

Lesson 2

Multitenant Architecture

Objectives

2

- Investigate the multitenant architecture
- Describe the container types: root, seed, pluggable
- Understand how Oracle Net establishes sessions
- Query the views that describe the environment
- Some features relevant to large deployments
 - Application containers
 - Proxy containers
 - Snapshot carousel

The CDB architecture

3

- Container database (CDB) hosts tenants (PDBs)
- There is one database and one instance
 - A (multiplexed) controlfile
 - An undo tablespace (12.2 permits one per PDB)
 - A single thread of redo
 - A set of background processes
 - One SCN
- In a RAC, replicate structures in the usual way

The CDB is in many ways a database like any other, and administered in the same way. The shared structures (controlfile, the redo log) are managed in exactly the same way as in a non-CDB. To the system administrator, a CDB is indistinguishable from a non-CDB: a set of files, a set of processes, and shared memory segments.

One shared undo tablespace may simplify the administration but it does remove some functionality. The decision on whether to use a shared undo tablespace or a local undo tablespace per pluggable container is not irrevocable.

The PDB architecture

4

- A PDB resides in the CDB
- No redo log or controlfile
- A PDB consists of a set of tablespaces
 - A system tablespace defining its local objects
 - Other tablespaces for user data
 - Temporary tablespace(s)
 - May or may not have its own undo tablespace
- Exposed to users as a database service
- Accessible only through Oracle Net connections

A PDB is nothing more than a set of tablespaces. These include a dictionary that defines all the objects local to the PDB: the data objects that exist as segments in its permanent tablespaces; programmatic objects such as stored PL/SQL and views; users and roles; grants.

There are two options for undo data. A single shared undo tablespace managed from the root container was the only option in the first release. From 12.2 it is possible to have separate undo tablespaces per PDB. This does involve a bit more management work and space usage, but enables some extra facilities that some, probably most, sites may require.

Each PDB has a name, unique within the CDB and ideally unique in the entire organization. This name is used to create a service that will be registered with the database listener. It is only by connecting through the listener to this service that users can log on to the PDB.

The CDB and the containers

5

- The CDB may contain many containers
- The cdb\$root container is the control structure
 - Manages redo and the controlfile (optionally, undo)
 - A dictionary of Oracle supplied and common objects
- Pluggable containers are the PDBs
 - Logically separate databases
 - A local dictionary, and access to common objects

The root container is the point from which the whole CDB can be administered. It has a complete data dictionary that understands the whole environment and stores definitions of the Oracle supplied shared objects. These objects are visible and usable in all pluggable containers, but exist only in the root. There is rarely going to be any reason for anyone other than the DBA (probably the senior DBA) to connect to the root container.

The pluggable containers, or PDBs, are for users. They may well have their own DBAs and developers who can for the most part work in perfect isolation from other containers.

The seed container

6

- All CDBs include a seed container, pdb\$seed
- Created at CDB creation time
- A read only PDB
- Not exposed as a service
- Four tablespaces: SYSTEM, SYSAUX, TEMP, (UNDO)
- Used as a source of new PDBs

The seed container will always exist. It is created at database creation time, and there is nothing one can (or would want) to do to it. The seed has only template files, including a data dictionary tablespace, that can be copied to create new PDBs. It may or may not include an undo tablespace.

Container Identifiers

7

- There are various unique identifiers
 - CON_ID 1 (root), 2 (seed), >=3 (pluggable containers)
 - DBID Generated on creation, stored in file headers
 - CON_UID Equal to DBID
 - GUID Globally unique ID, used by OMF
 - NAME Name of the container
- Conversion functions are available



The various identifiers are exposed in different views, with some inconsistencies in column naming:

v\$pdb\$
v\$containers
cdb_pdb\$
cdb_pdb_history

These functions convert one identifier to another:

CON_NAME_TO_ID('container_name')
CON_DBID_TO_ID(container_dbid)
CON_UID_TO_ID(container_uid)
CON_GUID_TO_ID(container_guid)

Session architecture

8

- Only the root container is visible to IPC
- PDBs require listener spawned sessions
- A session can move between containers
- A PDB is a service and a set of tablespaces
- The service restricts the scope of the session
 - Only the container's tablespaces are visible
 - Only local and common objects can be seen

An IPC connection to a database relies on setting an environment variable to the name of the local instance. The SQL*Net client can then connect directly to the SGA (which is, in effect, a named block of shared memory) and launch a server process. In a Multitenant environment, there is only one instance, therefore only one SGA. Using IPC will work, and connects you to the root container. Setting your ORACLE_SID to the name of a pluggable container cannot work. For regular users, the only way to connect to a PDB is through SQL*Net by requesting the service.

Once connected to the database, depending on the schema to which you connected and the permissions granted, it may be possible to move the session between containers. To do this, one must connect to a "common" user. Common users are defined in the root container, and propagated (though possibly with different privileges) to all pluggable containers.

