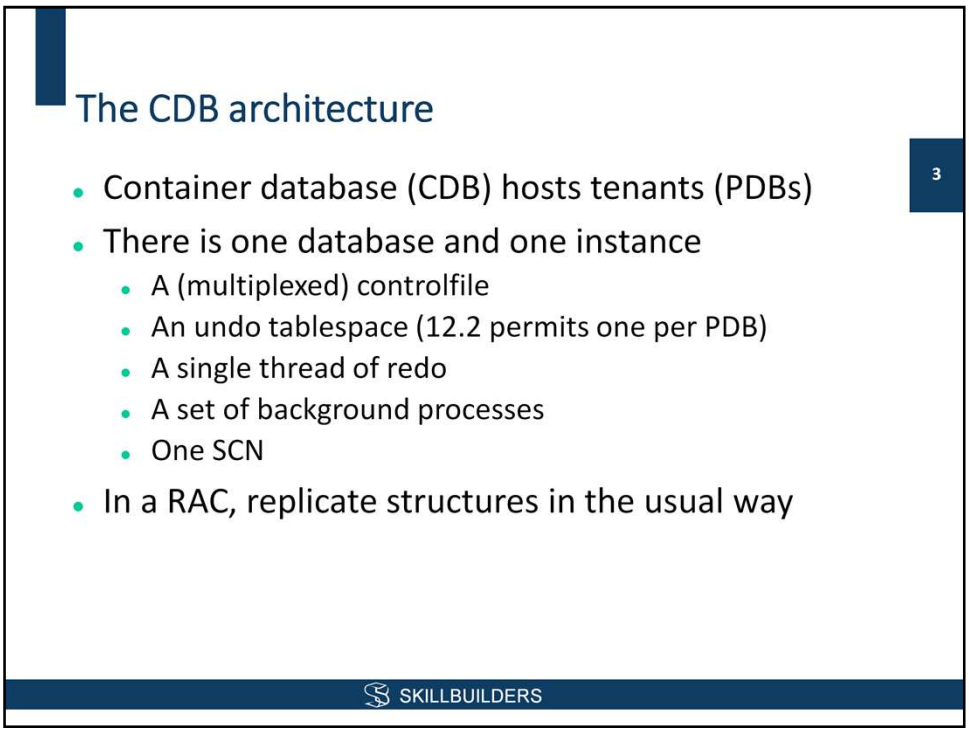
Lesson 2
Multitenant Architecture

SKILLBUILDERS

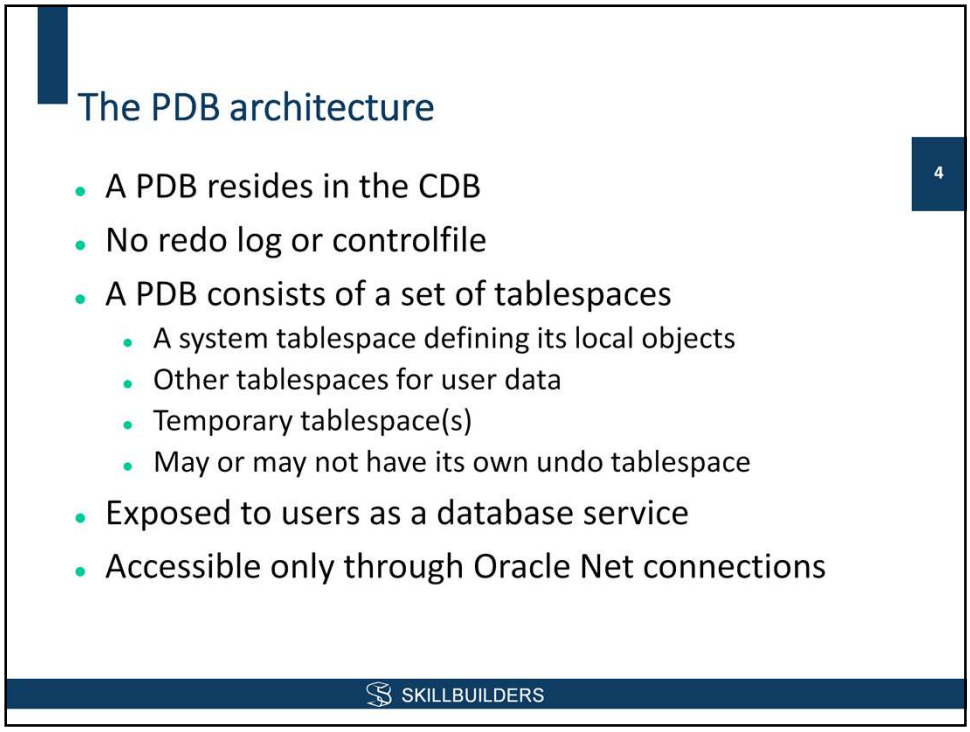
Objectives

- 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 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.

