

# Lesson 4

## Multitenant Administration

## Objectives

- Startup and shutdown a CDB
- Open and close pluggable containers
- Tablespace management
- Create local and common objects
- Understand use of roles and privileges
- Services in a multitenant environment
- Set instance parameters

## Database startup and shutdown

3

- True startup and shutdown occur only in the root
- Pluggable containers can be mounted or open
- Within a container, the usual syntax works
  - Shutdown commands transition from open to mount
  - Startup commands from mount to open
  - Open can be read write or read only
- By default, pluggable containers are mounted

As there is only one database and one instance and one controlfile, it is clear that startup of the instance and mounting the controlfile can be done only from the root container. The pluggable containers can then be opened and closed, as long as the root itself is open.

## Open and close pluggable containers

4

- Open mode seen in v\$containers, v\$pdb, cdb\_pdb
- From the root container:
  - `alter pluggable database <pdb> open [read write | only];`
  - `alter pluggable database <pdb> close [immediate];`
- From within the PDB:
  - `connect sys@<pdb> as sysdba`
  - `startup | alter database open`
  - Usual SHUTDOWN commands function
- PDBs do not shutdown
  - The PDB transitions to mount mode
  - The service remains registered with the listener
  - Only password file authenticated users can connect



Closing a PDB is similar to taking its tablespaces offline. The datafiles are checkpointed, and V\$DATAFILE\_HEADER reports that they are not fuzzy. They do however remain online.

If a non-CDB is mounted, none of its services will be registered with the database listener. In a CDB, the default service for each PDB is always registered as long as the root container is open; there is no need to register PDBs statically in the listener.ora file. This permits a SYSDBA connection over SQL\*Net which can be used to open the container, without the need to log on to the root container.

## Automating PDB startup

5

- To open PDBs automatically, create a trigger:
  - `alter session set container=cdb$root;`
  - `create trigger open_pdb after startup on database`
  - `begin`
  - `execute immediate 'alter pluggable database all open';`
  - `end;`
  - `/`
- Other trigger types:
  - `after clone on pluggable database`
  - `before unplug on pluggable database`
- Or preserve the PDB state following shutdown
  - `ALTER PLUGGABLE DATABASE ... SAVE STATE;`
  - `ALTER PLUGGABLE DATABASE ... DISCARD STATE;`
  - `ALTER PLUGGABLE DATABASE ALL SAVE STATE;`

 SKILLBUILDERS

By default, pluggable containers will be mounted when a CDB is opened. The `SAVE STATE` command instructs the CDB to open or mount the container on startup depending on its current state. The saved state is visible in `CDB_PDB_SAVED_STATES`. The triggers give more precise control.

The `AFTER CLONE` and `BEFORE UNPLUG` triggers should be created in PDBs. They fire once and then self-delete. Usage cases could include configuring external references such as database links, or controlling the availability of the job system, or setting user passwords. A simple example:

```
create trigger post_clone after clone on pluggable database
begin
execute immediate 'alter system set job_queue_processes=0';
end;
/
```

## Managing PDB tablespaces

6

- Normal syntax for tablespace and file management
  - Create/drop/alter tablespace
  - Rename/move/resize datafiles
- Each PDB sets its own default tablespaces
- OMF parameters can be different per PDB
- Logfile commands are not permitted in a PDB
- Set a space budget for a PDB:
  - `alter pluggable database ... storage (maxsize ... ) ;`

One issue that may arise in a large environment is controlling the use of space by individual PDBs. It is likely that different PDBs will have different administrators, and that they should reside on different disc groups or filesystems. Setting a space budget will prevent one DBA from inadvertently taking up excessive storage.

## Managing undo data

7

- Release 12.1: a single, shared, undo tablespace
  - Manage from the root
  - Undo commands in PDBs are questionable
- Release 12.2: local undo tablespace per container
  - Manage in each container
  - Permit use of some new features
- Conversion between modes is possible
  - Perform in upgrade mode
  - Tablespaces created automatically



Operating in shared undo mode simplifies administration and in a single tenant environment may be the best option. In a multi-tenancy database, local undo mode offers some advantages. For example, it is impossible for excessive undo data generated in one container (perhaps by badly structured transactions) to cause problems in other containers. Flashback of one container also becomes possible without the need for creating an auxiliary database, as is hot cloning of a container.

