ddl resulting from this compare
dbms_output.put_line(altddl);

END;
/
```

```
SQL> DECLARE
  2  h           NUMBER;   -- Open handle
  3  och         NUMBER;   -- openc handle
  4  th          NUMBER;   -- transform handle
  5  clob1       CLOB;     -- object1 sxml doc
  6  clob2       CLOB;     -- object2 sxml doc
  7  difdoc      CLOB;     -- difference doc
  8  altxml      CLOB;     -- alter xmldoc
  9  altddl      CLOB;     -- alter ddl doc
 10
 11  BEGIN
 12  -- Fetch SXML metadata for the first doc
 13  h    := dbms_metadata.open('TABLE');
 14  dbms_metadata.set_filter(h,'NAME', 'TEST1');
 15  th   := dbms_metadata.add_transform(h,'SXML');
 16  clob1 := dbms_metadata.fetch_clob(h);
 17  dbms_metadata.close(h);
```

```
 18
 19  -- Fetch SXML metadata for the second doc
 20  h    := dbms_metadata.open('TABLE');
 21  dbms_metadata.set_filter(h, 'NAME', 'TEST2');
 22  th    := dbms_metadata.add_transform(h, 'SXML');
 23  clob2 := dbms_metadata.fetch_clob(h);
 24  dbms_metadata.close(h);
 25
 26  -- Compare the two objects and generate an SXML difference
document
 27  och := dbms_metadata_diff.openc('TABLE');
 28  dbms_metadata_diff.add_document(och, clob1);
 29  dbms_metadata_diff.add_document(och, clob2);
 30  difdoc :=  dbms_metadata_diff.fetch_clob(och);
 31  dbms_metadata_diff.close(och);
 32
 33  -- Using the difdoc get the ALTER xmldoc
 34  h := dbms_metadata.openw('TABLE');
 35  th := dbms_metadata.add_transform(h,'ALTERXML');
 36  dbms_metadata.set_parse_item(h,'XPATH');
 37  dbms_metadata.set_parse_item(h,'ALTERABLE');
 38  dbms_metadata.set_parse_item(h,'CLAUSE_TYPE');
 39  dbms_metadata.set_parse_item(h,'NAME');
 40  dbms_metadata.set_parse_item(h,'COLUMN_ATTRIBUTE');
 41  dbms_metadata.set_parse_item(h,'CONSTRAINT_TYPE');
 42  dbms_metadata.set_parse_item(h,'CONSTRAINT_STATE');
 43  dbms_metadata.set_parse_item(h,'PARTITION_TYPE');
 44
 45  -- Get the alter xmldoc
 46  dbms_metadata.set_transform_param(th, 'BATCH_ALTER_DDL',
true);
 47  dbms_lob.createtemporary(altxml, TRUE);
 48  dbms_metadata.convert (h, difdoc, altxml);
 49  dbms_metadata.close(h);
 50
 51  -- convert the alter xmldoc to ddl
 52  h := dbms_metadata.openw('TABLE');
 53  th := dbms_metadata.add_transform(h,'ALTERDDL');
 54  dbms_lob.createtemporary(altddl, TRUE);
 55  dbms_metadata.convert (h, altxml, altddl);
 56
 57  -- display the ddl resulting from this compare
 58  dbms_output.put_line(altddl);
```

```
 59
 60  END;
 61  /
ALTER TABLE "SCOTT"."TEST1" DROP ("Y", "Z")
  ALTER TABLE "SCOTT"."TEST1" RENAME TO "TEST2"

PL/SQL procedure successfully completed.

SQL>
```

Observe that there is now one single `ALTER TABLE` command for the first two differences detected. The reason is that these two differences correspond to the same `DROP` clause required to make the first `SCOTT.TEST1` table look like the second `SCOTT.TEST2` table.

6. Re-create the tables and compare.

```
SQL> DROP TABLE scott.test1;

Table dropped.

SQL> CREATE TABLE scott.test1 (x NUMBER);

Table created.

SQL> DROP TABLE scott.test2;

Table dropped.

SQL> CREATE TABLE scott.test2 (x NUMBER, y VARCHAR2(40), t
VARCHAR2(30), z DATE);

Table created.

SQL>
```

7. Use the `DBMS_METADATA.SET_TRANSFORM_PARAM` procedure to compare the two tables with the new `BATCHED_ALTER_DDL` name of the transform parameter.

```
DECLARE
h           NUMBER;    -- Open handle
och         NUMBER;    -- openc handle
th          NUMBER;    -- transform handle
clob1       CLOB;      -- object1 sxml doc
```

```
clob2       CLOB;     -- object2 sxml doc
difdoc      CLOB;     -- difference doc
altxml      CLOB;     -- alter xmldoc
altddl      CLOB;     -- alter ddl doc

BEGIN
-- Fetch SXML metadata for the first doc
h    := dbms_metadata.open('TABLE');
dbms_metadata.set_filter(h,'NAME', 'TEST1');
th   := dbms_metadata.add_transform(h,'SXML');
clob1 := dbms_metadata.fetch_clob(h);
dbms_metadata.close(h);

-- Fetch SXML metadata for the second doc
h    := dbms_metadata.open('TABLE');
dbms_metadata.set_filter(h, 'NAME', 'TEST2');
th    := dbms_metadata.add_transform(h, 'SXML');
clob2 := dbms_metadata.fetch_clob(h);
dbms_metadata.close(h);

-- Compare the two objects and generate an SXML difference document
och := dbms_metadata_diff.openc('TABLE');
dbms_metadata_diff.add_document(och, clob1);
dbms_metadata_diff.add_document(och, clob2);
difdoc :=  dbms_metadata_diff.fetch_clob(och);
dbms_metadata_diff.close(och);

-- Using the difdoc get the ALTER xmldoc
h := dbms_metadata.openw('TABLE');
th := dbms_metadata.add_transform(h,'ALTERXML');
dbms_metadata.set_parse_item(h,'XPATH');
dbms_metadata.set_parse_item(h,'ALTERABLE');
dbms_metadata.set_parse_item(h,'CLAUSE_TYPE');
dbms_metadata.set_parse_item(h,'NAME');
dbms_metadata.set_parse_item(h,'COLUMN_ATTRIBUTE');
dbms_metadata.set_parse_item(h,'CONSTRAINT_TYPE');
dbms_metadata.set_parse_item(h,'CONSTRAINT_STATE');
dbms_metadata.set_parse_item(h,'PARTITION_TYPE');

-- Get the alter xmldoc
dbms_metadata.set_transform_param(th, 'BATCH_ALTER_DDL', true);
dbms_lob.createtemporary(altxml, TRUE);
dbms_metadata.convert (h, difdoc, altxml);
dbms_metadata.close(h);

-- convert the alter xmldoc to ddl
```

```
h := dbms_metadata.openw('TABLE');
th := dbms_metadata.add_transform(h,'ALTERDDL');
dbms_lob.createtemporary(altddl, TRUE);
dbms_metadata.convert (h, altxml, altddl);

-- display the ddl resulting from this compare
dbms_output.put_line(altddl);

END;
/
```

```
SQL> DECLARE
  2  h           NUMBER;   -- Open handle
  3  och         NUMBER;   -- openc handle
  4  th          NUMBER;   -- transform handle
  5  clob1       CLOB;     -- object1 sxml doc
  6  clob2       CLOB;     -- object2 sxml doc
  7  difdoc      CLOB;     -- difference doc
  8  altxml      CLOB;     -- alter xmldoc
  9  altddl      CLOB;     -- alter ddl doc
 10
 11  BEGIN
 12  -- Fetch SXML metadata for the first doc
 13  h    := dbms_metadata.open('TABLE');
 14  dbms_metadata.set_filter(h,'NAME', 'TEST1');
 15  th   := dbms_metadata.add_transform(h,'SXML');
 16  clob1 := dbms_metadata.fetch_clob(h);
 17  dbms_metadata.close(h);
 18
 19  -- Fetch SXML metadata for the second doc
 20  h    := dbms_metadata.open('TABLE');
 21  dbms_metadata.set_filter(h, 'NAME', 'TEST2');
 22  th    := dbms_metadata.add_transform(h, 'SXML');
 23  clob2 := dbms_metadata.fetch_clob(h);
 24  dbms_metadata.close(h);
 25
 26  -- Compare the two objects and generate an SXML difference
document
 27  och := dbms_metadata_diff.openc('TABLE');
 28  dbms_metadata_diff.add_document(och, clob1);
 29  dbms_metadata_diff.add_document(och, clob2);
 30  difdoc :=  dbms_metadata_diff.fetch_clob(och);
 31  dbms_metadata_diff.close(och);
```

```
 32
 33  -- Using the difdoc get the ALTER xmldoc
 34  h := dbms_metadata.openw('TABLE');
 35  th := dbms_metadata.add_transform(h,'ALTERXML');
 36  dbms_metadata.set_parse_item(h,'XPATH');
 37  dbms_metadata.set_parse_item(h,'ALTERABLE');
 38  dbms_metadata.set_parse_item(h,'CLAUSE_TYPE');
 39  dbms_metadata.set_parse_item(h,'NAME');
 40  dbms_metadata.set_parse_item(h,'COLUMN_ATTRIBUTE');
 41  dbms_metadata.set_parse_item(h,'CONSTRAINT_TYPE');
 42  dbms_metadata.set_parse_item(h,'CONSTRAINT_STATE');
 43  dbms_metadata.set_parse_item(h,'PARTITION_TYPE');
 44
 45  -- Get the alter xmldoc
 46  dbms_metadata.set_transform_param(th, 'BATCH_ALTER_DDL',
true);
 47  dbms_lob.createtemporary(altxml, TRUE);
 48  dbms_metadata.convert (h, difdoc, altxml);
 49  dbms_metadata.close(h);
 50
 51  -- convert the alter xmldoc to ddl
 52  h := dbms_metadata.openw('TABLE');
 53  th := dbms_metadata.add_transform(h,'ALTERDDL');
 54  dbms_lob.createtemporary(altddl, TRUE);
 55  dbms_metadata.convert (h, altxml, altddl);
 56
 57  -- display the ddl resulting from this compare
 58  dbms_output.put_line(altddl);
 59
 60  END;
 61  /
ALTER TABLE "SCOTT"."TEST1" ADD ("Y" VARCHAR2(40),
"T" VARCHAR2(30), "Z" DATE)
  ALTER TABLE "SCOTT"."TEST1" RENAME TO "TEST2"

PL/SQL procedure successfully completed.