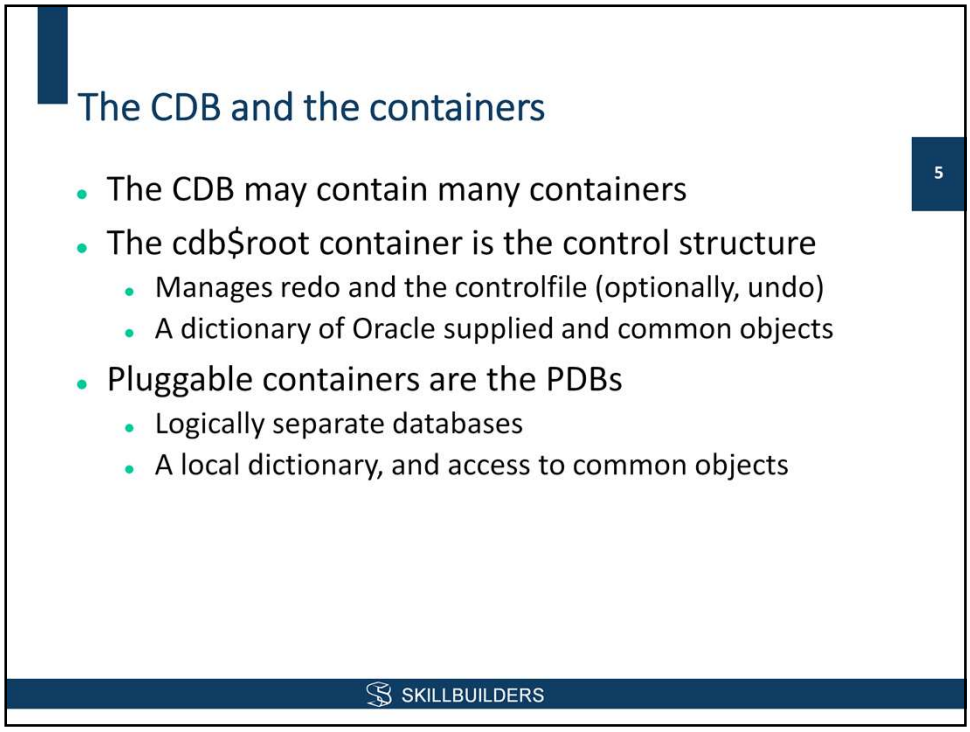
One shared undo tablespace may simplify the administration but it does remove some functionality. The decision on whether to use a shared undo tablespace or a local undo tablespace per pluggable container is not irrevocable.



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.

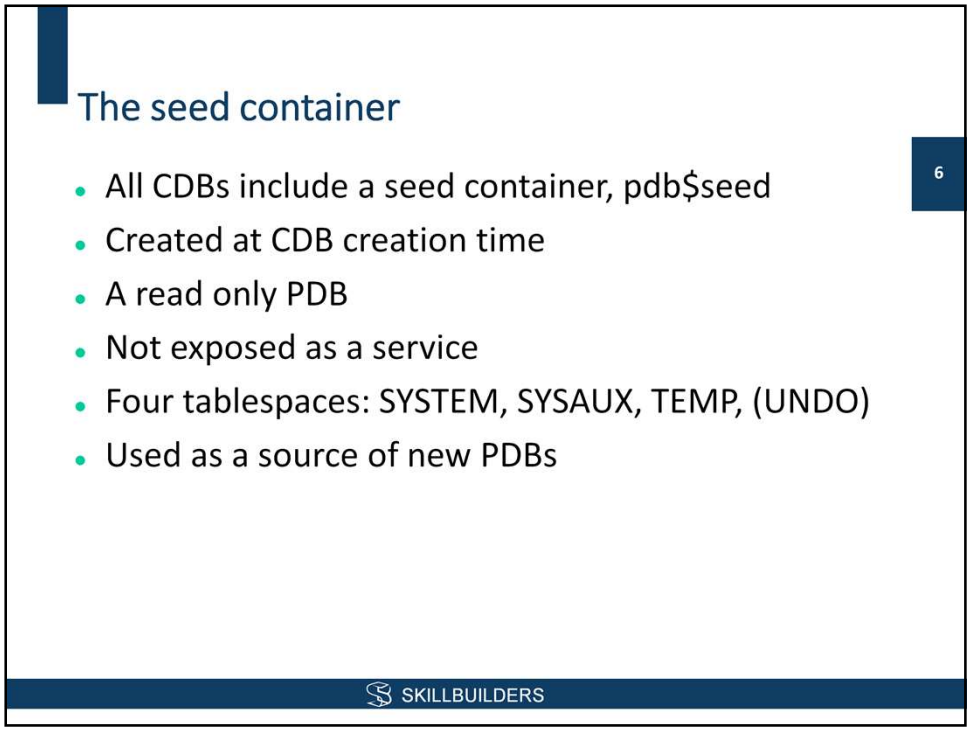
There are two options for undo data. A single shared undo tablespace managed from the root container was the only option in the first release. From 12.2 it is possible to have separate undo tablespaces per PDB. This does involve a bit more management work and space usage, but enables some extra facilities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

