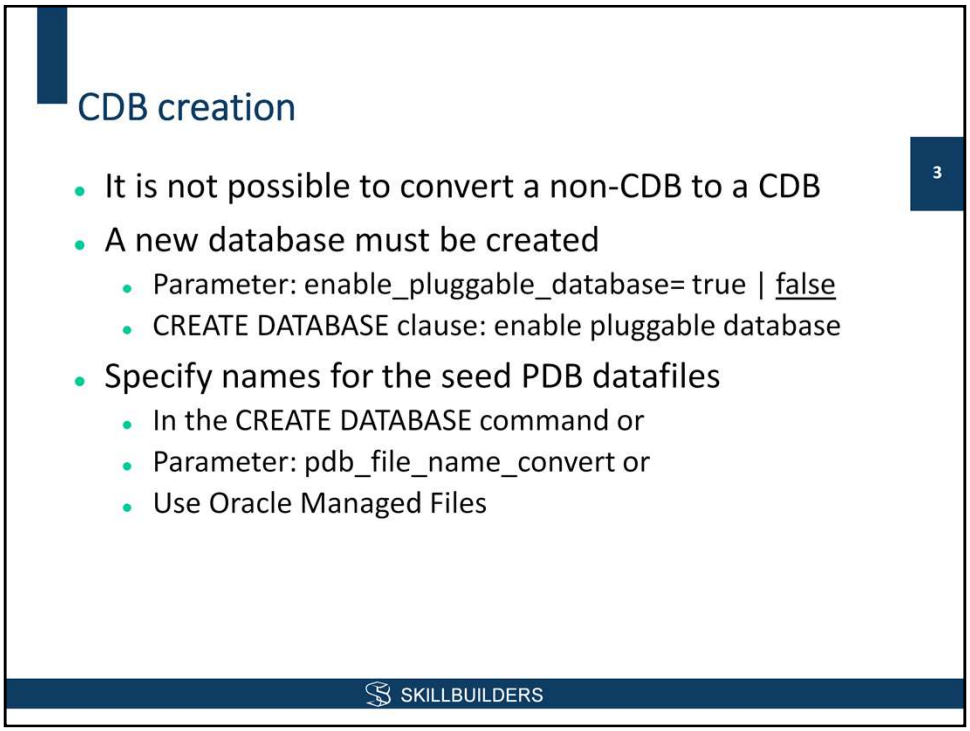
Lesson 3
Set Up A Multitenant Environment

SKILLBUILDERS

Objectives

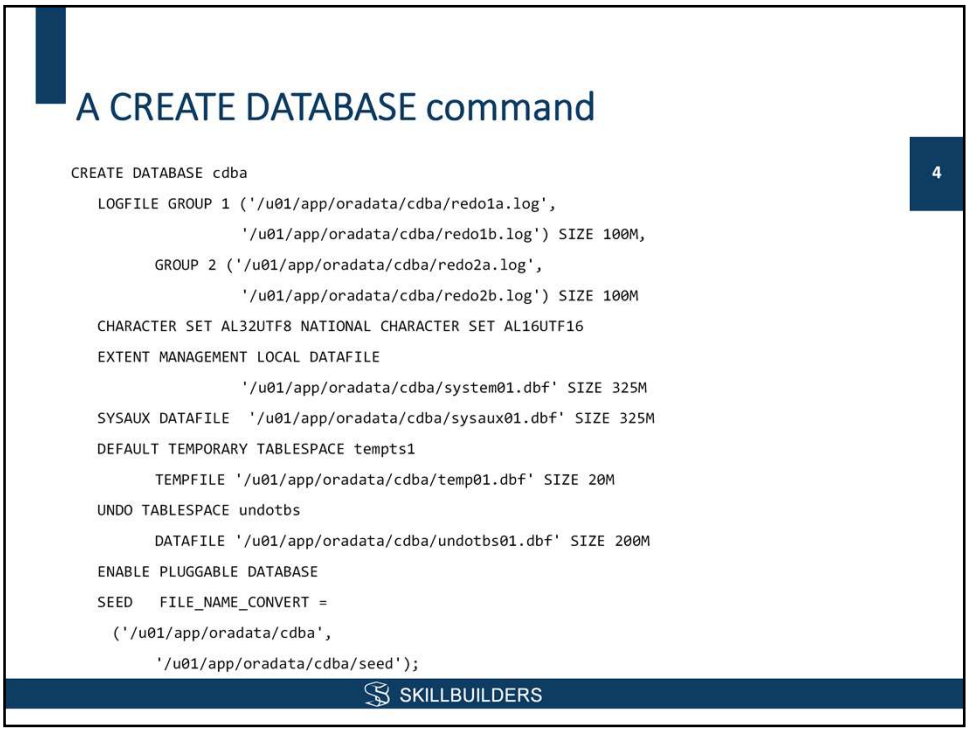
- Create a container database
- Create pluggable containers
- Convert a non-CDB to a pluggable container
- Relocate, unplug, and drop containers
- Hot clones of PDBs and non-CDBs
- Upgrade a CDB



The Multitenant option requires memory structures in the SGA that must be created when the instance is started and persistent structures in the data dictionary that are created when the database is created. The controlfile is also Multitenant aware (see the column V$DATABASE.CDB). It is not possible to convert a non-CDB into a CDB, not even by recreating the controlfile.

Attempting to mount a database with ENABLE_PLUGGABLE_DATABASE set inappropriately produces ORA-65101 or ORA-65093 errors.

A CDB always has a root container and a seed container. The CREATE DATABASE command has been enhanced to allow naming of file names for the seed container. There are various ways to generate the file names, but using Oracle Managed Files is undoubtedly the best way to do this. This is true for all databases, but in a Multitenant environment it is particularly helpful.