SQL>
```

Observe that there is now one single `ALTER TABLE` command for the first three differences detected. The reason is that these three differences correspond to the same `ADD` clause required to make the first `SCOTT.TEST1` table look like the second

SCOTT.TEST2 table.
In this case, Oracle Database 12c would display three ALTER TABLE commands for the first three differences detected:

```
ALTER TABLE "SCOTT"."TEST1" ADD ("Y" VARCHAR2(40))
  ALTER TABLE "SCOTT"."TEST1" ADD ("T" VARCHAR2(30))
  ALTER TABLE "SCOTT"."TEST1" ADD ("Z" DATE)
```

Q/ What happens if the BATCHED_ALTER_DDL name of the transform parameter is set to FALSE or is simply not defined?

```
DECLARE
h           NUMBER;   -- Open handle
och         NUMBER;   -- openc handle
th          NUMBER;   -- transform handle
clob1       CLOB;     -- object1 sxml doc
clob2       CLOB;     -- object2 sxml doc
difdoc      CLOB;     -- difference doc
altxml      CLOB;     -- alter xmldoc
altddl      CLOB;     -- alter ddl doc

BEGIN
-- Fetch SXML metadata for the first doc
h    := dbms_metadata.open('TABLE');
dbms_metadata.set_filter(h,'NAME', 'TEST1');
th   := dbms_metadata.add_transform(h,'SXML');
clob1 := dbms_metadata.fetch_clob(h);
dbms_metadata.close(h);

-- Fetch SXML metadata for the second doc
h    := dbms_metadata.open('TABLE');
dbms_metadata.set_filter(h, 'NAME', 'TEST2');
th    := dbms_metadata.add_transform(h, 'SXML');
clob2 := dbms_metadata.fetch_clob(h);
dbms_metadata.close(h);

-- Compare the two objects and generate an SXML difference document
och := dbms_metadata_diff.openc('TABLE');
dbms_metadata_diff.add_document(och, clob1);
dbms_metadata_diff.add_document(och, clob2);
difdoc := dbms_metadata_diff.fetch_clob(och);
dbms_metadata_diff.close(och);

-- Using the difdoc get the ALTER xmldoc
h := dbms_metadata.openw('TABLE');
th := dbms_metadata.add_transform(h,'ALTERXML');
```

```
dbms_metadata.set_parse_item(h,'XPATH');
dbms_metadata.set_parse_item(h,'ALTERABLE');
dbms_metadata.set_parse_item(h,'CLAUSE_TYPE');
dbms_metadata.set_parse_item(h,'NAME');
dbms_metadata.set_parse_item(h,'COLUMN_ATTRIBUTE');
dbms_metadata.set_parse_item(h,'CONSTRAINT_TYPE');
dbms_metadata.set_parse_item(h,'CONSTRAINT_STATE');
dbms_metadata.set_parse_item(h,'PARTITION_TYPE');

-- Get the alter xmldoc
dbms_metadata.set_transform_param(th, 'BATCH_ALTER_DDL', false);
dbms_lob.createtemporary(altxml, TRUE);
dbms_metadata.convert (h, difdoc, altxml);
dbms_metadata.close(h);

-- convert the alter xmldoc to ddl
h := dbms_metadata.openw('TABLE');
th := dbms_metadata.add_transform(h,'ALTERDDL');
dbms_lob.createtemporary(altddl, TRUE);
dbms_metadata.convert (h, altxml, altddl);

-- display the ddl resulting from this compare
dbms_output.put_line(altddl);

END;
/
```

```
SQL> DECLARE
  2  h           NUMBER;    -- Open handle
  3  och         NUMBER;    -- openc handle
  4  th          NUMBER;    -- transform handle
  5  clob1       CLOB;      -- object1 sxml doc
  6  clob2       CLOB;      -- object2 sxml doc
  7  difdoc      CLOB;      -- difference doc
  8  altxml      CLOB;      -- alter xmldoc
  9  altddl      CLOB;      -- alter ddl doc
 10
 11  BEGIN
 12  -- Fetch SXML metadata for the first doc
h    := dbms_metadata.open('TABLE');
 13   14  dbms_metadata.set_filter(h,'NAME', 'TEST1');
 15  th   := dbms_metadata.add_transform(h,'SXML');
 16  clob1 := dbms_metadata.fetch_clob(h);
 17  dbms_metadata.close(h);
```

```
 18
 19  -- Fetch SXML metadata for the second doc
 20  h    := dbms_metadata.open('TABLE');
 21  dbms_metadata.set_filter(h, 'NAME', 'TEST2');
 22  th    := dbms_metadata.add_transform(h, 'SXML');
 23  clob2 := dbms_metadata.fetch_clob(h);
 24  dbms_metadata.close(h);
 25
 26  -- Compare the two objects and generate an SXML difference
document
 27  och := dbms_metadata_diff.openc('TABLE');
 28  dbms_metadata_diff.add_document(och, clob1);
 29  dbms_metadata_diff.add_document(och, clob2);
 30  difdoc :=  dbms_metadata_diff.fetch_clob(och);
 31  dbms_metadata_diff.close(och);
 32
 33  -- Using the difdoc get the ALTER xmldoc
 34  h := dbms_metadata.openw('TABLE');
 35  th := dbms_metadata.add_transform(h,'ALTERXML');
 36  dbms_metadata.set_parse_item(h,'XPATH');
 37  dbms_metadata.set_parse_item(h,'ALTERABLE');
 38  dbms_metadata.set_parse_item(h,'CLAUSE_TYPE');
 39  dbms_metadata.set_parse_item(h,'NAME');
 40  dbms_metadata.set_parse_item(h,'COLUMN_ATTRIBUTE');
 41  dbms_metadata.set_parse_item(h,'CONSTRAINT_TYPE');
 42  dbms_metadata.set_parse_item(h,'CONSTRAINT_STATE');
 43  dbms_metadata.set_parse_item(h,'PARTITION_TYPE');
 44
 45  -- Get the alter xmldoc
 46  dbms_metadata.set_transform_param(th, 'BATCH_ALTER_DDL',
false);
 47  dbms_lob.createtemporary(altxml, TRUE);
 48  dbms_metadata.convert (h, difdoc, altxml);
 49  dbms_metadata.close(h);
 50
 51  -- convert the alter xmldoc to ddl
 52  h := dbms_metadata.openw('TABLE');
 53  th := dbms_metadata.add_transform(h,'ALTERDDL');
 54  dbms_lob.createtemporary(altddl, TRUE);
 55  dbms_metadata.convert (h, altxml, altddl);
 56
 57  -- display the ddl resulting from this compare
 58  dbms_output.put_line(altddl);
```

```
 59
 60  END;
 61  /
ALTER TABLE "SCOTT"."TEST1" ADD ("Y" VARCHAR2(40))
  ALTER TABLE "SCOTT"."TEST1" ADD ("T" VARCHAR2(30))
  ALTER TABLE "SCOTT"."TEST1" ADD ("Z" DATE)
  ALTER TABLE "SCOTT"."TEST1" RENAME TO "TEST2"

PL/SQL procedure successfully completed.

SQL> EXIT
$
```

A/ This is Oracle Database 12c behavior.

8. Drop the `SCOTT` schema account. Execute the `$HOME/labs/DB/cleanup_DDL.sh` shell script.

```
$ $HOME/labs/DB/cleanup_DDL.sh
…
$
```

Practices for Lesson 6:
Improving Performance

Practices for Lesson 6: Overview

Overview

In these practices, you configure and use Automatic In-Memory Management (AIM) and track in-memory expressions (IMEs) within a window capture.

Practice 6-1: Configuring and Using AIM

Overview

In this practice, you configure and use Automatic In-Memory (AIM) to avoid setting in-memory Automatic Data Optimization (ADO) policies.

Tasks

1. Before starting the practice, execute the `$HOME/labs/admin/glogin_6.sh` shell script. It sets formatting for all columns selected in queries.

```
$ $HOME/labs/admin/glogin_6.sh
…
$
```

2. Configure the in-memory column store size to 450M, and create and load in-memory tables `OE.PART`, `OE.SUPPLIER`, `OE.DATE_DIM`, `OE.CUSTOMER`, and `OE.LINEORDER`. Execute the `$HOME/labs/PERF/IM_tables.sh` shell script to complete these tasks.

```
$ $HOME/labs/PERF/IM_tables.sh
…
$
```

3. Verify that the in-memory column store size is set to 464M.

```
$ sqlplus / AS SYSDBA

SQL> SHOW PARAMETER inmemory_size

NAME                                 TYPE         VALUE
------------------------------------ ------------ ------------------------------
inmemory_size                        big integer 464M
SQL>
```

4. Verify that Automatic In-Memory (AIM) is not yet activated.

```
SQL> SHOW PARAMETER inmemory_automatic_level

NAME                                 TYPE         VALUE
------------------------------------ ------------ ------------------------------
inmemory_automatic_level             string       OFF
SQL>
```

5. Activate heat map statistics as you would when you define Automatic Data Optimization (ADO) policies that are able to evict in-memory segments from the IM column store when under pressure.

```
SQL> ALTER SYSTEM SET heat_map=ON;

System altered.
```

```
SQL>
```

6. Check whether in-memory tables are populated into the IM column store.

```
SQL> SELECT segment_name, bytes, inmemory_size,
            bytes_not_populated, populate_status
     FROM   v$im_segments;
 2    3
no rows selected

SQL>
```

Q/ Why are the in-memory tables not populated into the IM column store?

A/ The in-memory tables are not populated into the IM column store because AIM is not enabled and on-demand population has not been requested.

7. Perform an on-demand population of the `OE.PART`, `OE.SUPPLIER`, `OE.DATE_DIM`, `OE.CUSTOMER`, and `OE.LINEORDER` tables.

```
SQl> CONNECT system@PDB1
Enter password: password
Connected.
SQL> SELECT count(*) from oe.LINEORDER;

  COUNT(*)
----------
  13619440

SQL> SELECT count(*) from oe.PART;

  COUNT(*)
----------
   1600000

SQL> SELECT count(*) from oe.SUPPLIER;

  COUNT(*)
----------
     16000

SQL> SELECT count(*) from oe.CUSTOMER;

  COUNT(*)
```

```
----------
    240000

SQL> SELECT count(*) from oe.DATE_DIM;

  COUNT(*)
----------
      2554

SQL>
```

Q/ Are all in-memory tables populated into the IM column store?

```
SQL> SELECT segment_name, bytes, inmemory_size,
            bytes_not_populated, populate_status
     FROM   v$im_segments;
  2    3
SEGMENT_NAME       BYTES INMEMORY_SIZE BYTES_NOT_POPULATED
------------------------- ---------- ------------- ----------
POPULATE_STATUS
-----------------------------------
LINEORDER    1583767552     174653440          1180688384
STARTED


SQL>
```

After several minutes of loading, the populating status changes.