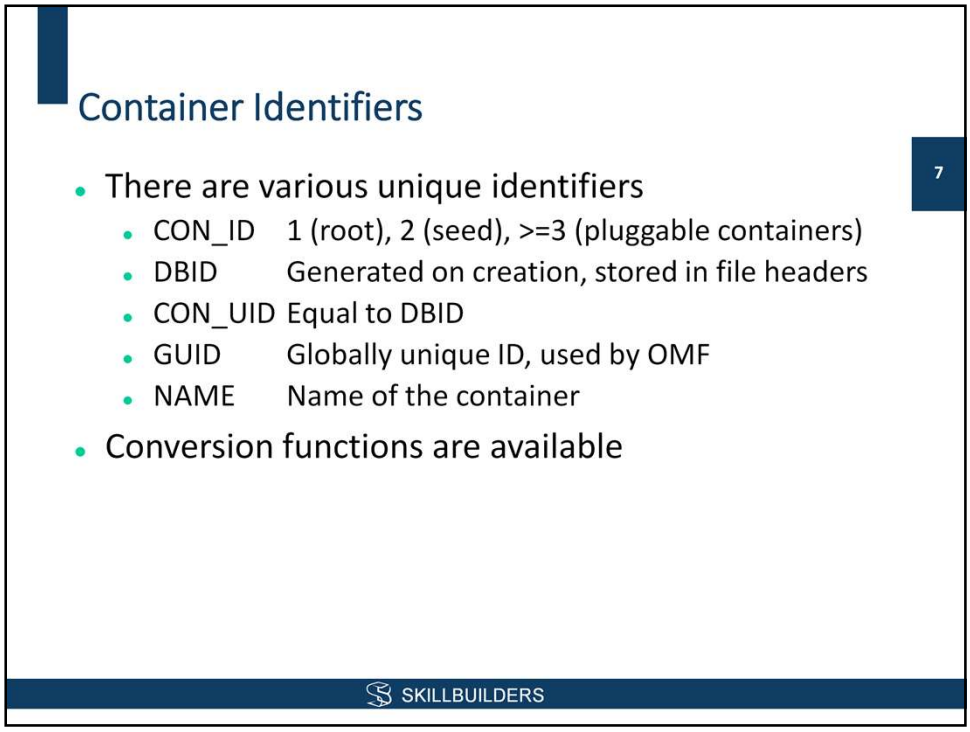


The root container is the point from which the whole CDB can be administered. It has a complete data dictionary that understands the whole environment and stores definitions of the Oracle supplied shared objects. These objects are visible and usable in all pluggable containers, but exist only in the 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 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.



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

v$pdbs

v$containers

cdb_pdbs

cdb_pdb_history

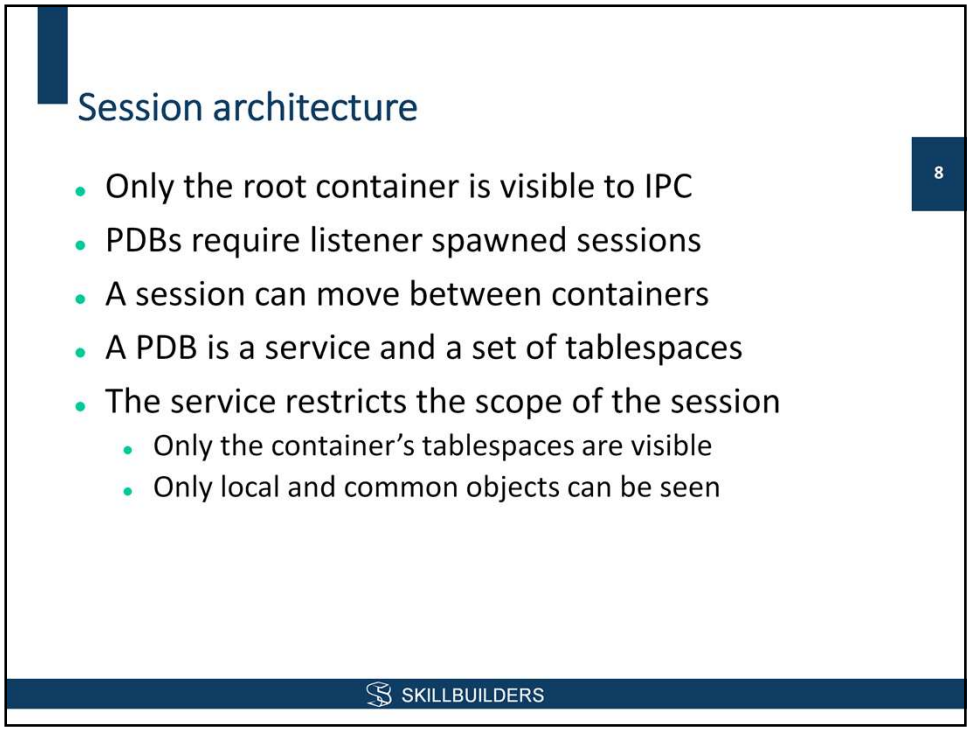
These functions convert one identifier to another:

CON_NAME_TO_ID('container_name')

CON_DBID_TO_ID(container_dbid)

CON_UID_TO_ID(container_uid)

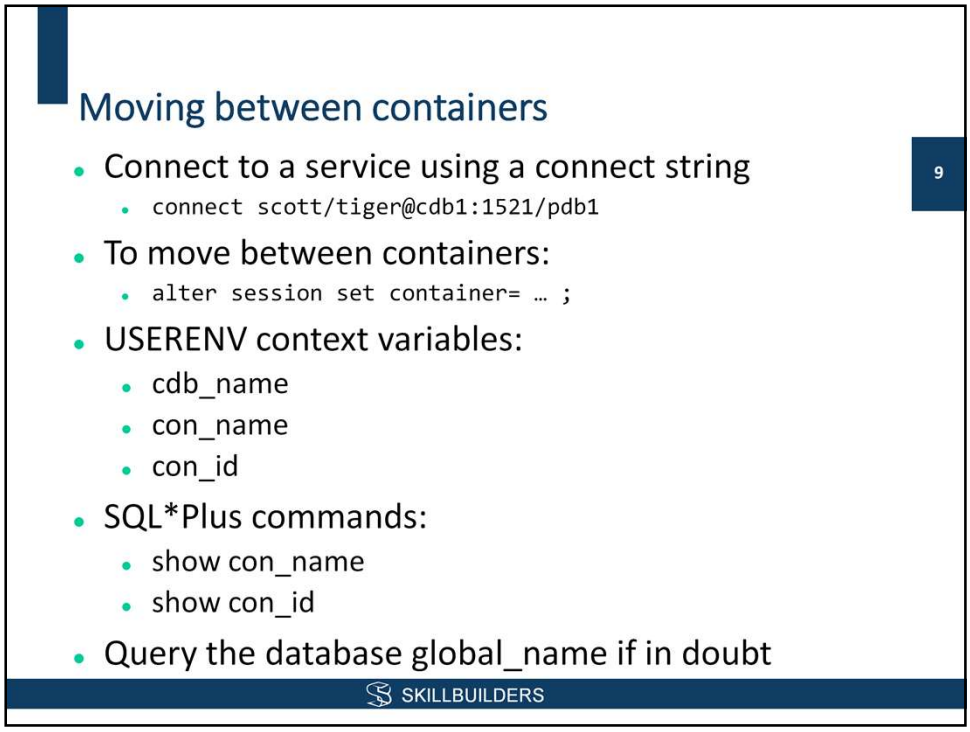
CON_GUID_TO_ID(container_guid)



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.

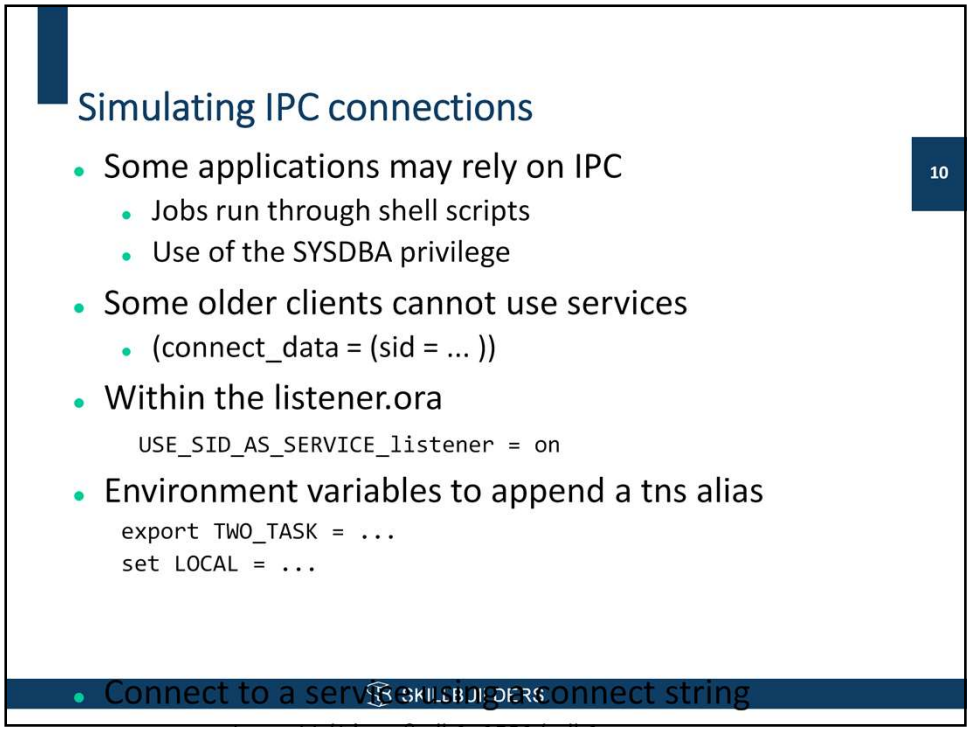
Once connected to the database, depending on the schema to which you connected and the permissions granted, it may be possible to move the session between containers. To do this, one must connect to a "common" user. Common users are defined in the root container, and propagated (though possibly with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



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.