Within a pluggable container, no matter how the session got there or who it is, it is absolutely impossible to see objects residing in another container – with the exception of explicitly shared objects defined in the root. If for application purposes it is necessary to see objects in another container this must be done through database links, exactly as though the containers were separate databases.

Moving between containers

9

- Connect to a service using a connect string
 - `connect scott/tiger@cdb1:1521/pdb1`
- To move between containers:
 - `alter session set container= ... ;`
- USERENV context variables:
 - `cdb_name`
 - `con_name`
 - `con_id`
- SQL*Plus commands:
 - `show con_name`
 - `show con_id`
- Query the database `global_name` if in doubt



Connection to a PDB through SQL*Net is the same as connecting to a non-CDB. The connection can be to a privileged account, such as SYS AS SYSDBA, but the authentication must be through the password file. Once connected, common users (more of which later...) granted the SET CONTAINER privilege can move between containers, if they have been granted CREATE SESSION in each container.

Even though developers should have no requirement for knowing that they are running in a Multitenant environment, their code may need to determine where it is running. This is a case where a minor change in coding may be required. It is not sufficient to determine the database name or the instance name; these are the same for all containers. Instead they must determine the container name, which is also used as the database `global_name` reported by the container.

Simulating IPC connections

10

- Some applications may rely on IPC
 - Jobs run through shell scripts
 - Use of the SYSDBA privilege
- Some older clients cannot use services
 - (connect_data = (sid = ...))
- Within the listener.ora

```
USE_SID_AS_SERVICE_listener = on
```
- Environment variables to append a tns alias

```
export TWO_TASK = ...
set LOCAL = ...
```

- Connect to a service using a connect string

In some circumstances, it may be awkward or impossible always to have to go through a TNS connect string when logging on to the database. Older Oracle clients may also not be capable (or configurable) of using service names in their connect strings. There are workarounds for these situations.

The current release includes a trigger in the root container, SYS.DBMS_SET_PDB, which may be helpful:

```
CREATE OR REPLACE NONEDITIONABLE TRIGGER "SYS"."DBMS_SET_PDB" after logon on database
WHEN (user = 'SYS' or user = 'SYSTEM') declare
  pdb_name varchar2(64);
begin
  DBMS_SYSTEM.get_env ('ORACLE_PDB_SID', pdb_name);
  if(pdb_name is not null)
  then
    EXECUTE IMMEDIATE 'alter session set container = ' || ''' || pdb_name
      || ''';
  end if;
exception
  when others then
    NULL;
end dbms_set_pdb;
```

CDBs and distributed systems

11

- Consolidation may be of distributed databases
- No change to DB link syntax
- Claimed to be faster than a normal database link
- Two phase commit does apply
- Changing container does not commit
- Replication may be an issue
 - Advanced Replication is desupported
 - Streams is deprecated
 - GoldenGate is separately licensed
 - Data Guard is Multitenant aware



In many cases, databases consolidated into a CDB will store related data. There is no change whatsoever to the way that database links function. Create them as usual, and when using one a second session against the instance will be launched by the listener, and this second session will connect to the service that maps to the appropriate PDB.

A possible advantage is that the listener can be configured with IPC listening addresses for each PDB. If this is done and the links use an appropriate connect string, then the communications do not need to go through the TCP stack which is sometimes claimed to improve performance.

Distributed systems that rely on replication may have problems. Advanced Replication will work between PDBs, but Advanced Replication is desupported from release 12.2. Streams (and therefore CDC) will not function in this environment. The separately licensed options X-Streams and GoldenGate are Multitenant aware.

Distributed systems that rely on Data Guard for replication require special consideration. It is possible to replicate only a subset of PDBs to a given standby, though the exact functionality is version dependent.

Data dictionary views

12

- Data dictionary views now a four level hierarchy
 - USER/ALL/DBA views show local and common objects
 - CDB views in the root show the whole environment
 - CDB views in a PDB are equivalent to DBA views
- CDB views include a CON_ID column
 - Identifies the container from which the row comes
 - Otherwise, columns are identical to DBA views
- Supplied common users see across containers
 - CDB views are filtered for non-Oracle maintained users
 - Other common users need a setting:

```
alter user c##x set container_data =  
all | default container = current ;
```



The CDB views, when queried in the root container, are in effect a UNION ALL of the DBA views for each open container. Closed containers are of necessity excluded because there is no access to their data dictionaries.

The CON_ID column ensures uniqueness. Join the views to CDB_PDBS to retrieve the container's name and other details.

The seed container, even though open read only, is by default excluded from most CDB views. This behaviour is controlled by an instance parameter,

EXCLUDE_SEED_CDB_VIEW = TRUE | FALSE

This parameter has become a hidden _underscore parameter from release 12.2.