```
SQL> SELECT segment_name, bytes, inmemory_size,
            bytes_not_populated, populate_status
     FROM   v$im_segments;
 2    3
SEGMENT_NAME       BYTES INMEMORY_SIZE BYTES_NOT_POPULATED
------------------------- ---------- ------------- ----------
POPULATE_STATUS
-----------------------------------
LINEORDER    1583767552     337838080           828366848
OUT OF MEMORY

SQL>
```

A/ You discover that the OE.SUPPLIER table is not populated into the IM column store because there is not enough IM column store space for all in-memory segments.

8. Set the INMEMORY_AUTOMATIC_LEVEL initialization parameter to ensure that any hot segment that is not populated because of memory pressure is populated first.
   Note that the AIM action will happen when the IMCO background process wakes up to act. The IMCO cycle is 2 minutes.

```
SQL> ALTER SYSTEM SET inmemory_automatic_level = MEDIUM;

System altered.

SQL> EXIT
$
```

9. Now execute the $HOME/labs/PERF/loop.sh shell script. The script executes many SELECT statements on the OE.SUPPLIER table. This generates heat map statistics that will trigger an eviction AIM action of the cold segment, OE.LINEORDER, and populate action of other hotter segments.

   Note: Because there is no ADO policy on tables allowing eviction and therefore overriding AIM, AIM can submit tasks to evict the cold segments and populate hotter segments.

```
$ $HOME/labs/PERF/loop.sh
…
$
```

10. Check that the task in charge of evicting the in-memory segment, OE.LINEORDER, and populating hotter in-memory segments is running. You will have to reiterate the query until the IMCO background process wakes up to act.

```
$ sqlplus system@PDB1

Enter password: password

SQL> SELECT object_name, action, im.status
     FROM   v$im_adotaskdetails im, dba_objects o
     WHERE im.obj# = o.object_id;
  2    3
OBJECT_NAME      ACTION           STATUS
-------------------------------- ---------------------------------------- ------------------------
  DATE_DIM      POPULATE          DONE
  CUSTOMER      POPULATE          DONE
 LINEORDER      POPULATE          PROCESSING
  SUPPLIER      POPULATE          DONE
      PART      POPULATE          DONE

SQL>
```

```
SQL> SELECT object_name, action, im.status
     FROM   v$im_adotaskdetails im, dba_objects o
     WHERE im.obj# = o.object_id;
  2    3
OBJECT_NAME    ACTION            STATUS
-------------------------------- ------------------------------ ------------------------
LINEORDER   PARTIAL   POPULATE   DONE
LINEORDER     PARTIAL  POPULATE  DONE
PART            POPULATE         DONE
PART            POPULATE         DONE
SUPPLIER        POPULATE         DONE
SUPPLIER        POPULATE         DONE
CUSTOMER        POPULATE         DONE
CUSTOMER        POPULATE         DONE
DATE_DIM        POPULATE         DONE
DATE_DIM        POPULATE         DONE

10 rows selected.

SQL>
```

11. Find the evicted in-memory segment and verify that the `OE.LINEORDER` table is now fully populated into the IM column store.

```
SQL> SELECT segment_name, bytes, inmemory_size,
            bytes_not_populated, populate_status
     FROM   v$im_segments;
 2    3
SEGMENT_NAME       BYTES INMEMORY_SIZE BYTES_NOT_POPULATED
-------------------------------- ---------- ------------- ------------------------------
POPULATE_STATUS
-----------------------------------
DATE_DIM          319488       1310720                     0
COMPLETED

PART           166526976      35258368                     0
COMPLETED

CUSTOMER        30670848       7602176                     0
COMPLETED

LINEORDER    1583767552     271908864           945807360
COMPLETED
```

```
SUPPLIER         1974272       1310720                    0
COMPLETED

SQL>
```

Q/ Why are there more and more actions from the IMCO background process whereas the hottest in-memory segments have been populated?

```
SQL> SELECT im.obj#, object_name, action, im.status
     FROM   v$im_adotaskdetails im, dba_objects o
     WHERE im.obj# = o.object_id;
  2    3
      OBJ# OBJECT_NAME    ACTION                         STATUS
---------- -------------- ------------------------------ ------------------------
     77666 LINEORDER      PARTIAL POPULATE DONE
     77666 LINEORDER      PARTIAL POPULATE DONE
     77666 LINEORDER      PARTIAL POPULATE DONE
     77667 PART           POPULATE         DONE
     77667 PART           POPULATE         DONE
     77667 PART           POPULATE         DONE
     77668 SUPPLIER       POPULATE         DONE
     77668 SUPPLIER       POPULATE         DONE
     77668 SUPPLIER       POPULATE         DONE
     77669 CUSTOMER       POPULATE         DONE
     77669 CUSTOMER       POPULATE         DONE
     77669 CUSTOMER       POPULATE         DONE
     77670 DATE_DIM       POPULATE         DONE
     77670 DATE_DIM       POPULATE         DONE
     77670 DATE_DIM       POPULATE         DONE

15 rows selected.

SQL> SELECT object_name, action, im.status
     FROM   v$im_adotaskdetails im, dba_objects o
     WHERE  im.obj# = o.object_id;
  2    3
OBJECT_NAME    ACTION                         STATUS
-------------- ------------------------------ ------------------------
LINEORDER   PARTIAL   POPULATE   DONE
LINEORDER   PARTIAL   POPULATE   DONE
LINEORDER      PARTIAL POPULATE DONE
LINEORDER      PARTIAL POPULATE DONE
```

```
PART            POPULATE          DONE
PART            POPULATE          DONE
PART            POPULATE          DONE
PART            POPULATE          DONE
SUPPLIER        POPULATE          DONE
SUPPLIER        POPULATE          DONE
SUPPLIER        POPULATE          DONE
SUPPLIER        POPULATE          DONE
CUSTOMER        POPULATE          DONE
CUSTOMER        POPULATE          DONE
CUSTOMER        POPULATE          DONE
CUSTOMER        POPULATE          DONE
DATE_DIM        POPULATE          DONE
DATE_DIM        POPULATE          DONE
DATE_DIM        POPULATE          DONE
DATE_DIM        POPULATE          DONE

20 rows selected.

SQL> EXIT
$
```

A/ The IMCO background process regularly checks whether there are no hotter in-memory segments to populate.

Practice 6-2: Tracking IM Expressions Within a Window Capture

Overview

In this practice, you track expressions populated into the in-memory column store within a defined capture window.

Tasks

1. Execute the `$HOME/labs/PERF/IM_tables.sh` shell script. It configures the in-memory column store size, and creates and loads in-memory tables. Even if the script was already executed in Practice 6-1, re-execute it.

```
$ $HOME/labs/PERF/IM_tables.sh
…
$
```

   In case you encounter disk space issues with an Archiver error, proceed with the following commands answering `YES` to both:

```
$ rman target /

RMAN> DELETE ARCHIVELOG ALL;
…
RMAN> DELETE OBSOLETE;
…
RMAN> EXIT;
$
```

2. In the current session called *Session1*, create another in-memory small table.

```
$ sqlplus system@PDB1

Enter password: password

SQL> CREATE TABLE OE.test ( c1 NUMBER, c2 NUMBER,
                            vc1 AS (c1+c2), vc2 AS (c1*2))
                    INMEMORY;
  2    3
Table created.

SQL> INSERT INTO OE.test (c1, c2) VALUES (1,2);

1 row created.

SQL> INSERT INTO OE.test (c1, c2) VALUES (2,3);
```

```
1 row created.

SQL> COMMIT;

Commit complete.

SQL>
```

3. Execute the `$HOME/labs/PERF/loop2.sql` SQL script that contains queries with expressions on the `OE.LINEORDER`, `OE.DATE_DIM`, and `OE.TEST` in-memory tables.

```
SQL> @$HOME/labs/PERF/loop2.sql
…
SQL>
```

4. In another terminal window, called *Session2*, while the loop is still executing, track expressions within a capture window. Log in to `PDB1` as `OE`.

```
$ sqlplus oe@PDB1
Enter password: password
Connected.
SQL>
```

a. In *Session2*, open a capture window to signal the beginning of an expression capture window.

```
SQL> exec DBMS_INMEMORY_ADMIN.IME_OPEN_CAPTURE_WINDOW()

PL/SQL procedure successfully completed.

SQL>
```

b. In *Session2,* get the current capture state of the expression capture window and the time stamp of the most recent modification.

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
       P_CAPTURE_STATE           varchar2(40);
       P_LAST_MODIFIED           timestamp;
     BEGIN
    DBMS_INMEMORY_ADMIN.IME_GET_CAPTURE_STATE(P_CAPTURE_STATE,
P_LAST_MODIFIED);
    dbms_output.put_line('-----------------------------------');
    dbms_output.put_line('State = '||P_CAPTURE_STATE);
    dbms_output.put_line('Date = '||P_LAST_MODIFIED);
    dbms_output.put_line('-----------------------------------');
    END;
/
-----------------------------------------------
State = OPEN
```

```
Date  = 05-DEC-17 09.05.09.162892 AM
-----------------------------------------------

PL/SQL procedure successfully completed.

SQL>
```

c. In *Session2*, close the window to signal the end of the expression capture window.

```
SQL> exec DBMS_INMEMORY_ADMIN.IME_CLOSE_CAPTURE_WINDOW()

PL/SQL procedure successfully completed.

SQL>
```

Q/ What is the state of the capture?

```
SQL> DECLARE
       P_CAPTURE_STATE           varchar2(40);
       P_LAST_MODIFIED           timestamp;
     BEGIN
    DBMS_INMEMORY_ADMIN.IME_GET_CAPTURE_STATE(P_CAPTURE_STATE,
P_LAST_MODIFIED);
    dbms_output.put_line('-----------------------------------');
    dbms_output.put_line('State = '||P_CAPTURE_STATE);
    dbms_output.put_line('Date = '||P_LAST_MODIFIED);
    dbms_output.put_line('-----------------------------------');
    END;
/
-----------------------------------------------
State = CLOSE
Date  = 05-DEC-17 10.20.06.273889 AM
-----------------------------------------------

PL/SQL procedure successfully completed.

SQL>
```

A/ The state displays CLOSE.

d. Still in *Session2*, add all the hot expressions captured in the previous window into the IM column store. The database considers statistics for expressions tracked in the most recent capture window.

```
SQL> exec DBMS_INMEMORY_ADMIN.IME_CAPTURE_EXPRESSIONS('WINDOW')

PL/SQL procedure successfully completed.

SQL>
```