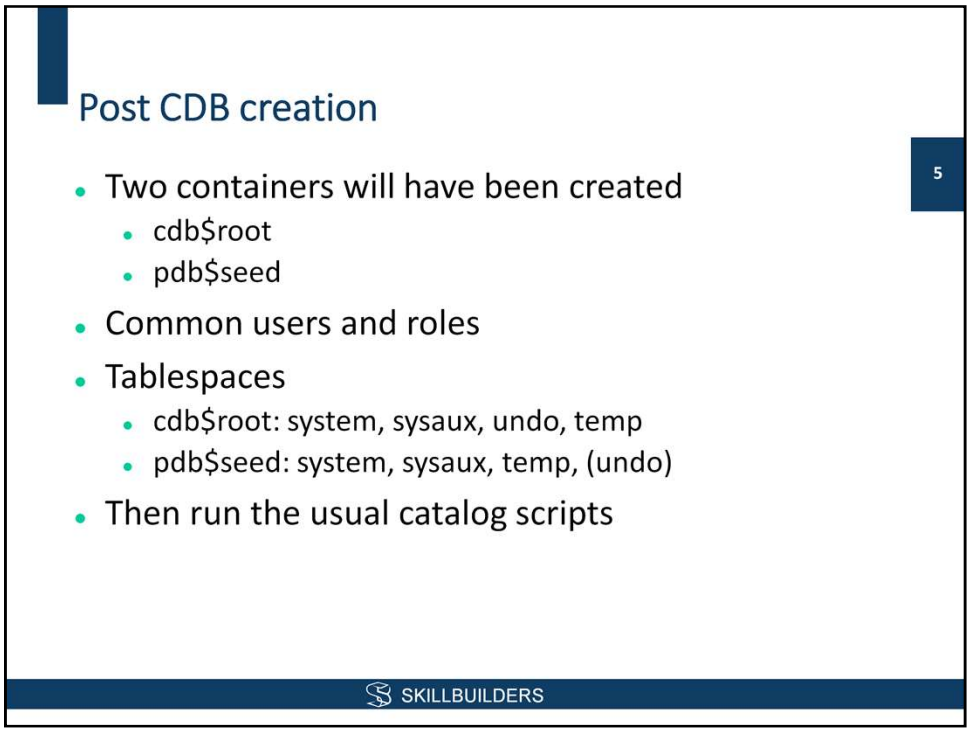


Before running the CREATE DATABASE command, the instance must have been started in nomount mode with ENABLE_PLUGGABLE_DATABASE=TRUE.

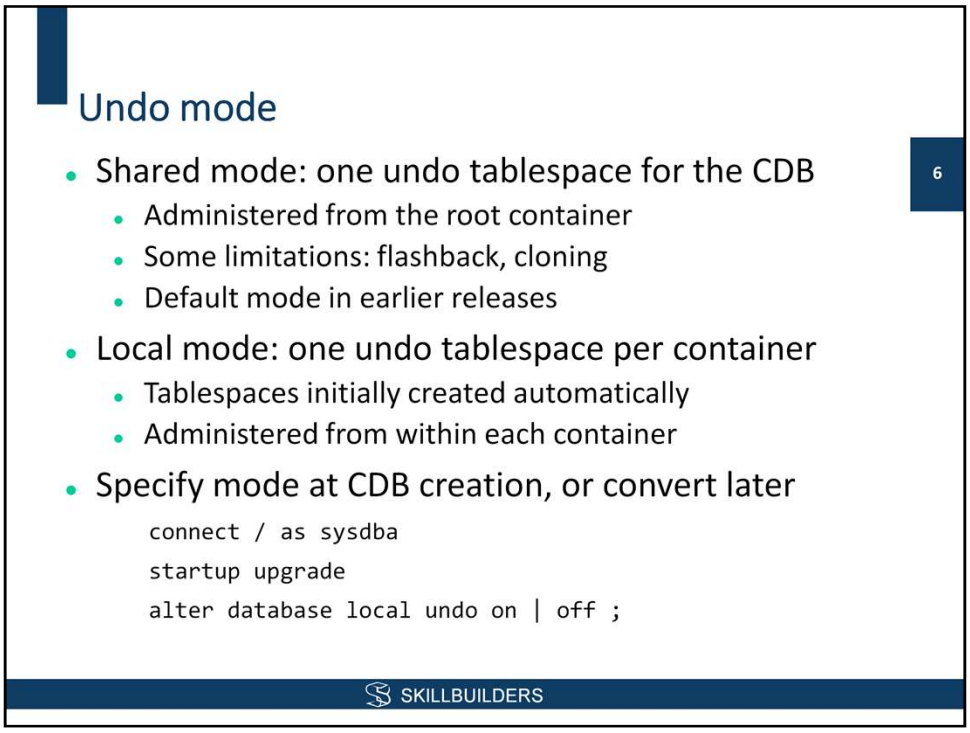
This command is one of many variations. The assumption is that OMF is not being used, hence the coding of full paths for online logfiles and the files for the root container's system, sysaux, temp, and undo tablespace datafiles. Names for the seed container's system, sysaux, and temp datafiles have not been explicitly specified. They will be generated by re-writing the root's filenames to insert another directory into the path. The same technique could be used to (for example) nominate an ASM disc group. It is also possible to specify several pairs of strings, or to provide fully qualified filenames.

If the OMF parameters had been set, the command could have been much simpler:

```
create database cdba enable pluggable database;
```