To convert between modes:

```
shutdown immediate;  
startup upgrade;  
alter database local undo ON | OFF ;  
shutdown immediate;  
startup;
```

An undo tablespace is created in the seed container, copied to all other containers when they open.

## Local and common users

8

- Local users exist in only one container
  - Cannot be created in the root container
  - Usual naming rules apply
- Common users exist in all containers
  - Created and managed in the root container
  - Must conform to the `common_user_prefix`
  - Propagated to all containers
- Oracle maintained users can break these rules
- Shown in [ CDB | DBA | ALL ] \_USERS

Within a pluggable container, user and schema management is exactly as in a non-CDB. These users are "local" users defined within the PDB's data dictionary. The username must be unique within the container, but can exist independently in multiple pluggable containers.

Users created in the root container are "common" users. Common users are automatically propagated into all PDBs that are open read write, and into closed or read only PDBs when they next open read write.

The `common_user_prefix` parameter specifies a prefix that cannot be used for local users and must be used for common users. The default is `C##`; changing this is likely to cause confusion. Oracle supplied common users (such as `SYSTEM`, or `CTXSYS`) do not conform to the prefix.

All administration of common users (such as password changes) is performed in the root container, though privileges can be granted in pluggable containers.

See the `%_USERS` views to determine whether a user is common or local, and Oracle maintained or user created.



## Roles and profiles

9

- Local roles and profiles
  - Create in a PDB as normal
  - Assign to local users or common users
- Common roles and profiles
  - Created and managed in the root container
  - Must conform to the common\_user\_prefix
  - Assign to local or common users per PDB
  - Assign to common users in the root container
- Shown in [CDB | DBA] \_ [ROLES | PROFILES ]

Along with users, roles and profiles are objects that may be common or local. Local roles and profiles are visible only within the container where they were created and are managed in the usual manner. Common roles and profiles are created in the root (named with the prefix) and propagated to all containers.

## Creating users

10

- Create a common user in the root:  

```
create user c##u1 identified by oracle  
[container=all];
```
- Create a local user in a pluggable container:  

```
create user scott identified by tiger  
[container=current];
```
- Common users are propagated to PDBs
  - On creation, to open read write PDBs
  - To remaining PDBs when opened read write
  - All user admin (other than grants) is in the root
  - Common users have a schema in each container
  - Dropping a common user drops from all containers

The CREATE USER command has a CONTAINER= clause. When in the root container, the default (and only permitted) value is ALL; when in a pluggable container, the default (and only permitted) value is CURRENT.

Even though a user may be common, his schema is not. The schema is managed individually in each container, and objects are private to that container. To create objects, the common user must be granted appropriate privileges and tablespace quotas in whatever container the objects will exist.

## Using roles

11

- Grant a common role commonly in the root:  

```
grant c##r1 to c##user1 [container=all] ;  
grant connect to c##user1 [container=all] ;
```
- Grant a common role locally in the root  

```
grant c##r1 to c##user1 [container=current] ;  
grant connect to c##user1 [container=current] ;
```
- Grant a local role locally in a PDB:  

```
grant r1 to user1 [container=current] ;
```
- Grant a common role to common user in a PDB:  

```
grant c##r1 to c##user1 [container=current] ;
```
- Grant a local role to a common user in a PDB:  

```
grant r1 to c##user1 [container=current] ;
```

Within a PDB, roles (common or local) can be granted to common or local users. The scope of the grant is that container, and `container=current` is default, and the only permitted syntax. In the root container (where all users are common) grants can be local (the default) or global according to the `container=` clause.

## Administrative roles

12

- The SYS% roles can be granted to any user
  - Local or common users
  - In the root with CONTAINER = ALL | CURRENT
  - In pluggable containers, local grant only
- Limited visibility in V\$PWFILERS
- Data dictionary view: CDB\_LOCAL\_ADMIN\_PRIVS
  - Populated only in the root container
  - Metadata linked object CDB\_LOCAL\_ADMINAUTH\$

The administrative privileges can be granted locally in any container or globally in the root. The password file itself is updated only when the grant is in the root. Grants within a PDB are written to the data dictionary.

When connected with one of these privileges, most privileged commands will run as normal syntactically though the effect when run in a pluggable container may be different from when run in the root container.