5. Display expression statistics.
   a. Still in *Session2,* display the statistics of expressions that have been tracked in the current window.

```
SQL> SELECT owner, table_name, evaluation_count AS COUNT,
            created, expression_text
     FROM   dba_expression_statistics
     WHERE  snapshot = 'WINDOW'
     AND    owner='OE';
 2    3    4    5
OWNER  TABLE_NAM       COUNT CREATED
-------------- ---------- ---------- ---------
EXPRESSION_TEXT
--------------------------------------------------------------------------------
OE     LINEORDER     108592 05-DEC-17
"LO_ORDERDATE"

OE     LINEORDER 450209408 05-DEC-17
"LO_DISCOUNT"

OE     LINEORDER 450209408 05-DEC-17
"LO_QUANTITY"

OE     LINEORDER 450209408 05-DEC-17
"LO_SHIPPRIORITY"

OE     LINEORDER 450209408 05-DEC-17
"LO_SHIPMODE"

OE     LINEORDER 450209408 05-DEC-17
"LO_EXTENDEDPRICE"

OE     LINEORDER 450209408 05-DEC-17
SYS_OP_BLOOM_FILTER(:BF0000,"LO_ORDERDATE")
```

```
OE     LINEORDER 450209408 05-DEC-17
"LO_TAX"

OE     LINEORDER 443278176 05-DEC-17
"LO_EXTENDEDPRICE"*(1-"LO_DISCOUNT")

OE     LINEORDER 443278176 05-DEC-17
1+"LO_TAX"

OE     DATE_DIM      217184 05-DEC-17
"D_DATEKEY"

OE     DATE_DIM      108592 05-DEC-17
TO_DATE(TO_CHAR("D_DATEKEY"),'YYYY-MM-DD')

OE     TEST               88 05-DEC-17
"C1"*2

OE     TEST              176 05-DEC-17
"C1"

OE     TEST              176 05-DEC-17
"C2"

OE     TEST               88 05-DEC-17
"C1"+"C2"


16 rows selected.

SQL>
```

Expression Statistics Store (ESS) information is stored in the data dictionary. The `DBA_EXPRESSION_STATISTICS` view shows the metadata that the optimizer has collected in the ESS.

b. In-Memory expressions are exposed as system-generated virtual columns, prefixed by the string `SYS_IME`, in the `USER_IM_EXPRESSIONS` view. The result may differ from the result below. It depends on the frequency and statistics collected.

```
SQL> SELECT table_name, column_name, sql_expression
     FROM   user_im_expressions;
 2
```

```
TABLE_NAM COLUMN_NAME
--------- ----------------------------------------------------------
SQL_EXPRESSION
--------------------------------------------------------------------------------------------------------------

DATE_DIM SYS_IME00010000000B474F
TO_DATE(TO_CHAR("D_DATEKEY"),'YYYY-MM-DD')

LINEORDER SYS_IME00010000000B4753
1+"LO_TAX"

LINEORDER SYS_IME00010000000B4754
"LO_EXTENDEDPRICE"*(1-"LO_DISCOUNT")

LINEORDER SYS_IME00010000000B475C
"LO_EXTENDEDPRICE"*(1-"LO_DISCOUNT")*(1+"LO_TAX")

LINEORDER SYS_IME0001000000099170
"LO_EXTENDEDPRICE"*(1-"LO_DISCOUNT")

SQL>
```

6. Still in *Session2*, clear the expression virtual columns. Before that, interrupt the loop in *Session1* in case it is still executing.
   a. Clear all `SYS_IME` expression virtual columns in the database, including those tracked in the last window.

```
SQL> exec DBMS_INMEMORY_ADMIN.IME_DROP_ALL_EXPRESSIONS()

PL/SQL procedure successfully completed.

SQL>
```

   b. Check that all `SYS_IME` expression virtual columns in the database are cleared.

   Q/ Are the expressions statistics that have been tracked in the current window cleared?

```
SQL> SELECT table_name, column_name, sql_expression
     FROM   user_im_expressions;
  2
no rows selected

SQL>
```

   A/ Yes.

Q/ Are the expressions statistics that have been tracked in the previous window cleared?

```
SQL> SELECT owner, table_name, evaluation_count AS COUNT,
            created, expression_text
     FROM   dba_expression_statistics
     WHERE  snapshot = 'WINDOW'
     AND    owner='OE';
 2    3    4    5
OWNER  TABLE_NAM       COUNT CREATED
-------------- -------------------- ---------- --------------------
EXPRESSION_TEXT
--------------------------------------------------------------------------------
OE     LINEORDER     108592 05-DEC-17
"LO_ORDERDATE"

OE     LINEORDER 450209408 05-DEC-17
"LO_DISCOUNT"

OE     LINEORDER 450209408 05-DEC-17
"LO_QUANTITY"

OE     LINEORDER 450209408 05-DEC-17
"LO_SHIPPRIORITY"

OE     LINEORDER 450209408 05-DEC-17
"LO_SHIPMODE"

OE     LINEORDER 450209408 05-DEC-17
"LO_EXTENDEDPRICE"

OE     LINEORDER 450209408 05-DEC-17
SYS_OP_BLOOM_FILTER(:BF0000,"LO_ORDERDATE")

OE     LINEORDER 450209408 05-DEC-17
"LO_TAX"

OE     LINEORDER 443278176 05-DEC-17
"LO_EXTENDEDPRICE"*(1-"LO_DISCOUNT")

OE     LINEORDER 443278176 05-DEC-17
1+"LO_TAX"
```

```
OE     DATE_DIM      217184 05-DEC-17
"D_DATEKEY"

OE     DATE_DIM      108592 05-DEC-17
TO_DATE(TO_CHAR("D_DATEKEY"),'YYYY-MM-DD')

OE     TEST              88 05-DEC-17
"C1"*2

OE     TEST             176 05-DEC-17
"C1"

OE     TEST             176 05-DEC-17
"C2"

OE     TEST              88 05-DEC-17
"C1"+"C2"


16 rows selected.

SQL>
```

A/ No.

Q/ What will happen when you open a new capture window?

```
SQL> exec DBMS_INMEMORY_ADMIN.IME_OPEN_CAPTURE_WINDOW()

PL/SQL procedure successfully completed.

SQL>
```

```
SQL> SELECT owner, table_name, evaluation_count AS COUNT,
            created, expression_text
     FROM   dba_expression_statistics
     WHERE  snapshot = 'WINDOW'
     AND    owner='OE';
  2    3    4    5
no rows selected
```

```
SQL> EXIT
$
```

A/ New expressions will be tracked and collected within the new capture window.

7. Quit *Session1*.

```
SQL> EXIT
$
```

8. Execute the `$HOME/labs/PERF/cleanup_IM_tables.sh` shell script to drop `in-memory tables in PDB1 and disable the IM column store usage`.

```
$ $HOME/labs/PERF/cleanup_IM_tables.sh
…
$
```

Practices for Lesson 7:
Handling Enhancements in Big Data and Data Warehousing

Practices for Lesson 7: Overview

Overview

In these practices, you query inlined external tables, populate external tables into the IM column store, use hierarchy-based predicates and calculated measures on analytic views, and use polymorphic table functions (PTFs).

Practice 7-1: Querying Inlined External Tables

Overview

In this practice, you query inlined external tables without creating a persistent object in the data dictionary.

Tasks

1. Before starting the practice, execute the `$HOME/labs/admin/glogin_7.sh` shell script. It sets formatting for all columns selected in queries.

```
$ $HOME/labs/admin/glogin_7.sh
…
$
```

2. Execute the `$HOME/labs/DW/ext_tables.sh` shell script. The script completes the following operations:
   - Creates the `TEST` user
   - Grants the `TEST` user the `required` privileges
   - Creates the `ext_dir` directory `AS '/home/oracle/labs'`
   - Grants `READ` and `WRITE` privileges to `TEST`

```
$ $HOME/labs/DW/ext_tables.sh
…
$
```

3. Display the content of the `/home/oracle/labs/DW/empext1.dat` external data file.

```
$ cat /home/oracle/labs/DW/empext1.dat
114,Den,Raphaely,DRAPHEAL,515.127.4561,07-DEC-
02,PU_MAN,11000,,100,30
115,Alexander,Khoo,AKHOO,515.127.4562,18-MAY-
03,PU_CLERK,3100,,114,30
116,Shelli,Baida,SBAIDA,515.127.4563,24-DEC-
05,PU_CLERK,2900,,114,30
117,Sigal,Tobias,STOBIAS,515.127.4564,24-JUL-
05,PU_CLERK,2800,,114,30
…
199,Douglas,Grant,DGRANT,650.507.9844,13-JAN-
08,SH_CLERK,2600,,124,50
200,Jennifer,Whalen,JWHALEN,515.123.4444,17-SEP-
03,AD_ASST,4400,,101,10
201,Michael,Hartstein,MHARTSTE,515.123.5555,17-FEB-
04,MK_MAN,13000,,100,20
202,Pat,Fay,PFAY,603.123.6666,17-AUG-05,MK_REP,6000,,201,20
203,Susan,Mavris,SMAVRIS,515.123.7777,07-JUN-
02,HR_REP,6500,,101,40
$
```

4. Connect as `TEST` and query the inlined external table.

```
$ sqlplus test@PDB1

Enter password: password

SQL> SELECT ext_emp.employee_id, ext_emp.first_name,
            ext_emp.last_name
     FROM EXTERNAL
     (( EMPLOYEE_ID NUMBER(6,0), FIRST_NAME VARCHAR2(20),
        LAST_NAME VARCHAR2(25), EMAIL VARCHAR2(25),
        PHONE_NUMBER VARCHAR2(20), HIRE_DATE DATE,
        JOB_ID VARCHAR2(10), SALARY NUMBER(8,2),
        COMMISSION_PCT NUMBER(2,2), MANAGER_ID NUMBER(6,0),
        DEPARTMENT_ID NUMBER(4,0))
     TYPE ORACLE_LOADER
     DEFAULT DIRECTORY ext_dir
     ACCESS PARAMETERS
      ( records delimited by newline
        badfile ext_dir:'empxt%a_%p.bad'
        logfile ext_dir:'empxt%a_%p.log'
        fields terminated by ','
               missing field values are null
          ( employee_id, first_name, last_name, email,
            phone_number, hire_date, job_id, salary,
            commission_pct, manager_id, department_id)
      )
      LOCATION ('empext1.dat')
      REJECT LIMIT UNLIMITED
    ) ext_emp;
2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17
18  19  20  21  22  23  24  25

EMPLOYEE_ID FIRST_NAME           LAST_NAME
----------- -------------------- -------------------------
        114 Den                  Raphaely
        115 Alexander            Khoo
…
        201 Michael              Hartstein
...
        202 Pat                  Fay
        203 Susan                Mavris

2078 rows selected.
```

```
SQL>
```

5. Query the inlined external table and display the number of employees for each department. Order the result by department number.

```
SQL> SELECT ext_emp.department_id, count(*) AS dept_count
     FROM EXTERNAL
     (( EMPLOYEE_ID NUMBER(6,0), FIRST_NAME VARCHAR2(20),
        LAST_NAME VARCHAR2(25), EMAIL VARCHAR2(25),
        PHONE_NUMBER VARCHAR2(20), HIRE_DATE DATE,
        JOB_ID VARCHAR2(10), SALARY NUMBER(8,2),
        COMMISSION_PCT NUMBER(2,2), MANAGER_ID NUMBER(6,0),
        DEPARTMENT_ID NUMBER(4,0))
     TYPE ORACLE_LOADER
     DEFAULT DIRECTORY ext_dir
     ACCESS PARAMETERS
      ( records delimited by newline
        badfile ext_dir:'empxt%a_%p.bad'
        logfile ext_dir:'empxt%a_%p.log'
        fields terminated by ','
               missing field values are null
          ( employee_id, first_name, last_name, email,
            phone_number, hire_date, job_id, salary,
            commission_pct, manager_id, department_id)
      )
      LOCATION ('empext1.dat')
      REJECT LIMIT UNLIMITED
    ) ext_emp
    GROUP BY ext_emp.department_id
    ORDER  BY ext_emp.department_id;
 2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17
18  19  20  21  22  23  24  25
DEPARTMENT_ID DEPT_COUNT
------------- ----------
           10        244
           20        507
           30        123
           40        257
           50        945
         4114          1
                       1

7 rows selected.
```

```
SQL> EXIT
$
```

Q/ Should you drop the inlined external table?