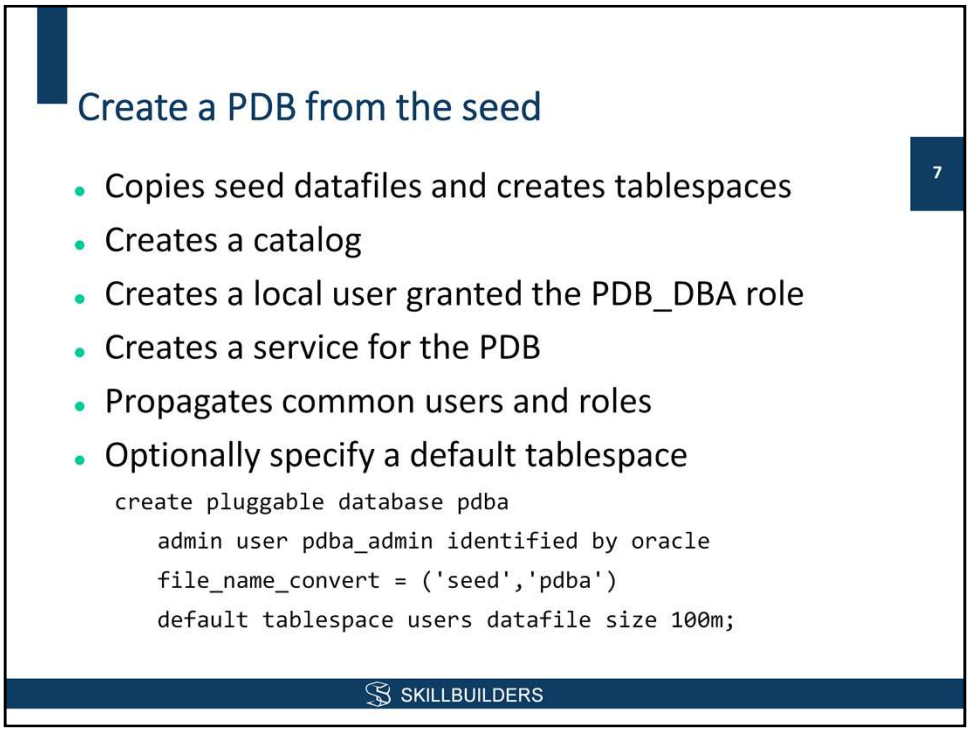
There are no particular actions needed after database creation. The Database Configuration Assistant can generate scripts that will automate the creation of one or more PDBs as post-creation actions. These scripts invoke the CREATE PLUGGABLE DATABASE command to create the PDBs from the seed container after creating the CDB.



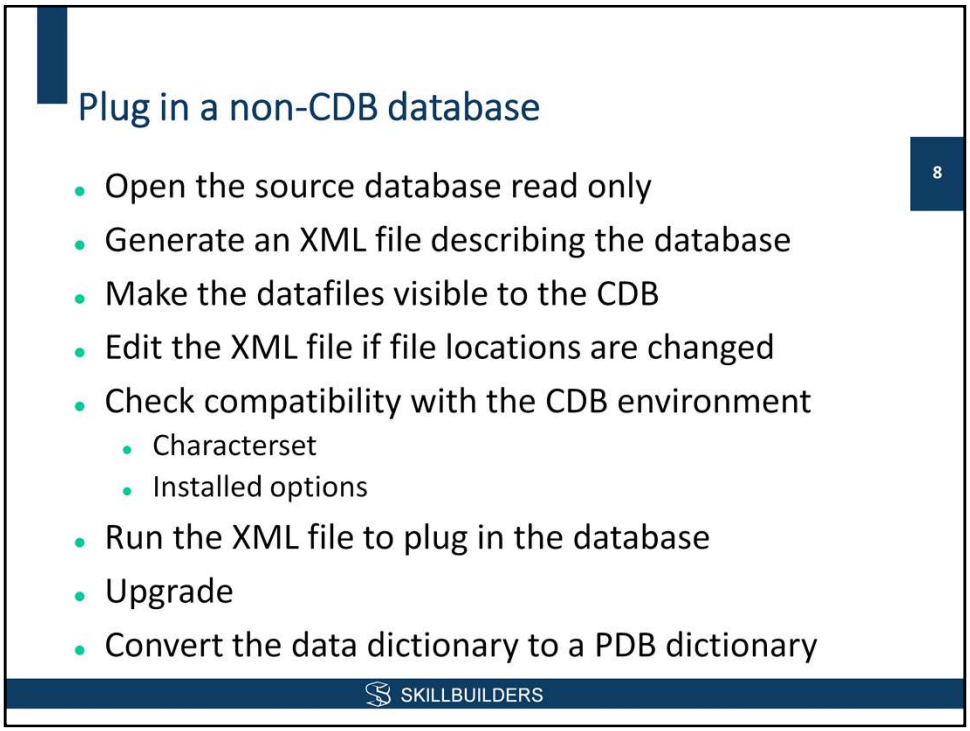
The initial release of Multitenant used a shared undo tablespace, administered from the root container. It is now possible to have one undo tablespace per container. While this does mean an increase in administration work, it also enables various capabilities that my out-weigh the small increase in DBA workload caused by the need to manage and monitor undo in each PDB.

Enable local undo with an ALTER DATABASE command, but in fact this is not a controlfile update but a data dictionary update. The setting is visible as a row in the DATABASE_PROPERTIES view. An undo tablespace will be created for each pluggable container when it is opened, which can be managed or replaced subsequently just as in a non-CDB.

In a single tenant environment there may appear to be little benefit to using local undo: It is just one more tablespace to manage. However, some operations (such as hot cloning) do become possible. In a more complex environment the advantages are significant. For example, it becomes possible to perform a database flashback operation on one PDB rather than the entire CDB.



This is the simplest and quickest way to create a PDB. There are various techniques for generating file names for the new container. The example above uses a conversion to replace one string in the source file name with another string to use for the destination filename. The same technique could be used to (for example) nominate an ASM disc group. It is also possible to specify several pairs of strings, or to provide fully qualified filenames. OMF would simplify the file naming process.