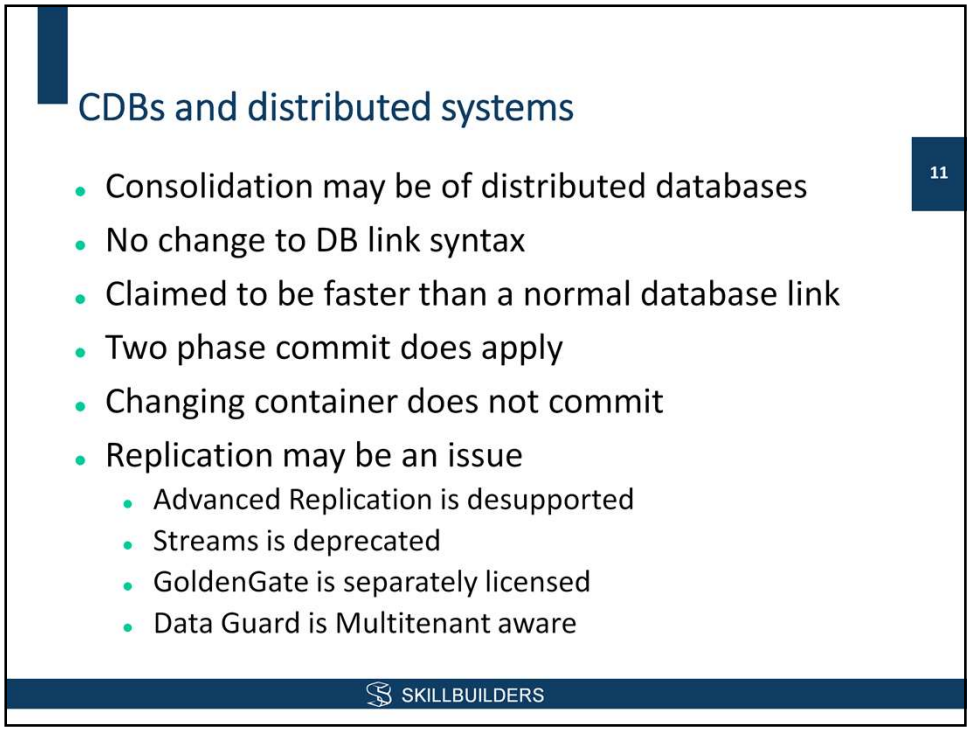
Even though developers should have no requirement for knowing that they are running in a Multitenant environment, their code may need to determine where it is running. This is a case where a minor change in coding may be required. It is not sufficient to determine the database name or the instance name; 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.