A/ There is no table to drop in the database because there is no persistent object in the data dictionary associated to the inlined external table.

Practice 7-2: Populating External Tables in the In-Memory Column Store

Overview

In this practice, you create, populate, and then query in-memory external tables.

Tasks

1. Because the in-memory external table loads its data from the `/home/oracle/labs/DW/empext1.dat` external file and requires space in the IM column store, execute the `$HOME/labs/DW/IM_ext_tables.sh` shell script that completes the following operations:
   - Configures the in-memory column store size to 750M
   - Creates the `TEST` user
   - Grants `TEST` the `required` privileges
   - Creates the `ext_dir` directory
   - Grants `READ` and `WRITE` privileges to `TEST`

```
$ $HOME/labs/DW/IM_ext_tables.sh
…
$
```

2. Log in to `PDB1` as `TEST` and create the in-memory external table.

```
$ sqlplus test@PDB1

Enter password: password

SQL> CREATE TABLE test.ext_emp (
        id NUMBER(6), first_name VARCHAR2(20),
        last_name VARCHAR2(25), email VARCHAR2(25),
        phone_number VARCHAR2(20), hire_date DATE,
        job_id VARCHAR2(10), salary NUMBER(8,2),
        commission_pct NUMBER(2,2), manager_id NUMBER(6),
        department_id NUMBER(4))
     ORGANIZATION EXTERNAL
    (TYPE ORACLE_LOADER DEFAULT DIRECTORY ext_dir
     ACCESS PARAMETERS
            ( records delimited by newline
              badfile ext_dir:'empxt%a_%p.bad'
              logfile ext_dir:'empxt%a_%p.log'
              fields terminated by ','
                     missing field values are null
             (ID, FIRST_NAME, LAST_NAME, EMAIL, PHONE_NUMBER,
              HIRE_DATE, JOB_ID, SALARY, COMMISSION_PCT,
```

```
               MANAGER_ID, DEPARTMENT_ID))
     LOCATION ('empext1.dat')
    )
    REJECT LIMIT UNLIMITED
    INMEMORY;
 2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17
18  19  20  21  22
Table created.

SQL>
```

3. Verify that the external table is an in-memory object.

```
SQL> SELECT table_name, inmemory, inmemory_compression
     FROM   user_external_tables;
 2
TABLE_NAME             INMEMORY INMEMORY_COMPRESS
---------------------- -------- ----------------
EXT_EMP                ENABLED FOR QUERY LOW

SQL>
```

4. Populate in-memory external tables by counting the number of rows in the table.

```
SQL> SELECT count(*) FROM test.ext_emp;

  COUNT(*)
----------
      2078

SQL>
```

Q/ Is the table populated into IMCS after the query? (Connect as `SYSTEM` to verify)

```
SQL> CONNECT system@PDB1
Enter password: password
Connected.
SQL> SELECT owner, segment_name, populate_status, con_id
     FROM  v$im_segments;
 2
no rows selected

SQL>
```

A/ No.

a. Populate the external table into the IM column store.

```
SQL> EXEC dbms_inmemory.populate ('TEST','EXT_EMP')

PL/SQL procedure successfully completed.

SQL>
```

b. Verify that the segment is populated into the IM column store.

```
SQL> SELECT owner, segment_name, populate_status, con_id
     FROM   v$im_segments;
 2
OWNER           SEGMENT_NAME POPULATE_STAT      CON_ID
--------------- ------------ --------------- ----------
TEST            EXT_EMP      COMPLETED               4

SQL>
```

5. Query in-memory external tables.

a. Log in to `PDB1` as user `TEST` and query the in-memory external table.

```
SQL> CONNECT test@PDB1
Enter password: password
Connected.
SQL> SELECT count(*) FROM test.ext_emp;

  COUNT(*)
----------
      2078

SQL>
```

b. Check whether the in-memory external table data is queried from the IM column store or buffer cache.

```
SQL> SELECT * FROM dbms_xplan.display_cursor();

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------
SQL_ID 7382qx3bj1x4n, child number 0
-------------------------------------
SELECT count(*) FROM test.ext_emp

Plan hash value: 546827939

------------------------------------------------------------------------------------------------------------------------
```

```
| Id  | Operation| Name              |  Rows   |  Cost
(%CPU)| Time     |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |         |       |   341
(100)|          |
|   1 |  SORT AGGREGATE             |         |     1 |
|          |
|   2 |   EXTERNAL TABLE ACCESS FULL| EXT_EMP |   102K|   341
(1)| 00:00:01 |
-------------------------------------------------------------------------------------------------------------------


14 rows selected.


SQL>
```

The execution plan shows that the table was accessed from the buffer cache.

c. To access the in-memory table data from the IM column store, set the `QUERY_REWRITE_INTEGRITY` parameter to `STALE_TOLERATED` in your session.

```
SQL> ALTER SESSION SET
                  QUERY_REWRITE_INTEGRITY = STALE_TOLERATED;
  2
Session altered.


SQL>
```

d. Re-query the table.

```
SQL> SELECT count(*) FROM test.ext_emp;


  COUNT(*)
----------
      2078


SQL>
```

Q/ Does the execution plan show that the table was still accessed from the buffer cache?

```
SQL> SELECT * FROM dbms_xplan.display_cursor();

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------------------------------
SQL_ID  7382qx3bj1x4n, child number 0
```

```
-------------------------------------------------------------------------------
SELECT count(*) FROM test.ext_emp

Plan hash value: 546827939

-------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                | Name    | Rows
| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |         |
    |   341 (100)|          |
|   1 |  SORT AGGREGATE                          |         |
    1 |            |          |
|   2 |   EXTERNAL TABLE ACCESS INMEMORY FULL | EXT_EMP |
102K|   341   (1)| 00:00:01 |
-------------------------------------------------------------------------------------------------------------------------------

14 rows selected.

SQL>
```

A/ No. The execution plan shows that the table is now accessed from the IM column store.

6. Update the external table to observe how queries on in-memory external tables behave.
   a. Add records to the external file.

```
SQL> host echo "202,Pat,Fay,PFAY,603.123.6666,17-AUG-
05,MK_REP,6000,,201,20" >> /home/oracle/labs/DW/empext1.dat

SQL> host echo "203,Susan,Mavris,SMAVRIS,515.123.7777,07-JUN-
02,HR_REP,6500,,101,40" >> /home/oracle/labs/DW/empext1.dat

SQL>
```

Q/ Is the in-memory external table automatically repopulated into the IM column store?

```
SQL> SELECT count(*) FROM test.ext_emp;

  COUNT(*)
----------
      2078

SQL>
```

A/ No. The query still displays the same number of employees.

b. Repopulate the in-memory external table.

```
SQL> EXEC dbms_inmemory.repopulate ('TEST','EXT_EMP')

PL/SQL procedure successfully completed.

SQL>
```

c. Verify that the in-memory external table is repopulated into the IM column store.

```
SQL> SELECT count(*) FROM test.ext_emp;

  COUNT(*)
----------
      2080

SQL>
```

7. Look at the system statistics related to external in-memory segments.

```
SQL> CONNECT system@PDB1
Enter password: password
Connected.
SQL> SELECT display_name, value FROM v$sysstat s, v$statname n
     WHERE s.statistic# = n.statistic#
     AND (display_name LIKE 'IM XT%'
          OR display_name LIKE 'IM%external%')
     AND value <> 0;
 2   3   4   5
DISPLAY_NAME                                             VALUE
-------------------------------------------------------- ----------
IM populate external table read time (ms)                   886
IM XT populate rows                                        4158
IM XT populate CUs                                            2
IM XT populate segments                                       2

SQL> EXIT
$
```

8. Execute the `$HOME/labs/DW/cleanup_IM_tables.sh` shell script to drop `the in-memory external table`.

```
$ $HOME/labs/DW/cleanup_IM_tables.sh
…
$
```

Practice 7-3: Using Hierarchy-Based Predicates and Calculated Measures on Analytic Views

Overview

In this practice, you query analytic views (AVs) with SQL by using the same capabilities as Microsoft's Multidimensional Expression (MDX) language, or by using the `DBMS_MDX_ODBO` package, the MDX interface provided by PL/SQL.