## Shared objects

13

- Create and use non-shared objects as normal
- Oracle maintained objects may be shared
  - Data links share metadata, with local content
  - Metadata links share DDL and data
  - Extended link objects have both root and local data
  - All maintenance is through Oracle supplied scripts
  - Documented in dba\_objects.sharing
- The CONTAINERS clause
  - Simulates data linked tables
  - Permits SQL access to all containers from the root
  - Objects must be the same structure in all containers
  - Owned by a common user
  - Views and synonyms extend functionality to local users



Extended metadata links have a shared root segment plus local segments.

If the same structured table, view, or synonym exists in a common user's schema in all containers, from the root SELECT and DML can be executed against it with the CONTAINERS clause. The object will have a CON\_ID column implicitly appended which identifies from which container any row comes:

```
cdba> sho con_name
```

```
CON_NAME
```

```
-----
```

```
CDB$ROOT
```

```
cdba> desc c##10.t1
```

Name	Null?	Type
-----		
C1		VARCHAR2(10)

```
cdba> select * from containers(c##10.t1) where con_id in (1,4);
```

```
C1          CON_ID
```

```
-----
```

root	1
jw2	4

cdba>

## Service names

14

- Each container has a unique `global_name`
  - Used as the default service for the container

```
ALTER DATABASE RENAME GLOBAL_NAME TO ... ;
```
- Create additional services with `DBMS_SERVICE`
  - The `SERVICE_NAMES` parameter cannot be set
  - The service name and net name must be unique
- Service conflicts checked when plugging in
- Grid Infrastructure associates services with PDBs

Use of services is essential in a multitenant environment, and slightly different from a non-CDB in that the `SERVICE_NAMES` parameter cannot be set. This should not be an issue, as more sophisticated techniques (the `DBMS_SERVICE` package and Grid Infrastructure) have been available for many years to manage services.

`DBMS_SERVICE.CREATE_SERVICE` will define the service in the data dictionary of the PDB where it is run. The `srvctl add service` command includes the switch `-pdb` to specify which PDB it is a part of.

## Instance parameters

15

- One spfile per CDB
- Parameters may have CDB or container scope
  - Documented in `v$parameter.ispdb_modifiable`
  - Some can be set in a PDB but will be ignored
- Locally set parameters: `v$system_parameter`
  - Stored in the container's dictionary
  - Move with the container
- Consider performance characteristics
  - PX limits set in the root, usage managed in the PDBs
  - Optimizer defaults set in the root, over-ride per PDB

Usually it is obvious whether a parameter is PDB modifiable, though there are some anomalies. As a general rule, anything to do with the shared structures will not be modifiable; anything that has a purely local scope should be.



## Memory and IO management

16

- Overall instance settings defined in the root
  - Automatic Memory Management
  - SGA and PGA targets and maximums
- In a PDB, minimum values can be imposed
  - For SGA, PGA, and individual pools
  - Total no more than 50% of the value set in the root
- Maximum IO rates can also be imposed per PDB
  - MAX\_IOPS, MAX\_MBPS
  - DBA\_HIST\_RSRC\_PDB\_METRIC may be useful

The Resource Manager is multitenant aware, and will be discussed later.

From release 12.2 it is possible to limit the impact of activity in one pluggable container on another by setting parameters to control memory usage. As in non-CDB, the parameters for individual structures (such as `DB_CACHE_SIZE`) determine minimums, and `SGA_TARGET` and `PGA_AGGREGATE_TARGET` specify maximums. `SGA_MIN_SIZE` specifies a minimum. If set, these parameters will control how much space in the buffer cache is occupied by blocks belonging to the datafiles of each container and how much library cache space is occupied by code running within it. In effect, the LRU algorithms are modified to give each container virtual caches with the instance's SGA.

The `v$cache` or `v$bh` views can be aggregated to see how much cache memory is actually being occupied by objects from a particular container at any given moment.

## Multitenant admin: summary

- Within a PDB, most admin is unchanged
- Startup/shutdown is always CDB level
- Individual PDBs can be opened and closed
- Users and roles may be local or common
- Services manage connecting sessions to PDBs

17

## Exercise

- Open and close PDBs
- Create tablespaces
- Create local and common users and roles
- Start services within PDBs

18

# SKILLBUILDERS



ORACLE DATABASE ADMINISTRATION

CLOUD ADMINISTRATION

ORACLE AND APEX HOSTING

APEX APPLICATION DESIGN AND DEVELOPMENT

TRAINING