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;
```

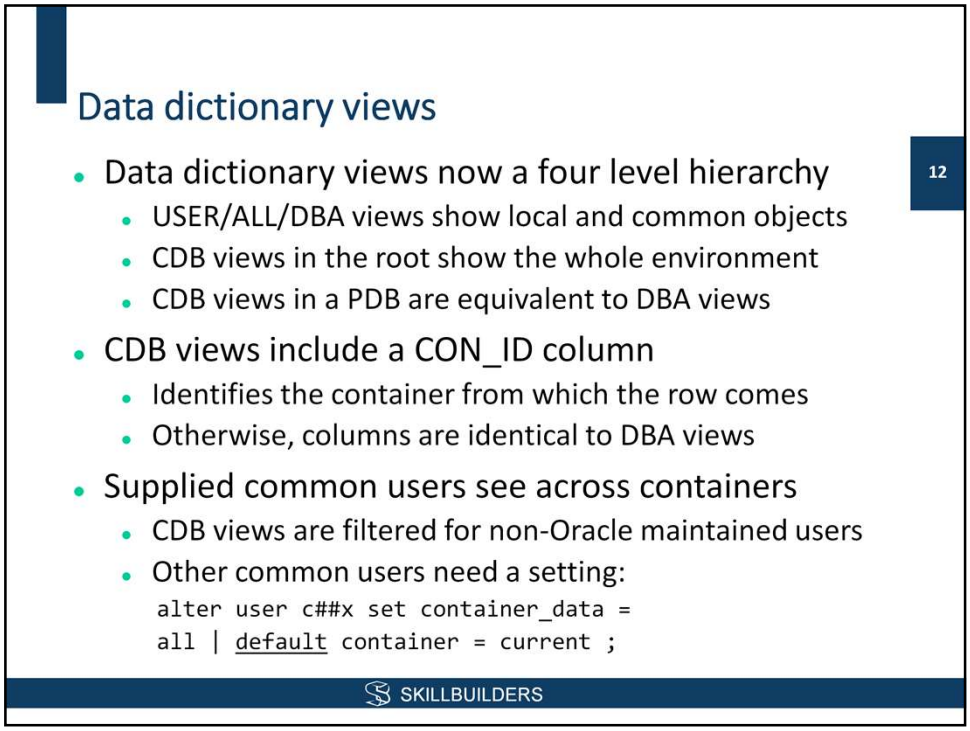


In many cases, databases consolidated into a CDB will store related data. There is no change whatsoever to the way that database links function. Create them as usual, and when using one a second session against the instance will be launched by the listener, and this second session will connect to the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



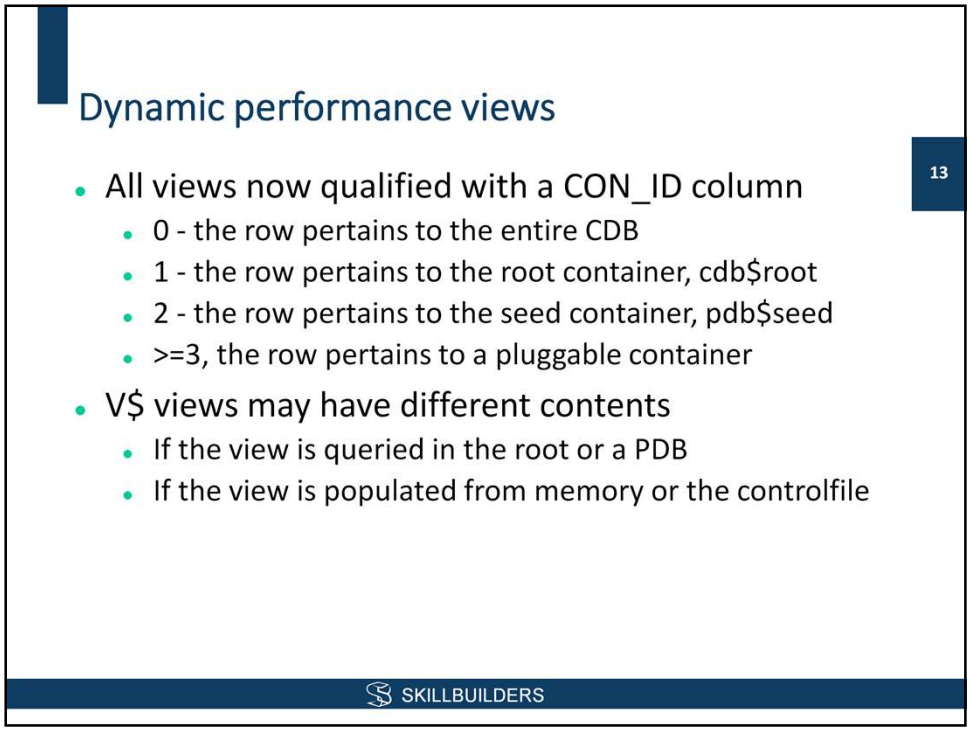
The CDB views, when queried in the root container, are in effect a UNION ALL of the DBA views for each open container. Closed containers are of necessity excluded because there is no access to their data dictionaries.

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

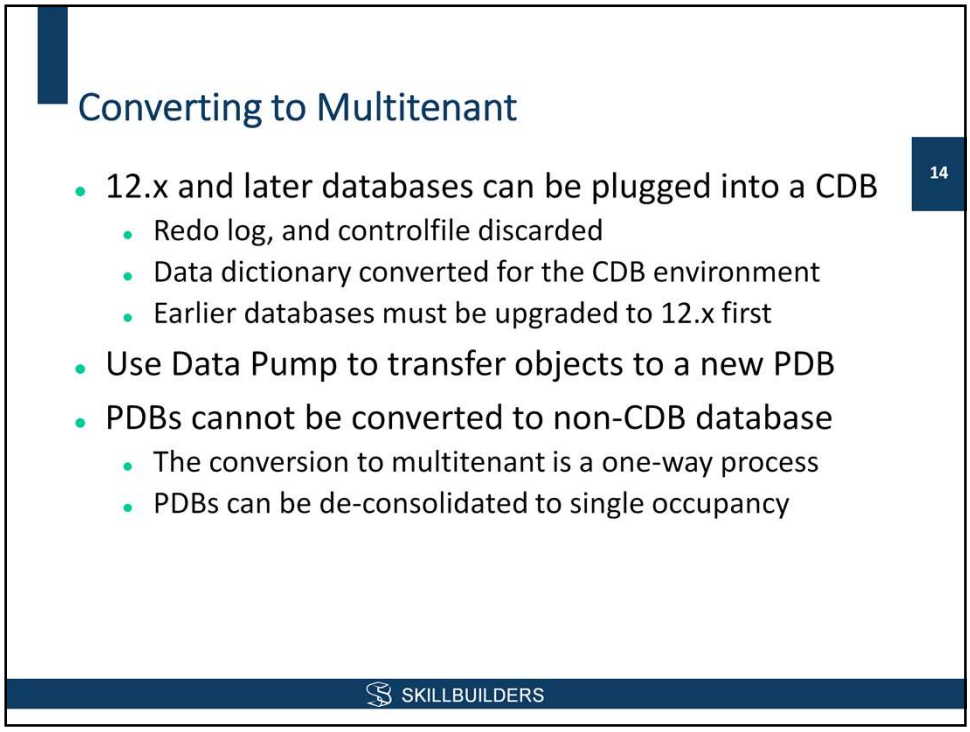