Tasks

1. Execute the `$HOME/labs/DW/AV.sh` shell script. The script creates an analytic view (AV) based on fact and dimension tables, all loaded with data.

```
$ $HOME/labs/DW/AV.sh
…
$
```

2. Query the AV.
   a. Look at the `TIME_HIER` hierarchy.

```
$ sqlplus av@PDB1

Enter password: password

SQL> SELECT year_name, quarter_name, month_name, depth,
            parent_level_name
     FROM   time_hier;
  2    3
YEAR_NAM QUARTER_ MONTH_NA       DEPTH PARENT_
-------- -------- -------- ---------- ------
CY2011   Q1CY2011 Feb-11            3 QUARTER
CY2011   Q1CY2011 Jan-11            3 QUARTER
CY2011   Q1CY2011 Mar-11            3 QUARTER
CY2012   Q1CY2012 Mar-12            3 QUARTER
CY2012   Q1CY2012 Jan-12            3 QUARTER
CY2012   Q1CY2012 Feb-12            3 QUARTER
CY2013   Q1CY2013 Jan-13            3 QUARTER
CY2013   Q1CY2013 Feb-13            3 QUARTER
CY2013   Q1CY2013 Mar-13            3 QUARTER
CY2014   Q1CY2014 Feb-14            3 QUARTER
CY2014   Q1CY2014 Mar-14            3 QUARTER
CY2014   Q1CY2014 Jan-14            3 QUARTER
CY2015   Q1CY2015 Feb-15            3 QUARTER
CY2015   Q1CY2015 Jan-15            3 QUARTER
CY2015   Q1CY2015 Mar-15            3 QUARTER
```

```
CY2011   Q2CY2011 Jun-11          3 QUARTER
CY2011   Q2CY2011 May-11          3 QUARTER
CY2011   Q2CY2011 Apr-11          3 QUARTER
CY2012   Q2CY2012 Jun-12          3 QUARTER
CY2012   Q2CY2012 Apr-12          3 QUARTER
CY2012   Q2CY2012 May-12          3 QUARTER
CY2013   Q2CY2013 May-13          3 QUARTER
CY2013   Q2CY2013 Jun-13          3 QUARTER
CY2013   Q2CY2013 Apr-13          3 QUARTER
CY2014   Q2CY2014 May-14          3 QUARTER
CY2014   Q2CY2014 Jun-14          3 QUARTER
CY2014   Q2CY2014 Apr-14          3 QUARTER
CY2015   Q2CY2015 Jun-15          3 QUARTER
CY2015   Q2CY2015 May-15          3 QUARTER
CY2015   Q2CY2015 Apr-15          3 QUARTER
CY2011   Q3CY2011 Jul-11          3 QUARTER
CY2011   Q3CY2011 Sep-11          3 QUARTER
CY2011   Q3CY2011 Aug-11          3 QUARTER
CY2012   Q3CY2012 Aug-12          3 QUARTER
CY2012   Q3CY2012 Sep-12          3 QUARTER
CY2012   Q3CY2012 Jul-12          3 QUARTER
CY2013   Q3CY2013 Sep-13          3 QUARTER
CY2013   Q3CY2013 Aug-13          3 QUARTER
CY2013   Q3CY2013 Jul-13          3 QUARTER
CY2014   Q3CY2014 Jul-14          3 QUARTER
CY2014   Q3CY2014 Sep-14          3 QUARTER
CY2014   Q3CY2014 Aug-14          3 QUARTER
CY2015   Q3CY2015 Aug-15          3 QUARTER
CY2015   Q3CY2015 Jul-15          3 QUARTER
CY2015   Q3CY2015 Sep-15          3 QUARTER
CY2011   Q4CY2011 Dec-11          3 QUARTER
CY2011   Q4CY2011 Oct-11          3 QUARTER
CY2011   Q4CY2011 Nov-11          3 QUARTER
CY2012   Q4CY2012 Nov-12          3 QUARTER
CY2012   Q4CY2012 Oct-12          3 QUARTER
CY2012   Q4CY2012 Dec-12          3 QUARTER
CY2013   Q4CY2013 Dec-13          3 QUARTER
CY2013   Q4CY2013 Nov-13          3 QUARTER
CY2013   Q4CY2013 Oct-13          3 QUARTER
CY2014   Q4CY2014 Dec-14          3 QUARTER
CY2014   Q4CY2014 Oct-14          3 QUARTER
CY2014   Q4CY2014 Nov-14          3 QUARTER
```

```
CY2015   Q4CY2015 Oct-15            3 QUARTER
CY2015   Q4CY2015 Dec-15            3 QUARTER
CY2015   Q4CY2015 Nov-15            3 QUARTER
CY2011   Q1CY2011                   2 YEAR
CY2012   Q1CY2012                   2 YEAR
CY2013   Q1CY2013                   2 YEAR
CY2014   Q1CY2014                   2 YEAR
CY2015   Q1CY2015                   2 YEAR
CY2011   Q2CY2011                   2 YEAR
CY2012   Q2CY2012                   2 YEAR
CY2013   Q2CY2013                   2 YEAR
CY2014   Q2CY2014                   2 YEAR
CY2015   Q2CY2015                   2 YEAR
CY2011   Q3CY2011                   2 YEAR
CY2012   Q3CY2012                   2 YEAR
CY2013   Q3CY2013                   2 YEAR
CY2014   Q3CY2014                   2 YEAR
CY2015   Q3CY2015                   2 YEAR
CY2011   Q4CY2011                   2 YEAR
CY2012   Q4CY2012                   2 YEAR
CY2013   Q4CY2013                   2 YEAR
CY2014   Q4CY2014                   2 YEAR
CY2015   Q4CY2015                   2 YEAR
CY2011                              1 ALL
CY2012                              1 ALL
CY2013                              1 ALL
CY2014                              1 ALL
CY2015                              1 ALL
                                    0

86 rows selected.

SQL>
```

b. Query the AV to find the values of sales for all years, per year and per quarter.

```
SQL> SELECT time_hier.member_name, sales
     FROM   sales_av HIERARCHIES(time_hier)
     WHERE  time_hier.level_name IN  ('ALL','YEAR','QUARTER')
     ORDER BY time_hier.hier_order;
 2    3    4
MEMBER_N       SALES
-------------------- --------------------------

ALL       1970258105
```

```
CY2011    593507775
Q1CY2011    7068888
Q2CY2011  556371578
Q4CY2011   30067309
CY2012    568622304
Q2CY2012  564675917
Q4CY2012    3946387
CY2013    611827904
Q1CY2013    5327980
Q2CY2013  592275258
Q3CY2013   10287982
Q4CY2013    3936684
CY2014    196300122
Q1CY2014    5327980
Q2CY2014  183612568
Q3CY2014    7359574

17 rows selected.

SQL>
```

c. Query the AV to find the values of sales for all months for the first and second quarters of all years.

```
SQL> SELECT time_hier.member_name, sales
     FROM   sales_av HIERARCHIES(time_hier)
     WHERE  time_hier.level_name = 'MONTH'
     AND    TO_CHAR(time_hier.month_end_date,'Q') IN (1,2)
     ORDER BY time_hier.hier_order;
 2    3    4    5
MEMBER_N       SALES
-------------------- ------------------------
Jan-11       1740908
Feb-11       5327980
Apr-11     556371578
Apr-12     564675917
Mar-13       5327980
Apr-13     590711509
May-13       1563749
Feb-14       5327980
Apr-14     183612568

9 rows selected.

SQL>
```

3. Use filter-before aggregate predicates to query the AV. The `USING` and `FILTER FACT` clauses allow a query to filter fact rows before being aggregated by the analytic view. Use the `FILTER FACT` clause to change the values of aggregate-level hierarchy members. The `USING` clause can be thought of as an inner query, similar to a `WITH` clause.

   Report sales at the year and quarter levels, but only for the first half of each year. This example filters months where the quarter of the year of the `MONTH_END_DATE` attribute is 1 or 2. The hierarchy used is `TIME_HIER`.

```
SQL> SELECT time_hier.member_name, sales
     FROM ANALYTIC VIEW (
     USING sales_av HIERARCHIES(time_hier)
     FILTER FACT (time_hier TO level_name = 'MONTH'
                  AND TO_CHAR(month_end_date,'Q') IN (1,2)))
     WHERE time_hier.level_name IN ('ALL','YEAR','QUARTER')
     ORDER BY time_hier.hier_order;
 2   3   4   5   6   7
MEMBER_N       SALES
-------------------- ------------------------
ALL          1914660169
CY2011        563440466
Q1CY2011        7068888
Q2CY2011      556371578
CY2012        564675917
Q2CY2012      564675917
CY2013        597603238
Q1CY2013        5327980
Q2CY2013      592275258
CY2014        188940548
Q1CY2014        5327980
Q2CY2014      183612568

12 rows selected.

SQL>
```

Q/ Do you get visual or nonvisual totals?

A/ The filter-before aggregate predicate for the 2011 year filtered out the fourth quarter (see `Q4CY2011` in the previous query). The resulting total of sales for year 2011 aggregates only the first two quarters. This is the same behavior for all years. The aggregated sum of sales for all years is smaller in the second query than in the previous query.

SQL execution with filter-before aggregate predicates manages to report visual totals.

a. Now filter on two hierarchies, `TIME_HIER` and `GEOGRAPHY_HIER`. Report sales for the first half of each year (first and second quarters) in Mexico and Canada.

```
SQL> SELECT time_hier.member_name AS time,
            geography_hier.member_name AS geography, sales
     FROM ANALYTIC VIEW (
     USING sales_av HIERARCHIES(time_hier, geography_hier)
     FILTER FACT (time_hier TO level_name = 'QUARTER' AND
                   (quarter_name like 'Q1%'
                     OR quarter_name like 'Q2%'),
                    geography_hier TO level_name = 'COUNTRY'
                     AND country_name in ('Mexico','Canada')))
     WHERE time_hier.level_name = 'YEAR'
     AND geography_hier.level_name = 'REGION'
     ORDER BY time_hier.hier_order;
 2   3   4   5   6   7   8   9  10  11  12
TIME   GEOGRAPHY              SALES
------ ------------------------------------ ----------
CY2011 North America       38931636
CY2012 North America       39129154
CY2013 North America       41586079
CY2014 North America       13119131

SQL>
```

b. Re-execute the same query, adding another country. Report sales for the first half of each year (first and second quarters) in Mexico, Canada, and Chile.

```
SQL> SELECT time_hier.member_name AS time,
            geography_hier.member_name AS geography, sales
     FROM ANALYTIC VIEW (
     USING sales_av HIERARCHIES(time_hier, geography_hier)
     FILTER FACT
       (time_hier TO level_name = 'QUARTER' AND
         (quarter_name like 'Q1%' OR quarter_name like 'Q2%'),
        geography_hier TO level_name = 'COUNTRY'
          AND country_name in ('Mexico','Canada' ,'Chile')))
     WHERE time_hier.level_name = 'YEAR'
     AND geography_hier.level_name = 'REGION'
     ORDER BY time_hier.hier_order;
 2   3   4   5   6   7   8   9  10  11  12
TIME   GEOGRAPHY              SALES
------ ------------------------------------ ----------
CY2011 South America        4880427
CY2011 North America       38931636
```

```
CY2012 South America        4764838
CY2012 North America       39129154
CY2013 North America       41586079
CY2013 South America        5423944
CY2014 North America       13119131
CY2014 South America        1618375

8 rows selected.

SQL>
```

c. Re-execute the same query and report the global sales total for each year of each region.

```
SQL> SELECT time_hier.member_name AS time,
            geography_hier.member_name AS geography, sales
     FROM ANALYTIC VIEW (
     USING sales_av HIERARCHIES(time_hier, geography_hier)
     FILTER FACT
       (time_hier TO level_name = 'QUARTER' AND
         (quarter_name like 'Q1%' OR quarter_name like 'Q2%'),
        geography_hier TO level_name = 'COUNTRY'
          AND country_name in ('Mexico','Canada' ,'Chile')))
     WHERE time_hier.level_name IN ('ALL','YEAR')
     AND geography_hier.level_name = 'REGION'
     ORDER BY time_hier.hier_order;
 2   3   4   5   6   7   8   9  10  11  12
TIME   GEOGRAPHY              SALES
-------- ---------------------------------------- ------------------------
ALL    South America       16687584
ALL    North America      132766000
CY2011 South America        4880427
CY2011 North America       38931636
CY2012 South America        4764838
CY2012 North America       39129154
CY2013 South America        5423944
CY2013 North America       41586079
CY2014 South America        1618375
CY2014 North America       13119131

10 rows selected.

SQL>
```

Q/ Does the sales total for North America include all years, all quarters, and all countries?