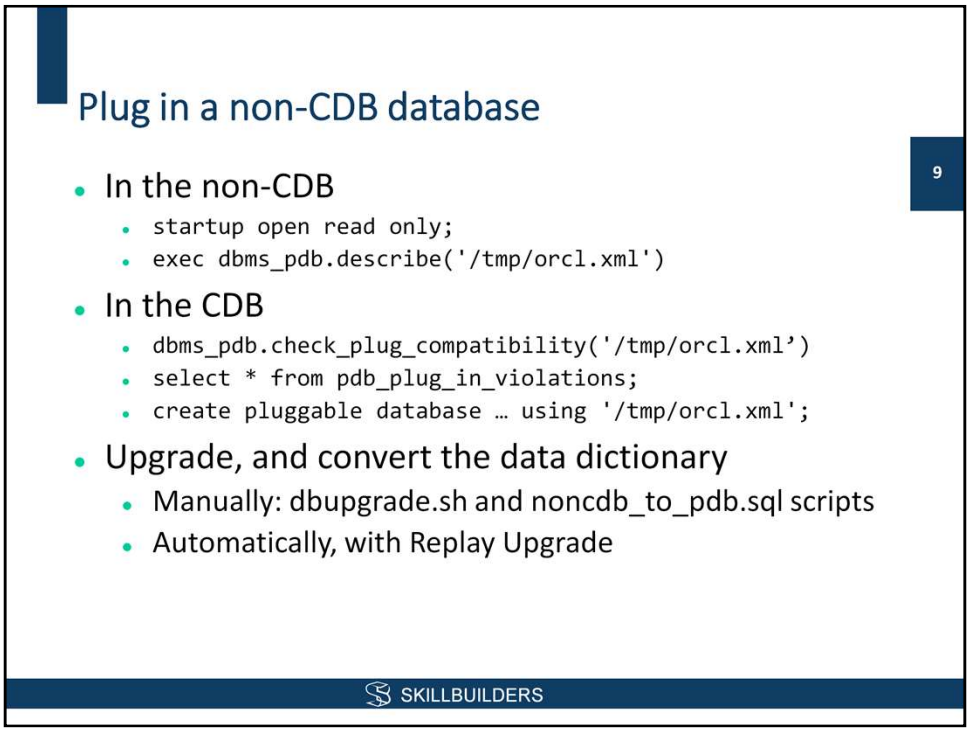


As a non-CDB can only be release 19 or earlier, plugging in to 23c will require upgrading. This can automatic or managed manually.

The source database is unavailable to users throughout the whole process, as the source must be in a transactionally consistent state.

In the initial release, the CDB's database characterset must be the same as that of the potential PDB. From release 12.2 this condition has been relaxed: as a general rule, if the CDB uses Unicode, then anything can be plugged in. Unicode AL32UTF8 is, from 12.2, the default database characterset and there will rarely be a reason not to use it.

The conversion of the data dictionary is irreversible. This is the process of adjustments necessary to run in this environment. It will usually be necessary to recompile all PL/SQL afterwards, as all the Oracle supplied objects on which PL/SQL may depend will have been recreated as linked objects from the root container.



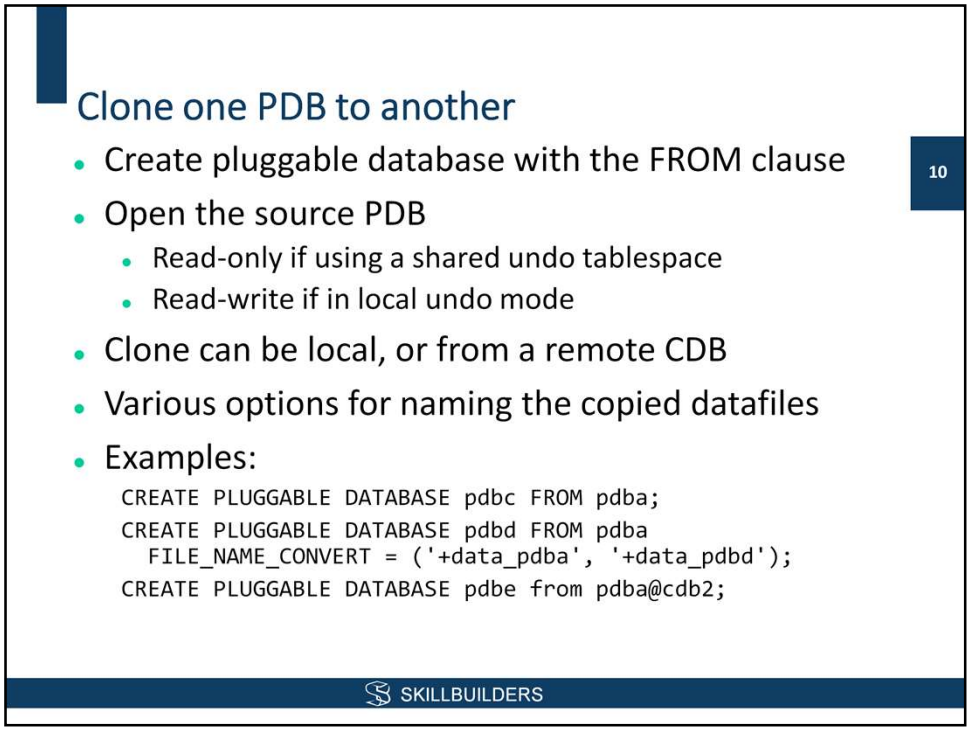
The dbms_pdb package is available in all 12.x and later databases, created when the database is created or upgraded. An 11.x database must be upgraded to at least 12.x before plugging in, and then the DBMS_PDB.DESCRIBE procedure can be run.

The CHECK_PLUG_COMPATIBILITY function will populate a view with information regarding any issues with plugging in. An example of running it:

```
set serveroutput on
declare ret boolean;
begin
ret:=dbms_pdb.CHECK_PLUG_COMPATIBILITY('c:\tmp\orclz.describe.xml');
if ret then dbms_output.put_line('ok');
else dbms_output.put_line('bad');
end if;
end;
/
select * from pdb_plug_in_violations;
```

The default on plugging in is COPY. This does mean that the source datafiles are untouched. Specifying NOCOPY will be faster and requires no disc space.

Other procedures in the package are for creating common objects in an application root container.



