



Integrated Cloud Applications & Platform Services



Oracle Database: Managing Multitenant Architecture

Student Guide

D105924GC10 | D106059

Learn more from Oracle University at education.oracle.com



Oracle Internal & Oracle Academy Only

Author

Dominique Jeunot

**Technical Contributors
and Reviewers**

Jean-François Verrier

Sravanti Tatiraju

Veerabhadra Rao Putrevu

Graphic Editor

Kavya Bellur

Editors

Moushmi Mukherjee

Adrita Biswas

Publishers

Sujatha Nagendra

Srividya Rameshkumar

Michael Sebastian Almeida

Raghunath M

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Disclaimer

This document contains proprietary information and is protected by copyright and other intellectual property laws. You may copy and print this document solely for your own use in an Oracle training course. The document may not be modified or altered in any way. Except where your use constitutes "fair use" under copyright law, you may not use, share, download, upload, copy, print, display, perform, reproduce, publish, license, post, transmit, or distribute this document in whole or in part without the express authorization of Oracle.

The information contained in this document is subject to change without notice. If you find any problems in the document, please report them in writing to: Oracle University, 500 Oracle Parkway, Redwood Shores, California 94065 USA. This document is not warranted to be error-free.

Restricted Rights Notice

If this documentation is delivered to the United States Government or anyone using the documentation on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS

The U.S. Government's rights to use, modify, reproduce, release, perform, display, or disclose these training materials are restricted by the terms of the applicable Oracle license agreement and/or the applicable U.S. Government contract.

Trademark Notice

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

1002012019

Contents

1 CDB Basics

Oracle Database 18c Multitenant Architecture 1-2
Objectives 1-3
Challenges 1-4
Non-CDB Architecture 1-5
Multitenant Architecture: Benefits 1-6
Other Benefits of Multitenant Architecture 1-7
Oracle Multitenant Container Database 1-8
Configurations 1-9
Database Objects in a non-CDB 1-10
User-Added Objects to a non-CDB 1-11
SYSTEM Objects in the USER Container 1-12
Provisioning a Pluggable Database 1-13
Multitenant Container Database Architecture 1-14
Containers 1-15
Tools 1-16
Data Dictionary and Dynamic Views 1-17
Terminology 1-18
Impacts 1-19
Summary 1-21
Practices Environment - 1 1-22
Practices Environment - 2 1-23
Practice 1: Overview 1-24
Multitenant Architecture Poster 1-25

2 CDB and Regular PDBs

Objectives 2-2
Goals 2-3
Creating a CDB 2-4
Creating a CDB: Using SQL*Plus 2-5
New Clause: SEED FILE_NAME_CONVERT 2-6
New Clause: ENABLE PLUGGABLE DATABASE 2-7
After CDB Creation: What's New in CDB 2-8
Data Dictionary Views: DBA_xxx 2-9
Data Dictionary Views: CDB_xxx 2-10

Data Dictionary Views: Examples	2-11
Data Dictionary Views: V\$xxx Views	2-12
After CDB Creation: To do List	2-13
Automatic Diagnostic Repository	2-14
Automatic Diagnostic Repository: alert.log File	2-15
Provisioning New Pluggable Databases	2-16
Tools	2-17
Create New PDB from PDB\$SEED	2-18
Steps: With FILE_NAME_CONVERT	2-19
Steps: Without FILE_NAME_CONVERT	2-20
Summary	2-21
Practice 2: Overview	2-22

3 Application PDBs and Application Installation

Objectives	3-2
Regular PDBs	3-3
PDBs and Applications	3-4
Application Containers	3-5
Application Containers: Other Features	3-6
Types of Containers	3-7
Creating Application PDBs	3-8
Application Name and Version	3-9
Installing Applications	3-10
Patching and Upgrading Applications	3-11
Application Common Objects	3-12
Use Cases for Application Containers	3-13
Use Case: Pure PDB-Based Versus Hybrid Model	3-14
Container Map	3-15
Container Map: Example	3-16
Query Routed Appropriately	3-17
Dynamic Container Map	3-18
Container Map and Containers Default	3-19
Query Across CDBs Using Application Root Replica	3-20
Durable Location Transparency	3-21
Data Dictionary Views	3-22
Terminology in Application Container Context	3-23
Commonality in Application Containers	3-24
Impacts	3-25
Summary	3-26
Practice 3: Overview	3-27

4 PDB Creation

- Objectives 4-2
- Cloning Regular PDBs 4-3
- Cloning Application Containers 4-4
- Plugging a Non-CDB into CDB 4-5
- Plugging a Non-CDB into CDB Using DBMS_PDB 4-6
- Replicating Non-CDB into CDB 4-7
- Cloning a Non-CDB or Remote PDB 4-8
- Plugging an Unplugged Regular PDB into CDB 4-9
- Flow 4-10
- Plugging Using Archive File 4-12
- Unplugging and Plugging Application PDBs 4-13
- Converting Regular PDBs to Application PDBs 4-14
- Unplugging and Plugging a PDB with Encrypted Data 4-15
- Local UNDO Mode Versus Shared UNDO Mode 4-16
- Cloning Remote PDBs in Hot Mode 4-17
- Near-Zero Downtime PDB Relocation 4-18
- Proxy PDB: Query Across CDBs Proxying Root Replica 4-20
- Creating a Proxy PDB 4-21
- Dropping PDBs 4-22
- Summary 4-23
- Practice 4: Overview 4-24

5 CDB and PDB Management

- Objectives 5-2
- Connection 5-3
- Switching Connection 5-4
- Creating Services 5-5
- Renaming Services 5-6
- Starting Up a CDB Instance 5-7
- Mounting a CDB 5-8
- Opening a CDB 5-9
- Opening a PDB 5-10
- Automatic PDB Opening 5-11
- Closing a PDB 5-12
- Shutting Down a CDB Instance 5-13
- Changing PDB Mode 5-14
- Modifying PDB Settings 5-15
- Instance Parameter Change Impact 5-16
- Instance Parameter Change Impact: Example 5-17
- Using ALTER SYSTEM Statement on PDB 5-18

Configuring Host Name and Port Number per PDB 5-19
Summary 5-20
Practice 5: Overview 5-21

6 Storage

Objectives 6-2
Objects in Tablespaces 6-3
Tablespaces Created During PDB Creation 6-4
Defining Default PermanentTablespaces 6-5
Temporary Tablespaces 6-6
UNDO Tablespaces 6-7
Summary 6-8
Practice 6: Overview 6-9

7 Security

Objectives 7-2
Creating Common Users in the CDB and PDBs 7-3
Creating Common Roles in the CDB and PDBs 7-4
Granting Privileges Commonly in the CDB and PDBs 7-5
Creating Common Profiles in the CDB and PDBs 7-6
Common Objects in Application Containers 7-7
Operations on Data-Linked Objects 7-8
Enabling Common Users to Access Data in PDBs 7-9
Finding Information About CONTAINER_DATA Attributes 7-10
Restricting Operations with PDB Lockdown Profiles 7-11
Restricting Operations in a PDB Lockdown Profile 7-12
PDB Lockdown Profiles Inheritance 7-13
Static and Dynamic PDB Lockdown Profiles 7-14
Auditing Actions in the CDB and PDBs 7-15
Managing Other Types of Security Policies in Application Containers 7-17
Securing Data with Oracle Database Vault 7-18
Oracle Database Vault-Enabled Strict Mode 7-19
Managing Keystore in the CDB and PDBs 7-20
Creating and Opening a Keystore 7-21
Setting TDE Master Encryption Keys 7-22
Managing Keystore in the CDB and PDBs 7-23
Keystore Management Changes for PDBs 7-24
Defining the Keystore Type 7-25
Isolating a PDB Keystore 7-26
Converting a PDB to Run in Isolated Mode 7-27
Converting a PDB to Run in United Mode 7-28

Migrating a PDB Between Keystore Types	7-29
Unplugging and Plugging a PDB with Encrypted Data	7-30
Per-PDB Wallet for PDB Certificates	7-31
Summary	7-32
Practice 7: Overview	7-33

8 Backup and Duplicate

Objectives	8-2
Goals	8-3
Syntax and Clauses in RMAN	8-4
CDB Backup: Whole CDB Backup	8-5
CDB Backup: Partial CDB Backup	8-6
PDB Backup: Partial PDB Backup	8-7
Using RMAN Backup to Plug an Unplugged PDB	8-8
Duplicating Pluggable Databases	8-9
Cloning Active PDB into an Existing CDB	8-10
Example: 1	8-11
Example: 2	8-12
Duplicating On-Premise CDB as Cloud Encrypted CDB	8-13
Duplicating On-Premise Encrypted CDB as Cloud Encrypted CDB	8-14
Migrating Cloud Encrypted CDB as On-Premise CDB	8-15
Checking for Block Corruption	8-16
Summary	8-17
Practice 8: Overview	8-18

9 Recovery and Flashback

Objectives	9-2
Goals	9-3
Instance Failure and Instance Recovery	9-4
NOARCHIVELOG Mode	9-5
PDB Tempfile Recovery	9-6
PDB SYSTEM or UNDO Tablespace Recovery	9-7
PDB non-SYSTEM Tablespace Recovery	9-8
PITR	9-9
Migrating a Non-CDB to a CDB	9-10
Migrating a Non-CDB and Transporting Non-CDB Backups to a CDB	9-11
Relocating/Plugging a PDB into Another CDB	9-12
Plugging a PDB and Transporting PDB Backups to a CDB - 1	9-13
Plugging a PDB and Transporting PDB Backups to a CDB - 2	9-14
Using PrePlugin Backups	9-15
To Be Aware Of	9-16

Example 9-17
CDB and PDB Flashback 9-18
PDB Flashback and Clean Restore Point 9-19
PDB Snapshot Carousel 9-20
Creating PDB Snapshot 9-21
Creating PDBs Using PDB Snapshots 9-22
Dropping PDB Snapshots 9-23
Flashbacking PDBs Using PDB Snapshots 9-24
Switching Over a Refreshable Cloned PDB 9-25
Unplanned Switchover 9-26
Summary 9-27
Practice 9: Overview 9-28

10 Performance

Objectives 10-2
Tuning a CDB 10-3
Sizing the CDB 10-4
Testing the Estimates 10-5
Managing SGA for PDBs 10-6
Managing PGA for PDBs 10-7
Monitoring PDB Memory Usage 10-8
AWR and ADDM Behavior 10-9
PDB-Level Snapshot Views 10-10
AWR Report 10-11
ADDM Tasks: At the CDB Level Only 10-12
Basic Rules: Statistics for Common Objects 10-13
Controlling the Degree of Parallelism of Queries 10-14
Heat Map and ADO Support 10-15
Managing Heat Map and ADO Policies in PDB 10-16
CDB Fleet 10-17
CDB Lead and CDB Members 10-18
Use Cases 10-19
Consolidated Database Replay Use Cases 10-20
Use Cases: Source Workloads 10-21
The Big Picture 10-22
Step 1 10-23
Step 2 10-24
Step 3 10-25
Step 4 10-26
Summary 10-27
Practice 10: Overview 10-28

11 Resources Allocation

- Objectives 11-2
- Allocating Resources in the CDB 11-3
- Resource Manager and Pluggable Databases 11-4
- Managing Resources Between PDBs 11-5
- CDB Resource Plan Basics: Limits 11-6
- PDB IO Rate Limit 11-8
- CDB Resource Plan: Full Example 11-9
- Maintaining a CDB Resource Plan 11-10
- Managing Resources Within a PDB 11-11
- Putting It Together 11-12
- Considerations 11-13
- PDB-Level Parallel Statement Queuing 11-14
- PDB-Level Parallel Statement Queuing: CPU_COUNT 11-16
- Session PGA Limit 11-17
- Performance Profiles 11-18
- Summary 11-19
- Practice 11: Overview 11-20

12 Data Movement

- Objectives 12-2
- Using Oracle Data Pump with PDBs 12-3
- Exporting from non-CDB and Importing into PDB 12-4
- Exporting and Importing Between PDBs 12-5
- Exporting from PDB and Importing into non-CDB 12-6
- Full Transportable Export/Import: Overview 12-7
- Full Transportable Export/Import: Usage 12-8
- Full Transportable Export/Import: Example 12-10
- Transporting a Database Over the Network: Example 12-11
- Using SQL*Loader with PDBs 12-12
- Summary 12-13
- Practice 12: Overview 12-14

13 Upgrade Methods

- Objectives 13-2
- Upgrading CDB and PDBs to 12c: Methods 13-3
- Upgrading a CDB Including PDBs from 12c to 18c 13-4
- Upgrading CDB Including PDBs from 12c to 18c 13-5
- Upgrading a Single Regular PDB from 12c to 18c 13-6
- Converting and Upgrading Regular PDBs to Application PDBs 13-7
- Practice 13: Overview 13-8

Cross-Platform Transportable PDB	13-9
Cross-Platform PDB Transport: Phase 1	13-10
Cross-Platform PDB Transport: Phase 2	13-11
Summary	13-12
Practice 13: Overview	13-13

14 Miscellaneous

Objectives	14-2
Using Xstreams with a CDB and PDB	14-3
Creating a Standby of a CDB	14-4
Instantiating a PDB on a Standby	14-6
Scheduling Operations in a PDB	14-7
Jobs Coordinator and Resources	14-8
Mining Statements of a PDB Using LogMiner	14-9
Summary	14-10
Practice 14: Overview	14-11

A Processes, Views, Parameters, Packages, and Privileges

Instance and Database	A-2
Multitenant Architecture	A-3
CDB and PDB Views and Columns	A-4
CDB and PDB Parameters	A-10
CDB and PDB New Packages	A-11

B Consolidated Database Replay Procedures

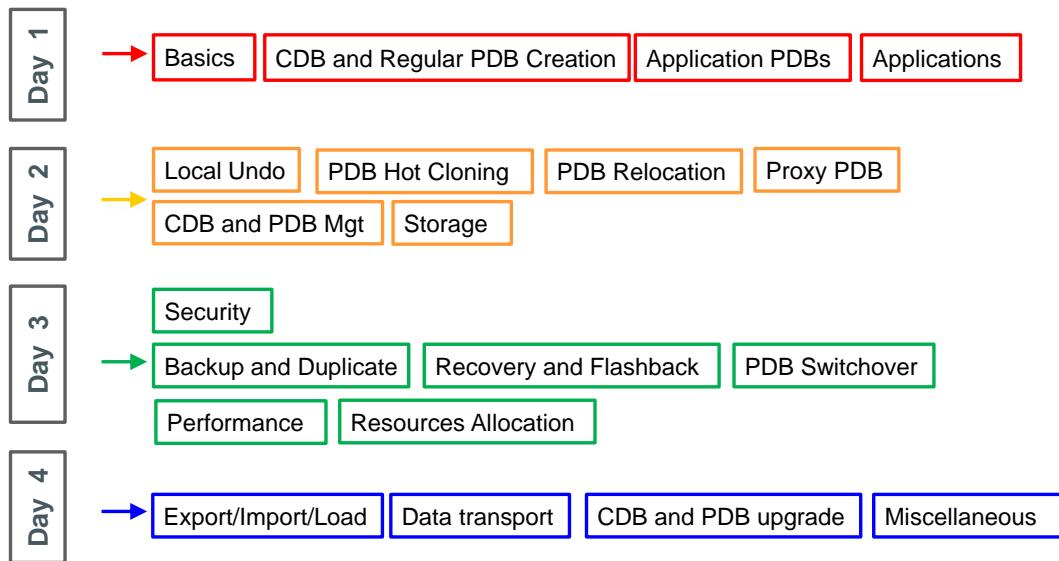
Consolidated Replay Steps	B-2
Procedures for Steps 4 and 5	B-3
Procedures for Steps 6 and 7	B-4
Procedure to Remap Connections with PDBs	B-5
Procedure to Prepare the Replay	B-6
Modes of Synchronization	B-7
Procedure to Start Replay	B-8
Views	B-9

CDB Basics

ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Oracle Database 18c Multitenant Architecture



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The first day will cover the multitenant architecture and the different types of pluggable databases (PDBs) (regular and application) in multitenant container databases (CDBs). After you understand the concept of regular and PDBs, you will create a CDB and then use different methods to create PDBs.

The second day will cover other methods of PDB creation. You will learn how to start and shut down a CDB and how to open and close a PDB.

The third day will cover security aspects in CDBs and PDBs in various areas like privileges and roles, lockdown profiles, auditing, Database Vault, and encryption. Day 3 will also cover availability through backup, duplicate, recovery, and flashback topics and then performance, monitoring, and resources allocation management in CDBs and PDBs.

The fourth day covers how you can move data from a non-CDB environment to a PDB. You will also learn how to move data between PDBs by using utilities such as the export and import features of Oracle Data Pump, SQL*Loader, external tables, and Oracle Recovery Manager. This day will cover upgrade methods for converting a non-CDB to a PDB, upgrading an Oracle Database release 12.2 PDB, or CDB to an 18c PDB or CDB.

Objectives

After completing this lesson, you should be able to:

- Describe the multitenant architecture
- Describe the CDB root and pluggable database containers
- Differentiate the CDB root from a pluggable database
- Explain the terminology of commonality
- List impacts in various areas



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

For a complete understanding of the multitenant architecture and usage, refer to the following Oracle documentation:

- “Changes in Oracle Database 18c, Version 18.1” in *Oracle Multitenant Administrator’s Guide 18c*

Refer to other sources of information available on YouTube, such as:

- [Oracle Multitenant](#)
- [Stale Standalone to Superb SaaS in a Short Series](#)

Other sources of information available to you are as follows:

- Oracle Learning Library (for example, Oracle by Example);
 - *Learning Path: 18c New Features for Multitenant*
 - *Oracle Database 12c New Features Series*
 - *Multitenant Architecture*
 - *Oracle Database 12c Multitenant Architecture Diagram Overview*
 - *Basics of CDB and PDB Architecture*
- MyOracle support notes (MOS), such as *Oracle Multitenant Option - 12c: Frequently Asked Questions (Doc ID 1511619.1)*
- Oracle Database documentation library, starting with *Oracle Multitenant: New Features In Oracle Database 12c Release 2 (12.2)*
- Oracle Technology Network (OTN), in the “Oracle 12c Multitenant” page, a white paper (<http://www.oracle.com/technetwork/database/multitenant/overview/multitenant-wp-12c-2078248.pdf>)

Challenges

Many Oracle customers have large numbers of “departmental” applications built on Oracle RDBMS that:

- Do NOT use a significant percentage of the hardware on which they are deployed
- Have instance and storage overhead preventing large numbers of “departmental” databases from being placed on the same physical and storage server
- Are NOT sufficiently complex to require 100 percent of the attention of a full-time administrator
- Do require significant time to patch or upgrade all applications



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Starting with Oracle Database 12c, the multitenant architecture enables you to have many pluggable databases inside a single Oracle Database instance.

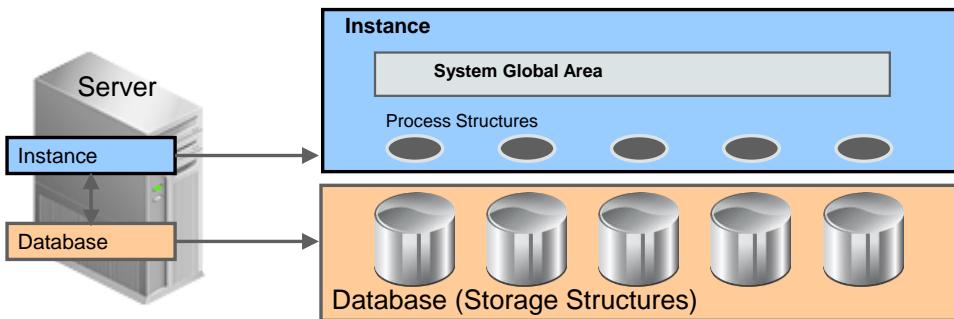
What is the benefit of using the multitenant architecture in Oracle Database 18c?

Currently, many Oracle customers have large numbers of “departmental” applications built on Oracle RDBMS.

- These applications rarely use a significant percentage of the hardware on which they are deployed. A significant number of instances and the amount of storage allocation for all these small databases prevent these from being placed on the same physical and storage server.
- Moreover, they are typically not sufficiently complex to require 100 percent of the attention of a full-time administrator.
- To better exploit hardware and DBA resources, customers would prefer to have most of these departmental applications consolidated on to a single Oracle RDBMS deployment.

The multitenant architecture allows DBAs to consolidate large numbers of small departmental database applications into a single, larger RDBMS installation.

Non-CDB Architecture



- Multiple non-CDBs share nothing:
 - Too many background processes
 - High shared/process memory
 - Many copies of Oracle metadata



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

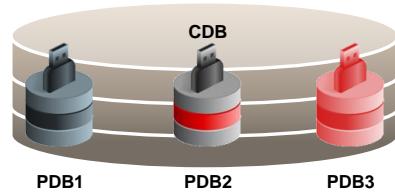
In Oracle Database release 11g, the only kind of database that is supported is a non-CDB. The old architecture is referred to as non-CDB architecture. The term non-CDB is shorthand for a pre-release 12.1 Oracle database. This type of database requires its own instance, which must have its own background processes and memory allocation for the SGA. It must store the Oracle metadata in its data dictionary. The database administrator can still create Oracle 18c non-CDBs with the same pre-12.1 architecture. These databases are not multitenant container databases; they are non-CDBs.

When you administer small departmental database applications, you must create as many databases as applications, which entails multiplying the number of instances. Consequently, this increases the number of background processes and memory allocation for the SGAs. You must therefore provision enough storage for all data dictionaries of these databases.

When you need to upgrade your applications to a new version, you have to upgrade each database, which is time-consuming for the DBA.

Multitenant Architecture: Benefits

- Operates **multiple databases in a centrally managed platform** to lower costs:
 - Less instance overhead
 - Less storage cost
- Reduces DBA resources costs and maintains security
 - No application changes
 - **Fast and easy provisioning**
 - Time saving for patching and upgrade
 - **Separation of duties** between:
 - Different application administrators
 - Application administrators and DBA
 - Users within application
- **Provides isolation**



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Consolidating many non-CDB databases onto a single platform reduces instance overhead, avoids redundant copies of data dictionaries, and consequently storage allocation. It benefits from fast provisioning, time-saving upgrading, and better security through separation of duties and application isolation. The new multitenant database that consolidates databases together is a multitenant container database or CDB and a database consolidated within a CDB, a pluggable database, or PDB.

- **No application change and very fast provisioning:** A new database can be provisioned quickly. A clone of a populated database can be created quickly. A populated database can be quickly unplugged from its CDB on one platform and quickly plugged into a CDB on a different platform. A non-CDB can quickly be plugged into a CDB.
- **Fast upgrade and patching of the Oracle Database version:** The cost (time taken and human effort needed) to upgrade many PDBs is the cost of upgrading a single Oracle Database occurrence. You can also upgrade a single PDB by unplugging it and plugging it into a CDB at a different Oracle database version.
- **Secure separation of duties:** The administrator of an application can perform the required tasks by connecting to the PDB that implements its back end. However, a user who connects to a PDB cannot see other PDBs. To manage PDBs as entities (for example, to create, drop, or unplug or plug one), a system administrator must connect to the CDB. A user connecting to a CDB must have special privileges.
- **Isolation of applications:** This task may not be achieved manually unless you use Oracle Database Vault, for example. A good example of isolation is dictionary separation enabling an Oracle database to manage the multiple PDBs separately from each other and from the CDB itself.

