Multitenant Workshop

Contents

Initial requirements

- SSH private key to Access the database server in the cloud. This private key is provided along with this manual.
- SSH client app, to login to the database server
- Database server public IP

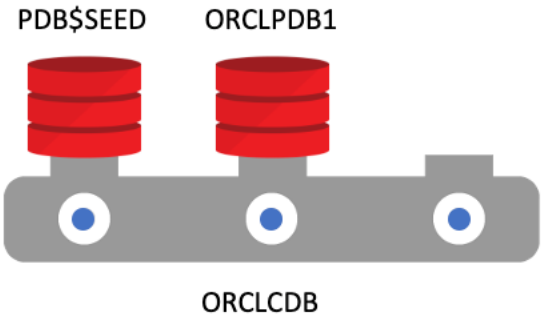
Multitenant

Access the CDB, PDB creation and administration

Below are described the first steps to access the container database (CDB) and create a pluggable database (PDB).

Access to the database server as "**oracle**" using ssh. Then use Sql*Plus to run the following commands:

```
ssh -i id_rsa oracle@<db server public ip>

[oracle@myoracledb ~]$ sqlplus / as sysdba

SQL*Plus: Release 21.0.0.0.0 - Production on Wed Oct 6 09:06:08 2021
Version 21.3.0.0.0

Copyright (c) 1982, 2021, Oracle.  All rights reserved.


Connected to:
Oracle Database 21c Enterprise Edition Release 21.0.0.0.0 - Production
Version 21.3.0.0.0

--- We can have a look at the existing PDB

SQL> show pdbs

    CON_ID CON_NAME                       OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         2 PDB$SEED                       READ ONLY  NO
         3 ORCLPDB1                       READ WRITE NO
```



Create a new PDB in the container database (CDB)

Follow the below steps to create a new PDB:

```
$ sqlplus / as sysdba

create pluggable database PDB1 admin user pdbadmin identified by "Oracle_4U";
show pdbs

    CON_ID CON_NAME                       OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         2 PDB$SEED                       READ ONLY  NO
         3 ORCLPDB1                       READ WRITE NO
         4 PDB1                           MOUNTED

alter pluggable database PDB1 open read write;
show pdbs

    CON_ID CON_NAME                       OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         2 PDB$SEED                       READ ONLY  NO
         3 ORCLPDB1                       READ WRITE NO
         4 PDB1                           READ WRITE NO
```

You've just created a new PDB from PDB$SEED.

Unplug a PDB from its CDB

PDB can be unplugged and plugged between CDB. For example, we might unplug PDB1 from container database CDB1, and plug it into container database CDB2.

In the following example, we will unplug a PDB from its CDB.

```
--- First we connect to the CDB.

sqlplus / as sysdba
```

```
show pdbs

    CON_ID CON_NAME                  OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         2 PDB$SEED                       READ ONLY  NO
         3 ORCLPDB1                       READ WRITE NO
         4 PDB1                           READ WRITE NO

alter pluggable database PDB1 close immediate;
alter pluggable database PDB1 unplug into '/home/oracle/PDB1.xml';

drop pluggable database PDB1 keep datafiles;

show pdbs;

    CON_ID CON_NAME                  OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         2 PDB$SEED                       READ ONLY  NO
         3 ORCLPDB1                       READ WRITE NO
```

Check the PDB compatibility before plugging it in the target CDB

Once the PDB has been unplugged, it's a best practice, whenever we plan to plug it in another CDB, to check the PDB compatibility in the target CDB. Should an incompatibility be encountered, the following PL/SQL block would report it:

```
sqlplus / as sysdba

set serveroutput on

DECLARE
   compatible BOOLEAN := FALSE;
BEGIN
   compatible := DBMS_PDB.CHECK_PLUG_COMPATIBILITY(
        pdb_descr_file => '/home/oracle/PDB1.xml');
   if compatible then
      DBMS_OUTPUT.PUT_LINE('Is pluggable PDB1.xml compatible? YES');
   else DBMS_OUTPUT.PUT_LINE('Is pluggable PDB1.xml compatible? NO');
   end if;
END;
/
```

Cloning, backup and restore of PDB, refreshable PDB

Plug a PDB by cloning from an unplugged PDB (as clone method)

In the following example, we create a clone from the metadata and datafiles of the unplugged PDB. Then we connect the original PDB back into its CDB using the nocopy method.

```
sqlplus / as sysdba

--- Create a clone of PDB1 using the manifest file generated during the unplug
command (copy the original datafiles)
create pluggable database PDB1_clone as clone using '/home/oracle/PDB1.xml';

-- Plug the unplugged DB with nocopy method (reuse the datafiles)
create pluggable database PDB1_nocopy using '/home/oracle/PDB1.xml' NOCOPY
TEMPFILE REUSE;

show pdbs

    CON_ID CON_NAME                  OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         2 PDB$SEED                       READ ONLY  NO
         3 ORCLPDB1                       READ WRITE NO
         5 PDB1_CLONE                     MOUNTED
         6 PDB1_NOCOPY                           MOUNTED

alter pluggable database PDB1_nocopy open read write;
alter pluggable database PDB1_CLONE open read write;

show pdbs

    CON_ID CON_NAME                  OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         2 PDB$SEED                       READ ONLY  NO
         3 ORCLPDB1                       READ WRITE NO
         5 PDB1_CLONE                     READ WRITE NO
         6 PDB1_NOCOPY                    READ WRITE NO

-- Now retrieve the datafiles names of PDB1_nocopy

alter session set container = PDB1_nocopy;

--- This query retrieves the PDB's datafile names
select file_name from dba_data_files;

--- Please write down these datafile names, as we will use them later on during
the backup/restore lab.

Example:
SQL> select file_name from dba_data_files
  2  ;

FILE_NAME
--------------------------------------------------------------------------------
-
```

```
/opt/oracle/oradata/ORCLCDB/CDADC32D998ECE2AE053E414010AA42A/datafile/o1_mf_und
otbs1_jotz78lt_.dbf

/opt/oracle/oradata/ORCLCDB/CDADC32D998ECE2AE053E414010AA42A/datafile/o1_mf_sys
aux_jotz78lr_.dbf

/opt/oracle/oradata/ORCLCDB/CDADC32D998ECE2AE053E414010AA42A/datafile/o1_mf_sys
tem_jotz78l7_.dbf

--- Note that PDB GUID is used in the path to the datafiles

select GUID from v$pdbs;

GUID
--------------------------------
CDADC32D998ECE2AE053E414010AA42A
```