A/ No.
Each hierarchy specifies a filter-before aggregate predicate, which serves to filter the leaves of that hierarchy before aggregating the measures. The fact rows are filtered to include only the leaf descendants of those members.

4. Use calculated measures to query the AV. The `USING` and `ADD MEASURES` clauses can be used to define a calculated measure within a `SELECT` statement. Define two calculated measures by adding the total of sales for the period prior to each period and the percent change of sales against the prior period.

```
SQL> SELECT time_hier.member_name, sales, sales_prior_period,
            ROUND(sales_prior_period_pct_change,3) AS
            percent_change_sales
     FROM ANALYTIC VIEW (
     USING sales_av HIERARCHIES(time_hier)
     ADD MEASURES (sales_prior_period AS (LAG(sales)
                          OVER (HIERARCHY time_hier OFFSET 1)),
                   sales_prior_period_pct_change AS
                     (LAG_DIFF_PERCENT(sales)
                          OVER (HIERARCHY time_hier OFFSET 1))
                   )
                        )
     WHERE time_hier.level_name = 'YEAR'
     ORDER BY time_hier.hier_order;
 2   3   4   5   6   7   8   9  10  11  12  13
MEMBER_N       SALES SALES_PRIOR_PERIOD PERCENT_CHANGE_SALES
---------- ---------- ------------------- ----------------------------------------
CY2011     593507775
CY2012     568622304           593507775                  -.042
CY2013     611827904           568622304                   .076
CY2014     196300122           611827904                  -.679

SQL>
```

5. Use both filter-before aggregate predicates and calculated measures to query the AV.
   a. Report sales, the total of sales for the period prior to each period, and the percent change of sales against the prior period for the first half of years in Mexico and Canada.

```
SQL> SELECT time_hier.member_name AS time,
            geography_hier.member_name AS geography, sales,
            sales_prior_period,
            ROUND(sales_prior_period_pct_change,3)
```

```
             AS percent_change_sales
     FROM ANALYTIC VIEW (
     USING sales_av HIERARCHIES(time_hier, geography_hier)
     FILTER FACT ( time_hier TO level_name = 'QUARTER' AND
                     (quarter_name like 'Q1%'
                      OR quarter_name like 'Q2%'),
                   geography_hier TO level_name = 'COUNTRY'
                      AND country_name in ('Mexico','Canada'))
     ADD MEASURES ( sales_prior_period AS (LAG(sales) OVER
                     (HIERARCHY time_hier OFFSET 1)),
                    sales_prior_period_pct_change AS
                        (LAG_DIFF_PERCENT(sales) OVER
                     (HIERARCHY time_hier OFFSET 1)))
                        )
     WHERE time_hier.level_name = 'YEAR'
     AND   geography_hier.level_name = 'REGION'
     ORDER BY time_hier.hier_order;
2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19
20  21
TIME    GEOGRAPHY           SALES SALES_PRIOR_PERIOD
------- --------------------------- ---------------------- ----------------------------------------
PERCENT_CHANGE_SALES
----------------------------------------
CY2011 North America   38931636
CY2012 North America   39129154           38931636
                .005
CY2013 North America   41586079           39129154
                .063
CY2014 North America   13119131           41586079
               -.685

SQL>
```

b. Add a `WHERE` clause and check whether it impacts the aggregated measure data.

```
SQL> SELECT time_hier.member_name AS time,
            geography_hier.member_name AS geography, sales,
            sales_prior_period,
            ROUND(sales_prior_period_pct_change,3)
             AS percent_change_sales
     FROM ANALYTIC VIEW (
     USING sales_av HIERARCHIES(time_hier, geography_hier)
     FILTER FACT ( time_hier TO level_name = 'QUARTER' AND
                     (quarter_name like 'Q1%'
                      OR quarter_name like 'Q2%'),
```

```
                    geography_hier TO level_name = 'COUNTRY'
                       AND country_name in ('Mexico','Canada'))
     ADD MEASURES ( sales_prior_period AS (LAG(sales) OVER
                      (HIERARCHY time_hier OFFSET 1)),
                    sales_prior_period_pct_change AS
                        (LAG_DIFF_PERCENT(sales) OVER
                      (HIERARCHY time_hier OFFSET 1)))
                       )
     WHERE time_hier.level_name = 'YEAR'
     AND   geography_hier.level_name = 'REGION'
     AND   sales > 30000000
     ORDER BY time_hier.hier_order;
2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19
20  21  22
TIME    GEOGRAPHY           SALES SALES_PRIOR_PERIOD
------- ------------------------- ---------- ----------------------------------------
PERCENT_CHANGE_SALES
----------------------------------------------
CY2011 North America   38931636
CY2012 North America   39129154           38931636
                .005
CY2013 North America   41586079           39129154
                .063

SQL> EXIT
$
```

An appended `WHERE` clause does not impact the aggregated measure data. It simply reduces the rows returned after the filter-before aggregation and the measures calculation are applied.

6. Execute the `$HOME/labs/DW/cleanup_AV.sh` shell script to drop `the AV schema, including fact and dimension tables.`

```
$ $HOME/labs/DW/cleanup_AV.sh
…
$
```

Practice 7-4: Using Polymorphic Table Functions

Overview

In this practice, you create and use polymorphic table functions (PTFs).

Tasks

1. Execute the `$HOME/labs/DW/PTF.sh` shell script. The script creates the `SCOTT.DEPT` and `SCOTT.EMP` tables.

```
$ $HOME/labs/DW/PTF.sh
…
$
```

2. In the first example, you create the `GET_COL` PTF that returns the columns of a given data type from a given table.
   a. Log in to the `PDB1` PDB as `SCOTT` to create a PTF that returns the columns of a given data type from a given table.

```
$ sqlplus scott@PDB1

Enter password: password

SQL>
```

   b. Create the PL/SQL package that contains the functions/procedures that the PTF implementation must use:
      - The supplied `DESCRIBE` function, which returns the new table "shape"
      - The supplied `FETCH_ROWS` procedure is not required because it does need to produce associated new column values for a given subset of rows.
      - In the package, the parameters are:
        - `tab`: Input table
        - `type_name`: A string representing the type of columns to return
        - `flip`: "False" (default value). Only matched columns with the given type name are returned; otherwise, the PTF ignores the columns with the given type name and returns the other columns.

```
CREATE OR REPLACE PACKAGE get_col_p AS
FUNCTION describe (tab IN OUT DBMS_TF.table_t,
                   type_name VARCHAR2,
                   flip VARCHAR2 DEFAULT 'False')
         RETURN DBMS_TF.describe_t;
END get_col_p;
/
```

```
SQL> CREATE OR REPLACE PACKAGE get_col_p AS
FUNCTION describe (tab IN OUT DBMS_TF.table_t,
                   type_name VARCHAR2,
                   flip VARCHAR2 DEFAULT 'False')
         RETURN DBMS_TF.describe_t;
END get_col_p;
/
 2   3   4   5   6   7
Package created.

SQL>
```

c. Create the package body containing the `DESCRIBE` function.

```
CREATE OR REPLACE PACKAGE BODY GET_COL_P as
FUNCTION describe (tab IN OUT DBMS_TF.table_t,
                   type_name VARCHAR2,
                   flip VARCHAR2 DEFAULT 'False')
         RETURN DBMS_TF.describe_t
AS
  typ constant varchar2(1024) := upper(trim(type_name));
BEGIN
  FOR i IN 1 .. tab.column.count() LOOP
      tab.column(i).pass_through := CASE upper(substr(flip,1,1))
          WHEN 'F' THEN
       DBMS_TF.Column_Type_Name(tab.column(i).description)=typ
          ELSE
       DBMS_TF.Column_Type_Name(tab.column(i).description)!=typ
         END /* case */;
   END LOOP;
RETURN null;
END;
END GET_COL_P;
/
```

```
SQL> CREATE OR REPLACE PACKAGE BODY GET_COL_P as
FUNCTION describe (tab IN OUT DBMS_TF.table_t,
                   type_name VARCHAR2,
                   flip VARCHAR2 DEFAULT 'False')
         RETURN DBMS_TF.describe_t
AS
  typ constant varchar2(1024) := upper(trim(type_name));
BEGIN
  FOR i IN 1 .. tab.column.count() LOOP
```

```
      tab.column(i).pass_through := CASE upper(substr(flip,1,1))
          WHEN 'F' THEN
       DBMS_TF.Column_Type_Name(tab.column(i).description)=typ
          ELSE
       DBMS_TF.Column_Type_Name(tab.column(i).description)!=typ
         END /* case */;
   END LOOP;
RETURN null;
END;
END GET_COL_P;
/
2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
18  19  20  21
Package body created.

SQL>
```

d. Create the PTF. Specify exactly one formal argument of type `TABLE`, specify the return type of the PTF as `TABLE`, specify the type of the PTF function (`ROW` or `TABLE POLYMORPHIC`), and indicate which package contains the actual PTF implementation.

```
SQL> CREATE OR REPLACE FUNCTION GET_COL (
                     tab       TABLE,
                        type_name VARCHAR2,
                        flip      VARCHAR2 DEFAULT 'False')
                  RETURN TABLE
                  PIPELINED ROW POLYMORPHIC using GET_COL_P;
/
2   3   4   5   6   7
Function created.

SQL>
```