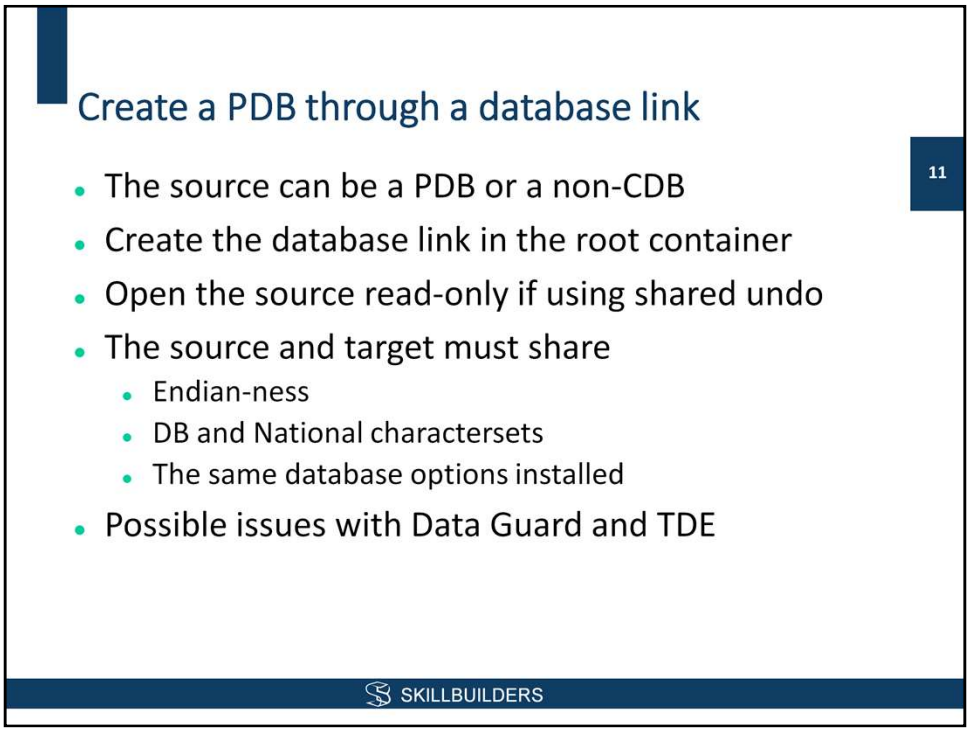
The process is little more than copying the datafiles and creating a new service. If using a database link, the network traffic may be considerable. If excessive, then using the unplug/copy/plugin route may be preferable.

An issue with cloning may be that the clone needs changes that must take immediate effect. For example, to ensure that jobs do not start to run create a trigger such as this in the source PDB before cloning:

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

Other actions could be setting passwords or re-mapping external references, such as directories or database links.

The trigger fires the first time the new PDB is opened, and then drops itself from the PDB.



The hot cloning capability requires that the source be in archivelog mode.

Create the link in the destination CDB root container. If the source is a CDB, it can connect to a common user in the root container of the remote database or in the pluggable container, but it must have the CREATE PLUGGABLE DATABASE privilege in the source PDB in all containers. The privilege should be granted directly, not through a role. Then:

```
create database link cdb2
  connect to system identified by oracle using 'cdb2';
```

If the source is a PDB, then for example:

```
create pluggable database pdbe FROM pdba@cdb2;
```

If the source is a non-CDB (which of necessity will be from release 12.x to release 19.x) then use the database name:

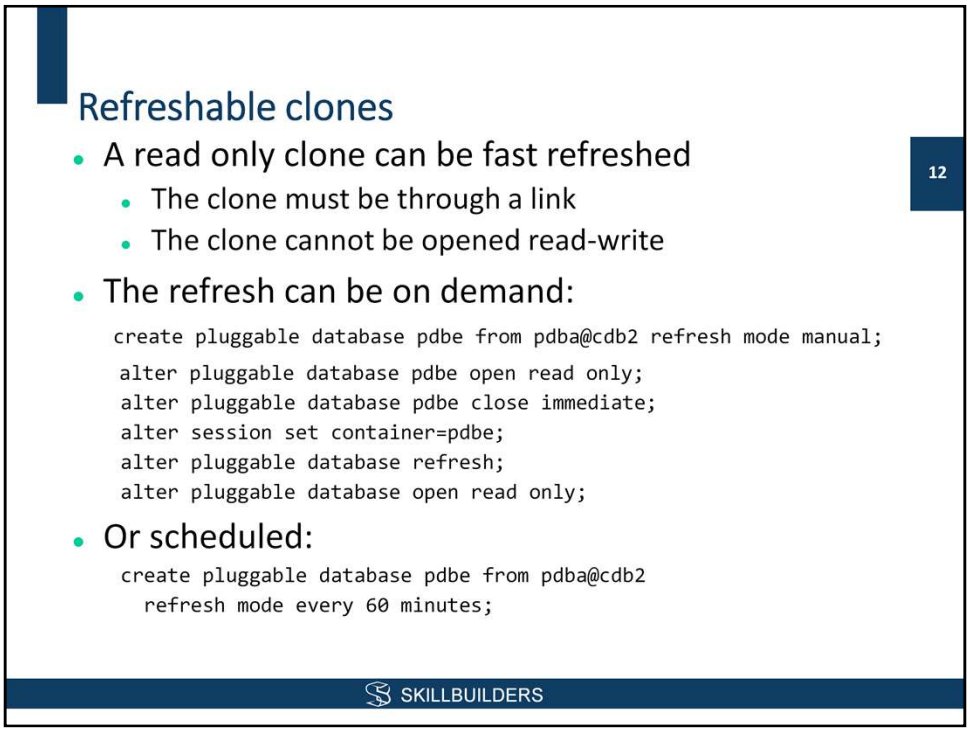
```
create pluggable database pdbf from orcl@orcl;
```

Following plugging in, the new PDB must be upgraded and the data dictionary converted.

The usual options for generating file names are available. The examples above assume that OMF has been configured.

It is possible to parallelize the process:

```
create pluggable database pdbe FROM pdba@cdb2 parallel 4;
```



The clone must be closed during a refresh. For scheduled refreshes, if the clone is open when the job runs, the job will fail and be retried at the next scheduled refresh time.