Create a new PDB from an existing PDB

In the following lab, we create a PDB clone using the "from PDB" method.

```
sqlplus / as sysdba

create pluggable database PDB1 from PDB1_nocopy;

alter pluggable database PDB1 open read write;

show pdbs

    CON_ID CON_NAME                       OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         2 PDB$SEED                       READ ONLY  NO
         3 ORCLPDB1                       READ WRITE NO
         4 PDB1                           READ WRITE NO
         5 PDB1_CLONE                     READ WRITE NO
         6 PDB1_NOCOPY                    READ WRITE NO
```

Create a refreshable PDB from an existing PDB

Refreshable PDB are read only clones of a source PDB. They can be refreshed by applying the redo log changes generated on the source PDB since the last refresh (incremental refresh). A refreshable PDB can be either in the same CDB as its source, or in another CDB. In both cases, it must be created with a database link.

In the following lab, we will create a refreshable PDB in the same CDB as the source PDB. We will create a privileged common user with the appropriate privileges, then create a database link connected to that user, and at last the refreshable PDB:

```
-- First we connect to the CDB$ROOT, and create a common user

sqlplus / as sysdba

--- Note the C## prefix in the username (common user), and the "container=ALL"
clause in the grant commands

create user C##_ADMIN_PDB identified by "Oracle_4U" container=ALL;
grant CREATE PLUGGABLE DATABASE to C##_ADMIN_PDB container=ALL;
grant create session to C##_ADMIN_PDB container=ALL;

-- Then we create a database link connected to that user. In our case, the
database link is a loopback, since we only have one CDB

create database link DBL_ORCLCDB connect to C##_ADMIN_PDB identified by
"Oracle_4U" using 'myoracledb:1521/ORCLCDB';


--- Check that the database link is working fine
select * from dual@DBL_ORCLCDB;

D
-
X

create pluggable database PDB1_REFRESH from PDB1@DBL_ORCLCDB refresh mode
manual;

--- A refreshable PDB can only be opened in read only mode. If you try to open
it in read write, it will be opened in read only anyway.

alter pluggable database PDB1_REFRESH open read only;

show pdbs

    CON_ID CON_NAME                       OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         2 PDB$SEED                       READ ONLY  NO
         3 ORCLPDB1                       READ WRITE NO
         4 PDB1                           READ WRITE NO
         5 PDB1_CLONE                     READ WRITE NO
         6 PDB1_NOCOPY                    READ WRITE NO
         7 PDB1_REFRESH                   READ ONLY  NO

-- Create some new data in the source PDB
--- Connect to PDB1 and create a local user with a table
sqlplus system/Oracle_4U@myoracledb:1521/PDB1
```

```
create tablespace USERS datafile size 50M;
create user TEST_REFRESH identified by "Oracle_4U" temporary tablespace TEMP
default tablespace USERS;
grant connect, resource to TEST_REFRESH;
alter user TEST_REFRESH quota unlimited on USERS;

--- Connect to TEST_REFRESH schema and create a new table

sqlplus TEST_REFRESH/Oracle_4U@myoracledb:1521/PDB1
create table TT (c1 number);
insert into TT values (999);
commit;

--- Connect to PDB1_REFRESH PDB as system, and check

sqlplus system/Oracle_4U@myoracledb:1521/PDB1_REFRESH
select username from dba_users where username = 'TEST_REFRESH';

-- We need to refresh this PDB

-- Then refresh the refreshable clone, and check the new data is there

sqlplus / as sysdba
alter pluggable database PDB1_REFRESH close immediate;
alter pluggable database PDB1_REFRESH refresh;
show pdbs

    CON_ID CON_NAME                       OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         2 PDB$SEED                       READ ONLY  NO
         3 ORCLPDB1                       READ WRITE NO
         4 PDB1                           READ WRITE NO
         5 PDB1_CLONE                     READ WRITE NO
         6 PDB1_NOCOPY                    READ WRITE NO
         7 PDB1_REFRESH                   MOUNTED

alter pluggable database PDB1_REFRESH open read only;

sqlplus TEST_REFRESH/Oracle_4U@myoracledb:1521/PDB1_REFRESH
select * from tt;

       C1
----------
       999
```

Create a snap clone on the refreshable PDB

Using the refreshable PDB as a TEST MASTER, create two snap clones.

```
-- On the PDB1_REFRESH PDB, used as a TEST MASTER, we create two sparse clones
```

```
-- Sparse clones are thin clones, that are sustained by the copy on write
technology of the storage
-- Sparse clones creation is extremely fast, as pointers to the TEST MASTER
datafiles are created, instead of physically cloning the TEST MASTER datafiles

sqlplus / as sysdba

show pdbs

    CON_ID CON_NAME                  OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         2 PDB$SEED                       READ ONLY  NO
         3 ORCLPDB1                       READ WRITE NO
         4 PDB1                           READ WRITE NO
         5 PDB1_CLONE                     READ WRITE NO
         6 PDB1_NOCOPY                    READ WRITE NO
         7 PDB1_REFRESH                   READ ONLY  NO

create pluggable database PDB1_SNAP1 from PDB1_REFRESH snapshot copy;
create pluggable database PDB1_SNAP2 from PDB1_REFRESH snapshot copy;
alter pluggable database PDB1_SNAP1 open;
alter pluggable database PDB1_SNAP2 open;

show pdbs

    CON_ID CON_NAME                  OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         2 PDB$SEED                       READ ONLY  NO
         3 ORCLPDB1                       READ WRITE NO
         4 PDB1                           READ WRITE NO
         5 PDB1_CLONE                     READ WRITE NO
         6 PDB1_NOCOPY                    READ WRITE NO
         7 PDB1_REFRESH                   READ ONLY  NO
         8 PDB1_SNAP1                     READ WRITE NO
         9 PDB1_SNAP2                     READ WRITE NO

-- Unlike their TEST MASTER, the snapshot copies are opened in read write,
allowing users to create their own data.

-- Connect to PDB1_SNAP1, and run:

sqlplus TEST_REFRESH/Oracle_4U@myoracledb:1521/PDB1_SNAP1

select * from tt;

       C1
----------
       999

insert into tt values (1000);
commit;

select * from tt;
```

```
      C1
----------
       999
      1000

-- Now connect to PDB1_SNAP2 and check the TT table:

sqlplus TEST_REFRESH/Oracle_4U@myoracledb:1521/PDB1_SNAP2

select * from tt;

      C1
----------
       999

-- Only the data from the TEST MASTER is visible, not the data created in
PDB1_SNAP1

delete tt;
insert into tt values (1001);
commit;
select * from tt;

      C1
----------
      1001

-- We can even modify (delete) the original data, without affecting the TEST
MASTER: connect to the TEST MASTER and check:

sqlplus TEST_REFRESH/Oracle_4U@myoracledb:1521/PDB1_REFRESH
select * from tt;

      C1
----------
       999

-- This illustrates the "copy on write" functionality
-- If we refresh the TEST MASTER, we will first drop the snapshot copies, and
re-create them after the TEST MASTER refresh.
```