Other Benefits of Multitenant Architecture

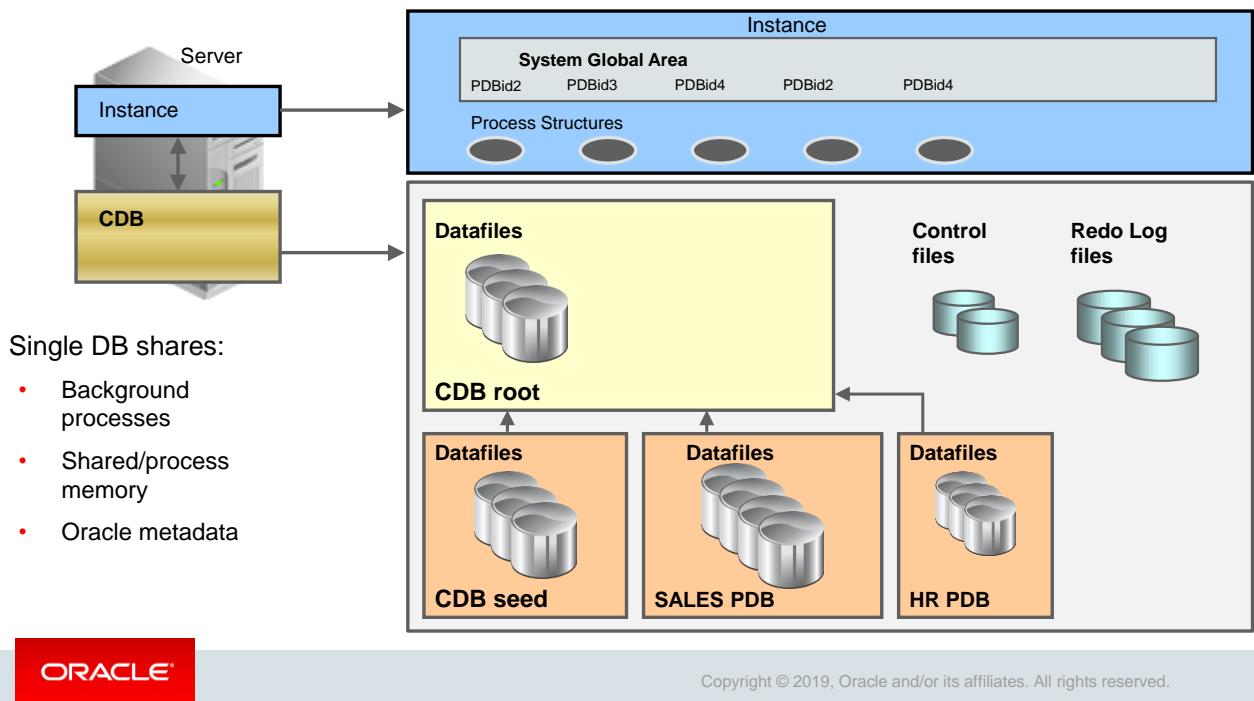
- Ensures **full backward-compatibility** with non-CDBs
- Fully operates with Oracle Real Application Cluster (Oracle RAC) and Data Guard
- Is supported by Oracle Enterprise Manager
- Is integrated with Resource Manager
- Allows central management and administration of multiple databases
 - Backups or disaster recovery
 - Patching and upgrades



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

- The multitenant architecture ensures the backward-compatibility principle. An example is the data dictionary views. The `DBA_OBJECTS` view shows the same results in a PDB as in a non-CDB for a particular application.
- The multitenant architecture is designed to be fully interoperable with RAC. Each instance in an Oracle RAC opens the CDB as a whole. A session sees only the single PDB it connects to.
- Enterprise Manager integrates CDBs and models the separation of duties of the CDB administrator and the PDB administrator.
 - A CDB can be defined as a target. An Enterprise Manager user can be given the credentials to act as a CDB administrator in such a target.
 - A PDB can be set up as a subtarget of a CDB target. An Enterprise Manager user can be given the credentials to act as a PDB administrator in such a target. An Enterprise Manager user that has been set up with the credentials to act as a PDB administrator for a particular PDB is able to connect to that one PDB and is unaware of the existence of peer PDBs in the same CDB. Moreover, when the intention is to carry out the duties of an application administrator, this Enterprise Manager user is unaware that the environment is a CDB and not a non-CDB.
- Resource Manager is extended with new between-PDB capabilities to allow the management of resources between the PDBs within a CDB. The backward-compatibility principle implies that Resource Manager must function in exactly the same way within a PDB as it does in a non-CDB.
- When you upgrade a whole CDB with n PDBs, you achieve the effect of upgrading n non-CDBs for the cost of upgrading one non-CDB.

Oracle Multitenant Container Database



This slide illustrates the consolidation of three applications that were deployed into three distinct non-CDBs into a single one. The graphic in the slide shows a multitenant container database with four containers: the CDB root and three pluggable databases. Each pluggable database has its own dedicated application and is managed either by its own DBA or by the container administrator that is `SYS` user of the CDB root container, a common `SYS` user. This common `SYS` user can manage the CDB root container and every pluggable database.

A pluggable database is a set of database schemas that appears logically to users and applications as a separate database. But at the physical level, the multitenant container database has a database instance and database files, just as a non-CDB does. Nothing changes: neither the client code nor the database objects.

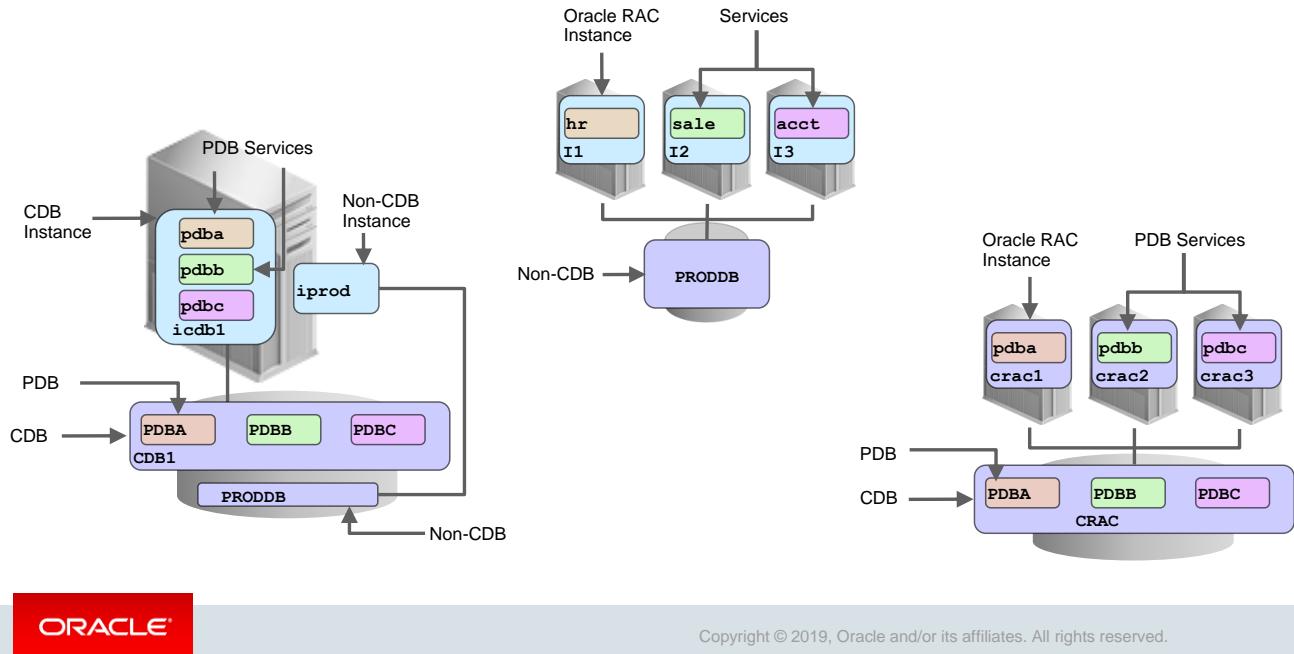
It is easy to plug non-CDBs into a CDB. A CDB avoids redundancy of:

- Background processes
- Memory allocation
- Oracle metadata in several data dictionaries

A CDB that groups several applications ends up with one instance. This instance will have one set of background processes, one SGA allocation, and one data dictionary in the CDB root container, common for all PDBs. Each PDB will maintain its own application data dictionary.

When you must patch or upgrade an application, you can perform maintenance operation only once on the CDB. Consequently, all the applications associated with the CDB will be patched or updated at the same time.

Configurations



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

What are the possible instances of database configurations?

- Each database instance can be associated with one and only one non-CDB or multitenant container database.
- In an Oracle RAC environment, several instances can be associated to a non-CDB or multitenant container database.
- An instance is associated with an entire CDB.

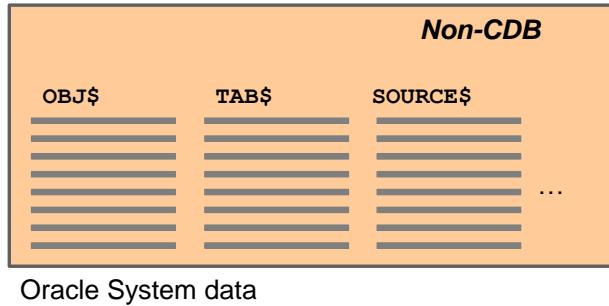
If there are multiple databases on the same server, then there is a separate and distinct instance for each non-CDB or CDB. An instance cannot be shared between a non-CDB and CDB.

There are three possible configuration options:

- **Multitenant configuration:** Typically more than one PDB per CDB, but can hold zero, one, or many PDBs at any one time, taking advantage of the full capabilities of the Multitenant architecture, which requires the licensed Oracle Multitenant option
- **Single-tenant configuration:** The special case of the Multitenant architecture, which does not require the licensed option
- **Non-CDB:** The Oracle Database 11g architecture

Database Objects in a non-CDB

After the initial database creation, the only objects are Oracle-supplied objects.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

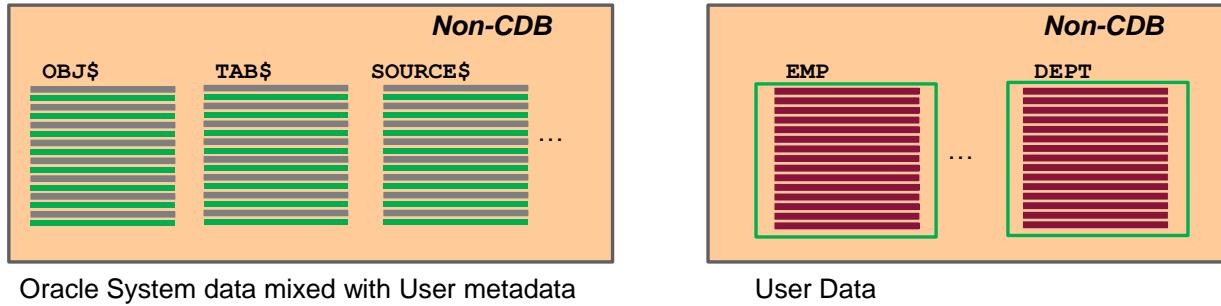
How is Oracle metadata shared between several applications in a non-CDB?

Immediately after the creation of the non-CDB, the only objects in the data dictionary are the Oracle-supplied objects. At this point, there is no user data.

The only schemas in the database are those required by the database system.

User-Added Objects to a non-CDB

In a non-CDB, user data is added: The metadata is mixed with the Oracle-supplied data in the data dictionary.



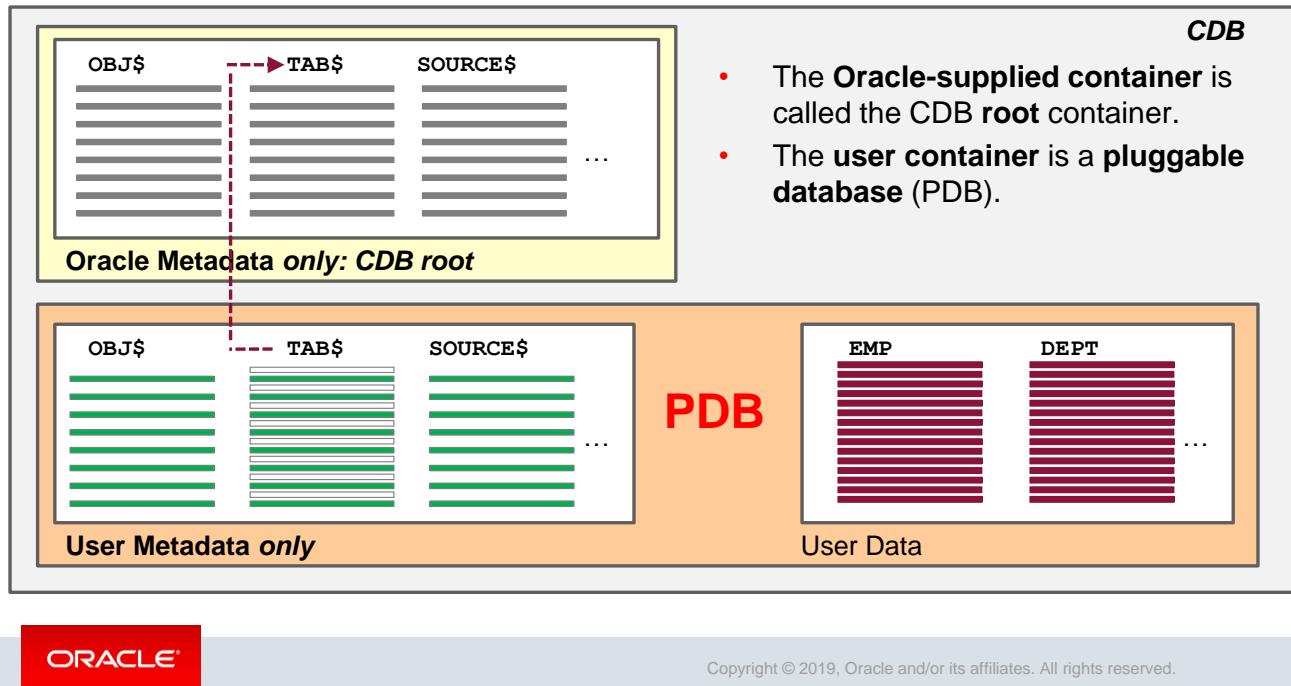
Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

In the non-CDB, users and user data are added.

The best practice is to add this data in tablespaces that are dedicated to user data. Storing the data in separate tablespaces enables you to protect, secure, and transport the data more easily, even though the user metadata is in the data dictionary along with the Oracle system metadata:

- Object definitions
- User definitions
- PL/SQL code
- Other user-created objects

SYSTEM Objects in the USER Container



In a multitenant container database, the concept of containers is introduced to separate the Oracle-supplied objects and the user data including the metadata into distinct containers.

Each container has a `SYSTEM` tablespace that holds a data dictionary.

- There is a dictionary in the Oracle metadata-only container that has the metadata for the Oracle-supplied objects.
- There is a dictionary in the user container holding the user metadata.

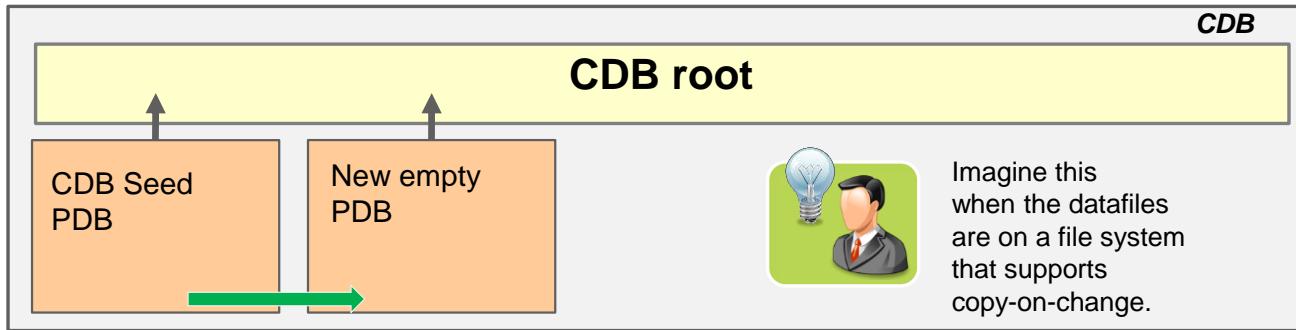
One of the goals of the multitenant architecture is that each container has a one-to-one relationship with an application.

Separating the metadata is the first step, the second is allowing the application or users inside the “user” container to access the Oracle-supplied objects. The user container is called a pluggable database (PDB).

The Oracle objects could have been duplicated in each PDB, but that takes a lot of space and would require every PDB to be upgraded each time an Oracle-supplied object changes, for example, with patches. The Oracle-supplied objects reside in a container called the CDB root container, which is named `CDB$ROOT`.

Pointers from a PDB to the Oracle-supplied objects allow the “system” objects to be accessed without duplicating them in the PDB. The PDB has the pieces it needs to be a complete environment for a database application. The application can run in the PDB just as it does in a non-CDB.

Provisioning a Pluggable Database



Different methods:

- **Create** new PDB from **CDB seed** pluggable database.
- **Plug** or **clone** a non-CDB as a PDB into a CDB.
- **Clone** or **relocate** a PDB from another PDB into the same or another CDB.
- **Plug an unplugged PDB** into the same or another CDB.
- **Create** a PDB as a **proxy** PDB.

ORACLE®

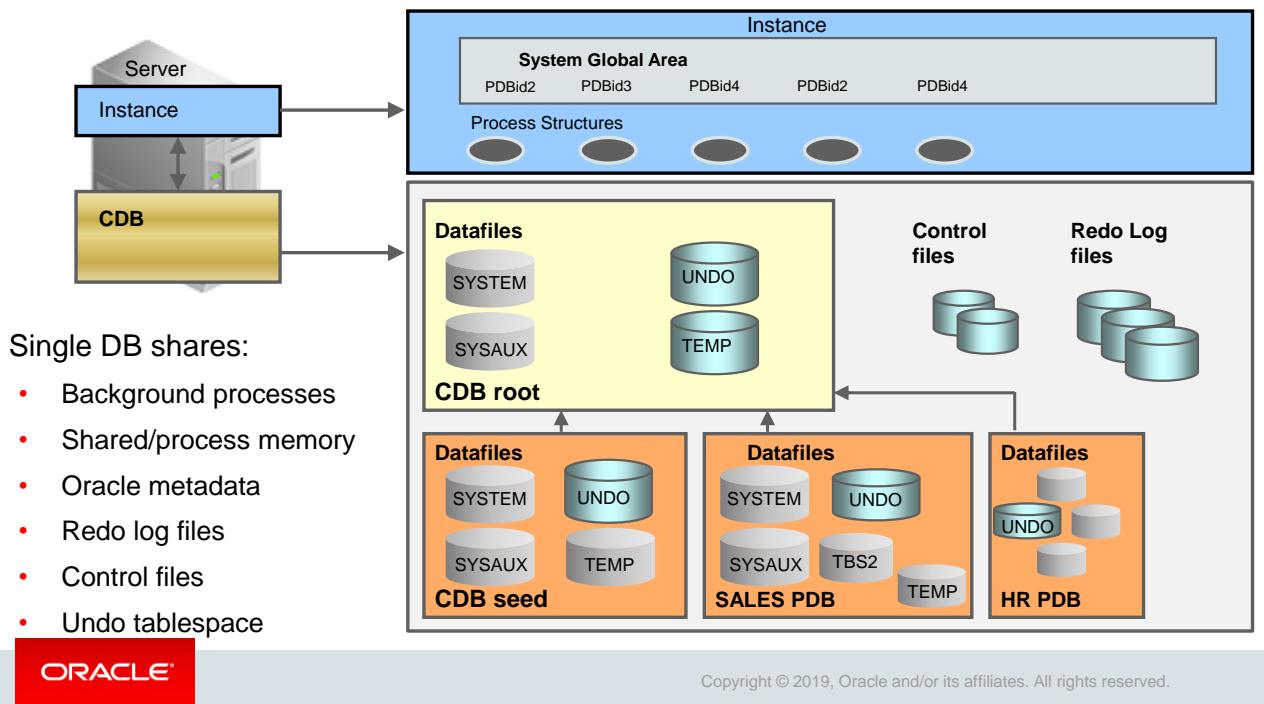
Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

To create a new PDB, use the provided CDB seed PDB. This CDB seed container is named `PDB$SEED` and is a part of every CDB. When you create a new PDB, the CDB seed PDB is cloned and gives the new PDB the name you specify. This operation is very fast. It is measured in seconds. The time that is taken is mostly for copying the files.

There are different methods to provision pluggable databases:

- **Create a new PDB from the `PDB$SEED` pluggable database:** This scenario is useful, for example, for a new application implementation.
- **Create a new PDB from a non-CDB:** Plug the non-CDBs in a CDB as PDBs as part of migration strategy. It is also a good way to consolidate the non-CDBs into a CDB.
- **Clone a non-CDB:** Clone the non-CDBs in a CDB as PDBs, as part of migration strategy. This is a good way to keep the non-CDB and therefore have the opportunity to compare the performance between the new PDB and the original non-CDB or at least wait until you consider that the PDB can work appropriately.
- **Clone a PDB from another PDB into the same or another CDB:** An example of this method is application testing. Relocate a PDB into another CDB so as to dispatch resources.
- **Plug an unplugged PDB into another CDB:** For example, instead of upgrading a multitenant container database from one release to another, you can unplug a pluggable database from one Oracle Database release and then plug it into a newly created multitenant container database from a higher release.
- **Proxy a PDB:** A proxy PDB provides fully functional access to another PDB in a remote CDB. This feature enables you to build location-transparent applications that can aggregate data from multiple sources that are in the same data center or distributed across data centers.

Multitenant Container Database Architecture

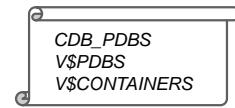


The graphic in the slide shows a CDB with four containers: the CDB root, the CDB seed, and two PDBs. The two applications use a single instance and are maintained separately.

At the physical level, the CDB has a database instance and database files, just as a non-CDB does.

- The redo log files are common for the whole CDB. The information it contains is annotated with the identity of the PDB where a change occurs. Oracle GoldenGate can understand the format of the redo log for a CDB. All PDBs in a CDB share the ARCHIVELOG mode of the CDB.
- The control files are common for the whole CDB. The control files are updated to reflect any additional tablespace and datafiles of plugged PDBs.
- An UNDO tablespace is by default local in each container. It is possible to have a single UNDO tablespace shared by all containers. In this case, there is one UNDO tablespace per instance in a RAC database.
- The CDB root or a PDB can have only one default temporary tablespace or tablespace group. Each PDB can have temporary tablespaces for use by local or common users in the PDB.
- Each container has its own data dictionary stored in its proper SYSTEM tablespace, containing its own metadata, and a SYSAUX tablespace.
- The PDBs can create tablespaces within the PDB according to application needs.
- Each datafile is associated with a specific container, named *CON_ID*.

Containers



Types of containers in V\$CONTAINERS:

- **CDB root**
 - The first **mandatory** container created at CDB creation
 - Oracle system-supplied common objects and metadata
 - Oracle system-supplied common users and roles
- **Pluggable database (PDBs)**
 - Tablespaces (permanent and temporary)
 - Schemas / Objects / Privileges
 - Created / cloned / unplugged / plugged / proxied
 - Particular PDB: CDB seed (`PDB$SEED`) used for fast provisioning of a new PDB



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

To summarize, a CDB is an Oracle database that contains the CDB root, the CDB seed, and possibly several pluggable databases (PDBs).

What is a PDB in a CDB? A PDB is the lower part of the horizontally partitioned data dictionary plus the user's quota-consuming data.

A non-CDB cannot contain PDBs. The multitenant architecture enables an Oracle database to contain a portable collection of schemas, schema objects, and non-schema objects that appear to an Oracle Net client as a separate database. For the PDBs to exist and work, the CDB requires a particular type of container, the CDB root, generated at the creation of the CDB. The CDB root is a system-supplied container that stores common users, which are users that can connect to multiple containers, and system-supplied metadata and data. The source code for system-supplied PL/SQL packages is stored in the CDB root.

There is only one CDB seed PDB in a CDB. The CDB seed PDB is a system-supplied template that is used to create new PDBs.

A CDB can contain up to 4,096 PDBs, including the CDB seed, the services being limited to 10,000.

The `V$CONTAINERS` view displays all PDBs, including the CDB root and the CDB seed.

Tools

	SQL*Plus	OUI	DBCA	EM Cloud Control	EM Database Express	SQL Developer	DBUA
Create a new CDB or PDB	Yes	Yes	Yes	Yes (PDB only)	Yes (PDB only)	Yes (PDB only)	
Explore CDB instance, architecture, and PDBs	Yes			Yes	Yes	Yes	
Upgrade a 12c CDB to 18c CDB				Yes			Yes



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

There are different tools to create and upgrade container databases. As shown in the slide, you can create a new CDB or new PDBs either using SQL*Plus or Database Configuration Assistant (DBCA) or during the installation of Oracle Database 18c. SQL Developer and EM Cloud Control allow you to create pluggable databases.

After you create a CDB, you can use views to explore the instance, database architecture, files, and pluggable databases of the CDB. Query views directly with SELECT statements using SQL*Plus or indirectly using GUI tools such as Enterprise Manager or SQL Developer.

You can upgrade an Oracle Database release 12c CDB to an Oracle Database 18c CDB with Enterprise Manager or Database Upgrade Assistant (DBUA).

Note: Oracle Enterprise Manager Database Express cannot be used to create a CDB, but it can be used to create PDBs and explore PDBs architecture or CDBs structures by using different port configurations.

Data Dictionary and Dynamic Views

CDB_XXX All objects in the multitenant container database across all PDBs

DBA_XXX All of the objects in a container or pluggable database

ALL_XXX Objects accessible by the current user

USER_XXX Objects owned by the current user

- CDB_pdb\$: All PDBs within CDB
- CDB_tablespaces: All tablespaces within CDB
- CDB_users: All users within CDB (common and local)

```
SQL> select OBJECT_ID, ORACLE_USERNAME, LOCKED_MODE, CON_ID from V$LOCKED_OBJECT;
OBJECT_ID  ORACLE_USERNAME  LOCKED_MODE  CON_ID
-----  -----
  83711    SYS            3           3 ← PDB1
  83710    DOM            3           4 ← PDB2
```



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

For backward-compatibility, DBA views show the same results in a PDB as in a non-CDB: DBA_OBJECTS shows the objects that exist in the PDB from which you run the query. This implies, in turn, that although the PDB and the CDB root have separate data dictionaries, each data dictionary view in a PDB shows results fetched from both of these data dictionaries. The DBA_XXX views in the CDB root shows, even in a populated CDB, only the Oracle-supplied system—as is seen in a freshly created non-CDB.

To support the duties of the CDB administrator, a family of data dictionary views is supported with names such as CDB_XXX. Each DBA_XXX view has a CDB_XXX view counterpart with an extra column, Con_ID, that shows from which container the listed facts originate. Query the CDB_XXX views from the CDB root and from any PDB. The results from a particular CDB_XXX view are the union of the results from the DBA_XXX view counterpart over the CDB root and all currently open PDBs. When a CDB_XXX view is queried from a PDB, it shows only the information that it shows in its DBA_XXX view counterpart. If you connect to the CDB root and query CDB_USERS, you get the list of users, common and local, of each container. If you query DBA_USERS, you get the list of common users. If you connect to a PDB and query CDB_USERS or DBA_USERS, you get the same list of users, common and local, of the PDB.

The same backward-compatibility principle also implies to each of the familiar V\$ views.