Dynamic performance views

13

- All views now qualified with a CON_ID column
 - 0 - the row pertains to the entire CDB
 - 1 - the row pertains to the root container, cdb\$root
 - 2 - the row pertains to the seed container, pdb\$seed
 - >=3, the row pertains to a pluggable container
- V\$ views may have different contents
 - If the view is queried in the root or a PDB
 - If the view is populated from memory or the controlfile

Join the views to V\$CONTAINERS or V\$PDBS to identify the source of the row by container name. Similar information is in CDB_PDBS, with more information regarding how a container was created in CDB_PDB_HISTORY.

Converting to Multitenant

14

- 12.x and later databases can be plugged into a CDB
 - Redo log, and controlfile discarded
 - Data dictionary converted for the CDB environment
 - Earlier databases must be upgraded to 12.x first
- Use Data Pump to transfer objects to a new PDB
- PDBs cannot be converted to non-CDB database
 - The conversion to multitenant is a one-way process
 - PDBs can be de-consolidated to single occupancy

The conversion process from non-CDB to PDB is irreversible. During the conversion certain physical structures are discarded, and the data dictionary is adjusted to remove all the Oracle supplied objects. These are in effect replaced by access to the shared objects maintained in the root container.

It is certainly possible to unplug and remove a PDB from a CDB that hosts multiple PDBs and plug it into another CDB as a sole tenant. If a PDB is running an application that requires the full resources of an instance, this will be necessary. An example might be a DML intensive application where redo generation is the performance bottleneck.

Application containers

15

- An Application Container stores shared objects
 - Objects to be common to a subset of PDBs
 - Effectively, a virtual CDB within the CDB
- Install applications into the application root
 - Common users
 - User defined shared objects
- Create PDBs from the application root
 - User PDBs
 - Optionally a seed PDB: a copy of the root as it is
- Patch and upgrade apps in the application root



Shared objects are created by Oracle supplied routines in the root container. As a general rule, users cannot create shared objects. Application containers make this possible – though not in the root container. An application container acts as the root container for a set of pluggable containers: in effect, a virtual CDB residing within the CDB. Within the application container, objects can be created which will be shared by all its dependent pluggable containers.

The application shared objects can be updated only in the application container. The changes can then be pushed out to child containers through application sync operation.

To make effective use of application containers, it is necessary to be a large site. This facility is aimed at sites hosting many databases (or containers) running the same applications. A SaaS organization hosting a set of applications for many customers on the cloud would be a suitable environment.

Proxy containers

- Extends the Application Container concept
 - Provides access to a remote ("referenced") PDB
 - An application root can support distributed containers
- The proxy PDB contains no user data
 - A proxy is local, accessing data in the remote PDB
 - Copies of SYSTEM, SYSAUX, TEMP and UNDO only
 - User tablespaces exist in the remote referenced PDB
- All SQL executes in the referenced PDB
- Possible usage cases
 - Relocate a PDB without changing any connect strings
 - Share a root container between multiple CDBs
 - Extreme scalability

16

 SKILLBUILDERS

As with application containers, proxy containers are unlikely to be deployed other than at the largest sites.

The PDB must be created (or plugged in) in one CDB. Then in another, create the proxy. The creation (though no subsequent activity) requires a database link. Users connect to the proxy and are transparently routed to the PDB, in a manner analogous to using a connection manager.

The use of proxy containers gets around the scalability issues of Multitenant. A CDB has only one SGA, only one set of background processes. This can limit the scalability on the environment. Use of proxy containers distributes the work of running applications across multiple CDBs, possibly on multiple physical servers, while retaining the central management.

PDB snapshot carousel

- A snapshot is a point-in-time copy of a PDB
- Files copies may be complete or sparse
- The carousel manages a library of snapshots
- Enterprise Edition on Exadata only

17

A snapshot is a read consistent copy of a PDB, created while the source is open or closed. Creation can be manual on demand, or automated with a schedule. If the underlying storage can use sparse clones, then the snapshot will be sparse; otherwise, it will be a full copy.

A set of snapshots, with automatic creation and deletion to maintain the population, is a snapshot carousel.

The carousel automates managing a library of snapshots. For example, snapshots could be taken every day and retained for a week.

Architecture: summary

18

- One database, one instance: the CDB
- A PDB is a service plus a set of tablespaces
- Session scope is restricted to the current container
- The root container manages common objects
- The seed container is used for creating PDBs
- Each PDB has a dictionary for local objects
- Oracle Net manages connections to containers

Exercise: Build A CDB

- Create a new container database
- Investigate the physical and memory structures

19

SKILLBUILDERS



ORACLE DATABASE ADMINISTRATION

CLOUD ADMINISTRATION

ORACLE AND APEX HOSTING

APEX APPLICATION DESIGN AND DEVELOPMENT

TRAINING