PDB Backup and restore

In the following lab, we will backup a PDB, simulate the loss of a datafile, restore the lost datafile and recover the PDB.

```
$ rman target=/

Recovery Manager: Release 21.0.0.0.0 - Production on Wed Oct 6 14:13:50 2021
Version 21.3.0.0.0
```

```
Copyright (c) 1982, 2021, Oracle and/or its affiliates.  All rights reserved.

connected to target database: ORCLCDB (DBID=2852045519)

RMAN> BACKUP PLUGGABLE DATABASE PDB1_NOCOPY;
RMAN> exit
```

Simulate the loss of one of the PDB datafiles. Just remove it from the operating system:

```
--- Remove a datafile from the operating system:
--- You need to use the file name for tablespace SYSTEM, as returned previously
in page 6
rm
/opt/oracle/oradata/ORCLCDB/CDADC32D998ECE2AE053E414010AA42A/datafile/o1_mf_sys
tem_jotz78l7_.dbf

--- Connect to the PDB

sqlplus system/Oracle_4U@myoracledb:1521/PDB1_NOCOPY

-- This fails:

ORA-27041: unable to open file
Linux-x86_64 Error: 2: No such file or directory
Additional information: 3
ORA-00604: error occurred at recursive SQL level 2
ORA-01116: error in opening database file 19
ORA-01110: data file 19:
'/opt/oracle/oradata/ORCLCDB/CDADC32D998ECE2AE053E414010AA42A/datafile/o1_mf_sy
s
tem_jotz78l7_.dbf'
ORA-27041: unable to open file
Linux-x86_64 Error: 2: No such file or directory
Additional information: 3
```

Execute the following RMAN code to restore and recover the PDB:

```
--- Ensure datafile ID (19) matches with your environment

$ rman target=/

RUN {
  alter pluggable database PDB1_NOCOPY close immediate;
  RESTORE datafile 19;
  RECOVER PLUGGABLE DATABASE PDB1_NOCOPY;
  ALTER PLUGGABLE DATABASE PDB1_NOCOPY open;
}

--- Check that the PDB is now sucessufully opened and accessible
sqlplus / as sysdba
```

```
show pdbs

    CON_ID CON_NAME                  OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         2 PDB$SEED                       READ ONLY  NO
         3 ORCLPDB1                       READ WRITE NO
         4 PDB1                           READ WRITE NO
         5 PDB1_CLONE                     READ WRITE NO
         6 PDB1_NOCOPY                    READ WRITE NO
         7 PDB1_REFRESH                   READ ONLY  NO
         8 PDB1_SNAP1                     READ WRITE NO
         9 PDB1_SNAP2                     READ WRITE NO

sqlplus system/Oracle_4U@myoracledb:1521/PDB1_NOCOPY

SQL*Plus: Release 21.0.0.0.0 - Production on Wed Oct 6 15:04:44 2021
Version 21.3.0.0.0

Copyright (c) 1982, 2021, Oracle.  All rights reserved.

Last Successful login time: Wed Oct 06 2021 12:58:09 +00:00

Connected to:
Oracle Database 21c Enterprise Edition Release 21.0.0.0.0 - Production
Version 21.3.0.0.0

SQL>
```

This concludes the backup and restore lab.

PDB Flashback

The flashback database functionality can be used at PDB level, without impacting neither the CDB, nor other PDBs.

```
-- Connect to ORCLPDB1 PDB, and create a local user

sqlplus system/"Oracle_4U"@myoracledb:1521/ORCLPDB1

create user TEST_FLASHBACK identified by "Oracle_4U" default tablespace USERS
temporary tablespace TEMP;
grant connect, resource to TEST_FLASHBACK;
alter user TEST_FLASHBACK quota unlimited on USERS;
exit

-- Connect to TEST_FASHBACK schema and create a table

sqlplus TEST_FLASHBACK/"Oracle_4U"@myoracledb:1521/ORCLPDB1
create table tt (c1 number);
insert into tt values (1);
insert into tt values (2);
commit;
```

```
exit

-- Now we create a restore point
sqlplus sys/Oracle_4U@myoracledb:1521/ORCLPDB1 as sysdba
create restore point RP_1 guarantee flashback database;
exit

-- Now connect to TEST_FLASHBACK schema and drop the table
sqlplus TEST_FLASHBACK/"Oracle_4U"@myoracledb:1521/ORCLPDB1
drop table tt;
exit

-- Now connect to the CDB and flashback the PDB to the restore point

sqlplus / a sysdba
alter pluggable database ORCLPDB1 close immediate;
flashback pluggable database ORCLPDB1 to restore point RP_1;

show pdbs

    CON_ID CON_NAME                        OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         2 PDB$SEED                       READ ONLY  NO
         3 ORCLPDB1                       MOUNTED

alter pluggable database ORCLPDB1 open resetlogs;

show pdbs

    CON_ID CON_NAME                        OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         2 PDB$SEED                       READ ONLY  NO
         3 ORCLPDB1                       READ WRITE NO

-- Now connect to TEST_FLASHBACK schema and check the table is back

sqlplus TEST_FLASHBACK/"Oracle_4U"@myoracledb:1521/ORCLPDB1

select * from tt;

       C1
----------
         1
         2
```