Terminology

- DBA, CDB_DBA, and PDB_DBA (roles assigned to administrators at different levels)
- Common vs Local:
 - Users
 - Privileges / Roles
 - Objects
 - Profiles
- CDB vs PDB level:
 - CDB Resource Manager plan vs PDB RM plan
 - Unified audit policies at CDB or PDB level
 - Encryption master keys at CDB and PDB level
 - Database Vault realms and command rules at CDB or PDB level
 - XStream at CDB or PDB level



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

There are different types of database administrators.

- In a non-CDB, the DBA is responsible for all administrative tasks at the database level.
- In a CDB, there are different levels of administration:
 - The DBA responsible for administering the CDB instance, the CDB root, and all PDBs
 - The DBAs responsible for administering their respective PDB

The terminology for entities in a CDB and in PDBs is the following:

- Common users, roles, and profiles exist in all containers and have the same name throughout these containers. Local users, roles, and profiles have a unique name for the container (PDB) in which they reside.
- Common privileges are privileges that are "commonly" granted for all containers in the CDB, rather than privileges that are granted locally within a PDB.
- Common objects exist in Oracle-supplied schemas. Local objects are created in PDBs in local schemas.
- CDB resource management works at the CDB level, and PDB resource management works at the PDB level.
- Audit policies can be created in the CDB root and also in each PDB. There is the same concept for encryption master keys and for Database Vault realms and command rules.
- XStream Out is only available at CDB level and XStream In only at PDB level.
XStream consists of Oracle Database components and application programming interfaces (APIs) that enable client applications to receive data changes from an Oracle database and send data changes to an Oracle database. XStream Out provides Oracle Database components and APIs that enable you to share data changes made to an Oracle database with other systems. XStream In provides Oracle Database components and APIs that enable you to share data changes made to other systems with an Oracle database.

Impacts

- Define a character set for the CDB and per PDB.
- Define PDB initialization parameters in a single SPFILE.
- Do not use PDB-qualified database object names. Instead use database links.

```
SQL> SELECT * FROM HR:apps.tab1;          SQL> SELECT * FROM apps.tab1@HR;
```

- Implement subset standbys at the PDB level.
- Configure Oracle Database Vault per PDB and on common objects.
- Create one TDE master encryption key per PDB to encrypt PDB data.
- Configure unified audit at CDB and PDB level.
- Benefit from Heat Maps and Automatic Data Optimization.
- Use Logminer for objects at all levels.
- Configure replication at PDB and application level with XStream and Oracle GoldenGate.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

- If the CDB has a unicode database character set of AL32UTF8, the CDB can contain PDBs with different database character sets, because all character sets can be converted to AL32UTF8. Character set of an existing PDB can be changed to any compatible character set using existing database character set migration steps. There is only one single spfile for the CDB. PDB parameters values are stored in a dictionary table.
- Use a database link to access an object in another PDB.
- If you use Oracle Active Dataguard for reporting purposes, then you do not need to replicate the PDBs. Oracle Database release 12c enables you to implement a subset of PDBs in the standby database by using the STANDBYS clause with a list of the PDBs to replicate. If a user executes the ALTER DATABASE SWITCHOVER TO ... VERIFY statement to switch over a standby database with the subset of PDBs to the primary database, a warning error message appears. However, these standby databases can become primary databases. In the following statement, PDB1 will be created on the standby databases `stdby1` and `stdby2`:

```
SQL> CREATE PLUGGABLE DATABASE pdb1 ... STANDBYS=(stdby1,stdby2);
```

- In Oracle Database Vault, each PDB has its own Database Vault metadata. Database Vault constructs, such as realms, are isolated within a PDB. Oracle Database release 12.2 introduced protection on common objects with common realms and command rules.
- Each PDB has its own master key used to encrypt data in the PDB. The TDE master encryption key must be transported from the source database keystore to the target database keystore when a PDB is moved from one host to another. For column encryption, each PDB maintains its own ENC\$, which is not a metadata-linked object.

- A unified audit configuration is visible and enforced across all PDBs. It enables administrators to avoid configuring auditing separately for each container. This provides the ability to not only create audit policies used by all PDBs but also audit policies used exclusively for each PDB. An audit configuration that is not enforced across all PDBs means it applies only within a PDB and is not visible outside it.
- Heat Map and ADO (Automatic Data Optimization) enable automation of Information Lifecycle Management (ILM) actions, automating movement of data to the appropriate storage format through compression and storage tiering. ADO relies on statistics reported and collected by Heat Map, a tracking activity at both the segment level and the block level. Heat Map and ADO are enabled in CDBs since Oracle Database 12c release 2.
- XStream is a programmatic interface to allow a client application access to the changes in the database, known as XStream Outbound Server. XStream Inbound Server allows a client application to feed changes into the database and takes advantage of the apply process available in the database. Oracle GoldenGate is the logical replication, and XStream is licensed via the Oracle GoldenGate (OGG) license. Capturing changes from the database must always be from a CDB root. The XStream outbound can be configured to capture changes from a PDB or the entire CDB. Applying changes via Oracle GoldenGate is done per PDB. An XStream inbound server is configured to apply changes into a specific PDB and performs all of its work within the context of the PDB.
Support in XStream and Oracle GoldenGate applies with no specific restrictions.
- Logminer ad hoc query (`V$LOGMNR_CONTENTS`, `DBMS_LOGMNR`) supports customer common objects in PDBs just as they support local objects in PDBs.
- In general, all scheduler objects created by the user can be exported or imported into the PDB using data pump. Predefined scheduler objects are not exported, and that means that any changes made to these objects by the user will have to be applied once again when the database is imported into the PDB. A job defined in a PDB runs only if a PDB is open.

Summary

In this lesson, you should have learned how to:

- Describe the multitenant architecture
- Describe the CDB root and pluggable database containers
- Differentiate the CDB root from a pluggable database
- Explain the concept of commonality
- List impacts in various areas



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Practices Environment - 1

Each student has two virtual machines (VM1 and VM2) to use during class.

- **On VM1:**
 - Oracle Database 18c installed:
 - With one CDB named `ORCL` and one PDB named `PDB1`. The CDB has been set up for you in advance so that you can explore an existing configuration.
 - You will create another CDB, `CDB18`, and its PDB, `PDB18`.
 - Oracle Database 12.2 installed:
 - With one CDB named `CDB12` and one PDB named `PDB12` to perform upgrade operations to Oracle Database 18c.
 - With a non-CDB named `NONCDB` to perform plugging or cloning operations into one of the Oracle Database 18c CDBs. This shows you how to migrate a non-CDB into a CDB.
- **On VM2:**
 - Enterprise Manager 13c deployed to monitor Oracle Database 18c targets.
 - Oracle Database 12.1 installed with one CDB named `cdbem` and one PDB named `PDBEM` used as the Oracle Management Repository (OMR).



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Practices Environment - 2

Pre-created databases and instances with their respective PDBs:

- Datafiles in /u02/app/oracle/oradata/
- CDB root datafiles in /u02/app/oracle/oradata/<db_name>
- PDB datafiles in /u02/app/oracle/oradata/<db_name>/<pdb_name>
- Control files in /u02/app/oracle/oradata/<db_name>
and /u03/app/oracle/fast_recovery_area/<db_name>
- All redo log files in /u04/app/oracle/redo/<db_name>
- All backup files in /u03/app/oracle/fast_recovery_area/<db_name>
- Password and init files in \$ORACLE_HOME/dbs
- Diagnostics files in /u01/app/oracle/diag/rdbms/orcl/ORCL/...
- TDE wallet in /u01/app/oracle/admin/<db_name>/tde_wallet
- Net files in \$ORACLE_HOME/network/admin



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The configuration of the environment used for the practices of the course matches the configuration used on the Oracle Cloud Virtual Machine for the pre-created ORCL database of the Oracle Database Cloud Service. The Oracle Database Cloud Service provides you the ability to deploy Oracle databases in the Cloud, with each database deployment containing a single Oracle database.

This is a good way to get familiar with the Oracle Database Cloud environment that is preconfigured for Cloud customers.

The configuration used on the Oracle Cloud Virtual Machine for the pre-created database of a database deployment will be covered in detail in the lesson titled “PDBs in Cloud.”

Practice 1: Overview

- 1-1: Discovering practices environment
- 1-2: Adding a CDB as a new target in Enterprise Manager Cloud Control
- 1-3: Checking named credentials
- 1-4: Using Enterprise Manager Express

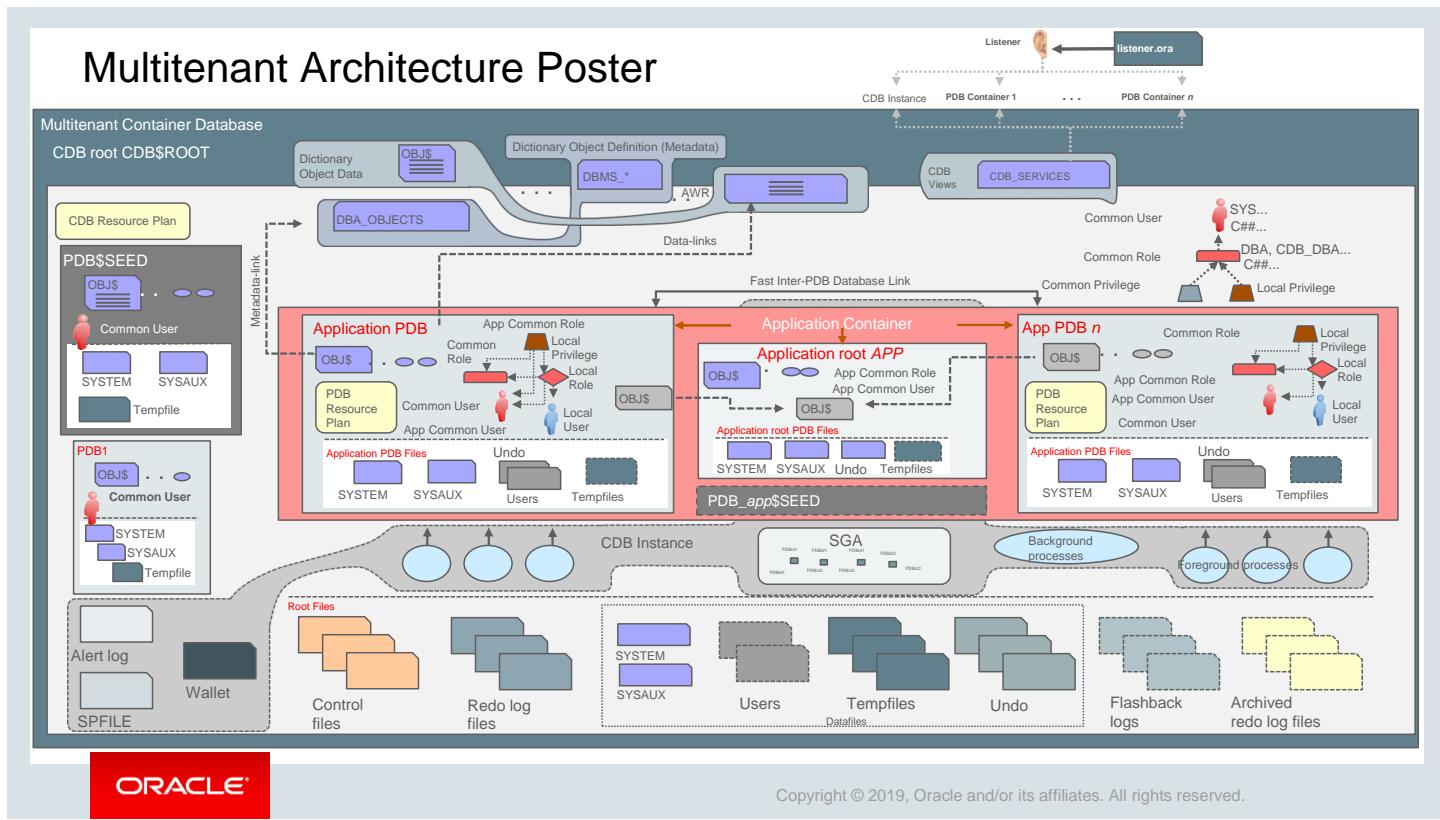
Note:

In most of the practices, you will have to execute setup and cleanup shell scripts and SQL scripts. The scripts may generate false errors in the following occurrences:

- A tablespace dropped because the tablespace was not created
- A pluggable database dropped because the pluggable database was not created
- A directory creation dropped because the directory already exists
Do not pay attention to the errors.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

ORACLE®

CDB and Regular PDBs

The Oracle logo, consisting of the word "ORACLE" in white capital letters inside a red rectangular box.

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Configure and create a CDB
- Create a new PDB from the CDB seed
- Explore the instance
- Explore the structure of PDBs
- Explore the ADR



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

For a complete understanding of CDB and PDBs creation procedures, refer to the Oracle documentation:

- *Oracle Multitenant Administrator's Guide 18c*

Goals

Create a multitenant container database:

- To consolidate many pre-12.1, 12c, and 18c non-CDBs into a single, larger database
- To prepare a container:
 - For plugging any future new application
 - For testing applications
 - For diagnosing application performance
- To simplify and reduce time for patching and upgrade

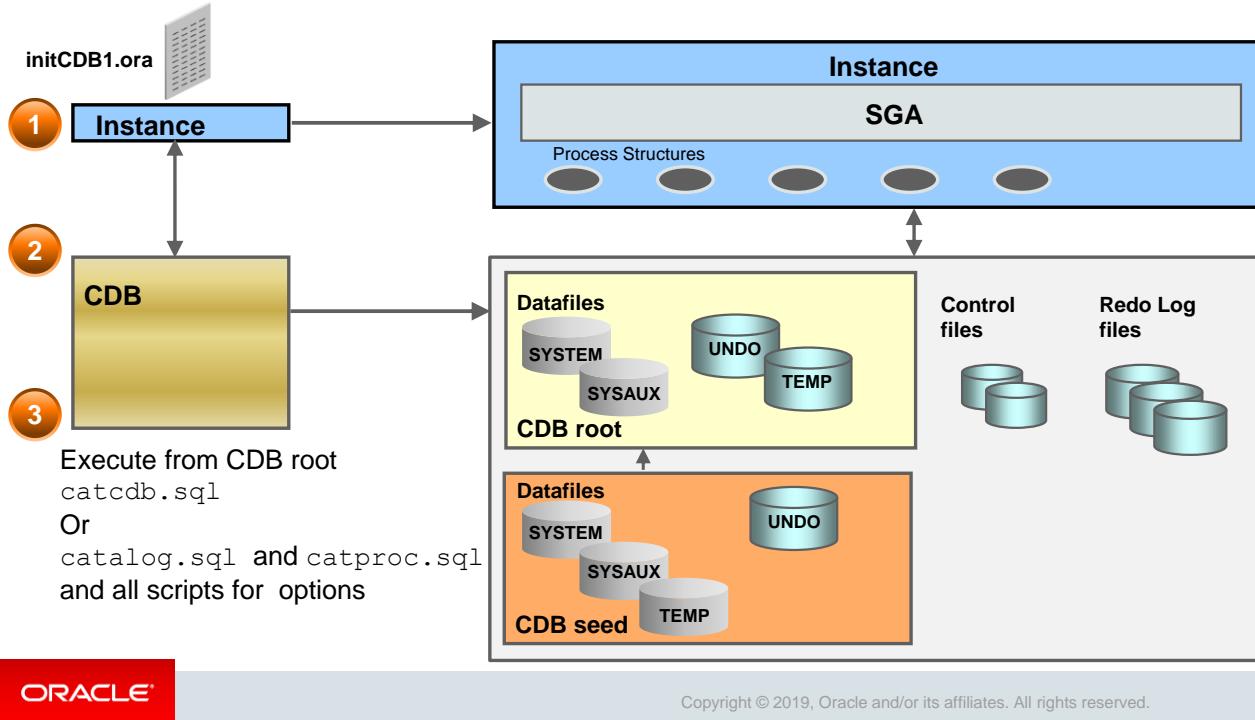


Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Why create multitenant container databases?

There are several reasons for creating a multitenant container database as explained in the slide.

Creating a CDB



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The steps required to create a new CDB, using either DBCA or SQL*Plus, are the same.

- The first step, as for any database, non-CDB or CDB, consists of configuring an instance with an `init.ora` parameter file and then starting the instance.
- The second step is the creation of the CDB using the `CREATE DATABASE` command with a new clause `ENABLE PLUGGABLE DATABASE` specifying that the database is a multitenant container database and not a non-CDB. The operation creates the controlfiles during the mount phase, the redo log files, and CDB root datafiles during the open phase. The CDB root datafiles are used for the `SYSTEM` tablespace containing the Oracle-supplied metadata and data dictionary, the `SYSAUX` tablespace for AWR, and the `UNDO` tablespace. It also creates the CDB seed with its own datafiles used for the `SYSAUX`, `SYSTEM` and `UNDO` tablespaces. You may use the new clause `SEED FILE_NAME_CONVERT` to define the datafiles location of the CDB seed pluggable database. The `FILE_NAME_CONVERT` clause creates the CDB seed. The CDB seed datafiles can be used as templates for future PDBs creation. If you omit this clause, Oracle Managed Files determines the names and locations of the CDB seed's files.
- The third step is the creation of the catalog with the execution of the `$ORACLE_HOME/rdbms/admin/catcdb.sql` script connected to the CDB root.

Creating a CDB: Using SQL*Plus

1. Start up the instance :

- Set ORACLE_SID=CDB1.
- Create the initCDB1.ora file and set parameters:
 - CONTROL_FILES to CDB control file names
 - DB_NAME to a CDB name
 - ENABLE_PLUGGABLE_DATABASE to TRUE

```
SQL> CONNECT / AS SYSDBA
SQL> STARTUP NOMOUNT
```

2. Create the database:

```
SQL> CREATE DATABASE cdb1 ENABLE PLUGGABLE DATABASE ...
      SEED FILE_NAME_CONVERT = ('/oracle/dbs','/oracle/seed');
```

→ CDB\$ROOT + PDB\$SEED created

3. Execute the \$ORACLE_HOME/rdbms/admin/catcdb.sql SQL script.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Below are the detailed steps to create a new CDB using SQL*Plus.

- Before starting the instance, create an init<SID>.ora parameter file with the parameters: DB_NAME, CONTROL_FILES if OMF is not used, and DB_BLOCK_SIZE. The global database name of the CDB root is the global database name of the CDB. The ENABLE_PLUGGABLE_DATABASE parameter set to TRUE is required to define that the instance is ready for a CDB and not a non-CDB creation. You can also use the MAX_PDBS parameter to limit the number of PDBs in the CDB. Set the ORACLE_SID environment variable. Launch SQL*Plus, connect as an OS authenticated user belonging to the DBA OS group, and execute the STARTUP NOMOUNT command. If you are using Oracle ASM storage to manage your disk storage, then you must start the Oracle ASM instance and configure your disk groups before performing the next steps.
- Use the CREATE DATABASE command with the clause ENABLE_PLUGGABLE_DATABASE to create a CDB and not a non-CDB. The command creates the CDB root and the CDB seed. You can use the clause SEED FILE_NAME_CONVERT to specify the location of the CDB seed's files. If you omit the clause, OMF determines the names and locations of the CDB seed's files. The FILE_NAME_CONVERT specifies the source directory of the CDB root datafiles and the target CDB seed directory. Omit the clause SEED FILE_NAME_CONVERT if you use the new init.ora parameter PDB_FILE_NAME_CONVERT, mapping names of the CDB root datafiles to the CDB seed datafiles. The directories must exist. The character set defined is the single one for the CDB.
- Run the catcdb.sql SQL script to build views on the data dictionary tables and install standard PL/SQL packages in the CDB root. You can also execute the catalog.sql and catproc.sql SQL scripts and all other SQL scripts related to the options installed.

New Clause: SEED FILE_NAME_CONVERT

```
SQL> CREATE DATABASE cdb1
  USER SYS IDENTIFIED BY p1 USER SYSTEM IDENTIFIED BY p2
  LOGFILE GROUP 1 ('/u01/app/oradata/CDB1/redo1a.log',
    '/u02/app/oradata/CDB1/redo1b.log') SIZE 100M,
  GROUP 2 ('/u01/app/oradata/CDB1/redo2a.log',
    '/u02/app/oradata/CDB1/redo2b.log') SIZE 100M
  CHARACTER SET AL32UTF8 NATIONAL CHARACTER SET AL16UTF16
  EXTENT MANAGEMENT LOCAL DATAFILE
    '/u01/app/oradata/CDB1/system01.dbf' SIZE 325M
  SYSAUX DATAFILE '/u01/app/oradata/CDB1/sysaux01.dbf' SIZE 325M
  DEFAULT TEMPORARY TABLESPACE tempts1
    TEMPFILE '/u01/app/oradata/CDB1/temp01.dbf' SIZE 20M
  UNDO TABLESPACE undotbs
    DATAFILE '/u01/app/oradata/CDB1/undotbs01.dbf' SIZE 200M
  ENABLE PLUGGABLE DATABASE
SEED FILE_NAME_CONVERT =('/u01/app/oradata/CDB1','/u01/app/oradata/CDB1/seed');
```



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

You can find in the slide an example of a full CREATE DATABASE statement.

What is new compared to the non-CDB CREATE DATABASE statement?

The first important clause required if you want the database to be a multitenant container database is **ENABLE PLUGGABLE DATABASE** clause correlated with the **ENABLE_PLUGGABLE_DATABASE** initialization parameter set to TRUE, and one way to declare the directory for the CDB seed datafiles is to use the **SEED FILE_NAME_CONVERT** clause. The **FILE_NAME_CONVERT** specifies the source directory of the CDB root datafiles and the target CDB seed directory.

The /u01/app/oradata/CDB1 CDB root directory and the /u01/app/oradata/CDB1/seed CDB seed directory must exist.

New Clause: ENABLE PLUGGABLE DATABASE

Without **SEED FILE_NAME_CONVERT**:

- OMF: **DB_CREATE_FILE_DEST** = '/u02/app/oradata'

```
SQL> CONNECT / AS SYSDBA
SQL> STARTUP NOMOUNT
SQL> CREATE DATABASE cdb2
  USER SYS IDENTIFIED BY p1
  USER SYSTEM IDENTIFIED BY p2
  EXTENT MANAGEMENT LOCAL
  DEFAULT TEMPORARY TABLESPACE temp
  UNDO TABLESPACE undotbs
  DEFAULT TABLESPACE users
ENABLE PLUGGABLE DATABASE;
```

- Or new instance parameter: **PDB_FILE_NAME_CONVERT** = '/u02/app/oradata/CDB2','/u02/app/oradata/seed'



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Oracle Managed Files

If you do not use explicit datafile names, use Oracle Managed Files (OMF):

- Set the **DB_CREATE_FILE_DEST** instance parameter with the value of the destination directory of the datafiles of the **SYSTEM**, **SYSAUX**, **UNDO**, and **USERS** tablespaces specified in the statement. Oracle chooses default sizes and properties for all datafiles, control files, and redo log files. In the first example, the **/u01/app/oradata** directory must exist.
If you use Oracle ASM storage, you can set **DB_CREATE_FILE_DEST** to **+data**, where **+data** would be a preconfigured ASM disk group.

PDB_FILE_NAME_CONVERT Instance Parameter

If you do not use the **SEED FILE_NAME_CONVERT** clause, use a new instance parameter:

- The **PDB_FILE_NAME_CONVERT** instance parameter maps names of existing files (the root datafiles in your case) to new file names (the seed datafiles in this case).
In the example, both **/u02/app/oradata/CDB1** and **/u02/app/oradata/seed** directories must exist.

After CDB Creation: What's New in CDB

A CDB has new characteristics compared to non-CDBs:

- **Two containers:**
 - The CDB **root** (CDB\$ROOT)
 - The CDB **seed** (PDB\$SEED)
- **Several services:** One per container
 - Name of CDB root service = name of the CDB (`cdb2`)
 - Maximum number of services: 10000
 - Max nb of services per PDB<= max nb of services in CDB
- **Common users** in CDB root and CDB seed: `SYS`, `SYSTEM` ...
- **Common privileges** granted to common users
- **Predefined common roles**
- Tablespaces and datafiles associated with each container:
 - `SYSTEM`, `SYSAUX`, and `UNDO`



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

After the CDB is created, there are new CDB components and objects such as:

- Two containers: The CDB root and the CDB seed (maximum number of PDBs: 4096)
- As many services as containers: The service name for the root container is the CDB name given at the CDB creation concatenated with the domain name. Each new PDB is assigned a service name: the service name is the PDB name given at PDB creation concatenated with domain name. If you create or plug a `PDBtest` PDB, its service name would be `PDBtest` concatenated with the domain name. You can find all service names maintained in a CDB in the `CDB_SERVICES` view. To connect to the CDB, you connect to the root, using either local OS authentication or the root service name. For example, if you set the `ORACLE_SID` to the CDB instance name and use `CONNECT / AS SYSDBA`, you are connected to the root under the common `SYS` user granted system privileges to manage and maintain all PDBs. To connect to a desired PDB, use either easyconnect or the `tnsnames.ora` file.
For example, `CONNECT username/password@net_service_name`.
- Common users: `SYS`, `SYSTEM`, created in all containers, the root, and the seed
- Common privileges granted to all users in all containers
- Predefined common roles in all containers, the root, and the seed
- Tablespaces: `SYSTEM`, `SYSAUX` associated with each container (maximum number of datafiles: 65534)

Data Dictionary Views: **DBA_xxx**

DBA_xxx All objects in the root or a pluggable database

ALL_xxx Objects accessible by the current user in a PDB

USER_xxx Objects owned by the current user in a PDB

DBA dictionary views providing information within PDB:

```
SQL> SELECT table_name FROM dict WHERE table_name like 'DBA%';
```

- **DBA_tablespaces**: All tablespaces of the PDB
- **DBA_data_files**: All datafiles of the PDB
- **DBA_tables**: All tables in the PDB
- **DBA_users**: All common and local users of the PDB

ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

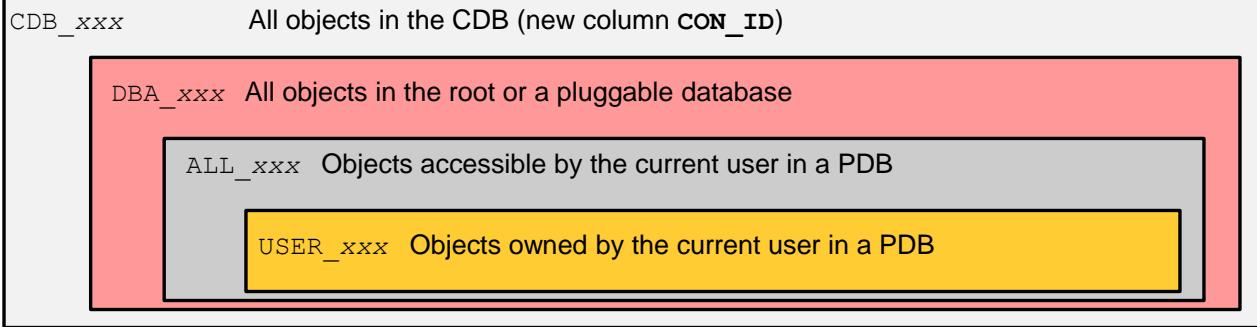
For backward-compatibility, DBA views show the same results in a PDB as in a non-CDB. For example, the **DBA_OBJECTS** view shows the objects that exist in the PDB from which you run the query.