The seed container, even though open read only, is by default excluded from most CDB views. This behaviour is controlled by an instance parameter,

EXCLUDE_SEED_CDB_VIEW = TRUE | FALSE

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

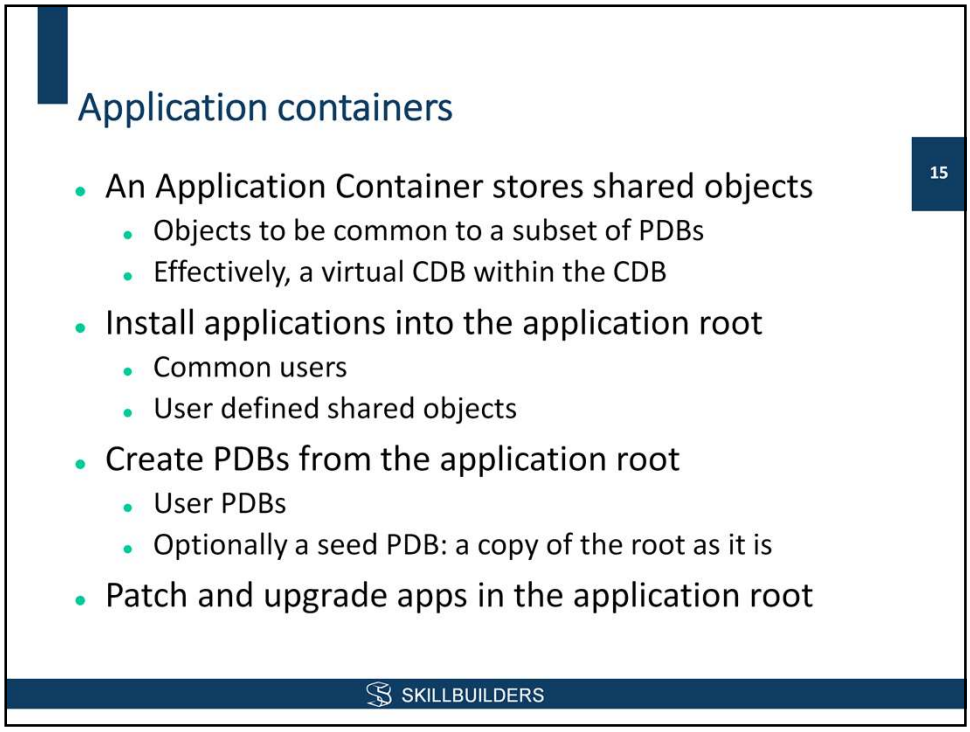


Join the views to V$CONTAINERS or V$PDBS to identify the source of the row by container name. Similar information is in CDB_PDBS, with more information regarding how a container was created in CDB_PDB_HISTORY.



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.

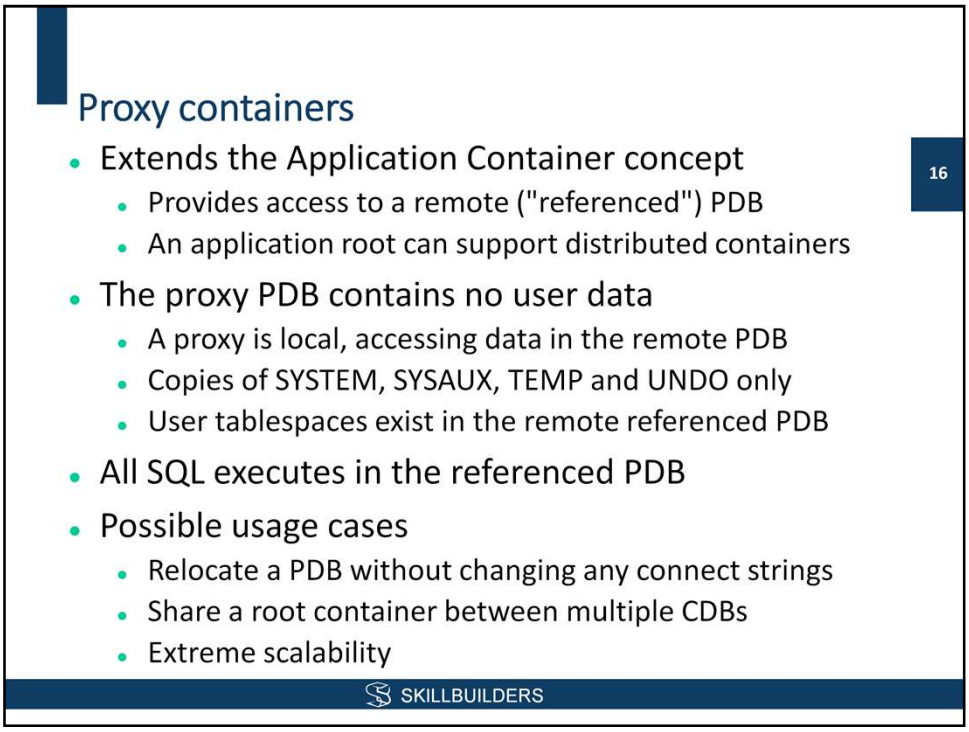
It is certainly possible to unplug and remove a PDB from a CDB that hosts multiple PDBs and plug it into another CDB as a sole tenant. If a PDB is running an application that requires the full