Clean-up the environment

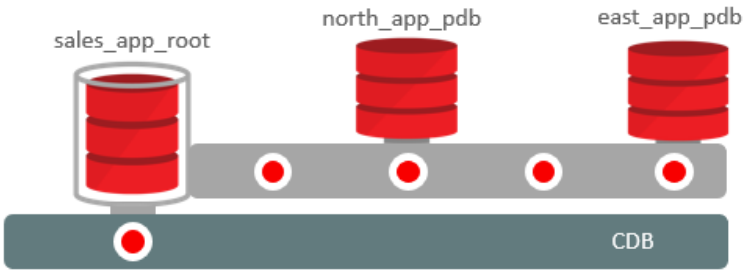
```
sqlplus / as sysdba

alter pluggable database all close;
```

```
show pdbs

    CON_ID CON_NAME                        OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         2 PDB$SEED                       READ ONLY  NO
         3 ORCLPDB1                       MOUNTED
         4 PDB1                           MOUNTED
         5 PDB1_CLONE                     MOUNTED
         6 PDB1_NOCOPY                    MOUNTED
         7 PDB1_REFRESH                   MOUNTED
         8 PDB1_SNAP1                     MOUNTED
         9 PDB1_SNAP2                     MOUNTED

drop pluggable database PDB1_SNAP2 including datafiles;
drop pluggable database PDB1_SNAP1 including datafiles;
drop pluggable database PDB1_REFRESH including datafiles;
drop pluggable database PDB1_NOCOPY including datafiles;
drop pluggable database PDB1_CLONE including datafiles;
drop pluggable database PDB1 including datafiles;
alter pluggable database ORCLPDB1 open;

show pdbs

    CON_ID CON_NAME                        OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         2 PDB$SEED                       READ ONLY  NO
         3 ORCLPDB1                       READ WRITE NO
```

Application Container

Create an application root

In the following lab, we'll create and use an application container. Application containers were introduced in 12.2. They are meant to be an additional container between the CDB and the application PDB. In the application container reside one or several applications, composed by metadata, code and data common to all the application PDB.



At first, we connect to the CDB and create a PDB, in which we will install an application.

```
$ sqlplus / as sysdba
set sqlprompt CDB$ROOT>
set linesize 1000
col app_name format a30
col name format a20
SELECT con_id, name, open_mode, application_root app_root,
                 application_pdb app_pdb, application_seed app_seed
         from v$containers
         order by con_id;

    CON_ID NAME            OPEN_MODE  APP APP APP
---------- ------------------- ---------- --- --- ---
         1 CDB$ROOT         READ WRITE NO  NO  NO
         2 PDB$SEED         READ ONLY  NO  NO  NO
         3 ORCLPDB1         READ WRITE NO  NO  NO

--- Create an application container: this is a special PDB

CREATE PLUGGABLE DATABASE sales_app_root AS APPLICATION CONTAINER
         ADMIN USER appadmin IDENTIFIED BY "Oracle_4U";

SELECT con_id, name, open_mode, application_root app_root,
                application_pdb app_pdb, application_seed app_seed
         from v$containers
         order by con_id;

    CON_ID NAME            OPEN_MODE  APP APP APP
---------- ------------------- ---------- --- --- ---
         1 CDB$ROOT         READ WRITE NO  NO  NO
         2 PDB$SEED         READ ONLY  NO  NO  NO
         3 ORCLPDB1         READ WRITE NO  NO  NO
         4 SALES_APP_ROOT   MOUNTED    YES NO  NO

alter pluggable database sales_app_root open;
show pdbs

    CON_ID CON_NAME                  OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         2 PDB$SEED                  READ ONLY  NO
         3 ORCLPDB1                  READ WRITE NO
         4 SALES_APP_ROOT            READ WRITE NO
exit
```

Install an application in the application root

Once created, we connect to the application container, in order to install our first application.

```
sqlplus sys/"Oracle_4U"@myoracledb:1521/sales_app_root as sysdba

set sqlprompt SALES_APP_ROOT>
select app_name, app_version, app_id, app_status,
                      app_implicit implicit
```

```
              from dba_applications;

select CON_ID, name, CON_UID, Guid from v$containers;

---- Begin the installation of application sales_app

ALTER PLUGGABLE DATABASE APPLICATION sales_app BEGIN INSTALL '1.0';

col app_name format a40
select app_name, app_version, app_id, app_status, app_implicit implicit from
dba_applications;
```

We create a common user, and grant some privileges.

```
-- Create a common user

CREATE USER sales_app_user IDENTIFIED BY "Oracle_4U" CONTAINER=ALL;
GRANT CREATE SESSION, create procedure, CREATE TABLE, unlimited tablespace TO
sales_app_user;
```

We create a table that will be shared in all the application PDB. Note that metadata only is shared, meaning that the table definition is common to all application PDB, but that each application will have its own private data in that table.

```
CREATE TABLE sales_app_user.customers SHARING=METADATA
               ( cust_id    NUMBER constraint cust_pk primary key,
                 cust_name  varchar2(30),
                 cust_add   varchar2(30),
                 cust_zip   NUMBER
               );
```

Then we execute the application installation with an "end install" sentence.

```
ALTER PLUGGABLE DATABASE APPLICATION sales_app END INSTALL '1.0';

 select app_name, app_version, app_id, app_status, app_implicit implicit from
dba_applications;
```

Create application PDBs and SYNC them with the application root

At this stage, we have created common objects for the "sales_app" application, into the application root sales_app_root. In the next steps, we will create an application PDB that will somehow inherit from these common objects.

Create an application PDB north_app_pdb from the seed PDB (PDB$Seed):

Note we are running the creation still from the application root:

```
CREATE PLUGGABLE DATABASE north_app_pdb
               ADMIN USER pdb_admin IDENTIFIED BY "Oracle_4U";

show pdbs

    CON_ID CON_NAME                  OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         4 SALES_APP_ROOT            READ WRITE NO
         5 NORTH_APP_PDB             MOUNTED

ALTER PLUGGABLE DATABASE north_app_pdb OPEN;
show pdbs

    CON_ID CON_NAME                  OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         4 SALES_APP_ROOT            READ WRITE NO
         5 NORTH_APP_PDB             READ WRITE NO

conn sys/"Oracle_4U"@myoracledb:1521/north_app_pdb as sysdba

set sqlprompt NORTH_APP_PDB>

desc sales_app_user.customers -- The table shoudn't exist yet

ALTER PLUGGABLE DATABASE APPLICATION sales_app SYNC;

SELECT con_id, name, open_mode, application_root app_root,
            application_pdb app_pdb, application_seed app_seed
from v$containers
order by con_id;

desc sales_app_user.customers -- Now the table should exist

Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
-
 CUST_ID                                   NOT NULL NUMBER
 CUST_NAME                                          VARCHAR2(30)
 CUST_ADD                                           VARCHAR2(30)
 CUST_ZIP                                           NUMBER
```