The refresh is accomplished by use of redo. Any archive logs necessary must therefore be available.

To open read write:

```
alter pluggable database … refresh mode none;
alter pluggable database … open read write;
```

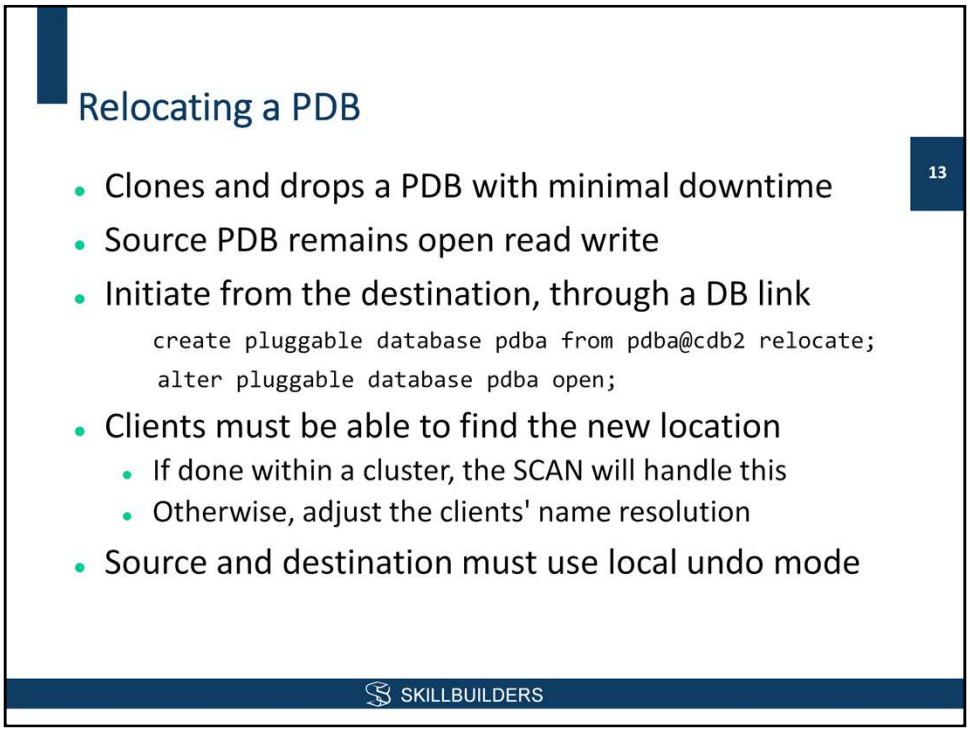
Exadata adds a switchover capability: the source and destination PDBs can reverse roles.

Usage cases of refreshable clones include:

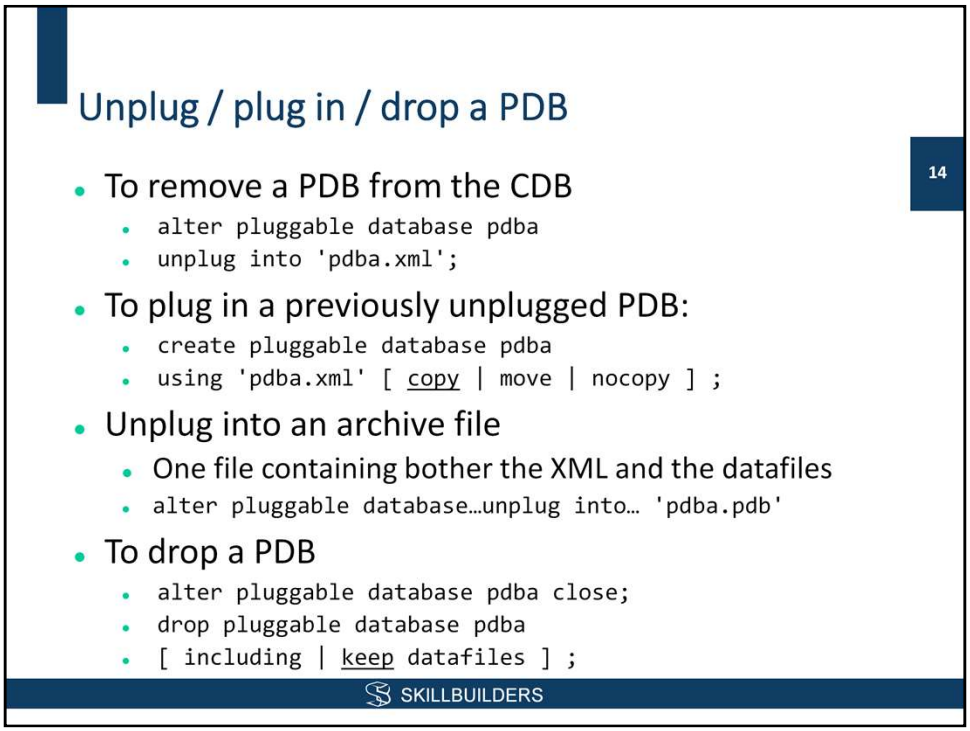
Near zero data loss. If a refresh is scheduled frequently (perhaps every few minutes) then fault tolerance similar to Data Guard will be achieved. This will work in Standard Edition as well as Enterprise.

Near zero downtime for platform migrations. Clone to a CDB on different hardware, frequent refresh, and a final refresh before opening read write. This will minimize downtime for a switchover.

A staging area for creating a set of clones. One read only PDB used as a source for several clones to be opened read write for various purposes.



During the copy of the datafiles through the database link, DML continues on the source PDB. When the clone is opened, redo from the source (in the form of a foreign archive logfile) is used to recover it and the source is dropped.