- In the root, DBA_xxx views only show objects contained in the root.
- In a PDB, the DBA_xxx views only show objects contained in the PDB.

Examples of DBA views:

- While connected to the root, you query **DBA_USERS**. You get the list of common users created from the root (in the root, only common users exist).
- While connected to a PDB, you query **DBA_USERS**. You get the list of users, common and local, of the PDB.

Data Dictionary Views: CDB_xxx



CDB dictionary views provide information across PDBs:

```
SQL> SELECT view_name FROM dba_views WHERE view_name like 'CDB%';
```

- CDB_pdbs: All PDBs within the CDB
- CDB_tablespaces: All tablespaces within the CDB
- CDB_users: All users within the CDB (common and local)



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

In a CDB, for every DBA_* view, a CDB_* view is defined.

- In the CDB root, CDB views can be used to obtain information about tables, tablespaces, users, privileges, parameters, PDBs, and other types of objects contained in the CDB root and all open PDBs.
- In a PDB, the CDB_* views show objects visible through a corresponding DBA_* view only.

In addition to all the columns found in a given DBA_* view, the corresponding CDB_* view also contains the CON_ID column, which identifies a container whose data a given CDB_* row represents. In a non-CDB, the value of a CON_ID column is 0. In a CDB, the value can be either 1 used for rows containing data pertaining to the CDB root only or n where n is the applicable container ID.

Examples of CDB views:

- Connected to the CDB root and querying CDB_USERS, you get the list of users, common and local, of each container.
- Connected to a PDB and querying CDB_USERS or DBA_USERS, you get the same list of users, common and local, of the PDB.
- Connected to the CDB root and querying the CDB_PDBS view, you get the list of all PDBs. Querying the CDB_TABLESPACES view, you get the list of all tablespaces of all PDBs.

Data Dictionary Views: Examples

- Comparisons:

SQL> CONNECT / AS SYSDBA
1 SQL> SELECT role, common, con_id FROM cdb_roles;
2 SQL> SELECT role, common FROM dba_roles;

SQL> CONNECT sys@PDB1 AS SYSDBA
3 SQL> SELECT role, common, con_id FROM cdb_roles;
4 SQL> SELECT role, common FROM dba_roles;

- Access to V\$ views showing data from PDBs can be secured using privilege.

SQL> SELECT name,open_mode FROM v\$pdbs;

NAME	OPEN_MODE
PDB\$SEED	READ ONLY
PDB1	READ WRITE



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

In the slide, there are some examples to make comparisons between DBA_xxx and CDB_xxx views.

- In the first example, connected to the CDB root and querying CDB_ROLES, you get the list of roles, common and local, of each container. Note that the new column CON_ID displays the container the role belongs to.
- In the second example, querying DBA_ROLES, you get all common roles of the CDB root only (there cannot be any local roles in the CDB root).
- In the third example, connected to the PDB1 and querying CDB_ROLES, you get the list of roles, common and local, of the container you are connected to. The CON_ID displays the same value in all rows.
- In the fourth example, querying DBA_ROLES, you get the same list except that there is no CON_ID column. Because the CON_ID column in CDB_ROLES displays the same value in all rows, this value is not helpful.

The same backward-compatibility principle applies also to each of the familiar v\$views.

Data Dictionary Views: V\$xxx Views

SGA accessed by all containers: V\$ views and CON_ID column

```
SQL> SELECT distinct status, con_id FROM v$bh order by 2;
```

STATUS	CON_ID
cr	1
free	1
xcur	1
xcur	2
cr	3

1 → CDB root

2 → CDB seed

3 → PDB1 PDB

```
SQL> select OBJECT_ID, ORACLE_USERNAME, LOCKED_MODE, CON_ID from V$LOCKED_OBJECT;
```

OBJECT_ID	ORACLE_USERNAME	LOCKED_MODE	CON_ID
83711	SYS	3	3
83710	DOM	3	4

3 ← PDB1 PDB

4 ← PDB2 PDB



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

In the slide, there are some examples of V\$xxx views. The new column CON_ID in V\$xxx views display how the single SGA is accessed by any PDB within the CDB.

In the first example, the V\$BH view provides the list of distinct status of the block buffers currently in the buffer cache. The blocks are clearly identified in the CON_ID column, each block being accessed by a specific container. 1 stands for the CDB root, 2 for the CDB seed, and 3 for the PDB1 pluggable database.

In the second example, the V\$LOCKED_OBJECT view provides the list of locks currently held on objects in different PDBs. The locks are clearly identified in the CON_ID column as locked by a specific container. 3 stands for one PDB and 4 for another PDB.

After CDB Creation: To do List

After CDB creation, the CDBA has to:

- Create the SPFILE from the PFILE.
- Execute the `$ORACLE_HOME/rdbms/admin/utlrp.sql` script.
- Optionally plug non-CDBs and create new PDBs.
- Test startup/shutdown procedures.
- Automate PDBs opening.
- Create backup and recovery procedures.

After PDB creation, each PDBA in its own PDB has to:

- Set a default tablespace.
- Optionally create additional temporary tablespaces.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

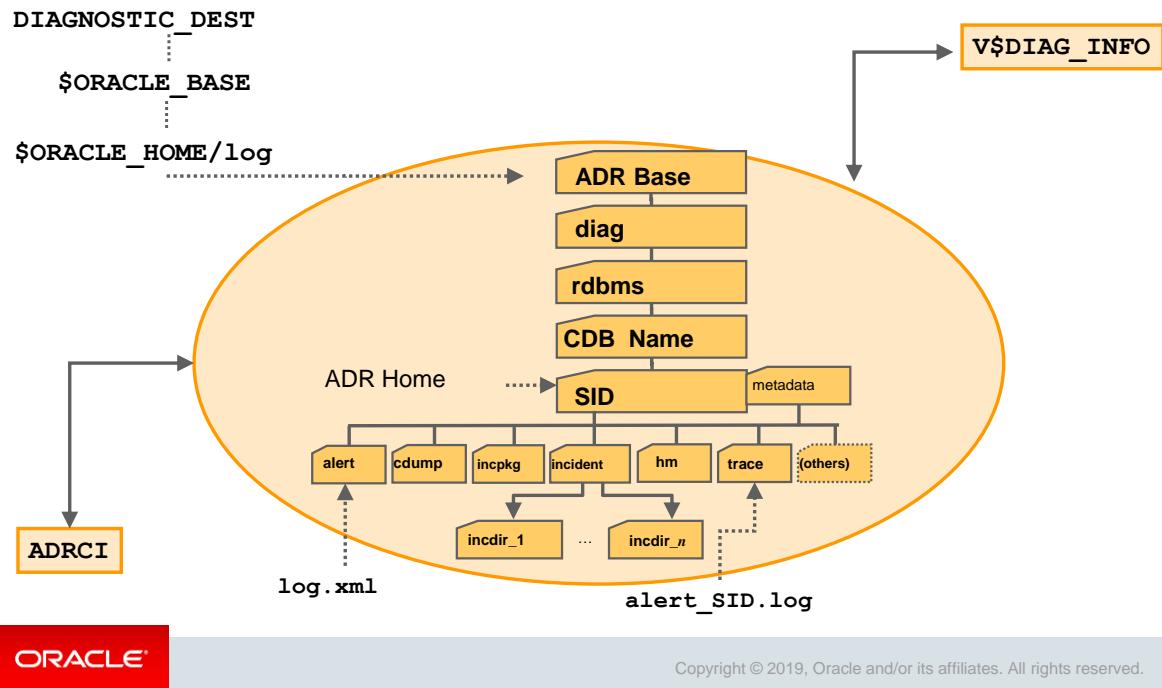
After the CDB is created, the container database administrator (CDBA) has to complete administrative tasks:

- Create the SPFILE from the PFILE.
- Execute the `$ORACLE_HOME/rdbms/admin/utlrp.sql` SQL script.
- Optionally plug non-CDBs if the initial plan was to consolidate several non-CDBs into a single one.
- Test startup and shutdown procedures.
- Create new event triggers to automate PDBs opening.
- Create backup and recovery procedures.

After possible PDB creation during the CDB creation, the pluggable database administrator (PDBA) has to complete administrative tasks in its own PDB:

- Set a default permanent tablespace.
- Create additional temporary tablespaces if specific amount of temporary space is required in the PDB.

Automatic Diagnostic Repository



All traces, incident dumps and packages, the alert log, Health Monitor reports, core dumps, and more files are stored in the Automatic Diagnostic Repository (ADR), a file-based repository for database diagnostic data. It has a unified directory structure across multiple instances and multiple products stored outside of any database. It is, therefore, available for problem diagnosis when the database is down. Nothing is changed with the arrival of container databases. Each instance of each product stores diagnostic data underneath its own ADR home directory. Each CDB, linked to a single instance, stores trace files in the same ADR home directory.

Its location is set by the `DIAGNOSTIC_DEST` initialization parameter. If this parameter is omitted or left null, the database sets `DIAGNOSTIC_DEST` upon startup as follows: if the environment variable `ORACLE_BASE` is set, `DIAGNOSTIC_DEST` is set to `$ORACLE_BASE`. If the environment variable `ORACLE_BASE` is not set, `DIAGNOSTIC_DEST` is set to `$ORACLE_HOME/log`.

Automatic Diagnostic Repository: `alert.log` File

The `alert_CDB1.log` shows new DDL statements.

```
CREATE DATABASE cdb1
...
ENABLE PLUGGABLE DATABASE
SEED FILE_NAME_CONVERT=('/u01/app/oradata/CDB1','/u01/app/oradata/seed');

CREATE PLUGGABLE DATABASE PDB$SEED AS CLONE USING ...
CREATE PLUGGABLE DATABASE pdb1 ... ;
ALTER PLUGGABLE DATABASE pdb1 UNPLUG INTO ... ;
ALTER PLUGGABLE DATABASE ALL OPEN ;
ALTER PLUGGABLE DATABASE pdb2 CLOSE IMMEDIATE ;
```



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The `alert.log` file of a CDB shows new DDL statements such as:

- CREATE PLUGGABLE DATABASE
- ALTER PLUGGABLE DATABASE
- DROP PLUGGABLE DATABASE

Provisioning New Pluggable Databases

- Create a new PDB from the CDB seed.
- Plug an unplugged PDB into the same CDB or into another CDB.
- Plug a non-CDB in a CDB.
- Clone a PDB from another PDB (local or remote CDB, hot or cold).
- Relocate a PDB from a CDB into another CDB.
- Proxy a PDB from another PDB.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

There are different methods to provision new PDBs in a CDB.

- Create a new PDB from the CDB seed, the `PDB$SEED`, for example for a new application implementation. This type of PDB creation is nearly instantaneous.
- Plug an unplugged PDB into another CDB or into the same CDB. For example, you have to upgrade a PDB to the latest Oracle version, but you do not want to apply it on all PDBs. Instead of upgrading a CDB from one release to another, you can unplug a PDB from one Oracle Database release and then plug it into a newly created CDB from a higher release. In case you unplugged a PDB inappropriately, you can still replug it into the same CDB.
- Plug non-CDBs in a CDB as PDBs, as part of the migration strategy. It is also a good way to consolidate several non-CDBs into a CDB.
- Clone a PDB from another PDB of the same CDB. For example, you want to test the application patch of your production. You first clone your production application in a cloned PDB and patch the cloned PDB to test.
- Relocate a PDB into another CDB so as to dispatch resources.
- Proxy a PDB. A proxy PDB provides fully functional access to another PDB in a remote CDB. This feature enables you to build location-transparent applications that can aggregate data from multiple sources that are in the same data center or distributed across data centers.

Tools

To provision new PDBs, you can use:

- SQL*Plus
- SQL Developer
- Enterprise Manager Cloud Control
- Enterprise Manager Database Express
- Database Configuration Assistant (DBCA)
 - Clone from CDB seed
 - Clone from an existing PDB
 - Plug an unplugged PDB



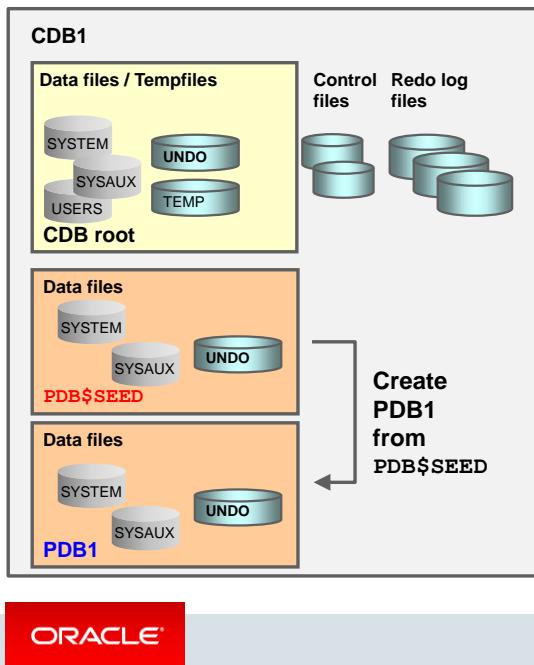
Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

There are different tools to provision new PDBs in a CDB.

- SQL*Plus
- SQL Developer
- Enterprise Manager Cloud Control
- Enterprise Manager Database Express

To create a new PDB from the CDB seed or from an existing PDB or by plugging an unplugged PDB method, you can also use Database Configuration Assistant (DBCA).

Create New PDB from PDB\$SEED



- Copies the datafiles from PDB\$SEED datafiles
- Creates tablespaces SYSTEM, SYSAUX, UNDO
- Creates a full catalog including metadata pointing to Oracle-supplied objects
- Creates common users:
 - SYS
 - SYSTEM
- Creates a local user (PDBA), granted local PDB_DBA role
- Creates a new default service

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The creation of a new PDB from the CDB seed is nearly instantaneous. The operation copies the datafiles from the READ ONLY seed PDB to the target directory defined in the CREATE PLUGGABLE DATABASE statement.

It creates tablespaces such as SYSTEM, to store a full catalog including metadata pointing to Oracle-supplied objects, SYSAUX for local auxiliary data, and UNDO for local undo segments.

It creates default schemas and common users that exist in the CDB seed, SYS who continues to have all superuser privileges, and SYSTEM who can administer the PDB.

It creates a local user (the PDBA), granted a local PDB_DBA role. Until the PDB SYS user grants privileges to the local PDB_DBA role, the new PDBA cannot perform any other operation than connecting to the PDB.

A new default service is also created for the PDB.

Steps: With FILE_NAME_CONVERT

Create a new PDB from the seed using **FILE_NAME_CONVERT**:

1. Connect to the CDB root as a common user with the CREATE PLUGGABLE DATABASE system privilege:

```
SQL> CREATE PLUGGABLE DATABASE pdb1
ADMIN USER admin1 IDENTIFIED BY p1 ROLES=(CONNECT)
FILE_NAME_CONVERT = ('PDB$SEEDdir', 'PDB1dir');
```

2. Use views to verify:

```
SQL> CONNECT / AS SYSDBA
SQL> SELECT * FROM cdb_pdbs;
SQL> SELECT * FROM cdb tablespaces;
SQL> SELECT * FROM cdb data_files;
SQL> ALTER PLUGGABLE DATABASE pdb1 OPEN RESTRICTED;
SQL> CONNECT sys@pdb1 AS SYSDBA
SQL> CONNECT admin1@pdb1
```

Note: The STATUS of the PDB is NEW.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The steps to create a new PDB from the CDB seed are the following:

If you do not use OMF (Oracle Managed Files):

- Connect to the CDB root as a common user with the CREATE PLUGGABLE DATABASE system privilege and execute the CREATE PLUGGABLE DATABASE statement as shown in the slide. The ADMIN USER clause defines the PDBA user created in the new PDB with the CONNECT and PDB_DBA roles (empty role). The clause FILE_NAME_CONVERT designates first the source directory of the CDB seed datafiles and second the destination directory for the new PDB datafiles.
- When the statement completes, use views to verify that the PDB is correctly created.
 - The CDB_PDBS view displays the list of the PDBs and the CDB_TABLESPACES view displays the list of the tablespaces of the new PDB (SYSTEM, SYSAUX, UNDO).
 - Still connected to the CDB root, open the PDB. Then try to connect to the new PDB under common user, SYS who always exists in any PDB, or the user defined in the ADMIN USER clause, admin1.

The CDB_PDBS view shows the STATUS of the new PDB: it is NEW. The PDB has never been opened. It must be opened in READ WRITE or RESTRICTED mode for Oracle to perform processing that is needed to complete the integration of the PDB into the CDB and mark it NORMAL. An error will be thrown if an attempt is made to open the PDB read only.

Steps: Without FILE_NAME_CONVERT

Create a new PDB from seed without **FILE_NAME_CONVERT**:

- Use OMF: **DB_CREATE_FILE_DEST** = '/u01/app/oradata/CDB1/**pdb1**'
- Or
- Use the instance parameter: **PDB_FILE_NAME_CONVERT** = '/u01/app/oradata/CDB1/seed', '/u01/app/oradata/CDB1/**pdb1**'

```
SQL> CREATE PLUGGABLE DATABASE pdb1
ADMIN USER pdb1_admin IDENTIFIED BY p1 ROLES=(CONNECT);
```

Or

- Use the clause in the CREATE PLUGGABLE DATABASE command:
- ```
CREATE_FILE_DEST = '/u01/app/oradata/CDB1/pdb1'
```



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

If you use OMF or **PDB\_FILE\_NAME\_CONVERT**, then first connect to the CDB root as SYS.

- With OMF, set, in `init.ora`, the `DB_CREATE_FILE_DEST` instance parameter to a target directory for the datafiles of the new PDB.
- Without OMF, set the `PDB_FILE_NAME_CONVERT` new instance parameter to both the source directory of the CDB seed datafiles and the target directory for the new PDB datafiles.

The `/u01/app/oradata/CDB1/pdb1` directory must exist.

Then use the `cdb_pdbs` view to verify that the new PDB and its tablespaces exist:

```
SQL> SELECT * FROM cdb_pdbs;
SQL> SELECT * FROM cdb_tablespaces;
SQL> SELECT * FROM cdb_data_files;
```

## Summary

In this lesson, you should have learned how to:

- Configure and create a CDB
- Create a new PDB from the CDB seed
- Explore the instance
- Explore the structure of PDBs
- Explore the ADR



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Practice 2: Overview

- 2-1: Exploring CDB architecture and structures
- 2-2: Creating a new CDB
- 2-3: Creating a new PDB



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

3

# Application PDBs and Application Installation



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

# Objectives

After completing this lesson, you should be able to:

- Describe application containers in CDBs
- Explain the purpose of application root and application seed
- Define application PDBs
- Create application PDBs
- Explain application installation on top of application containers
- Install an application
- Upgrade and patch applications
- Describe the commonality concept in application contexts
- Use a dynamic container map
- Describe enhancements in various areas



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

For a complete understanding about application PDBs, refer to “Administering Application Containers” in *Oracle Multitenant Administrator’s Guide 18c*.

Refer to other sources of information available under YouTube like:

[Stale Standalone to Superb SaaS in a Short Series](#)

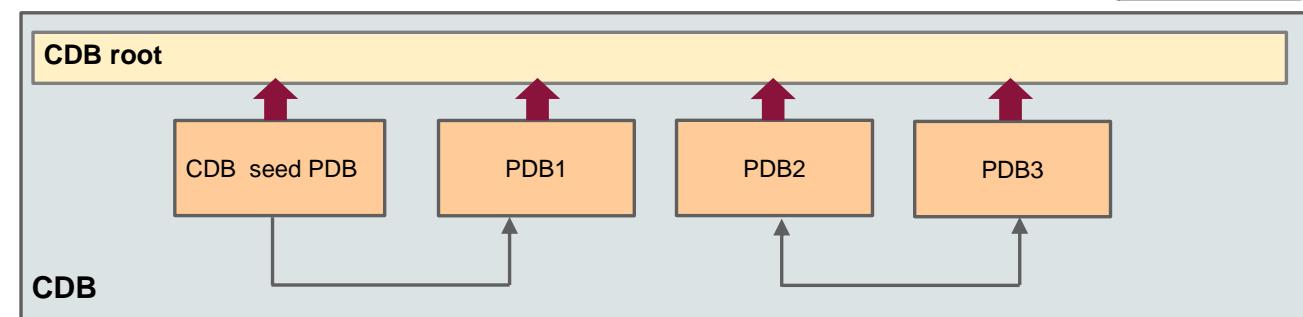
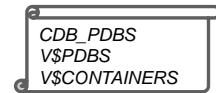
Refer to other sources of information available under Oracle Learning Library:

- *Oracle By Example (OBE): Learning Path: 18c New Features for Multitenant*
  - [using\\_dynamic\\_container\\_map](#)

Refer to MOS note: *Oracle Multitenant Option - 12c: Frequently Asked Questions (Doc ID 1511619.1)*

## Regular PDBs

- A regular PDB is a PDB within a CDB, storing data in objects independently of other PDBs.
- A regular PDB can be created from the CDB seed or from another PDB (*cloning* or *unplugging/plugging*).



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

A regular PDB stores data in objects independently of other PDBs. It can be created from the CDB seed or from another PDB (by *cloning* or *unplugging/plugging* methods).

## PDBs and Applications

Applications in regular PDBs need to be upgraded or patched in the same CDB or across many CDBs.



The upgrade script has to be executed in all regular PDBs individually.



No single master definition of application



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

What happens when an application needs to be upgraded or patched in the same CDB or across many CDBs?

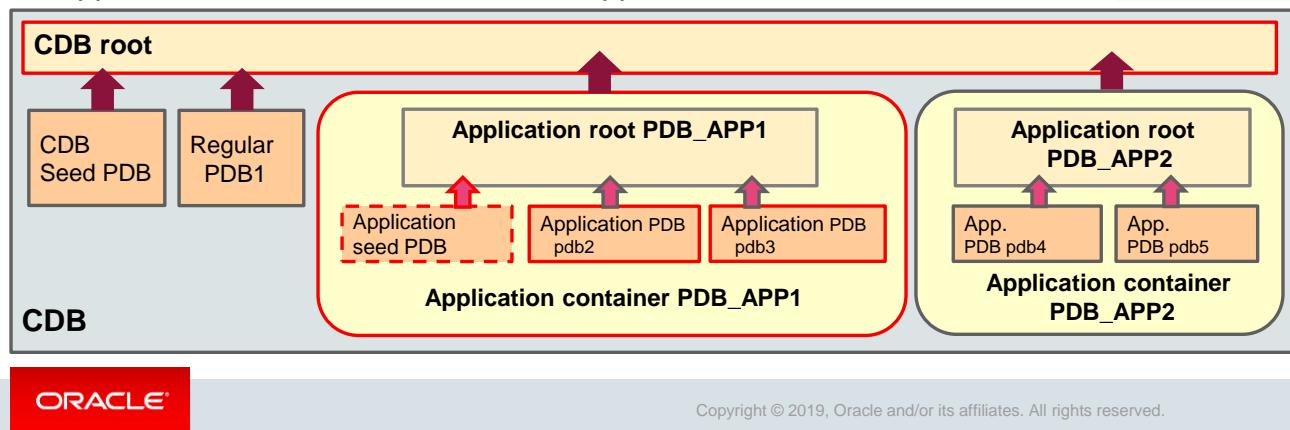
- There is no single master definition of an application on top of regular PDBs.
- The upgrade script has to be executed in all regular PDBs individually. This is time-consuming.

# Application Containers

An application container is a collection of PDBs grouped together within a CDB to store data for an application.

- The application root
- An optional application seed
- Application PDBs associated with the application root

New columns  
CDB\_PDBS  
V\$PDBS  
V\$CONTAINERS



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Application Container

- The concept of application containers offers the capability to have a single master application definition. The master application definition is created in the application root of the application container.

## Application Root

- An application root is a hybrid between the CDB root and a PDB in that it belongs to the CDB root and shares descriptions of Oracle-supplied common objects, while at the same time it allows the creation of application common objects, which are shared only by application PDBs that belong to the application root. Such objects are not visible to the CDB root, other application roots, or PDBs that do not belong to the application root.