Next we create the "east_app_pdb" application PDB:

```
conn sys/"Oracle_4U"@myoracledb:1521/sales_app_root as sysdba

set sqlprompt SALES_APP_ROOT>


CREATE PLUGGABLE DATABASE east_app_pdb
               ADMIN USER pdb_admin IDENTIFIED BY "Oracle_4U";
```

```
alter pluggable database east_app_pdb open;

show pdbs

    CON_ID CON_NAME                  OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         4 SALES_APP_ROOT           READ WRITE NO
         5 NORTH_APP_PDB            READ WRITE NO
         6 EAST_APP_PDB             READ WRITE NO

SELECT c.name,  aps.con_uid,   aps.app_name,   aps.app_version,
aps.app_status
      FROM   dba_app_pdb_status aps
       JOIN v$containers c
               ON c.con_uid = aps.con_uid
      WHERE  aps.app_name = 'SALES_APP';

NAME
--------------------------------------------------------------------------------
-
   CON_UID APP_NAME
---------- ----------------------------------------
APP_VERSION                APP_STATUS
-------------------------------- ------------
NORTH_APP_PDB
1113140000 SALES_APP
1.0                        NORMAL

-- We need to SYNC east_app_pdb

conn sys/"Oracle_4U"@myoracledb:1521/east_app_pdb as sysdba

set sqlprompt EAST_APP_PDB>

desc sales_app_user.customers

ERROR:
ORA-04043: object sales_app_user.customers does not exist

ALTER PLUGGABLE DATABASE APPLICATION sales_app SYNC;

desc sales_app_user.customers

Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
-
 CUST_ID                                   NOT NULL NUMBER
 CUST_NAME                                          VARCHAR2(30)
 CUST_ADD                                           VARCHAR2(30)
 CUST_ZIP                                           NUMBER

-- Show the objects shared through the application root
```

```
select app.app_name, obj.owner, obj.object_name, obj.object_type,
               obj.sharing, obj.application
      from dba_objects obj, dba_applications app
      where obj.owner in
          (select username from dba_users
           where oracle_maintained = 'N')
          and obj.application = 'Y'
      and obj.created_appid = app.app_id;
```

Upgrade the application

Connect to the application root, and add some new objects, that will be synchronized with the application PDB later.

```
conn sys/"Oracle_4U"@myoracledb:1521/sales_app_root as sysdba
set sqlprompt SALES_APP_ROOT>

ALTER PLUGGABLE DATABASE APPLICATION sales_app BEGIN UPGRADE '1.0' TO '2.0';

alter user SALES_APP_USER quota 50m on system;

select app_name, app_version, app_id, app_status,
                app_implicit implicit
         from dba_applications
         where app_name = 'SALES_APP';

APP_NAME                         APP_VERSION
-------------------------------- ------------------------------
    APP_ID APP_STATUS     I
---------- ------------ -
SALES_APP                        1.0
         3 UPGRADING N

--- Create an "extended data" table

create table SALES_APP_USER.zip_codes
                sharing=extended data
                   (zip_code number,
                   country varchar2(20));

insert into sales_app_user.zip_codes values (1, 'Spain(root)');
commit;

-- Create a data shared table

create table SALES_APP_USER.products
                 sharing=data
                   (prod_id number,
                    prod_name varchar2(20),
                    price number);

insert into SALES_APP_USER.products values (1, 'prod1 (root)', 111);
```

```
commit;

ALTER PLUGGABLE DATABASE APPLICATION sales_app END UPGRADE TO '2.0';
```

SYNC the changes in the application PDBs

Connect to "east_app_pdb" application PDB, and synchronize the changes introduced in version 2.0 of the application.

```
conn sys/"Oracle_4U"@myoracledb:1521/east_app_pdb as sysdba
set sqlprompt SALES_APP_EAST>

alter pluggable database application sales_app sync;
```

Connect to "north_app_pdb" application PDB, and synchronize the changes introduced in version 2.0 of the application.

```
conn sys/"Oracle_4U"@myoracledb:1521/north_app_pdb as sysdba

set sqlprompt SALES_APP_NORTH>

alter pluggable database application sales_app sync;


select * from sales_app_user.zip_codes;

  ZIP_CODE COUNTRY
---------- --------------------
         1 Spain(root)

select * from sales_app_user.products;

   PROD_ID PROD_NAME                    PRICE
---------- -------------------- ----------
         1 prod1 (root)            111
```

If we try to add some data into the "products" table, we will get an error, as this table was shared in "data" mode from the application root. Data in this table must be added/modified from the application root only.

```
SQL> insert into  sales_app_user.products values (2, 'prod2(north)', 111);
insert into   sales_app_user.products values (2, 'prod2(north)', 111)
              *
ERROR at line 1:
ORA-65097: DML into a data link table is outside an application action
```

We try to add a new row in the "zip_codes" table: this operation is successful, as this table was shared in "extended data" mode from the application root. The rows that were created from the application root are common to all the application PDB, and can be modified only from the application root. But each application PDB might add some private data to that table.

```
insert into sales_app_user.zip_codes values (2, 'USA (north)');
commit;
1 row created.

SQL>

Commit complete.

SQL>
SQL>
SQL> select * from sales_app_user.zip_codes;

  ZIP_CODE COUNTRY
---------- --------------------
         1 Spain(root)
         2 USA (north)
```

We can also add a row in the "customers" table, as it has been shared in "metadata" mode from the application root. In this table, each application PDB can create its own private data.

```
insert into sales_app_user.customers
              values ('1', 'Cust1(north)', 'USA (north) address', 2);
commit;

select * from sales_app_user.customers;



   CUST_ID CUST_NAME                      CUST_ADD
CUST_ZIP
---------- ------------------------------ ------------------------------ ------
----
         1 Cust1(north)                   USA (north) address
2
```

If we try to add a duplicate key in this table, for example with CUST_ID=1, it will fail, as CUST_ID is the primary key of this table.

```
SQL> insert into sales_app_user.customers values ('1', 'Another Cust1(north)',
'USA (north) address', 2);
insert into sales_app_user.customers values ('1', 'Another Cust1(north)', 'USA
(north) address', 2)
*
ERROR at line 1:
```

```
ORA-00001: unique constraint (SALES_APP_USER.CUST_PK) violated
```

Add local objects in the applications PDBs

In each application PDB, we can still create local objects, independent from the application.