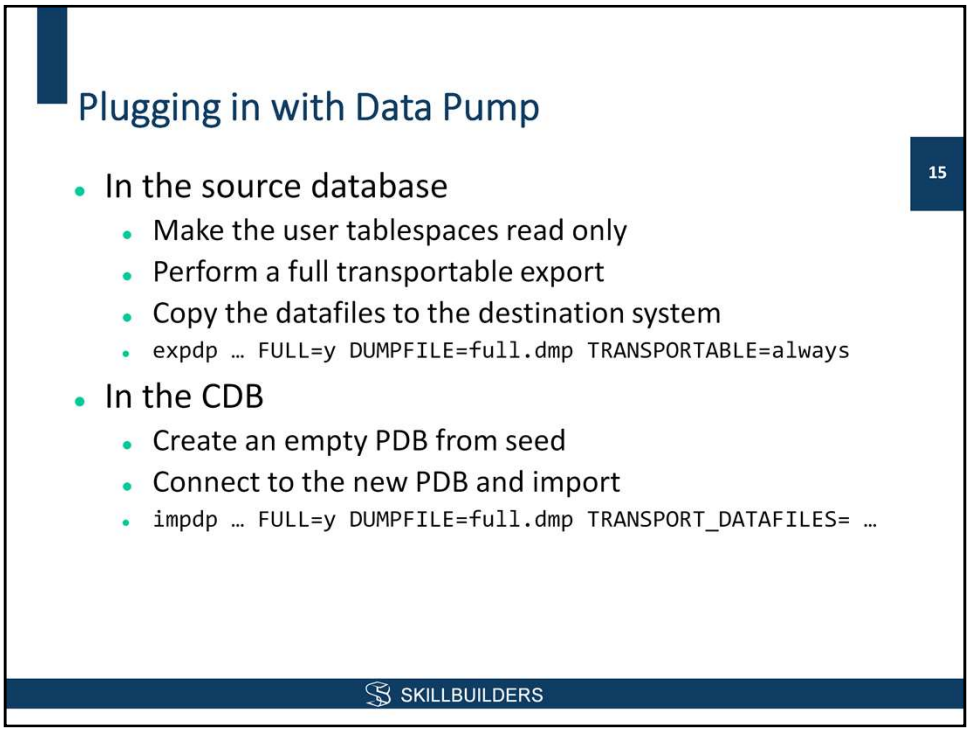


The UNPLUG INTO command generates an XML file describing the PDB, identical to that created by the dbms_pdb.describe procedure. A variation is to unplug into a zipped archive file, suffixed .pdb. The one file generated is a complete copy of the PDB that can be transferred to another CDB and plugged in.

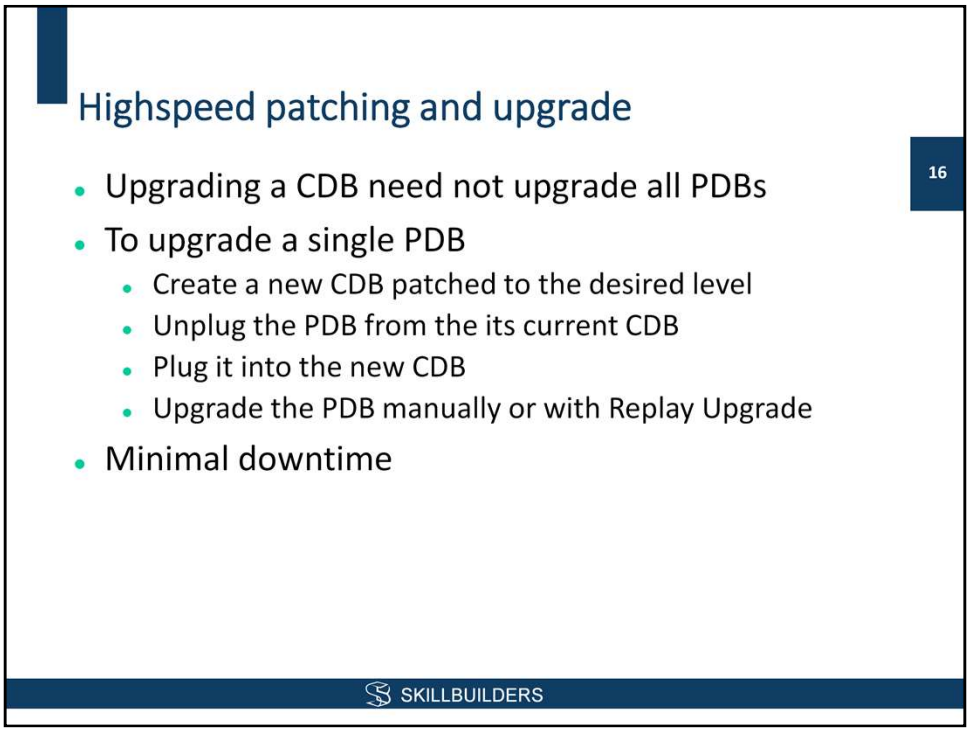
When plugging in a database, by default the files referenced in the XML file will be used. If you want to perform multiple plugins from the same set of files, specify COPY and new copies will created named by whatever rules are in place (OMF is your friend).

When dropping a PDB, the datafiles will by default survive. It is possible to recover the PDB by generating an XML describe file with the DBMS_PDB.RECOVER procedure.

A pdb archive (creation of which is determined by the file name suffix) is all necessary files: the xml description and the datafiles, zipped up. When plugging in, do not specify the copy clause. To rename files, use a file_name_convert clause.



Using the Data Pump method (which is basically transportable tablespaces) is the only way to plug in a database that is not release 12 or higher. The usual rules for tablespace transport apply.