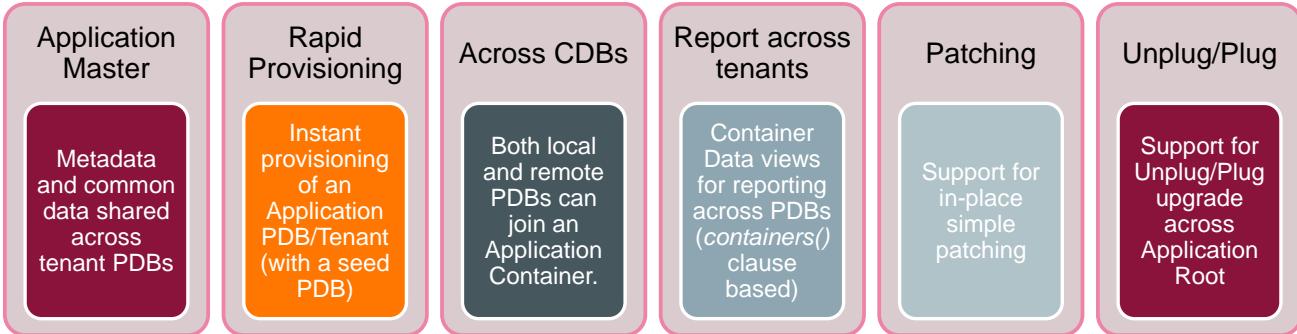
## Application Seed PDB

- The application seed is optional. After an application PDB is created, all the statements that are used for application installation, patch, or upgrade must be re-applied in the application PDB by synchronization. This can be a time-consuming process.
- An application seed, which is tied to only one application root, can be created for instantaneous provisioning of application PDBs. Synchronization of the application code in the application seed must be completed before application PDBs creation. Any application PDB that is created is a full clone of the application seed.

## Application PDB

- An application PDB can belong to only one application root. An application root container enables the creation of common objects, users, roles, and profiles, as well as the granting of privileges and roles commonly, which means that they apply only within the application root and the application PDBs that are associated with it.
- Changes made in the application root require synchronization across all application PDBs associated with the application root.

# Application Containers: Other Features



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Application Master

- Application common objects exist only in the application root of an application container.
- Metadata or data is shared only by the application PDBs associated with the application root in the application container.
- Local objects can be created in application PDBs outside of the application definition if needed.

## Rapid Provisioning

You can perform instant provisioning of an application PDB by using different methods:

- With a seed PDB defined in the application container
- By unplugging or plugging or cloning local and remote PDBs to join an application container

## Reporting

Queries can be executed across the application PDBs within a CDB and also across CDBs.

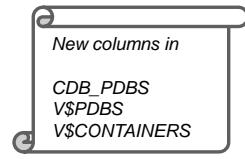
## Patching

Application containers support in-place simple application patching.

## Unplugging/Plugging

Unplug/Plug upgrade is supported across the application root.

# Types of Containers



- The **CDB root container** (`CDB$ROOT`)
  - The first **mandatory** container created at CDB creation
  - Oracle system-supplied common objects and metadata
  - Oracle system-supplied common users and roles
- **Pluggable database containers** (PDBs)
  - The CDB seed (`PDB$SEED`)
    - The second **mandatory** container created at CDB creation
    - Oracle system-supplied common entities for new PDBs
  - Regular PDBs
  - Application containers
    - Application root PDB
    - Optional application seed PDB (`application_container_root_name$SEED`)
    - Application PDBs



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

To summarize, a CDB is an Oracle database that contains the CDB root, the CDB seed, and optionally several PDBs. PDBs can be regular or federated together within application containers, with each application container including the application root PDB and an optional application seed.

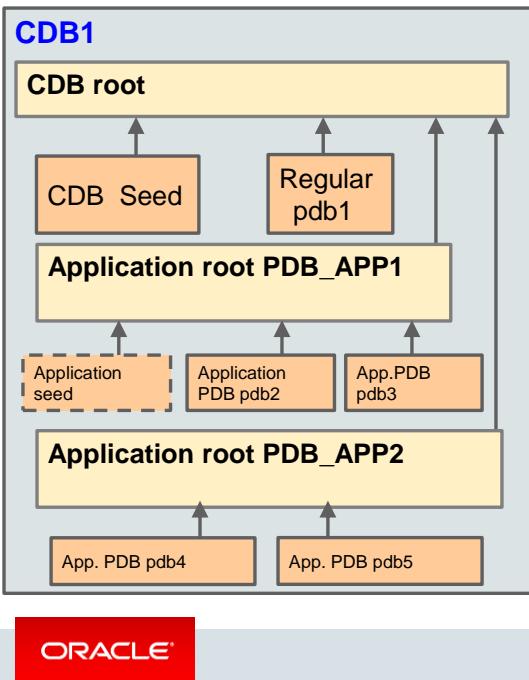
In a CDB, there is only one CDB root and one CDB seed. The CDB seed is a system-supplied template that is used to create new PDBs.

There is always one application root PDB for each application container that has been created. Each application container may have an application seed PDB as well.

A CDB can contain up to 4096 PDBs, including the CDB seed, the services being limited to 10000. The new `MAX_PDBS` initialization parameter specifies a limit on the number of PDBs that can be created in a CDB or in an application root. Only user-created PDBs are counted. `PDB$SEED`, application seeds, and application root clones are ignored.

The `V$CONTAINERS` view displays all containers, including the CDB root, the CDB seed, regular PDBs, application roots, application seeds, and application PDBs.

## Creating Application PDBs



1. Connect to the **CDB1** CDB root.
2. Create the **PDB\_APP1** PDB as the application root.
 

```
SQL> CONNECT / AS SYSDBA
SQL> CREATE PLUGGABLE DATABASE pdb_app
 AS APPLICATION CONTAINER ...;
```
3. Connect to the **PDB\_APP1** application root.
4. Install the application.
5. Optionally, create the application seed for the application PDBs in the application root.
6. Create the **PDB2** PDB as an application PDB within the **PDB\_APP1** application root.
7. Create other application PDBs if required.
8. Synchronize all application PDBs with the application installed if step 5 was not completed.

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.



The **CREATE PLUGGABLE DATABASE** statement uses the new **AS APPLICATION CONTAINER** clause to tag a newly created PDB as an application root.

You can then connect to the application root to create the application seed if required and application PDBs.

The **CREATE PLUGGABLE DATABASE** statement uses the new **AS SEED** clause to tag a newly created PDB as an application seed. The application seed of an application root container is used as a template for newly created PDBs within the application root container. It is therefore logical to install the application objects (common users, common tables, common granted privileges and roles) in the application root before creating the application seed so that the application seed inherits the common objects of the application root.

An application seed, other than the CDB seed, can be opened in read-write mode so that updates to the application root can be propagated to it and, therefore, on newly created application PDBs.

Use the **CREATE PLUGGABLE DATABASE** statement to create application PDBs.

If the application PDBs are created by using the application seed, it is not necessary to ask for synchronization with the application root. On the contrary, if the application PDBs are created from scratch, it is necessary to ask for synchronization with the application root.

An application root cannot be unplugged if any application PDB belongs to it. Unplugging an application root with its application PDBs requires two steps:

- First unplug all the application PDBs that belong to the application root.
- Then unplug the application root.

The type of PDB is displayed in new columns in the **V\$CONTAINERS**, **V\$PDBS**, and **CDB\_PDBS** views such as **application\_root**, **application\_seed**, **application\_pdb**, and **application\_root\_con\_id**.

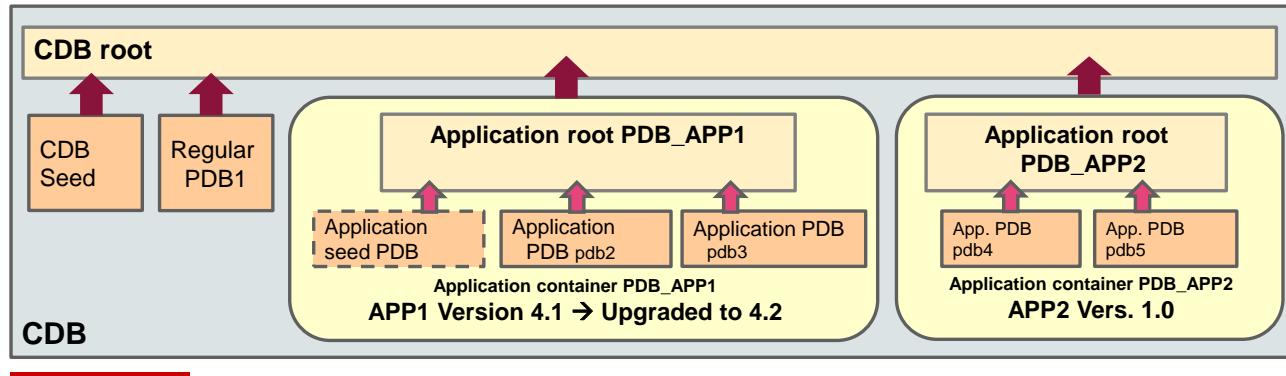
# Application Name and Version

An application container can be tagged with:

- An application name
- An application version

*DBA\_APPLICATIONS  
DBA\_APP\_VERSIONS  
DBA\_APP\_PATCHES  
DBA\_APP\_ERRORS  
DBA\_APP\_STATEMENTS*

An application can be patched, upgraded, or uninstalled.



**ORACLE®**

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Application Installation

If an application root and application PDBs store data for the same application, they can be tagged with an Application Name and Application Version.

The application is installed in the application root by using customer-supplied scripts. The application installation boundary needs to be indicated by using `ALTER PLUGGABLE DATABASE APPLICATION <app_name> BEGIN INSTALL 'n'` and `ALTER PLUGGABLE DATABASE APPLICATION <app_name> END INSTALL 'n'`. This value is the application version that is used for future upgrade and patching operations.

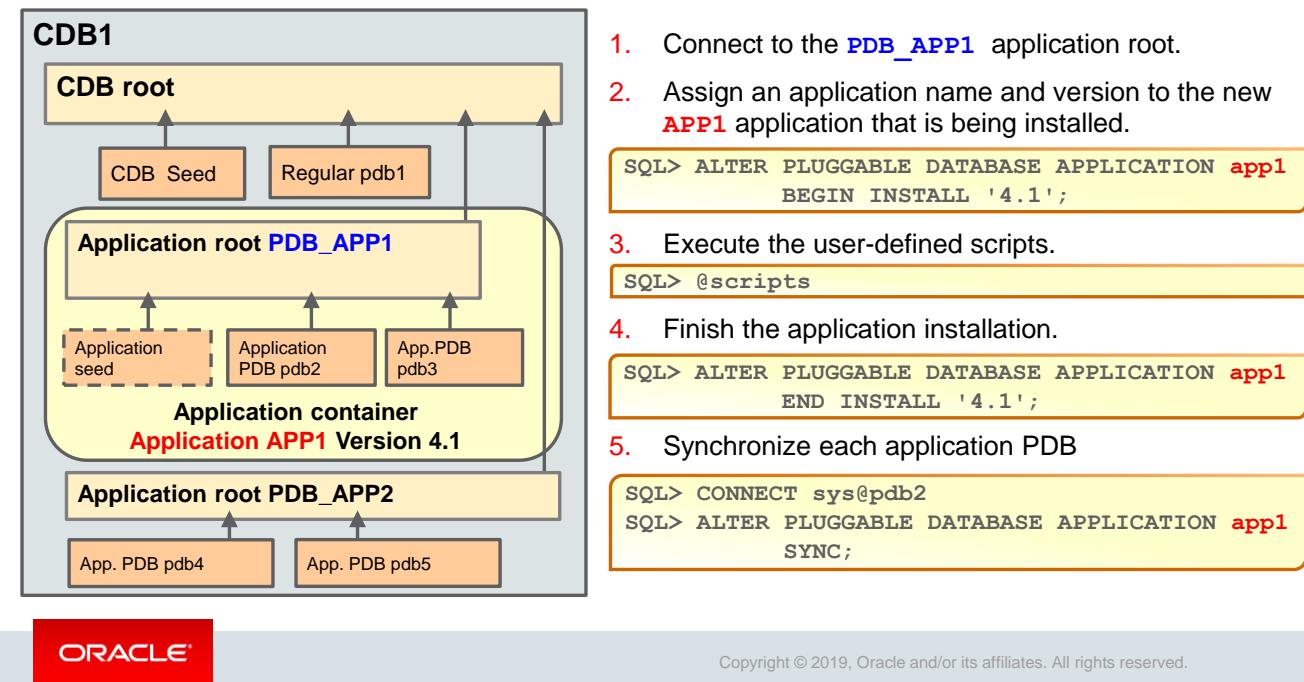
After the application installation is completed in the application root, the application needs to be propagated to the application PDBs. The application PDBs need to be synchronized with the application root unless the application PDB has inherited common objects from the application seed if it exists and is up-to-date.

Different applications can be installed on top of an application container that is tagged with distinct application names and application versions.

## Application Patches and Upgrades

When application patches or upgrades are performed, they are executed in the application root, based on the current application version. These operations also require start and end boundaries and then propagation to the application PDBs.

# Installing Applications



1. Connect to the **PDB\_APP1** application root.
2. Assign an application name and version to the new **APP1** application that is being installed.

```
SQL> ALTER PLUGGABLE DATABASE APPLICATION app1
 BEGIN INSTALL '4.1';
```

3. Execute the user-defined scripts.

```
SQL> @scripts
```

4. Finish the application installation.

```
SQL> ALTER PLUGGABLE DATABASE APPLICATION app1
 END INSTALL '4.1';
```

5. Synchronize each application PDB

```
SQL> CONNECT sys@pdb2
SQL> ALTER PLUGGABLE DATABASE APPLICATION app1
 SYNC;
```



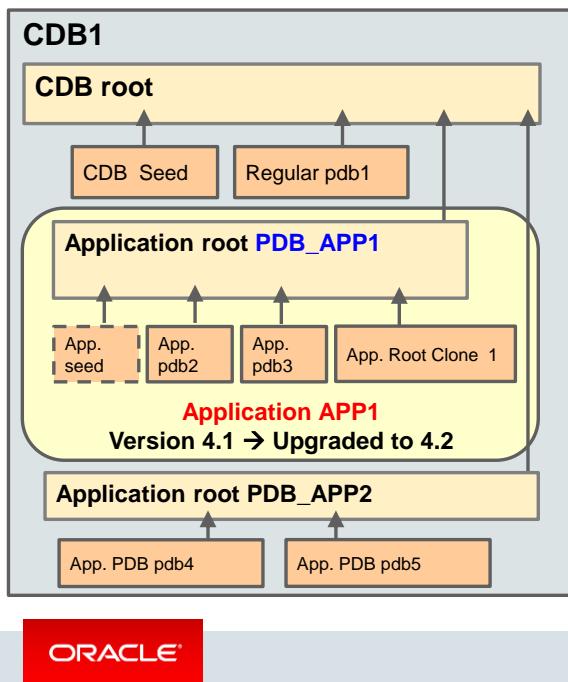
Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Application Installation

To implement an application on application PDBs within the same application container, perform the following steps:

1. Connect to the application root to start installing an application.
2. Tag the application with a name and version. This value is the application version that is used for future upgrade and patching operations. Before running the customer-supplied scripts, indicate installation start by using `ALTER PLUGGABLE DATABASE APPLICATION <app_name> BEGIN INSTALL 'n'`.
3. Run the customer-defined scripts of the application installation. The scripts include the users, roles, and objects creation common to all application PDBs within the application container, as well as the privileges and roles that are commonly granted to the common users of the application PDBs.
4. Indicate installation end by using `ALTER PLUGGABLE DATABASE APPLICATION <app_name> END INSTALL 'n'`. Until the end of the operation is declared, the application is still under the **INSTALLING** status.
5. Check that the application installation is complete and successful by displaying the `DBA_APPLICATIONS` view.
6. Connect to each application PDB to synchronize it with the application root.

# Patching and Upgrading Applications



1. Connect to the **PDB\_APP1** application root of the **APP1** application.
2. Check the current version of the **APP1** application before starting the upgrade.
3. Start the application upgrade to a higher version.

```
SQL> ALTER PLUGGABLE DATABASE APPLICATION app1
 BEGIN UPGRADE '4.1' TO '4.2';
```

4. Complete the application upgrade.

```
SQL> @scripts
SQL> ALTER PLUGGABLE DATABASE APPLICATION app1
 END UPGRADE TO '4.2';
```

5. Synchronize each application PDB.

```
SQL> CONNECT sys@pdb2
SQL> ALTER PLUGGABLE DATABASE APPLICATION app1
 SYNC;
```

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Application Upgrade

Applying a patch on an application or upgrading an application requires that you know the current application version, which is retrievable from the **DBA\_APPLICATIONS** view.

Major changes to an application constitute application upgrades.

Use the **ALTER PLUGGABLE DATABASE APPLICATION <app\_name> BEGIN UPGRADE '4.1' TO '4.2'** and **ALTER PLUGGABLE DATABASE APPLICATION <app\_name> END UPGRADE TO '4.2'** statements to indicate the start and end boundaries of the operation, respectively.

Until the end of the operation is declared, the application is still under the **UPGRADING** status.

At the beginning of any upgrade operation, a new application root is automatically created, which is an application root clone. It is primarily meant for metadata lookup.

## Application Patching

Minor changes to an application constitute application patches. Examples of minor changes can include bug fixes and security patches. You can patch an application in an application container.

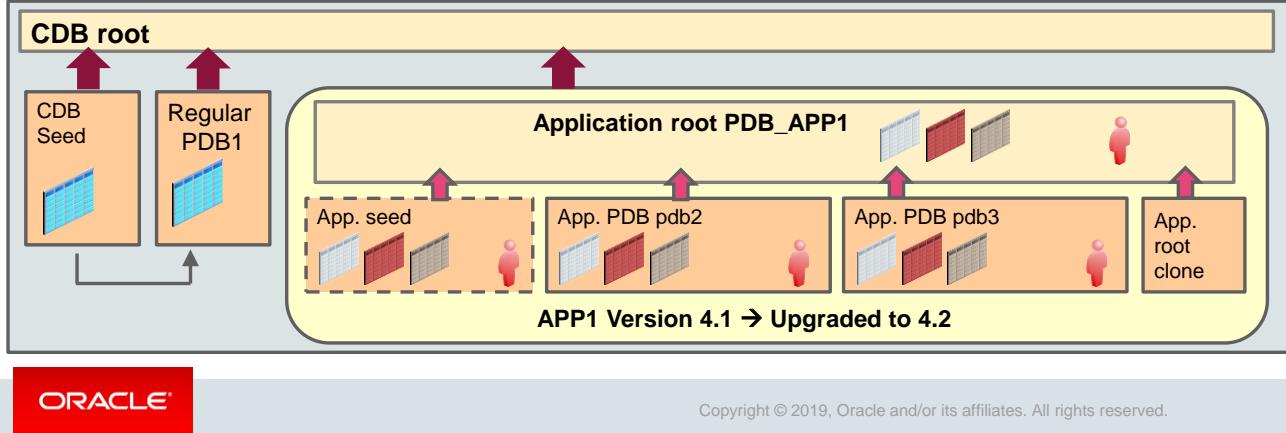
Use the **ALTER PLUGGABLE DATABASE APPLICATION <app\_name> BEGIN PATCH nnn MINIMUM VERSION 'n'** and **ALTER PLUGGABLE DATABASE APPLICATION <app\_name> END PATCH nnn** statements to indicate the start and end boundaries of the operation, respectively.

The value of **MINIMUM VERSION** indicates the minimum application version at which an application installation should be before the patch can be applied to it—the patch cannot be applied to an application installation that is at a lower application version than the given minimum application version.

Until the end of the operation is declared, the application is still under the **PATCHING** status.

# Application Common Objects

- The application root holds the common objects:
  - Users, roles, granted privileges, profiles, tables, views, and so on
- Synchronization of application PDBs with the application root is required.
- If an application is patched or upgraded, resynchronization of application PDBs is required.



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

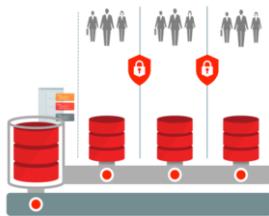
## Application

An application includes common objects that are sharable by all application PDBs in the application container such as common users, common roles, privileges granted commonly, common profiles, and common tables. Commonality within an application container limits the scope of the objects to the application root and the application PDBs associated with the application root. Therefore, the common objects in one application container are not visible to the CDB root and other application roots.

# Use Cases for Application Containers

## Pure SaaS

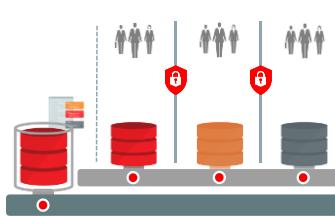
- Each customer's data resides in an individual PDB.
- All PDB-level operations are applicable on individual customer data.
- Customer data can be securely managed.
- Thousands of tenants can be handled.



ORACLE®

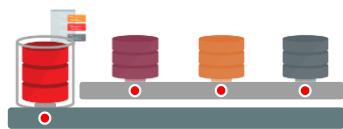
## Hybrid SaaS

- Large customers reside in individual PDBs.
- Smaller customers share a PDB.
- It is suitable for applications with a high density of customers.
- Similar types of customers can be grouped in a PDB.
- Hundreds of thousands of tenants can be handled.



## Logical DW

- Customers may address data sovereignty issues: *Country or region data will be segregated into a separate PDB.*
- There is efficient execution of ETLs for every region without impacting each other.
- The best execution plans are based on actual data distribution.



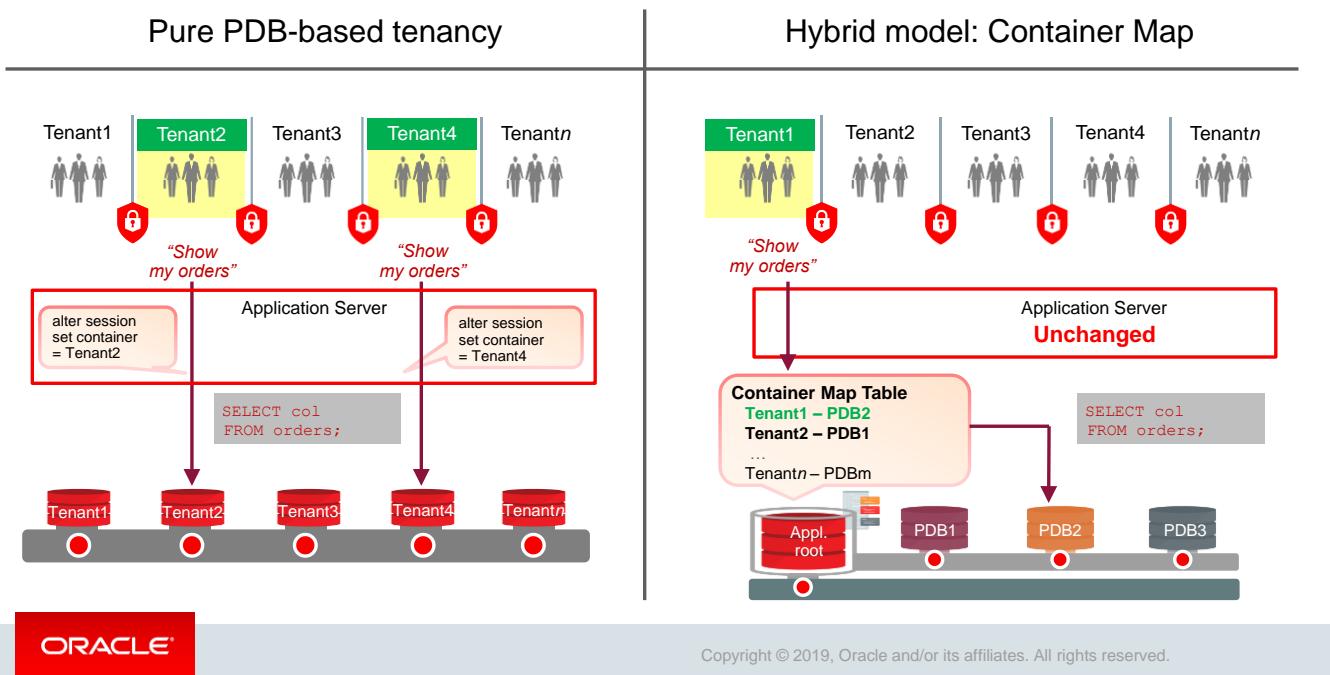
Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

In a pure PDB-based tenancy, each customer's data resides in an individual PDB. All pluggable database-level operations, such as unplug, plug, and clone, are applicable on the individual customer data. Customer data is securely managed, and thousands of tenants can be managed in this case.

In a hybrid model, large customers may reside in individual PDBs and smaller customers may share a PDB. This model is suitable for applications with a high density of customers. In this case, similar types of customers are grouped in a PDB, and hundreds of thousands of tenants can be managed.

In a logical Data Warehouse, customers may address data sovereignty issues such as country or region data that must be segregated into a separate PDB. This requires efficient execution of ETLs for every region without any impact on one another. A type of implementation is to create dimension tables in the application root and the fact tables in the application PDBs.