The following example creates a local table in the application PDB north_app_pdb.

```
create table sales_app_user.local_tbl(id number);

insert into sales_app_user.local_tbl values (1);

commit;

select * from sales_app_user.local_tbl;

        ID
----------
         1
```

Check and use the application container

Now let's check that data added to "customers" and "zip_codes" tables in "north_app_pdb" is not visible when connected to "east_app_pdb".

```
conn sys/"Oracle_4U"@myoracledb:1521/east_app_pdb as sysdba

set sqlprompt SALES_APP_EAST>


SQL> select * from sales_app_user.zip_codes;

  ZIP_CODE COUNTRY
---------- --------------------
         1 Spain(root)

select * from sales_app_user.customers;

no rows selected

select * from sales_app_user.products;

   PROD_ID PROD_NAME                 PRICE
---------- -------------------- ----------
         1 prod1 (root)                111

insert into sales_app_user.zip_codes values (2, 'USA (east)');
commit;
select * from sales_app_user.zip_codes;
```

```
  ZIP_CODE COUNTRY
---------- --------------------
         1 Spain(root)
         2 USA (east)

insert into sales_app_user.customers
             values ('1', 'Cust1(east)', 'USA (east) address', 2);
commit;
select * from sales_app_user.customers;

   CUST_ID CUST_NAME                      CUST_ADD
CUST_ZIP
---------- ------------------------------ ------------------------------ ------
----
         1 Cust1(east)                    USA (east) address
2

-- A select against "north_app_pdb" local table fails

select * from sales_app_user.local_tbl;
select * from sales_app_user.local_tbl
              *
ERROR at line 1:
ORA-00942: table or view does not exist

-- we can create our own private table here

create table sales_app_user.local_tbl(id number);
insert into sales_app_user.local_tbl values (2);
commit;

select * from sales_app_user.local_tbl;

        ID
----------
         2

--- Now let's run some queries from the application root

conn sys/"Oracle_4U"@myoracledb:1521/sales_app_root as sysdba

set sqlprompt SALES_APP_ROOT>


--- Let's run a select * from sales_app_user.customers;

select * from sales_app_user.customers;

no rows selected

--- execute a "show pdbs" command to get the CON_ID of the application PDB

show pdbs
```

```
    CON_ID CON_NAME                  OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         4 SALES_APP_ROOT           READ WRITE NO
         5 NORTH_APP_PDB            READ WRITE NO
         6 EAST_APP_PDB             READ WRITE NO

--- Now we use the container clause to get a consolidated view of the customers
table accross the application PDB

select * from containers(sales_app_user.customers)
               where CON_ID in (5,6); -- 6,7 are the CON_ID of the application
PDB

   CUST_ID CUST_NAME                      CUST_ADD
---------- ------------------------------ ------------------------------
  CUST_ZIP     CON_ID
---------- ----------
         1 Cust1(east)                    USA (east) address
         2         6

         1 Cust1(north)             USA (north) address
         2         5
```

We can update the application so that access to the " sales_app_user.customers " is made implicitely with the CONTAINER clause, without the need to specify it.

This must be configured using the "containers_default" clause:

```
ALTER PLUGGABLE DATABASE APPLICATION sales_app
               begin UPGRADE '2.0' TO '2.1';
ALTER TABLE sales_app_user.customers ENABLE containers_default;

ALTER PLUGGABLE DATABASE APPLICATION sales_app
               end UPGRADE  TO '2.1';

conn sys/"Oracle_4U"@myoracledb:1521/north_app_pdb as sysdba

set sqlprompt NORTH_APP_PDB>

ALTER PLUGGABLE DATABASE APPLICATION sales_app SYNC;

conn sys/"Oracle_4U"@myoracledb:1521/east_app_pdb as sysdba

ALTER PLUGGABLE DATABASE APPLICATION sales_app SYNC;

conn sys/"Oracle_4U"@myoracledb:1521/sales_app_root as sysdba

set sqlprompt SALES_APP_ROOT>

show pdbs --- (to get the CON_ID)

    CON_ID CON_NAME                  OPEN MODE  RESTRICTED
```

```
---------- ------------------------------ ---------- ----------
         4 SALES_APP_ROOT                 READ WRITE NO
         5 NORTH_APP_PDB                  READ WRITE NO
         6 EAST_APP_PDB                   READ WRITE NO

select * from sales_app_user.customers;

   CUST_ID CUST_NAME                      CUST_ADD
---------- ------------------------------ ------------------------------
  CUST_ZIP     CON_ID
---------- ----------
         1 Cust1(north)                   USA (north) address
         2          5

         1 Cust1(east)                    USA (east) address
         2          6
```

Clean-up the environment (Optional):

```
conn sys/"Oracle_4U"@myoracledb:1521/sales_app_root as sysdba
alter pluggable database NORTH_APP_PDB close immediate;
alter pluggable database EAST_APP_PDB close immediate;
drop pluggable database NORTH_APP_PDB including datafiles;
drop pluggable database EAST_APP_PDB including datafiles;
conn / as sysdba
alter pluggable database SALES_APP_ROOT close immediate;
drop pluggable database SALES_APP_ROOT including datafiles;
exit
```

Management of users, parameters and resources with Multitenant

In this section, we will see how to create users and objects, modify parameters and manage resources, in a CDB as well as in a PDB.