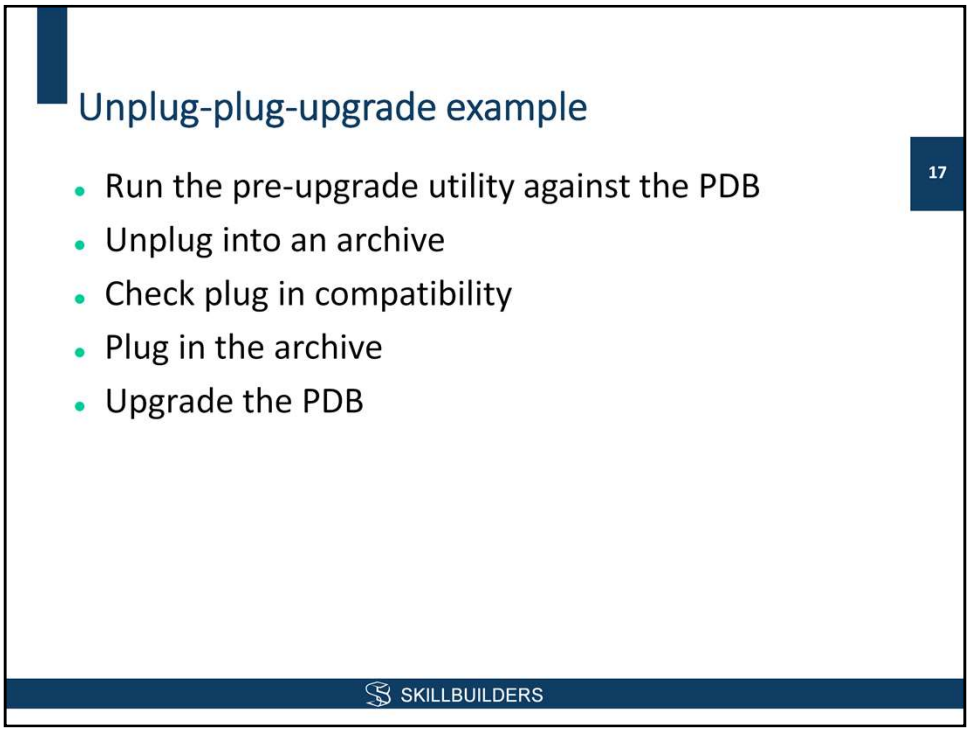


A database upgrade is two stages: upgrade the instance by updating the binaries in the Oracle Home, then upgrade the database by running scripts against the data dictionary. There should be nothing necessary to upgrade user objects, other than (perhaps) recompilation.

Once the

As a CDB has only one copy of the Oracle supplied objects, in the root container, all PDBs are automatically at the same release as the CDB itself. In practice, it will often be necessary to run scripts against the PDBs. The documentation for the patch or patchset will make this clear.

Applying an RU is similar. Upgrade the binaries, then use the datapatch utility to run any SQL scripts against all the containers.



To upgrade a 12.c PDB, pdb3, to 19c:

1. Run the pre-upgrade information tool from the new Oracle home:

java -jar $OH19/rdbms/admin/preupgrade.jar -c 'pdb3' TERMINAL TEXT:

Follow any instructions, such as purging the recyclebin.

2. Copy the PDB out into an archive:

alter pluggable database PDB3 close;

alter pluggable database PDB3 unplug into '/home/oracle/pdb3.pdb';

drop pluggable database PDB3 including datafiles;

3. In the destination CDB, check the compatibility:

dbms_pdb.check_plug_compatibility(-

pdb_descr_file => /home/oracle/pdb3.pdb', pdb_name => 'PDB3')

And address any issues.

4. Plugin and open the PDB:

create pluggable database pdb3 using '/home/oracle/pdb3.pdb'
file_name_convert=('/home/oracle', '/u02/oradata/CDB2/pdb3');

alter pluggable database PDB3 open upgrade;

5. Upgrade the PDB:

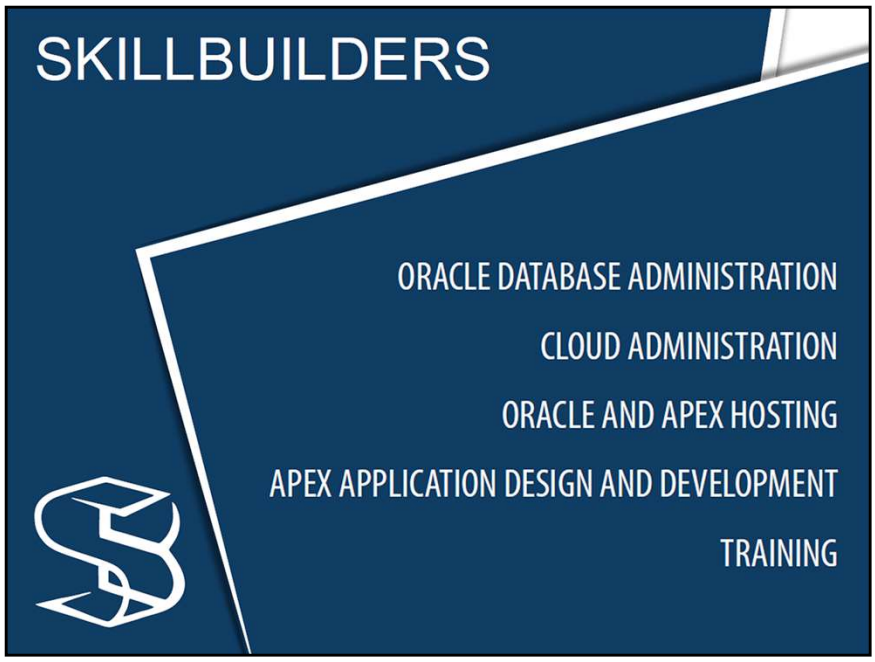
dbupgrade -c 'PDB3'

CDB and PDB set up: summary

- Multitenant must be enabled at creation time
  - You cannot convert non-CDB to CDB
  - Also, you cannot convert a PDB to a non-CDB
- PDBs can be created in various ways
  - From the seed container
  - Clone an existing container
  - Plugin an non-CDB
  - From non-CDB or PDB over a database link
- CDB and potential PDBs must be compatible
  - Character set
  - Installed components

Exercise: create PDBs

- Create pluggable containers in various ways
- Plug-in databases and unplug containers
- Configure SQL*Net for connections to PDBs


SKILLBUILDERS
ORACLE DATABASE ADMINISTRATION
CLOUD ADMINISTRATION
ORACLE AND APEX HOSTING
APEX APPLICATION DESIGN AND DEVELOPMENT
TRAINING