## Use Case: Pure PDB-Based Versus Hybrid Model



In a pure PDB-based tenancy, a customer storing its data in a specific PDB must connect to that PDB to query its own data.

```
ALTER SESSION SET CONTAINER = 'TENANT2';
```

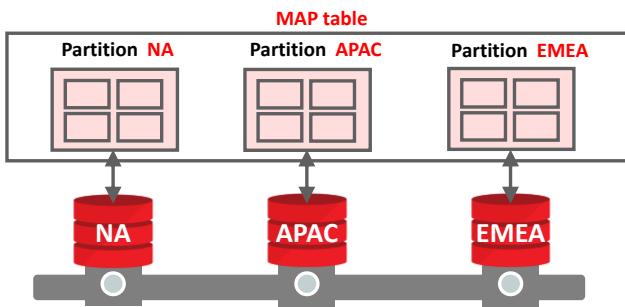
In this case, the query `select <columns> from orders` would be converted to:

```
SELECT <columns> FROM orders WHERE con_id = 4;
```

In a hybrid model, customers that share PDBs to store their data store the data in application containers and, therefore, do not have to connect to a specific PDB to query their data. The application request can be automatically routed to the appropriate application PDB within the application container via a container map table that provides data repartitioning in application PDBs.

## Container Map

- Define a PDB-based partition strategy based on the values stored in a column.
- Select a column that is commonly used and never updated.
  - Time Identifier (versus creation\_date) / Region Name
- Set the database property `CONTAINER_MAP` in the application root.



Each PDB corresponds to data for a particular partition.

```
DATABASE_PROPERTIES
PROPERTY_NAME = CONTAINER_MAP
PROPERTY_VALUE = app.tabapp
DESCRIPTION = value of container mapping table
```

ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The use of the clause `CONTAINERS (table or view)` in a query in the CDB root accesses a table or view in the CDB root and in each of the opened PDBs and returns a `UNION ALL` of the rows from the table or view. This concept is extended to work in an application container. `CONTAINERS (table or view)` queried in an application root accesses the table or view in the application root and in each of the opened application PDBs of the application container. `CONTAINERS (table or view)` can be restricted to access a subset of PDBs by using a predicate on `CON_ID`. `CON_ID` is an implicitly generated column of `CONTAINERS (table or view)`.

```
SELECT fname, lname FROM CONTAINERS(emp) WHERE con_id IN (44,56,79);
```

One drawback of `CONTAINERS()` is that queries need to be changed to add a `WHERE` clause on `CON_ID` if only certain PDBs should be accessed. Often, rows of tables or views are horizontally partitioned across PDBs based on a user-defined column.

The `CONTAINER_MAP` database property provides a declarative way to indicate how rows in metadata-linked tables or views are partitioned across PDBs.

The `CONTAINER_MAP` database property is set in the application root. Its value is the name of a partitioned table (the map object). The names of the partitions of the map object match the names of the PDBs in the application container. The columns that are used in partitioning the map object should match the columns in the metadata-linked object that is being queried. The partitioning schemes that are supported for a `CONTAINER_MAP` map object are `LIST`, `HASH`, and `RANGE`.

**Note:** Container maps can be created in CDB root, but the best practice is to create them in application roots.

## Container Map: Example

```

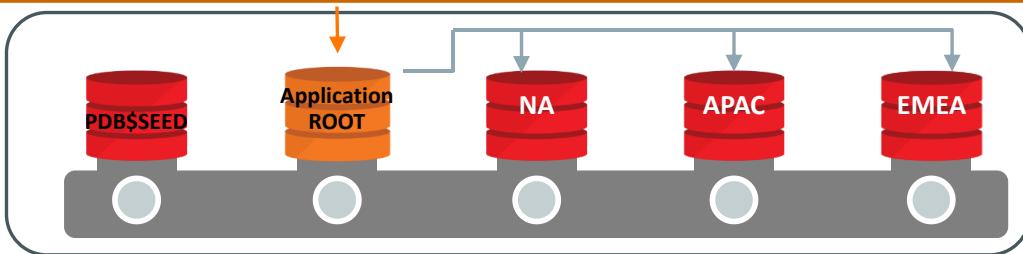
CREATE TABLE tab1 (region ..., ...);
CREATE TABLE tab2 (... , region ...);

CREATE TABLE app1.app_map (columns ... , region VARCHAR2(20))
PARTITION BY LIST (region)
(PARTITION NA VALUES ('AMERICA', 'MEXICO', 'CANADA'),
 PARTITION EMEA VALUES ('UK', 'FRANCE', 'GERMANY'),
 PARTITION APAC VALUES ('INDIA', 'CHINA', 'JAPAN'));

ALTER PLUGGABLE DATABASE SET CONTAINER_MAP = 'app1.app_map';
ALTER TABLE tab1 ENABLE container_map;

```

DBA\_TABLES  
CONTAINER\_MAP\_OBJECT = YES



ORACLE®

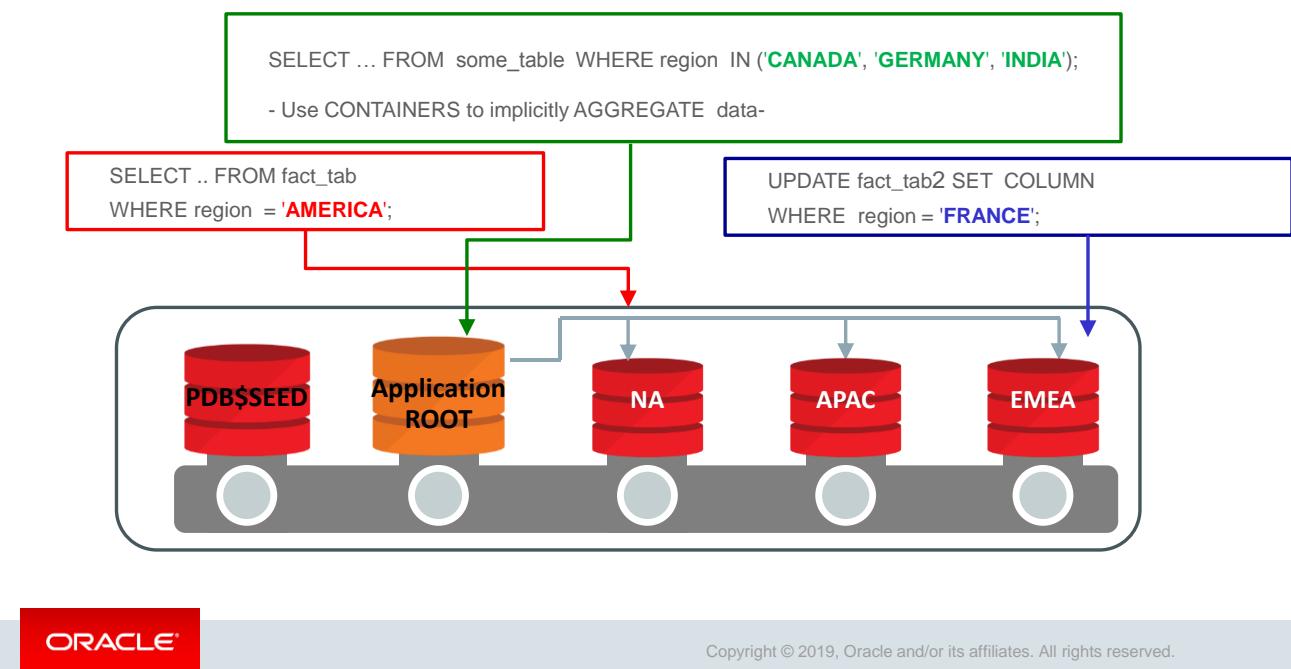
Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

In a hybrid model, you can create common partitioned tables in the application root, mapping a partition of the table to an application PDB of the application container where, for example, the `TENANT_GRP1` partition would store data for customers of group1 in the `Tenant_GRP1` application PDB and where the `TENANT_GRP2` partition would store data for customers of group2 in the `Tenant_GRP2` application PDB.

In a Data Warehouse model, you can create common partitioned tables in the application root, which are partitioned on a column such as `REGION` in our example, where data is segregated into separate application PDBs of the application container.

In the example in the slide, the `NA` partition stores data for `AMERICA`, `MEXICO`, and `CANADA` as defined in the list, in the `NA` application PDB. The `EMEA` partition stores data for `UK`, `FRANCE`, and `GERMANY` as defined in the list, in the `EMEA` application PDB.

## Query Routed Appropriately

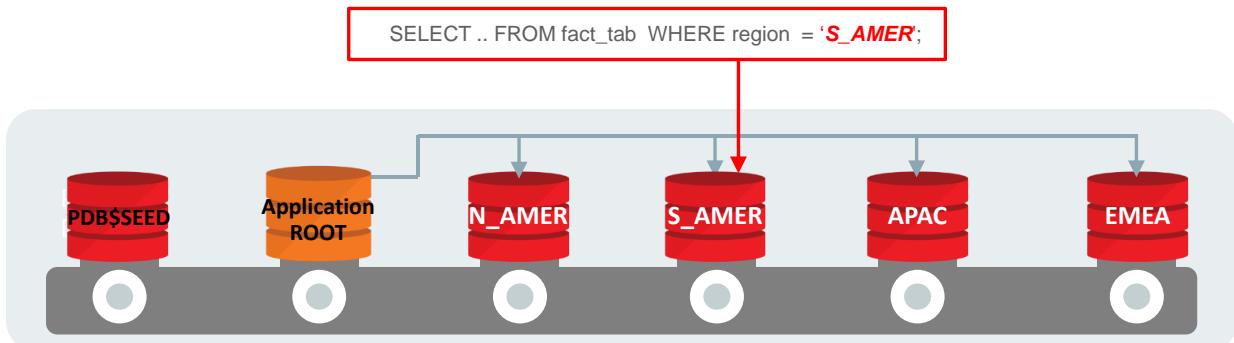


Because data is segregated into separate application PDBs of the application container, querying a container map table, for example the data for AMERICA, automatically retrieves data from the NA application PDB. The query is appropriately routed to the relevant partition and therefore to the relevant application PDB.

If you need to retrieve data from a table that is spread over several application PDBs within an application container, use the CONTAINERS clause to aggregate rows from partitions from several application PDBs.

## Dynamic Container Map

```
CREATE PLUGGABLE DATABASE s_amer ...
CONTAINER_MAP UPDATE (ADD PARTITION s_amer VALUES ('PERU','ARGENTINA'));
```



```
CREATE PLUGGABLE DATABASE s_amer_peru ...
CONTAINER_MAP UPDATE (SPLIT PARTITION s_amer
INTO (partition s_amer ('ARGENTINA'), partition s_amer_peru));
```

ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

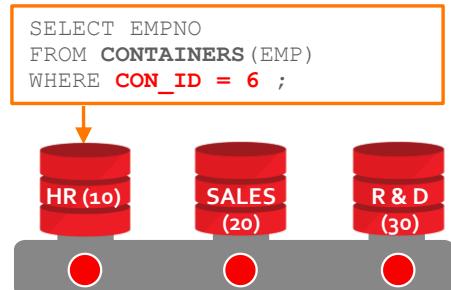
In Oracle Database 18c, when a PDB is created, dropped, or renamed, CONTAINER\_MAP defined in CDB root or application root or both can be dynamically updated to reflect the change.

The CREATE PLUGGABLE DATABASE statement takes an optional clause that describes the key values:

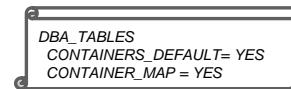
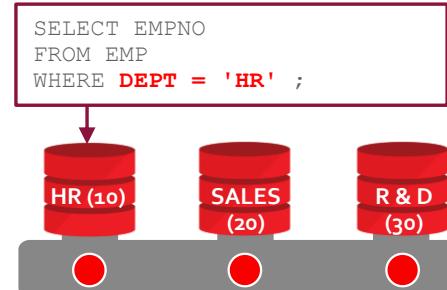
- Affiliated with the new PDB: CONTAINER\_MAP UPDATE (ADD PARTITION ...)
- Affiliated with the PDBs generated by the split operation from the original PDB: CONTAINER\_MAP UPDATE (SPLIT PARTITION ... INTO (PARTITION ... , PARTITION ...))

# Container Map and Containers Default

`CONTAINERS_DEFAULT` allows you to wrap the `CONTAINERS()` clause around any table.



`CONTAINER_MAP`, when used in conjunction with `CONTAINERS_DEFAULT`, prunes the partitions (PDBs) based on the key passed to the query.



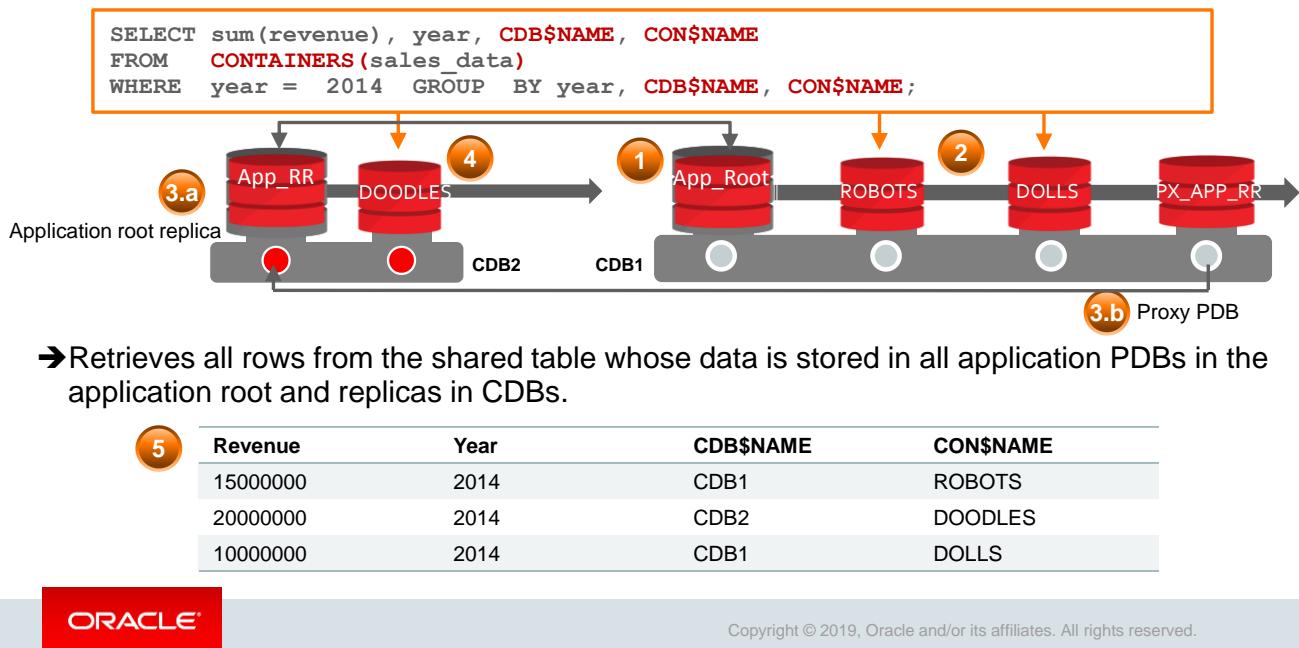
Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Metadata-linked objects in application containers are not automatically enabled for `CONTAINERS()`. This can be turned off or on by resetting and setting the `CONTAINERS_DEFAULT` attribute on the table.

`CONTAINER_MAP` and `CONTAINERS_DEFAULT`, when used together, allow `CONTAINER_MAP` to prune the partitions and therefore the application PDBs, based on the key that is passed to the query.

It is not mandatory to use `CONTAINERS_DEFAULT` with `CONTAINER_MAP`.

## Query Across CDBs Using Application Root Replica



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The `CONTAINERS()` clause in a query adds implicit columns, `CON_ID`, `CON$NAME`, and `CDB$NAME`. These are hidden columns and thus have to be explicitly referenced if their values are to be displayed. These columns are particularly useful when `CONTAINERS()` is used in an application root that has a proxy PDB, which allows SQL statements execution in a remote PDB as if it were a local PDB in the CDB.

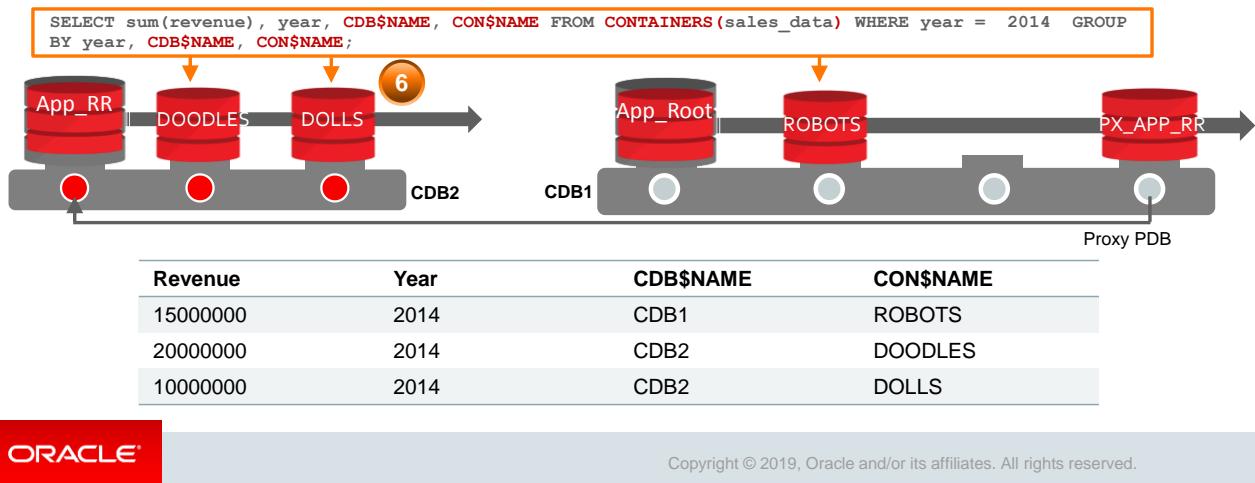
1. In `cdb1`, create the `app_root` application root container and install the application.
2. Create the `robots` and `dolls` application PDBs in `app_root` and synchronize them with the application that is installed in the application root.
3. In `cdb2`, create an application root replica of `app_root`. An application root replica is an exact clone of an application root that provides the ability to synchronize changes from the master application root to the root replicas:
  - a. Create a remote clone of the `app_root` application root, named `app_rr`.
  - b. In `cdb1`, in the `app_root` application, create the `px_app_rr` proxy PDB that references the application root replica, `app_rr`, in `cdb2`.
4. Create the `doodles` application PDB in `app_rr`.
5. Write the application code to aggregate data across the `robots`, `dolls`, and `doodles` application PDBs.

**Note:** Proxy PDB creation is detailed in the lesson titled “PDB Creation.”

## Durable Location Transparency

Load balance by relocating one of the application PDBs:

- ➔ The query still retrieves all the rows from the shared table in all the PDBs under the application roots in the CDBs.
- ➔ The application code is unchanged.

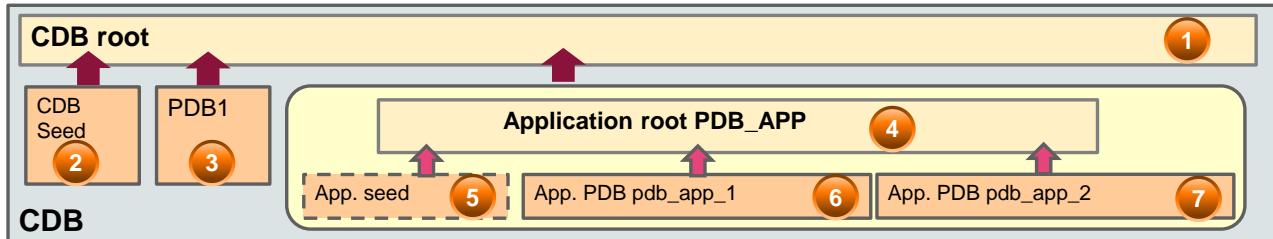


6. To load balance, relocate the `dolls` application PDB from `app_root` to `app_rr`. Relocation implies performing a transparent unplug or plug operation and/or a proxy PDB operation.

**Note:** PDB relocation and proxy PDB are detailed in the lesson titled “PDB Creation.”

The application code continues to run. This is an example of durable location transparency.

## Data Dictionary Views



```
SQL> SELECT name, con_id, application_root "APP_ROOT", application_seed "APP_Seed",
 application_pdb "APP_PDB", application_root_con_id "APP_ROOT_CONID"
 FROM v$containers order by con_id;
```

| NAME          | CON_ID | APP_ROOT | APP_Seed | APP_PDB | APP_ROOT_CONID |
|---------------|--------|----------|----------|---------|----------------|
| CDB\$ROOT     | 1      | NO       | NO       | NO      |                |
| PDB\$SEED     | 2      | NO       | NO       | NO      |                |
| PDB1          | 3      | NO       | NO       | NO      |                |
| PDB_APP       | 4      | YES      | NO       | NO      |                |
| PDB_APP\$SEED | 5      | NO       | YES      | YES     | 4              |
| PDB_APP_1     | 6      | NO       | NO       | YES     | 4              |
| PDB_APP_2     | 7      | NO       | NO       | YES     | 4              |



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Retrieve the hierarchy between an application root and its associated application PDBs from the new columns in the V\$CONTAINERS view:

- APPLICATION\_ROOT: A “YES” value means that the PDB is an application root.
- APPLICATION\_SEED: A “YES” value means that the PDB is an application seed whose application root container ID is described in the APPLICATION\_ROOT\_CON\_ID column.
- APPLICATION\_PDB: A “YES” value means that the PDB is an application PDB whose application root container ID is described in the APPLICATION\_ROOT\_CON\_ID column. Note that the application seed is also defined as an application PDB.

**Remark:** Regular PDBs and application roots have APPLICATION\_ROOT\_CON\_ID set to NULL.

You can also get the same new columns in the CDB\_PDBS and V\$PDBS views.

# Terminology in Application Container Context

Common versus Local:

- Users
- Privileges / Roles
- Objects
- Profiles
- Auditing policies and FGA policies
- Application context and VPD policies
- Transparent sensitive data protection (TSDP) policies
- Database Vault realms and common command rules

**Note:** Any statement that can be issued in a CDB root can also be issued in an application root.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The terminology for common users, roles, granted privileges and roles, profiles, and tables is valid in the context of regular and application containers.

- Common users, roles, and profiles are users, roles, and profiles existing in all containers in the CDB or in an application container with the same name versus local users, roles, and profiles with a unique name existing in one container only. A common user, role, or profile can be created in an application root. Common users, roles, and profiles are replicated in all application PDBs within the application container when the DBA synchronizes the application PDBs with the application root and are visible only in the application PDBs within the application container.
- Common privileges are privileges granted “commonly” to users or roles in all containers in the CDB or in an application container versus privileges granted “locally” to users or roles within a PDB. The same concept exists for roles granted commonly to users or roles in all containers in the CDB or in an application container. Roles granted locally to users or roles are granted to users or roles in a specific PDB. The prefix that is used for common users and roles at the CDB level does not apply in the context of application containers.
- Common objects exist in Oracle-supplied schemas in the CDB root. Users can create application common objects in an application root. The common object is visible to all application PDBs within the application container when the application PDBs have been synchronized with the application root.
- Common unified auditing allows the creation of auditing policies and FGA policies in application containers.
- Common application context and VPD policies can be created in application containers.
- Common transparent sensitive data protection (TSDP) policies can be managed in application containers.
- A Database Vault can protect common objects and commands in application containers via common realms and common command rules.

## Commonality in Application Containers

In an application root, statements to create common entities can be issued only as part of an application operation.

| Application Operation       | Common Entity                                                                                                                                                                                                         |
|-----------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| BEGIN INSTALL / END INSTALL | Create, alter, or drop a common user.<br>Create, alter, or drop a common role.<br>Create, alter, or drop a common profile.<br>Commonly grant privileges or roles to or revoke them from a common user or common role. |
| BEGIN UPGRADE / END UPGRADE | Create, alter, and drop common objects.                                                                                                                                                                               |
| BEGIN PATCH / END PATCH     |                                                                                                                                                                                                                       |



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

In an application root, statements to create, alter, or drop a common user, a common role, a common profile, or a common object or grant privileges and roles commonly can be issued only as part of an application INSTALL, UPGRADE, or PATCH operation.

This implies that the statements to create, alter, or drop entities that are common to the application PDBs in an application container are issued between two statements, issued from an application root connection:

```
SQL> CONNECT sys@app_root as sysdba
SQL> ALTER PLUGGABLE DATABASE APPLICATION <app_name> BEGIN INSTALL ...
```

and

```
SQL> ALTER PLUGGABLE DATABASE APPLICATION <app_name> END INSTALL ...
```

or

```
SQL> ALTER PLUGGABLE DATABASE APPLICATION <app_name> BEGIN UPGRADE ...
```

```
SQL> ALTER PLUGGABLE DATABASE APPLICATION <app_name> END UPGRADE ...
```

or

```
SQL> ALTER PLUGGABLE DATABASE APPLICATION <app_name> BEGIN PATCH ...
```

```
SQL> ALTER PLUGGABLE DATABASE APPLICATION <app_name> END PATCH ...
```

## Impacts

- Per PDB character set:
  - Enables storing multilingual data
  - Facilitates conversion of existing non-CDBs to PDBs
  - Facilitates fast and seamless unplug/plug of PDBs across CDBs that have different compatible character sets
  - Is the same for all PDBs in an application container
  - Is supported with the LogMiner data dictionary
- Common unified and FGA policies in application containers
- Database Vault common realms and command rules at CDB level
- Common objects in application PDBs supported by LogMiner



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

- If the CDB has a Unicode database character set of AL32UTF8, the CDB can contain PDBs with different database character sets. It becomes easier to convert existing non-CDBs to PDBs because non-CDBs can be plugged without having to perform character set conversion to match the CDB's character set. If an application container contains any common object, the use of multiple character sets across application PDBs is disallowed to protect user data against truncation and corruption issues.
- A unified audit configuration is visible and enforced across all PDBs, which enables administrators to avoid configuring auditing separately for each container. This provides the ability to create audit policies that are used by all PDBs and also audit policies that are used exclusively for each PDB. Common audit policies can be created in the context of application containers. An audit configuration that is not enforced across all PDBs means that it applies only within a PDB and is not visible outside it. An audit configuration that is enforced across all the PDBs of an application container means that it applies only within the application PDBs of the application container and is not visible outside it.
- Each PDB has its own Database Vault metadata. Database Vault constructs, such as realms, are isolated within a PDB. The protection on common objects is possible with common realms and command rules.
- The LogMiner ad hoc queries (`V$LOGMNR_CONTENTS`, `DBMS_LOGMNR`) support customer common objects in application PDBs just as they support objects in regular PDBs.