User creation at CDB and PDB level

Create common users (CDB level)

Create a common user using the CONTAINER clause:

```
sqlplus / as sysdba

--- Common user are created from CDB$ROOT
--- Their name must start with "C##"

CREATE USER c##user1 IDENTIFIED BY "Oracle_4U" CONTAINER=ALL;
GRANT CREATE SESSION TO c##user1 CONTAINER=ALL;
```

Create a common user without using the CONTAINER clause:

```
SQL> conn / as sysdba

--- The CONTAINER clause is implicit when creating a common user from CDB$ROOT

CREATE USER c##user2 IDENTIFIED BY "Oracle_4U";
GRANT CREATE SESSION TO c##user2;
```

Check the created users and their status:

```
SQL> Conn / as sysdba

select
USERNAME,ACCOUNT_STATUS,PROFILE,CREATED,DEFAULT_TABLESPACE,TEMPORARY_TAB
LESPACE from dba_users where username like 'C##%';

USERNAME
--------------------------------------------------------------------------------
ACCOUNT_STATUS
--------------------------------
PROFILE
--------------------------------------------------------------------------------
CREATED   DEFAULT_TABLESPACE              TEMPORARY_TABLESPACE
--------- ------------------------------ ------------------------------
C##USER2
OPEN
DEFAULT
07-OCT-21 USERS                          TEMP
C##USER1
OPEN
DEFAULT
07-OCT-21 USERS                          TEMP
```

Create local users

- The user that creates other users must have been granted the CREATE USER privilege.
- A local user name cannot start with "C##" or "c##".
- The local user name must be unique into the PDB.

Connect to another container while connected to a common user:

```
set echo on

show pdbs

    CON_ID CON_NAME                  OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         2 PDB$SEED                       READ ONLY  NO
         3 ORCLPDB1                       READ WRITE NO

CONN / AS SYSDBA
ALTER SESSION SET CONTAINER = ORCLPDB1;
```

Create a new tablespace for user orclpdb1_user_local1:

```
create tablespace data_tbs;
```

Show the datafiles for this new tablespace:

```
col FILE_NAME format a60
col TABLESPACE_NAME format a50
select tablespace_name, File_name from dba_data_files where
tablespace_name='DATA_TBS';

TABLESPACE_NAME
--------------------------------------------------
FILE_NAME
------------------------------------------------------------
DATA_TBS
/opt/oracle/oradata/ORCLCDB/C9E72D01B8EAF15EE0538701000AAEFE
/datafile/o1_mf_data_tbs_joxmsh7c_.dbf
```

Create the local user using the CONTAINER clause:

```
CREATE USER orclpdb1_user_local1 IDENTIFIED BY "Oracle_4U" default tablespace
DATA_TBS CONTAINER=CURRENT;

GRANT CREATE SESSION TO orclpdb1_user_local1;
```

Connect to system user in the PDB:

```
CONN system/"Oracle_4U"@myoracledb:1521/orclpdb1
```

Create another local user:

```
CREATE USER orclpdb2_user_local2 IDENTIFIED BY "Oracle_4U" default tablespace
DATA_TBS;

GRANT CREATE SESSION TO orclpdb2_user_local2;
```

Show the users created in the PDB:

```
set lines 999
set pages 999
col username format a20
col profile format a15
select
USER_ID,USERNAME,ACCOUNT_STATUS,DEFAULT_TABLESPACE,TEMPORARY_TABLESPACE,CREATED
,PROFILE from dba_users where username like 'ORCLPDB%';

   USER_ID USERNAME       ACCOUNT_STATUS                   DEFAULT_TABLESPACE
      TEMPORARY_TABLESPACE            CREATED PROFILE
---------- -------------------- ------------------------------ --------------
---------------- ------------------------------ --------- ---------------
       112 ORCLPDB1_USER_LOCAL1 OPEN                           DATA_TBS
      TEMP                         07-OCT-21 DEFAULT
       113 ORCLPDB2_USER_LOCAL2 OPEN                           DATA_TBS
      TEMP                         07-OCT-21 DEFAULT
```

Grant unlimited quota for those users on the "data_tbs" tablespace:

```
ALTER USER orclpdb2_user_local2 QUOTA UNLIMITED ON DATA_TBS;
ALTER USER orclpdb1_user_local1 QUOTA UNLIMITED ON DATA_TBS;
```

Grant CREATE TABLE to those users:

```
grant create table to ORCLPDB1_USER_LOCAL1;
grant create table to ORCLPDB2_USER_LOCAL2;
```

Create a sample table:

```
CONN orclpdb1_user_local1/"Oracle_4U"@myoracledb:1521/orclpdb1

CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

Create a new row and select it:

```
INSERT INTO Persons (PersonID, LastName, FirstName, Address, City)
VALUES (1, 'Garcia', 'Pedro', 'Calle Amatista, 43', 'Madrid');
Commit;

Select * from Persons;
```

Resource manager PDB performance profiles

We can use Resource Manager PDB performance profiles to manage the resources among PDB.

In the following steps, we will:

- Create a new CDB plan
- Assign PDB profile directives to this plan
- Activate the new plan

```
--- Connect to the CDB and create a new plan

sqlplus / as sysdba

DECLARE
  l_plan VARCHAR2(30) := 'CDB_PDB_PROFILE';
BEGIN
  DBMS_RESOURCE_MANAGER.clear_pending_area;
  DBMS_RESOURCE_MANAGER.create_pending_area;

  DBMS_RESOURCE_MANAGER.create_cdb_plan(
    plan    => l_plan,
    comment => 'A test CDB resource plan using PDB profiles');

  DBMS_RESOURCE_MANAGER.create_cdb_profile_directive(
    plan                  => l_plan,
    profile               => 'GOLD',
    shares                => 3,
    utilization_limit     => 100,
    parallel_server_limit => 100);

  DBMS_RESOURCE_MANAGER.create_cdb_profile_directive(
    plan                  => l_plan,
    profile               => 'SILVER',
    shares                => 2,
    utilization_limit     => 50,
    parallel_server_limit => 50);

  DBMS_RESOURCE_MANAGER.validate_pending_area;
  DBMS_RESOURCE_MANAGER.submit_pending_area;
END;
/
```

```
--- Check the new plan

COLUMN plan FORMAT A30
COLUMN comments FORMAT A30
COLUMN status FORMAT A10
SET LINESIZE 100

SELECT plan_id,
       plan,
       comments,
       status,
       mandatory
FROM   dba_cdb_rsrc_plans
WHERE  plan = 'CDB_PDB_PROFILE';

   PLAN_ID PLAN                      COMMENTS                  STATUS     MAN
---------- ------------------------------ ------------------------------ ------
---- ---
     77205 CDB_PDB_PROFILE         A test CDB resource plan using     NO
                                    PDB profiles

COLUMN plan FORMAT A30
COLUMN pluggable_database FORMAT A25
COLUMN profile FORMAT A25
SET LINESIZE 150 VERIFY OFF