resources of an instance, this will be necessary. An example might be a DML intensive application where redo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



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.

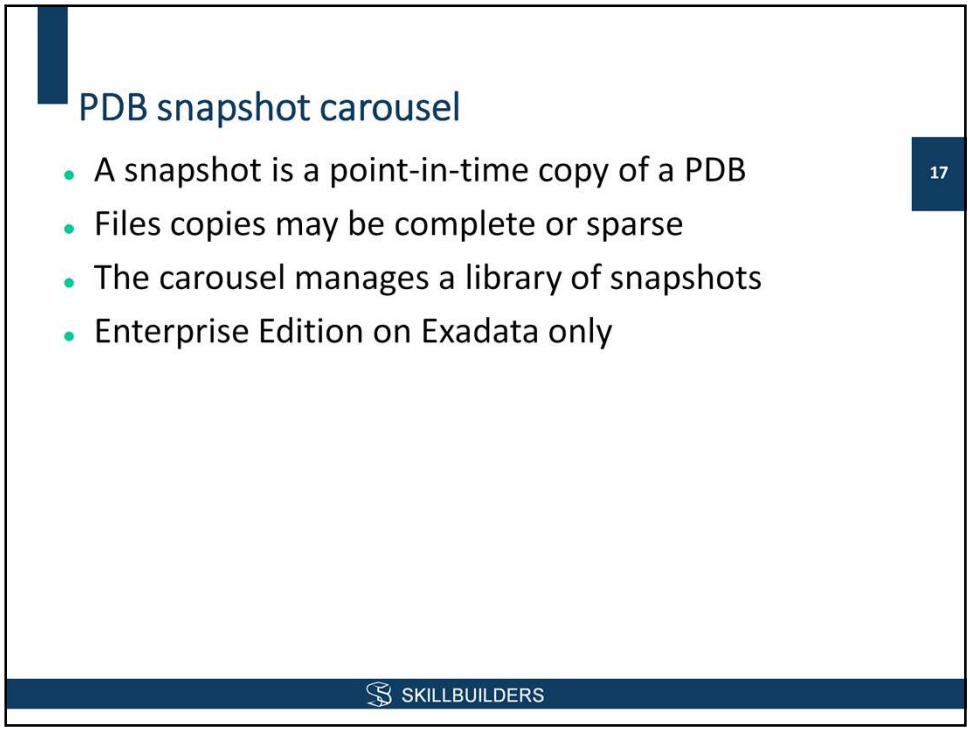
To make effective use of application containers, it is necessary to be a large site. This facility is aimed at sites hosting many databases (or containers) running the same applications. A SaaS organization hosting a set of applications for many customers on the cloud would be a suitable environment.



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.



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.

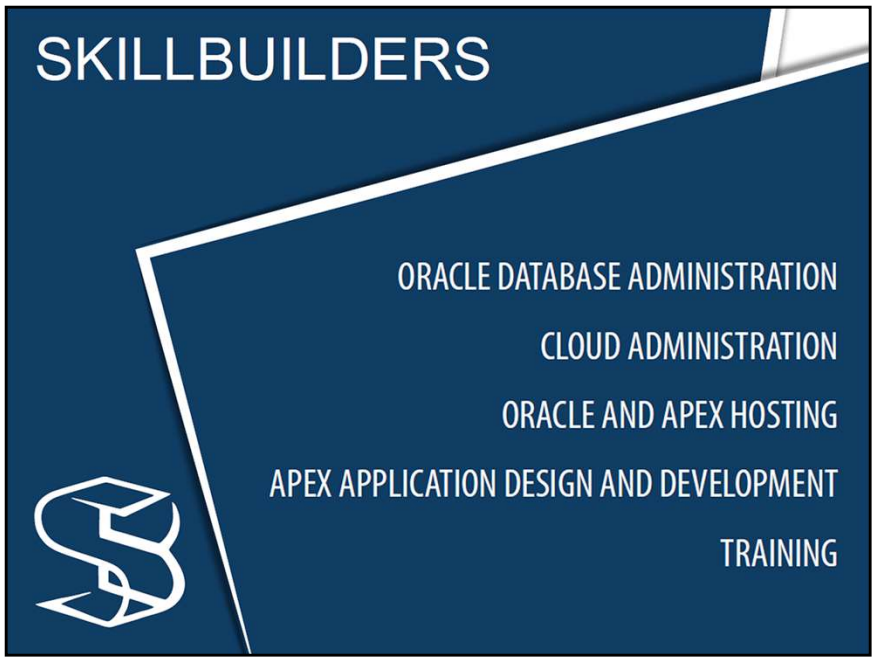
Architecture: summary

- 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