## Summary

In this lesson, you should have learned how to:

- Describe application containers in CDBs
- Explain the purpose of application root and application seed
- Define application PDBs
- Create application PDBs
- Explain application installation on top of application containers
- Install an application
- Upgrade and patch applications
- Describe the commonality concept in application contexts
- Use a dynamic container map
- Describe enhancements in various areas



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

- Describe application containers in CDBs
- Explain the purpose of application root and application seed
- Define application PDBs
- Create application PDBs
- Explain application installation on top of application containers
- Install an application
- Upgrade and patch applications
- Describe the commonality concept in application contexts
- Use a dynamic container map
- Describe enhancements in various areas

## Practice 3: Overview

- 3-1: Installing an application in an application container
- 3-2: Upgrading an application in an application container
- 3-3: Querying data across application PDBs in CDB



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.



# PDB Creation



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

# Objectives

After completing this lesson, you should be able to:

- Clone a regular PDB
- Clone an application container
- Unplug and plug or clone a non-CDB
- Unplug and plug a regular PDB
- Unplug and plug an application container
- Convert regular PDBs to application PDBs
- Configure and use the local UNDO mode
- Perform hot cloning
- Perform near-zero downtime PDB relocation
- Create and use a proxy PDB
- Drop PDBs



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

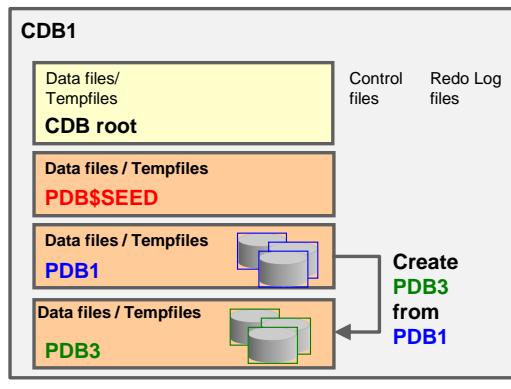
For a complete understanding of the PDB provisioning procedures, refer to the following guide in the Oracle documentation:

- *Oracle Multitenant Administrator's Guide 18c*

Refer to other sources of information available under YouTube like:

[Stale Standalone to Superb SaaS in a Short Series](#)

# Cloning Regular PDBs



PDB3 owns:

- SYSTEM, SYSAUX, UNDO tablespaces
- Full catalog
- SYS, SYSTEM common users
- Same local administrator name
- New service name

1. Define how Oracle will find the location of the data files:

- In init.ora, set DB\_CREATE\_FILE\_DEST= 'PDB3dir'
- In init.ora, set PDB\_FILE\_NAME\_CONVERT='PDB1dir', 'PDB3dir'
- Using the CREATE\_FILE\_DEST= 'PDB3dir' clause

2. Connect to the CDB root to close PDB1.

3. Clone PDB3 from PDB1.

```
SQL> CREATE PLUGGABLE DATABASE pdb3 FROM pdb1
 CREATE_FILE_DEST = 'PDB3dir';
```

4. Open PDB3 in read write mode.

```
SQL> ALTER PLUGGABLE DATABASE pdb3 OPEN;
```

**Note:** Cloning metadata only with NO DATA



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

This technique copies a source PDB from a CDB and plugs the copy into a CDB. The source PDB is in the local CDB.

The steps to clone a PDB within the same CDB are the following:

1. Define the location for the datafiles of the new PDB:

- In init.ora, set DB\_CREATE\_FILE\_DEST= 'PDB3dir' (OMF) or PDB\_FILE\_NAME\_CONVERT= 'PDB1dir', 'PDB3dir' (non OMF)
- Use the CREATE\_FILE\_DEST clause during the CREATE PLUGGABLE DATABASE statement (OMF).
- Use the FILE\_NAME\_CONVERT= ('pdb1dir', ' pdb3dir') clause to define the directory of the source files to copy from PDB1 and the target directory for the new files of PDB3 (non-OMF)

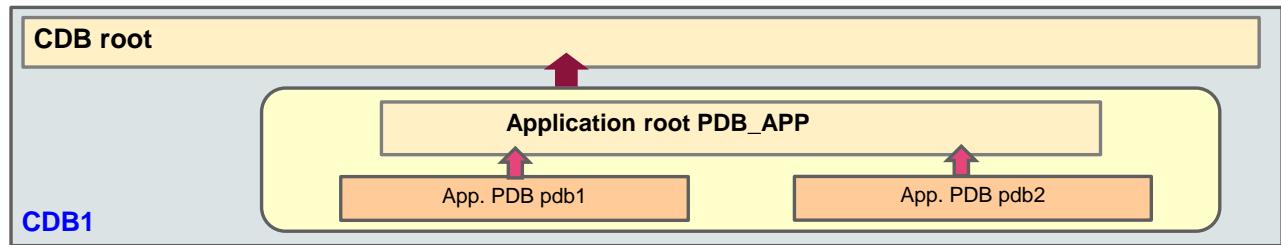
2. Connect to the CDB root as a common user with the CREATE PLUGGABLE DATABASE privilege.

3. Use the command CREATE PLUGGABLE DATABASE to clone the PDB pdb3 from pdb1.

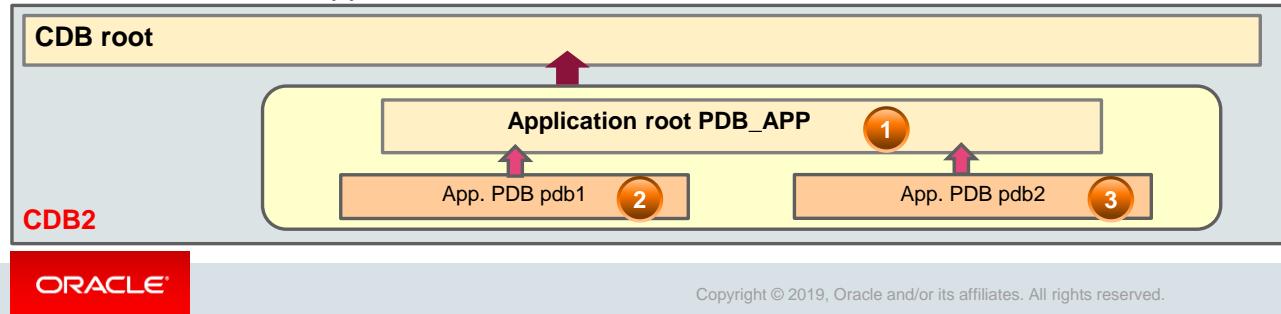
4. Then open the new pdb3 with the command ALTER PLUGGABLE DATABASE OPEN.

**Note:** NO DATA allows PDB metadata cloning, which is an interesting option that can be used for cloning an empty PDB and later importing data for testing.

## Cloning Application Containers



- Clone the application root.
- Then clone all the application PDBs.



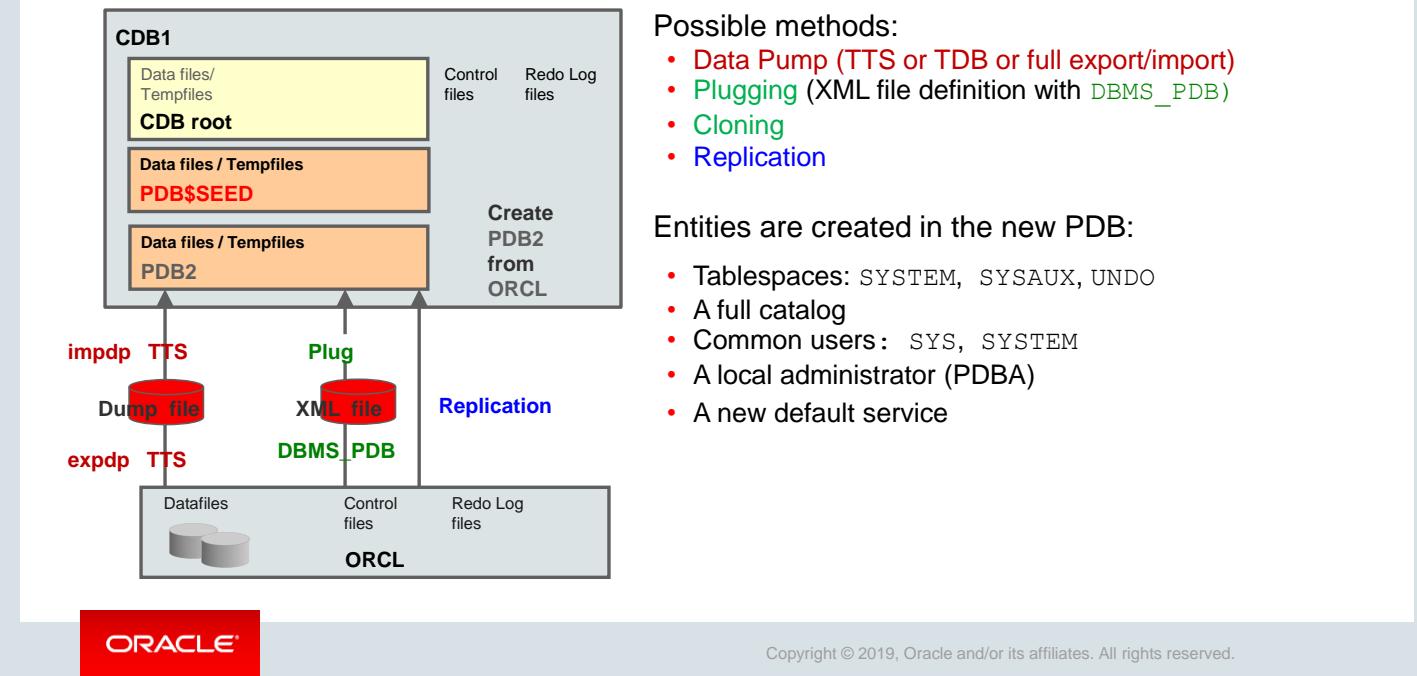
Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

This technique copies a source application container into another application container in a CDB.

The steps to clone an application container within the same CDB are the following:

1. Define the location for the datafiles of the new PDB:
  - In `init.ora`, set `DB_CREATE_FILE_DEST= 'PDB3dir'` (OMF) or `PDB_FILE_NAME_CONVERT= 'PDB1dir', 'PDB3dir'` (non OMF).
  - Use the `CREATE_FILE_DEST` clause during the `CREATE PLUGGABLE DATABASE` statement (OMF).
  - Use the `FILE_NAME_CONVERT=('pdb1dir', ' pdb3dir')` clause to define the directory of the source files to copy from `PDB1` and the target directory for the new files of `PDB3` (non-OMF).
2. Connect to the target CDB root as a common user with the `CREATE PLUGGABLE DATABASE` privilege.
3. Use the command `CREATE PLUGGABLE DATABASE` to clone the application root `pdb_app` from `pdb_app` into `cdb2`.
4. Then still in `cdb2`, open the new `pdb_app` with the command `ALTER PLUGGABLE DATABASE OPEN`.
5. Connect to the new application root `pdb_app` in `cdb2` and use the command `CREATE PLUGGABLE DATABASE` to clone the application PDB `pdb1` from `pdb1` of `cdb1` into `pdb_app` of `cdb2`.
6. Clone the application PDB `pdb2` from `pdb2` of `cdb1` into `pdb_app` of `cdb2`.
7. Open the `pdb1` and `pdb2` of `cdb2`.

## Plugging a Non-CDB into CDB

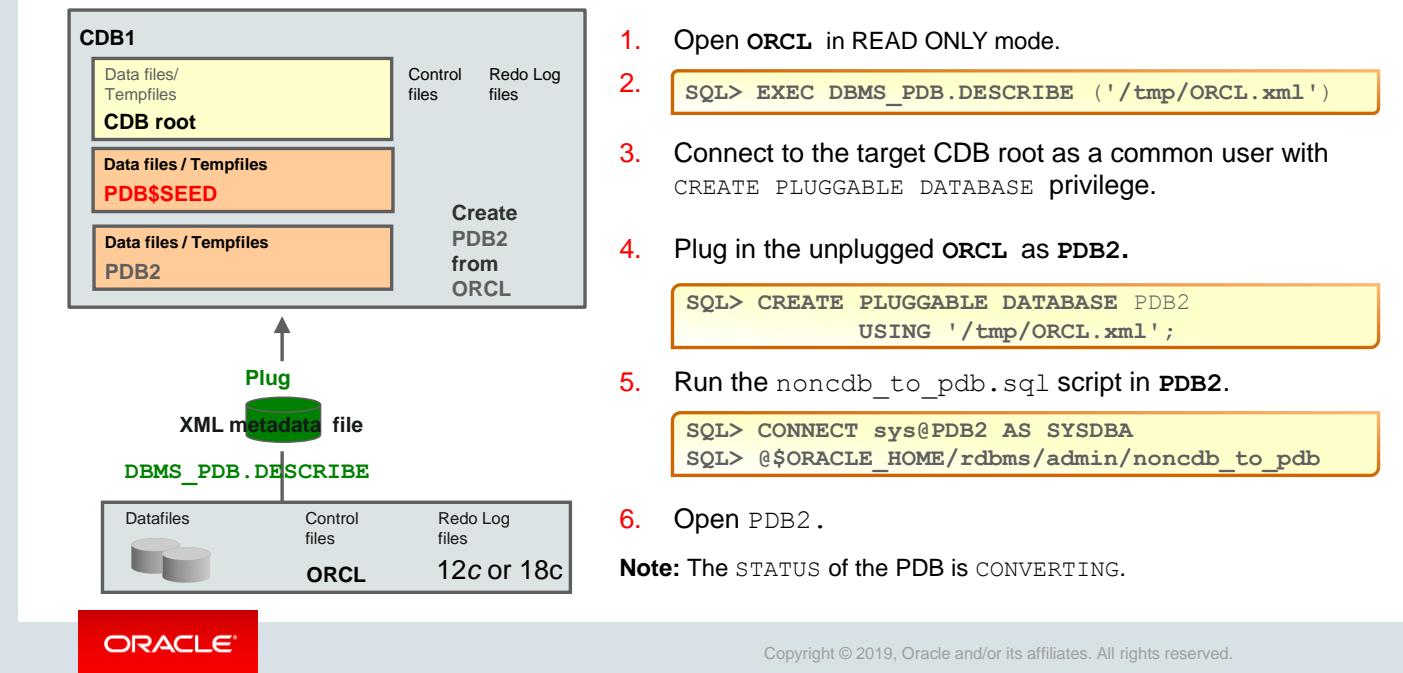


There are different possible methods to migrate data from a non-CDB database into a CDB.

Whichever method is used, you have to get the non-CDB into a transactionally consistent state and open it in restricted mode.

- It is appropriate to use Oracle Data Pump when:
  - The source database is an 11g database.
  - Both source and target databases are different endian.
  - The source character set is not equal to the target character set and is not a binary subset of the target.
  - Use either transportable tablespace (TTS) or full conventional export / import or full transportable database (TDB) provided that in the last one any user-defined object resides in a single user-defined tablespace. Data Pump full transportable database does not support movement of XDB or AWR repositories. Only user-generated XML schemas are moved.
- In other cases, using the `DBMS_PDB` package is the easiest option. The `DBMS_PDB` package constructs an XML file describing the non-CDB data files to plug the non-CDB into the CDB as a PDB. It is also a good way to quickly consolidate several non-CDBs into a CDB.
- Cloning non-CDBs in a CDB is a good way to keep the non-CDB and therefore have the opportunity to compare the performance between the new PDB and the original non-CDB or at least wait until you consider that the PDB can work appropriately.
- If the `DBMS_PDB` package cannot be used as this would be the case because the non-CDB is an 11g database, export/import is usually simpler than using GoldenGate replication, but export/import might require more downtime during the switch from the non-CDB to the PDB.

## Plugging a Non-CDB into CDB Using DBMS\_PDB



The technique with DBMS\_PDB package creates an unplugged PDB from an Oracle Database 12c or Oracle Database 18c non-CDB. The unplugged PDB can then be plugged into a CDB (of the same version) as a new PDB.

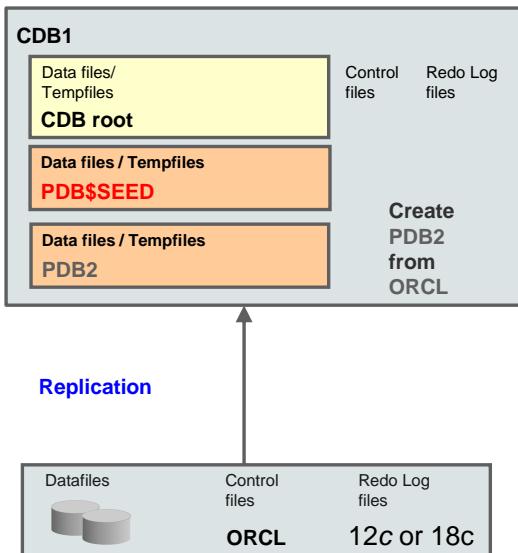
If the non-CDB is an Oracle Database 12c database and the target CDB is an Oracle Database 18c database, first upgrade the non-CDB to an Oracle Database 18c non-CDB. Then execute the DBMS\_PDB package to create an unplugged Oracle Database 18c PDB and finally plug it into the Oracle Database 18c CDB.

Running the DBMS\_PDB.DESCRIBE procedure on the non-CDB generates an XML file that describes the future PDB. You can plug in the unplugged PDB in the same way that you can plug in any unplugged PDB, using the XML file and the non-CDB datafiles. The steps are the following:

1. Connect to non-CDB ORCL and ensure that the non-CDB ORCL is in read only mode.
2. Execute the DBMS\_PDB.DESCRIBE procedure, providing the file name that will be generated. The XML file contains the list of datafiles to be plugged. The XML file and the data files described in the XML file comprise an unplugged PDB.
3. Connect to the target CDB to plug the unplugged ORCL as PDB2.
4. Before plugging the unplugged non-CDB, make sure it can be plugged into a CDB using the DBMS\_PDB.CHECK\_PLUG\_COMPATIBILITY procedure. Execute the CREATE PLUGGABLE command using the clause USING 'XMLfile.xml'. The list of datafiles from ORCL is read from the XMLfile to locate and name the datafiles of PDB2.
5. Run the ORACLE\_HOME/rdbms/admin/noncdb\_to\_pdb.sql script to delete unnecessary metadata from the PDB SYSTEM tablespace. This script is required for plugging non-CDBs only and must be run before the PDB is opened for the first time.
6. Open PDB2 to verify that the application tables are in PDB2.

If the non-CDB is an Oracle Database 11g database (11.2.0.3 or 11.2.0.4), first upgrade it to Oracle Database 12c or Oracle Database 18c.

# Replicating Non-CDB into CDB



1. Connect to the CDB root as a common user with CREATE PLUGGABLE DATABASE privilege.
2. Create new PDB2 (from PDB\$SEED).
3. Open PDB2 in read write mode.
4. Configure unidirectional replication environment from ORCL to PDB2.
5. Check application data.

```
SQL> CONNECT sys@PDB2
SQL> SELECT * FROM dba_tables;
SQL> SELECT * FROM HR.EMP;
```

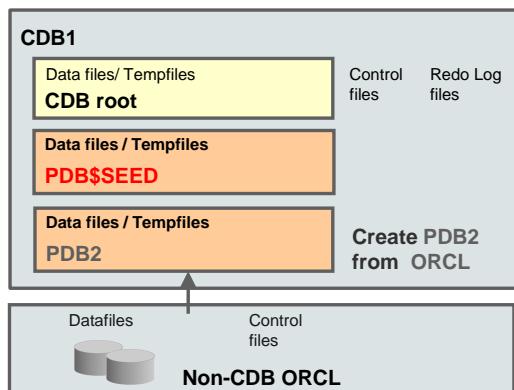


Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

If you choose the replication method, the steps would be the following:

1. Connect to the CDB root as a common user with the CREATE PLUGGABLE DATABASE privilege.
2. Create the new PDB2 from the CDB seed that will be the container for ORCL data.
3. Open PDB2 in read write mode.
4. Configure an Oracle GoldenGate unidirectional replication environment with the non-CDB ORCL as the source database and the PDB2 as the destination database.
5. When the data at PDB2 catches up with the data at the non-CDB ORCL, switch to PDB2.

## Cloning a Non-CDB or Remote PDB



**PDB\_ORCL** owns:

- SYSTEM, SYSAUX, UNDO tablespaces
- Full catalog
- A temporary tablespace
- SYS, SYSTEM common users
- New service name

1. Set **ORCL** in READ ONLY mode.
2. Connect to the CDB to create the database link:

```
SQL> CREATE DATABASE LINK link_orcl
 CONNECT TO system IDENTIFIED BY ***
 USING 'orcl';
```

3. Clone the non-CDB:

```
SQL> CREATE PLUGGABLE DATABASE pdb_orcl
 FROM NON$CDB@link_orcl
 CREATE_FILE_DEST = '.../PDB_orcl';
```

4. Run the **noncdb\_to\_pdb.sql** script.

```
SQL> CONNECT sys@pdb_orcl AS SYSDBA
SQL> @$ORACLE_HOME/rdbms/admin/noncdb_to_pdb
```

5. Open **PDB\_ORCL** in read-write mode.

```
SQL> ALTER PLUGGABLE DATABASE pdb_orcl OPEN;
```



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

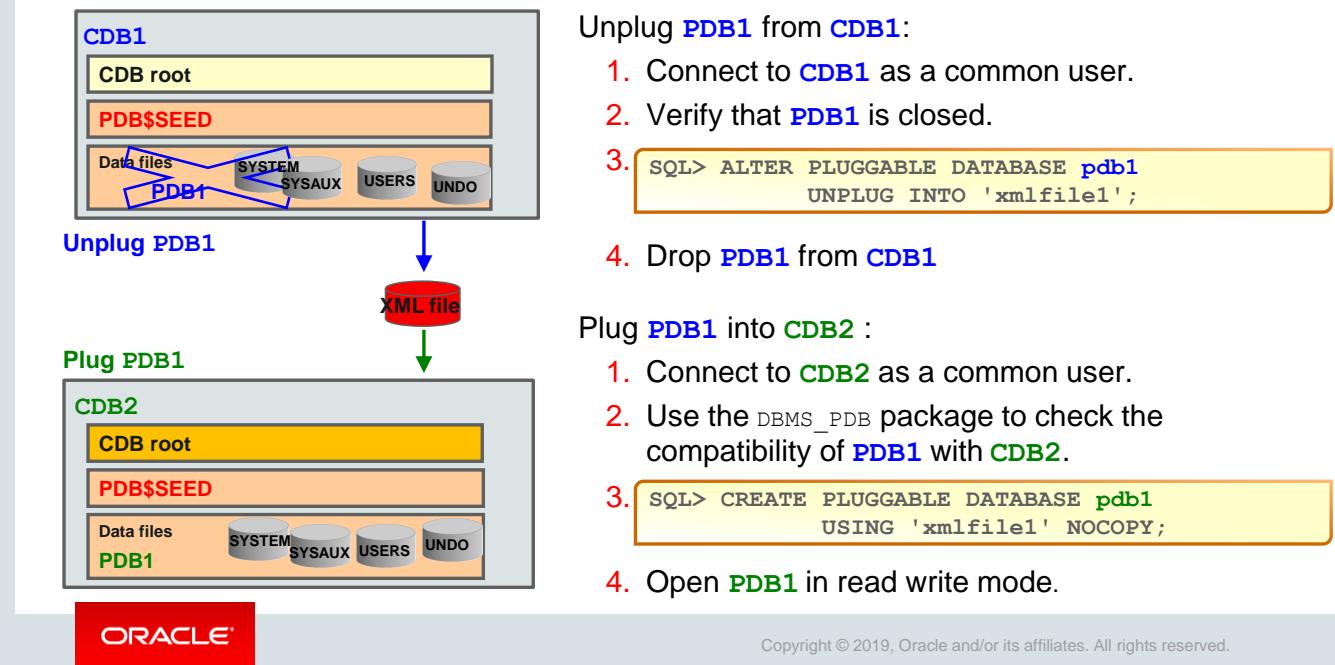
This technique copies a non-CDB or remote PDB and plugs the copy into a CDB.

The steps to clone a non-CDB or remote PDB into a CDB are the following:

1. Set the non-CDB or remote PDB in READ ONLY mode.
2. Connect to the root of the target CDB as a common user with CREATE PLUGGABLE DATABASE privilege.
3. Create a database link that allows a connection to the remote non-CDB or PDB as a user with CREATE PLUGGABLE DATABASE privilege.
4. Use the CREATE PLUGGABLE DATABASE command to clone the non-CDB as described in the slide. If you clone a remote PDB, use the source PDB name in place of **NON\$CDB**. Ensure that the new PDB does not conflict with a name of any container within the CDB.
5. If the cloned source is a non-CDB, it is necessary to run the **\$ORACLE\_HOME/rdbms/admin/noncdb\_to\_pdb.sql** script.
6. Then open the new PDB with the ALTER PLUGGABLE DATABASE command.
7. Finally, you can re-open the non-CDB or remote PDB.