SELECT plan,
       pluggable_database,
       profile,
       shares,
       utilization_limit AS util,
       parallel_server_limit AS parallel
FROM   dba_cdb_rsrc_plan_directives
WHERE  plan = 'CDB_PDB_PROFILE'
ORDER BY plan, pluggable_database, profile;

PLAN                           PLUGGABLE_DATABASE   PROFILE                    SHARES
UTIL   PARALLEL
------------------------------ ------------------------ ----------------------
--- ---------- ---------- ----------
CDB_PDB_PROFILE            ORA$AUTOTASK
90
CDB_PDB_PROFILE            ORA$DEFAULT_PDB_DIRECTIVE                          1
CDB_PDB_PROFILE                                 GOLD                          3
CDB_PDB_PROFILE                                 SILVER                        2
50     50
```

Shares represent the proportion of the CDB resources available to the PDB: if in a CDB we have two PDB:

- PDB1 with the GOLD profile (3 shares)

- PDB2 with the SILVER profile (2 shares)

PDB1 will be given 3/(3+2) = 3/5 = 60% of the resources and PDB2 will be given 2/(3+2) = 2/5 = 40% of the resources.

If limits are defined, like for example in profile SILVER (50% CPU Utilization and 50% parallel servers), they represent a hard limit to the CDB resources utilization.

If we add a new PDB (PDB3) in the CDB, with no PDB profile, it will inherit the number of shares and limits of the " ORA$DEFAULT_PDB_DIRECTIVE ", in this case 1 share and no limit.

So now, the percentage of resources available to each PDB changes:

- PDB1: 3/(3+2+1) = 50% (instead of 60%)
- PDB2: 2/(3+2+1) = 33.3% (instead of 40%)
- PDB3: 1(3+2+1) = 16.7%

So shares allow a dynamic redistribution of the CDB resources when PDB are created in or removed from the CDB.

```
-- Now we will add a new PDB profile, called BRONZE, to our plan

DECLARE
  l_plan VARCHAR2(30) := 'CDB_PDB_PROFILE';
BEGIN
  DBMS_RESOURCE_MANAGER.clear_pending_area;
  DBMS_RESOURCE_MANAGER.create_pending_area;

  DBMS_RESOURCE_MANAGER.create_cdb_profile_directive(
    plan                  => l_plan,
    profile               => 'BRONZE',
    shares                => 1,
    utilization_limit     => 25,
    parallel_server_limit => 25);

  DBMS_RESOURCE_MANAGER.validate_pending_area;
  DBMS_RESOURCE_MANAGER.submit_pending_area;
END;
/

--- Check your plan directives
COLUMN plan FORMAT A30
COLUMN pluggable_database FORMAT A25
COLUMN profile FORMAT A25
SET LINESIZE 150 VERIFY OFF

SELECT plan,
       pluggable_database,
       profile,
       shares,
       utilization_limit AS util,
       parallel_server_limit AS parallel
```

```
FROM   dba_cdb_rsrc_plan_directives
WHERE  plan = 'CDB_PDB_PROFILE'
ORDER BY plan, pluggable_database, profile;

PLAN                         PLUGGABLE_DATABASE   PROFILE                    SHARES
UTIL   PARALLEL
------------------------------ ------------------------ ---------------------
--- ---------- ---------- ----------
CDB_PDB_PROFILE              ORA$AUTOTASK
90
CDB_PDB_PROFILE              ORA$DEFAULT_PDB_DIRECTIVE                          1
CDB_PDB_PROFILE                                  BRONZE                         1
25     25
CDB_PDB_PROFILE                                  GOLD                           3
CDB_PDB_PROFILE                                  SILVER                         2
50     50

-- A new directive has been created
-- We can also modify an existing directive

DECLARE
  l_plan VARCHAR2(30) := 'CDB_PDB_PROFILE';
BEGIN
  DBMS_RESOURCE_MANAGER.clear_pending_area;
  DBMS_RESOURCE_MANAGER.create_pending_area;

  DBMS_RESOURCE_MANAGER.update_cdb_profile_directive(
    plan                     => l_plan,
    profile                  => 'bronze',
    new_shares               => 1,
    new_utilization_limit    => 20,
    new_parallel_server_limit => 20);

  DBMS_RESOURCE_MANAGER.validate_pending_area;
  DBMS_RESOURCE_MANAGER.submit_pending_area;
END;
/

--- Now we can assign a profile to a PDB
--- Connect to ORCLPDB1 and check its profile
alter session set container=ORCLPDB1;

show parameter DB_PERFORMANCE_PROFILE

NAME                                 TYPE     VALUE
------------------------------------ ----------- ------------------------------
db_performance_profile               string

--- This is NULL by default, meaning that ORA$DEFAULT_PDB_DIRECTIVE has been
applied to that PDB. Change the PDB profile to GOLD:

ALTER SYSTEM SET DB_PERFORMANCE_PROFILE=gold SCOPE=SPFILE;
ALTER PLUGGABLE DATABASE CLOSE IMMEDIATE;
ALTER PLUGGABLE DATABASE OPEN;
```

```
SHOW PARAMETER DB_PERFORMANCE_PROFILE

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
db_performance_profile               string      GOLD

--- Create a new PDB

connect / as sysdba
create pluggable database ORCLPDB2 from ORCLPDB1;
alter pluggable database ORCLPDB2 open;

show pdbs

    CON_ID CON_NAME                  OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         2 PDB$SEED                     READ ONLY  NO
         3 ORCLPDB1                     READ WRITE NO
         4 ORCLPDB2                     READ WRITE NO

-- Connect to ORCLPDB2 and check its profile

alter session set container = ORCLPDB2;
SHOW PARAMETER DB_PERFORMANCE_PROFILE

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
db_performance_profile               string      GOLD

--- This was inherited from the source database, but might be changed:
ALTER SYSTEM SET DB_PERFORMANCE_PROFILE=bronze SCOPE=SPFILE;
ALTER PLUGGABLE DATABASE CLOSE IMMEDIATE;
ALTER PLUGGABLE DATABASE OPEN;

SHOW PARAMETER DB_PERFORMANCE_PROFILE

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
db_performance_profile               string      BRONZE

--- For the plan to be active, we need to enable it at CDB level

connect / as sysdba
ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = 'CDB_PDB_PROFILE';
```