3. Query tables by using the `GET_COL` PTF.

a. Query the `SCOTT.DEPT` table.

```
SQL> SELECT * FROM scott.dept;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON

SQL>
```

b. Use the GET_COL PTF to report from the SCOTT.DEPT table only columns whose type is VARCHAR2.

```
SQL> SELECT * FROM GET_COL(scott.dept, 'varchar2');

DNAME          LOC
-------------- ------------------
ACCOUNTING     NEW YORK
RESEARCH       DALLAS
SALES          CHICAGO
OPERATIONS     BOSTON

SQL>
```

c. Use the same GET_COL PTF to report from another table, SCOTT.EMP table, only columns whose type is not VARCHAR2.

```
SQL> SELECT *
     FROM GET_COL(scott.emp, 'varchar2', flip => 'True');
 2
EMPNO     MGR HIREDATE      SAL    COMM      DEPTNO
---------- ---------- --------- ---------- ---------- ----------
  7369    7902 17-DEC-80    800                   20
  7499    7698 20-FEB-81   1600     300          30
  7521    7698 22-FEB-81   1250     500          30
  7566    7839 02-APR-81   2975                   20
  7654    7698 28-SEP-81   1250    1400          30
  7698    7839 01-MAY-81   2850                   30
  7782    7839 09-JUN-81   2450                   10
  7788    7566 19-APR-87   3000                   20
  7839         17-NOV-81   5000                   10
  7844    7698 08-SEP-81   1500       0          30
  7876    7788 23-MAY-87   1100                   20
  7900    7698 03-DEC-81    950                   30
  7902    7566 03-DEC-81   3000                   20
  7934    7782 23-JAN-82   1300                   10

14 rows selected.

SQL>
```

d. Use the GET_COL PTF to report for ANALYST and PRESIDENT employees only columns whose type is VARCHAR2.

```
SQL> SELECT * FROM GET_COL(scott.emp, 'varchar2')
     WHERE job IN ('ANALYST' , 'PRESIDENT');
 2
```

```
ENAME      JOB
---------- ---------
SCOTT      ANALYST
KING       PRESIDENT
FORD       ANALYST

SQL>
```

Only ENAME and JOB columns of data type VARCHAR2 are displayed.

e. Use the GET_COL PTF to report about employees whose commission is greater than 1000 and only columns whose type is not VARCHAR2.

```
SQL> SELECT *
     FROM GET_COL(scott.emp, 'varchar2', flip => 'True')
     WHERE comm > 1000;
 2   3
 EMPNO     MGR HIREDATE      SAL    COMM      DEPTNO
------- ------ --------- ------- ------- ----------
  7654   7698 28-SEP-81    1250    1400          30

SQL>
```

All columns except the ENAME and JOB columns are displayed.

f. Use the GET_COL PTF to report about employees whose commission is greater than 1000 and only columns whose type is VARCHAR2.

```
SQL> SELECT * FROM GET_COL(scott.emp, 'varchar2')
     WHERE comm > 1000;
WHERE comm > 1000
      *
ERROR at line 1:
ORA-00904: "COMM": invalid identifier

SQL>
```

Because COMM is not a VARCHAR2 type of column, the selection cannot apply.

g. Use the GET_COL PTF to report about employees whose job is either ANALYST or PRESIDENT and only columns whose type is not VARCHAR2.

```
SQL> SELECT *
     FROM GET_COL(scott.emp, 'varchar2', flip => 'True')
```

```
     WHERE job IN ('ANALYST' , 'PRESIDENT');
WHERE job IN ('ANALYST' , 'PRESIDENT')
      *
ERROR at line 2:
ORA-00904: "JOB": invalid identifier

SQL>
```

Because JOB is a VARCHAR2 type of column, the selection cannot apply.

4. In a second example, you create the CHANGE_CASE_P PTF that changes the case of all the VARCHAR2 columns in the input table to the specified case.
   a. Create the PL/SQL package body. The PL/SQL package that contains the functions/procedures for the PTF implementation must use:
      - The supplied DESCRIBE function that returns the new table “shape”
      - The supplied FETCH_ROWS procedure required to produce the associated new column values for a given subset of rows
      - In the package, the parameters are:
        - tab: Input table
        - type_name: The case-change function ('upper', 'lower', or 'initcap')

```
SQL> CREATE OR REPLACE PACKAGE CHANGE_CASE_P AS
         FUNCTION Describe(tab IN OUT DBMS_TF.Table_t,
                           new_case VARCHAR2)
                  RETURN DBMS_TF.describe_t;
         PROCEDURE Fetch_Rows(new_case varchar2);
     END CHANGE_CASE_P;
/
 2   3   4   5   6   7
Package created.

SQL>
```

   b. Create the package body containing the DESCRIBE function and FETCH_ROWS procedure.

```
SQL> CREATE OR REPLACE PACKAGE BODY CHANGE_CASE_P AS
       FUNCTION Describe(tab IN OUT DBMS_TF.table_t,
                         new_case VARCHAR2)
                RETURN DBMS_TF.describe_t
       AS
        new_cols DBMS_TF.columns_new_t;
        col_id   pls_integer := 1;
```

```
        BEGIN
          FOR i IN 1 .. tab.column.count() LOOP
            IF tab.column(i).description.type =
               DBMS_TF.TYPE_VARCHAR2 then
               tab.column(i).pass_through := FALSE;
               tab.column(i).for_read     := TRUE;
               new_cols(col_id) := tab.column(i).description;
               col_id := col_id + 1;
            END IF;
          END LOOP;
         RETURN DBMS_TF.describe_t(new_columns => new_cols);
         END;

         PROCEDURE Fetch_Rows(new_case varchar2)
         AS
          case_function constant varchar2(100) :=
          upper(trim(new_case));
          rowset        DBMS_TF.Row_Set_t;
         BEGIN
          DBMS_TF.Get_Row_Set(rowset);
           IF case_function in ('UPPER', 'LOWER', 'INITCAP')
           THEN
            FOR c IN 1 .. rowset.count() LOOP
             FOR r IN 1 .. rowset(c).tab_varchar2.count() LOOP
               rowset(c).tab_varchar2(r) :=
                 CASE case_function WHEN 'UPPER'
                   THEN upper(rowset(c).tab_varchar2(r))
                                    WHEN 'LOWER'
                   THEN lower(rowset(c).tab_varchar2(r))
                   ELSE initcap(rowset(c).tab_varchar2(r))
                 END;
            END LOOP;
           END LOOP;
          END IF;
          DBMS_TF.Put_Row_Set(rowset);
         END;
END CHANGE_CASE_P;
/
 2   3   4  …
Package body created.

SQL>
```

c. Create the PTF. Specify exactly one formal argument of type `TABLE`, the return type of the PTF as `TABLE`, the type of the PTF function (`ROW` or `TABLE POLYMORPHIC`), and indicate which package contains the actual PTF implementation.

```
SQL> CREATE OR REPLACE FUNCTION CHANGE_CASE
           (tab table, new_case varchar2)
            RETURN TABLE
            PIPELINED ROW POLYMORPHIC using CHANGE_CASE_P;
/
  2    3    4    5
Function created.

SQL>
```

5. Query tables by using the `CHANGE_CASE` PTF.

a. Display the content of the `SCOTT.DEPT` table after changing the case of all the `VARCHAR2` columns to the specified case.

```
SQL> SELECT * FROM CHANGE_CASE(scott.dept, 'initcap');

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 Accounting     New York
        20 Research       Dallas
        30 Sales          Chicago
        40 Operations     Boston

SQL>
```

b. Display the content of another table, the `SCOTT.EMP` table, after changing the case of all the `VARCHAR2` columns to the specified case.

```
SQL> SELECT * FROM CHANGE_CASE(scott.emp, 'lower')
     WHERE comm > 1000;
 2
 EMPNO     MGR HIREDATE      SAL    COMM      DEPTNO ENAME
---------- ---------- --------- ---------- ---------- ----------
JOB
---------
  7654    7698 28-SEP-81    1250    1400          30 martin
salesman


SQL>
```

c. Display the content of the SCOTT.EMP table after changing the case of all the VARCHAR2 columns to the specified case and selecting only rows whose JOB is SALESMAN.

```
SQL> SELECT * FROM CHANGE_CASE(scott.emp, 'lower')
     WHERE job ='SALESMAN';
 2
no rows selected

SQL>
```

Q/ Why are there no rows selected?

A/ Because JOB is a VARCHAR2 type of column, the PTF being applied on this VARCHAR2 column, SALESMAN, becomes salesman.

```
SQL> SELECT * FROM CHANGE_CASE(scott.emp, 'lower')
     WHERE job ='salesman';
 2
 EMPNO     MGR HIREDATE      SAL    COMM      DEPTNO ENAME
------------ ------------ --------------------- ------------ ------------ ------------------------ ------------------------
JOB
---------------------
  7499   7698 20-FEB-81   1600     300          30 allen
salesman

  7521   7698 22-FEB-81   1250     500          30 ward
salesman

  7654   7698 28-SEP-81   1250    1400          30 martin
salesman

  7844   7698 08-SEP-81   1500       0          30 turner
salesman


SQL>
```

d. View the execution plan. The POLYMORPHIC TABLE FUNCTION operation indicates the row source for a polymorphic table function, which is a table function whose return type is determined by its arguments.

```
SQL> SELECT * FROM dbms_xplan.display_cursor();

PLAN_TABLE_OUTPUT
```

```
----------------------------------------------------------------------------------------------------------------------
SQL_ID  090hgbkyy2jy0, child number 0
-------------------------------------
SELECT * FROM CHANGE_CASE(scott.emp, 'lower') WHERE job
='salesman'

Plan hash value: 3956160932
----------------------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name         | Rows |
Bytes |
Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |              |       |
|
    2 (100)|          |
|   1 |  POLYMORPHIC TABLE FUNCTION | CHANGE_CASE |    14 |
1218 |
            |          |
|   2 |   TABLE ACCESS FULL          | EMP          |    14 |
1218 |
    2   (0)| 00:00:01 |
----------------------------------------------------------------------------------------------------------------------
Note
-----
   - dynamic statistics used: dynamic sampling (level=2)

18 rows selected.

SQL>
```

Q/ What happens if you use both PTFs on the `SCOTT.EMP` table?

```
SQL> SELECT *
     FROM CHANGE_CASE(GET_COL(scott.emp, 'varchar2'));
FROM CHANGE_CASE(GET_COL(scott.emp, 'varchar2'))
                 *
ERROR at line 1:
ORA-62569: nested polymorphic table function is disallowed

SQL>
```

A/ A nested polymorphic table function is disallowed.

6. To use both PTFs, consider using a with-clause.

```
SQL> WITH T AS (
                SELECT * FROM GET_COL(scott.emp, 'varchar2')
               )
     SELECT * FROM CHANGE_CASE(T, 'initcap');
 2   3   4
ENAME      JOB
---------- ---------
Smith      Clerk
Allen      Salesman
Ward       Salesman
Jones      Manager
Martin     Salesman
Blake      Manager
Clark      Manager
Scott      Analyst
King       President
Turner     Salesman
Adams      Clerk
James      Clerk
Ford       Analyst
Miller     Clerk

14 rows selected.

SQL> EXIT
$
```

7. Execute the `$HOME/labs/DW/cleanup_PTF.sh` shell script to drop the PTFs.

```
$ $HOME/labs/DW/cleanup_PTF.sh
…
$
```

Practices for Lesson 8:
Describing Sharding Enhancements

Practices for Lesson 8

There are no practices for this lesson.