There are additional clone options like **SNAPSHOT COPY**. Refer to the *Oracle Database Administrator's Guide 18c Release 1 (18.1)*.

## Plugging an Unplugged Regular PDB into CDB



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

You can create a PDB in a CDB by the unplugging / plugging method.

Unplugging a PDB disassociates the PDB from a CDB. You unplug a PDB when you want to move the PDB to a different CDB or when you no longer want the PDB to be available.

The first step is to unplug **PDB1** from **CDB1**. The second step is to plug **PDB1** into **CDB2**.

To unplug **PDB1** from **CDB1**, first connect to the source CDB root and check that the PDB is closed using the `V$PDBS` view. Then use `ALTER PLUGGABLE DATABASE` with `UNPLUG` clause to specify the database to unplug and the XML file to unplug it into. The `STATUS` in `CDB_PDBS` of the unplugged PDB is `UNPLUGGED`. A PDB must be dropped from the CDB before it can be plugged back into the same CDB. If the PDB is plugged into another CDB, the PDB does not need to be dropped if the datafiles are copied.

Before plugging **PDB1** into **CDB2**, you can optionally check whether the unplugged PDB is compatible with the `CDB2` with the `DBMS_PDB.CHECK_PLUG_COMPATIBILITY` function.

To plug **PDB1** into **CDB2**, connect to **CDB2** root and use `CREATE PLUGGABLE DATABASE pdb1 USING 'xmlfile1.xml'`. The last step is opening the PDB.

## Flow

Several clauses can be used in conjunction:

Are new PDB files based on same files that were used to create existing PDB in CDB?

If not, AS CLONE clause is required, and so it ensures that Oracle Database generates unique PDB DBID, GUID, and other identifiers expected for the new PDB.

Does XML file accurately describe current locations of files?

If not, the SOURCE\_FILE\_NAME\_CONVERT clause is required.

Are files are in correct location?

If not, specify COPY to copy files to new location or MOVE to move them to another location. If yes, use NOCOPY. COPY is the default.

- FILE\_NAME\_CONVERT clause of CREATE PLUGGABLE DATABASE statement
- OMF: DB\_CREATE\_FILE\_DEST parameter
- PDB\_FILE\_NAME\_CONVERT parameter

Do you want to specify storage limits for PDB?

If yes, specify the STORAGE clause.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Several clauses can be used in conjunction according to different assumptions:

First question: Are the files of the new PDB based on the same files that were used to create an existing PDB in CDB? If not, the AS CLONE clause is required, and so it ensures that Oracle Database generates unique PDB DBID, GUID, and other identifiers expected for the new PDB.

Second question: Does the XML file accurately describe the current locations of the files? If not, the SOURCE\_FILE\_NAME\_CONVERT clause is required.

Third question: Are the files are in the correct location? If not, specify COPY to copy the files to a new location or MOVE to move them to another location . If yes, use NOCOPY. COPY is the default.

Use one of these techniques to specify the target location:

- In the CREATE PLUGGABLE DATABASE statement, include a FILE\_NAME\_CONVERT clause that specifies the target locations based on the names of the source files.
- Enable OMF to determine the target location using the DB\_CREATE\_FILE\_DEST initialization parameter.
- Specify the target locations in the PDB\_FILE\_NAME\_CONVERT initialization parameter.

If you use more than one of these techniques, then the precedence order is:

- FILE\_NAME\_CONVERT clause
- Oracle Managed Files using the DB\_CREATE\_FILE\_DEST initialization parameter
- The PDB\_FILE\_NAME\_CONVERT initialization parameter

Fourth question: Do you want to specify storage limits for the PDB? If yes, specify the STORAGE clause.

That's why according to the different combinations, you can use different syntaxes:

```
CREATE PLUGGABLE DATABASE PDB1 AS CLONE USING
'/disk1/usr/salespdb.xml' COPY;
```

```
CREATE PLUGGABLE DATABASE PDB1 USING '/disk1/usr/salespdb.xml'
SOURCE_FILE_NAME_CONVERT = ('/disk1/oracle/sales',
'/disk2/oracle/sales') NOCOPY STORAGE (MAXSIZE 500M
MAX_SHARED_TEMP_SIZE 100M);
```

```
CREATE PLUGGABLE DATABASE PDB1 USING '/disk1/usr/salespdb.xml' COPY
FILE_NAME_CONVERT = ('/disk1/oracle/sales', '/disk2/oracle/sales');
```

To know if a PDB is unplugged, display the STATUS column in the CDB\_PDBS view: the value is UNPLUGGED:

```
SQL> select pdb_name , status from CDB_PDBS;
```

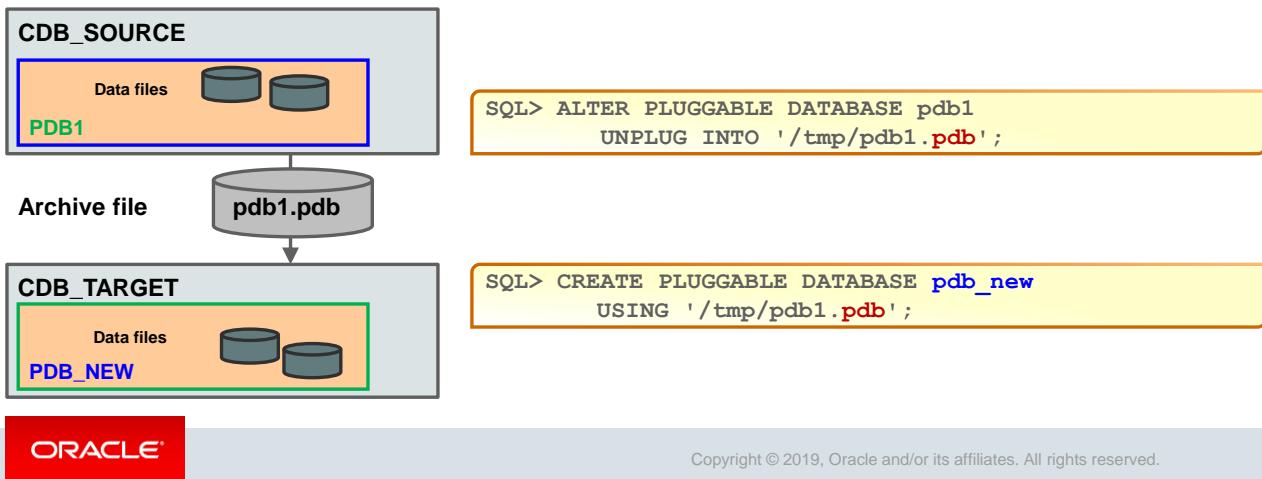
| PDB_NAME  | STATUS    |
|-----------|-----------|
| PDB1      | NORMAL    |
| PDB\$SEED | NORMAL    |
| PDB3      | UNPLUGGED |

## Plugging Using Archive File

1. Unplugging a PDB into a single archive file includes:

- XML file
- Data files

2. Plugging the PDB requires only the archive file.

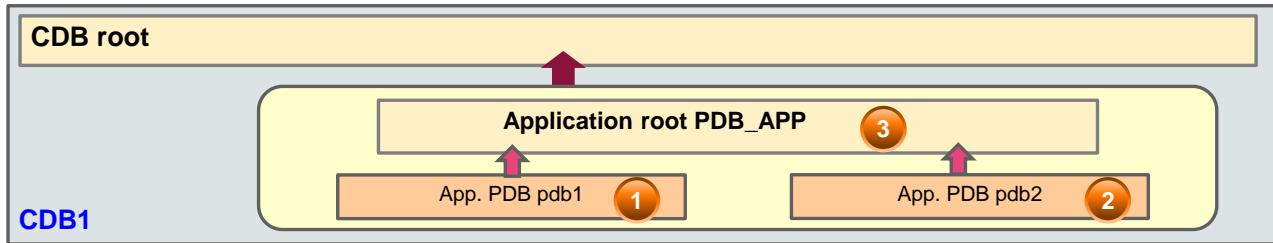


Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

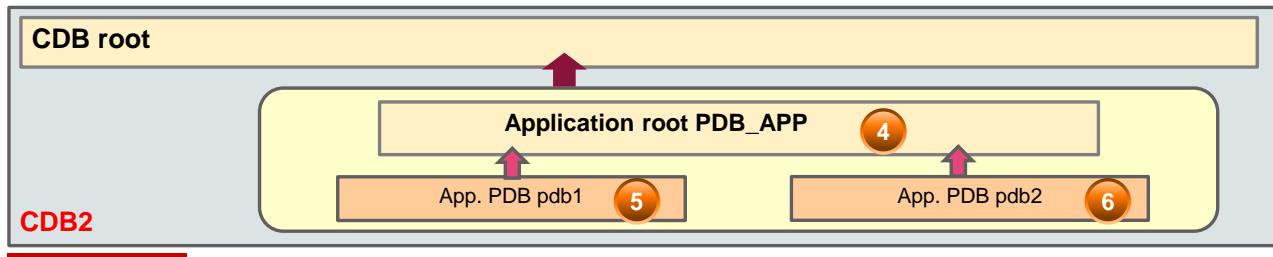
When a PDB is unplugged, all the datafiles associated with the PDB along with the PDB manifest must be copied or moved individually over to the remote server where it will be plugged into another CDB. You can choose to create a single PDB archive file, a compressed file with the .pdb extension, which contains the PDB manifest and all the datafiles when unplugging a PDB. When plugging in a PDB, the presence of a .pdb file is interpreted, and the PDB is plugged into the CDB. You can choose to run the PDB plug-in compatibility test directly on the PDB archive without extracting the PDB manifest file from the archive.

This feature provides ease of managing the unplugging and plugging of PDBs across CDBs.

## Unplugging and Plugging Application PDBs



1. In the source CDB, unplug all application PDBs and then the application root.
2. In the target CDB, plug the application root first and then all the application PDBs.



**ORACLE®**

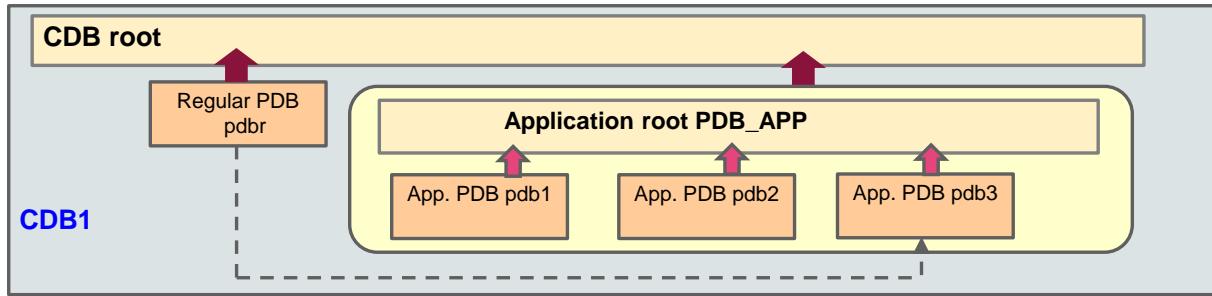
Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

To plug an application container from one CDB to another, proceed with the following sequence:

1. Unplug the application PDBs of the application container first and then unplug the application root of the application container.
2. Plug the application root of the application container first and then plug the application PDBs.

**Note:** When the application PDB is being plugged in, that PDB must have an application name that matches the application name of the application root due to the synchronization requirement that uses the application name and version.

## Converting Regular PDBs to Application PDBs



- Two methods to convert the regular PDB to an application PDB:
  - Clone the regular PDB into an application root.
  - Unplug the regular PDB to plug it into an application root.
- Connect to the application PDB to execute the `pdb_to_apppdb.sql` script.
- Synchronize the application PDB with the application root.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Applications that are already installed in a regular PDB can also take advantage of application containers.

1. Either plug or clone a regular PDB into an application root on top of which an application is already installed. Ensure that you are connected to the application root to complete this operation.
2. Connect to the new application PDB and execute the `$ORACLE_HOME/rdbms/admin/pdb_to_apppdb.sql` script so that the object definitions of objects marked as common in the application root are replaced with links in the application PDB. For example, users and roles that exist as common in the application root are marked as common in the plugged application PDB.
3. Finally synchronize the new application PDB with its application root.

## Unplugging and Plugging a PDB with Encrypted Data

1. Unplugging an encrypted PDB exports the master encryption key of the PDB.

```
SQL> ALTER PLUGGABLE DATABASE pdb1
 UNPLUG INTO '/tmp/pdb1.xml'
 ENCRYPT USING "tpwd1";
```

PDB wallet opened



2. Plugging the encrypted PDB imports the master encryption key of the PDB into the CDB keystore.

```
SQL> CREATE PLUGGABLE DATABASE pdb1
 USING '/tmp/pdb1.xml'
 KEYSTORE IDENTIFIED BY keystore_pwd1
 DECRYPT USING "tpwd1";
```

Target CDB wallet opened



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The unplugging operation of an encrypted PDB is a one-step operation, and the plugging operation of the new encrypted PDB is also a one-step operation.

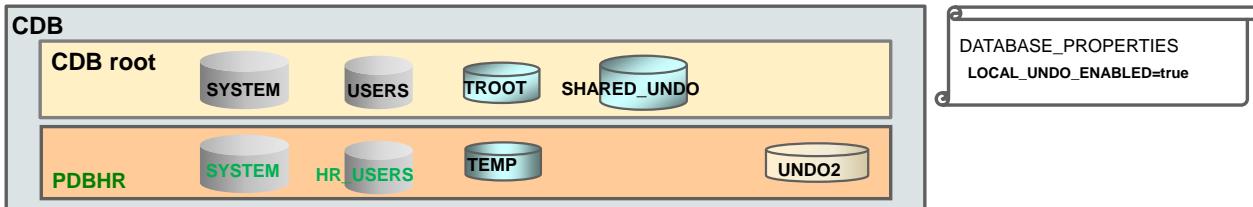
1. As part of the unplug command, the administrator must specify a temporary transport password to protect the implicitly exported master keys, and this can be achieved only if the wallet on the source database is already opened.
2. Trusted administrators must then be able to plug the encrypted PDB into a CDB without separately importing the master keys used by the PDB by specifying the transport password that was specified at unplug time to decrypt the implicitly exported master keys for import into the target database. This assumes that the “trusted administrator” is defined as someone who is authorized to have access to the wallet for the target CDB and has the privilege to plug the PDB into the CDB.

The preceding commands require the `SYSKM` administrative privilege as well as the privilege to unplug and plug the PDB.

## Local UNDO Mode Versus Shared UNDO Mode

Two UNDO modes: SHARED versus LOCAL

- There is only one shared UNDO tablespace (in CDB root).
- There can be a local UNDO tablespace in each PDB.



When is local UNDO mode required?

- Hot cloning
- Near-zero downtime PDB relocation

```
SQL> STARTUP UPGRADE
SQL> ALTER DATABASE LOCAL UNDO ON;
```

ORACLE®

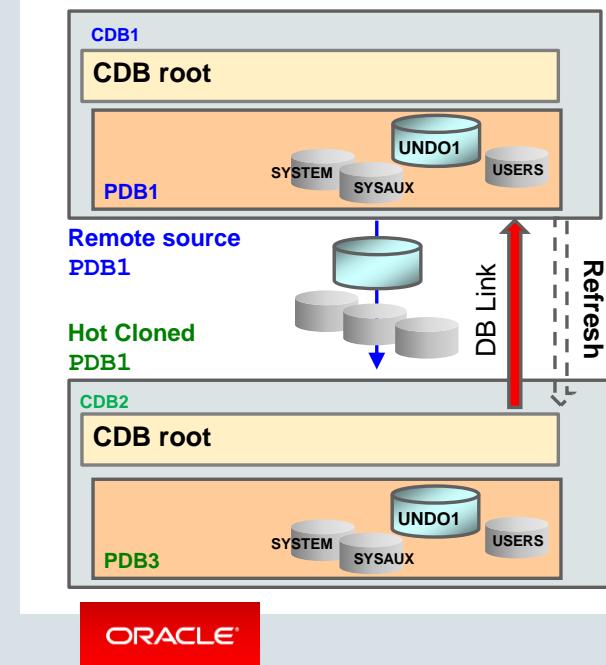
Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Using the local UNDO mode is required when cloning a PDB in hot mode or performing a near-zero downtime PDB relocation or refreshing PDBs or using proxy PDBs.

You can set a CDB in local UNDO mode either at CDB creation or by altering the CDB property.

- When the database property `LOCAL_UNDO_ENABLED` is `FALSE`, which is the default, there is only one UNDO tablespace that is created in the CDB root, and that is shared by all containers.
- When `LOCAL_UNDO_ENABLED` is `TRUE`, every container in the CDB uses local undo, and each PDB must have its own local UNDO tablespace. To maintain ease of management and provisioning, UNDO tablespace creation happens automatically and does not require any action from the user. When a PDB is opened and an UNDO tablespace is not available, it is automatically created.

## Cloning Remote PDBs in Hot Mode



**Remote source PDB still up and fully functional:**

1. Connect to the target **CDB2** root to create the database link to **CDB1**.
2. Switch the shared UNDO mode to local UNDO mode in both the CDBs.
3. Clone the remote **PDB1** to **PDB3**.
4. Open **PDB3** in read-only or read-write mode.

**Incremental refreshing:**

- Manual
- Automatic (predefined interval)

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Cloning a production PDB to get a test PDB copies the remote production PDB into a CDB while the remote production PDB is still up and fully functional.

Hot remote cloning requires both CDBs to switch from shared UNDO mode to local UNDO mode, which means that each PDB uses its own local UNDO tablespace.

### Refreshable Copy

In addition, hot cloning allows incremental refreshing in that the cloned copy of the production database can be refreshed at regular intervals. Incremental refreshing means refreshing an existing clone from a source PDB at a point in time that is more recent than the original clone creation to provide fresh data. A refreshable copy PDB can be opened only in read-only mode.

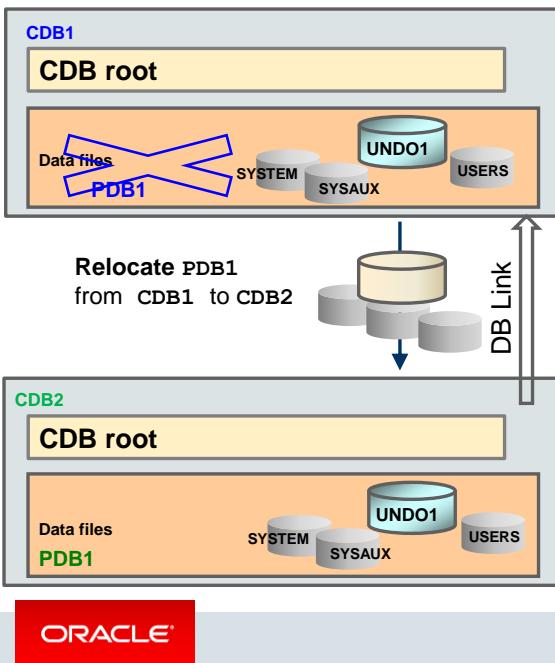
Propagating changes from the source PDB can be performed in two ways:

- Manually (on demand)
- Automatically at predefined time intervals

If the source PDB is not accessible at the moment the refresh copy needs to be updated, archive logs are read from the directory specified by the `REMOTE_RECOVERY_FILE_DEST` parameter to refresh the cloned PDB.

Oracle Database 18c allows you to clone an existing PDB with Database Configuration Assistant (DBCA).

# Near-Zero Downtime PDB Relocation



Use a single statement to relocate **PDB1** from **CDB1** into **CDB2**:

1. Switch the shared UNDO mode to local UNDO mode in both CDBs.
2. Set ARCHIVELOG mode in both CDBs.
3. Grant SYSOPER to the user connected to **CDB1** via the database link created in **CDB2**.
4. Connect to **CDB2** as a common user to create the database link.
5. Use the CREATE PLUGGABLE DATABASE statement with the new RELOCATE clause.
6. Open **PDB1** in read-write mode.

There is no need to:

- Unplug the PDB from the source CDB
- Copy or transfer the datafiles to a new location
- Plug the PDB in the target CDB
- Drop the source PDB from the source CDB

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

To get the same result as unplugging and plugging a PDB from a remote source CDB into another CDB, you can take advantage of the feature such as near-zero downtime PDB relocating.

A single DDL statement can relocate a PDB, using the “pull” mode, connected to the CDB where the PDB will be relocated to pull it from the CDB where the PDB exists, managing draining existing connections and migrating new connections without requiring any changes to the application.

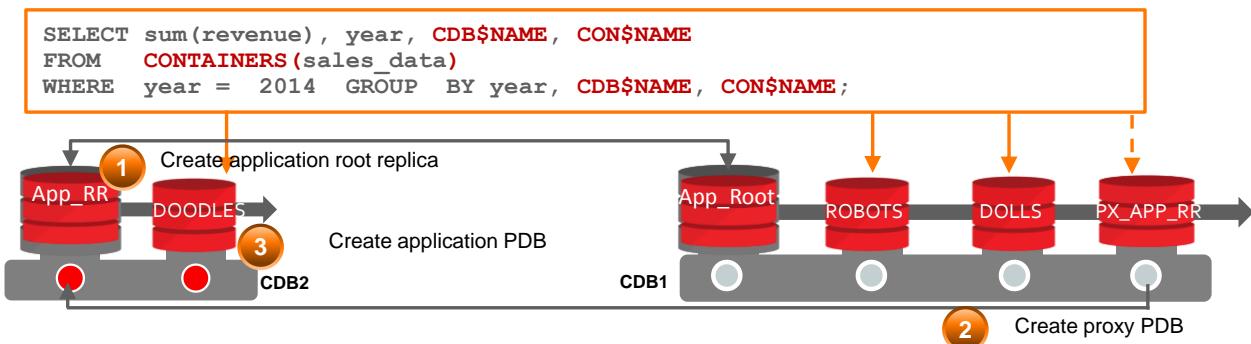
There are two relocation methods:

- **Normal availability mode**
  - When the newly created PDB is opened in read-write mode for the first time, the source PDB is automatically closed and dropped, and the relocation operation is completed with the relocated PDB being fully available. This is the “normal availability” default mode.
  - This method can be used to relocate application PDBs too.
- **Maximum availability mode**
  - The maximum availability mode reduces application impact by handling the migration of connections, preserving the source CDB in mount state to guarantee connection forwarding of the listener to the remote listener where the PDB is relocated. In this case, you cannot create a PDB with the same name as the source PDB because it will conflict with the listener forwarding. It is expected that connect strings are updated at a time that is convenient for the application. After this is done and all the clients connect to the new host without forwarding, the DBA can drop the source PDB.

- If AVAILABILITY MAX is specified during the CREATE PLUGGABLE DATABASE RELOCATE command, additional handling is performed to ensure smooth migration of workload and persistent connection forwarding from the source to the target. The PDB is always first opened in read-only mode. This makes the PDB available as a target for new connections before the source PDB is closed. During this operation, listener information of the target CDB is automatically sent to the source and a special forwarding registration is performed with the source PDB's current listener. New connections to the existing listener are automatically forwarded to connect to the new target. This forwarding persists even after the relocation operation has been completed and effectively allows for no changes to connect strings.
- It is still recommended that connect strings are updated eventually at a time that is convenient for the application, but availability is not dependent on when this action is performed.

PDB relocation requires enabling the local UNDO mode and ARCHIVELOG mode in both CDBs.

## Proxy PDB: Query Across CDBs Proxying Root Replica



→ Retrieves rows from the shared table whose data is stored in application PDBs in the application root and replicas in CDBs

| Revenue  | Year | CDB\$NAME | CON\$NAME |
|----------|------|-----------|-----------|
| 15000000 | 2014 | CDB1      | ROBOTS    |
| 20000000 | 2014 | CDB2      | DOODLES   |
| 10000000 | 2014 | CDB1      | DOLLS     |



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

A proxy PDB allows you to execute SQL statements in a remote PDB as if it were a local PDB in the CDB. This type of PDB is very helpful to query data spread over PDBs in different CDBs.

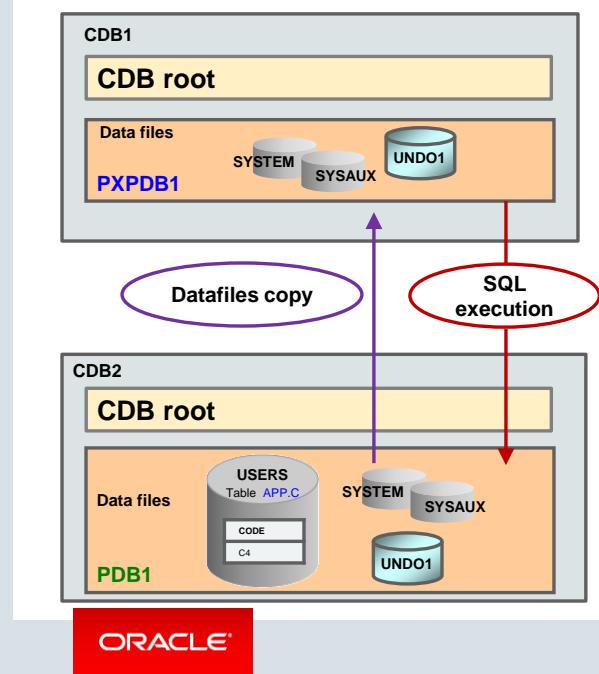
In the example in the slide, when connected to the `toys_root` application root, a query on a table shared in the application PDBs, `robots`, `dolls`, and `doodles` cannot retrieve rows from the remote `doodles` if two conditions are not satisfied:

- An application root replica is created in the remote CDB to replicate the application root and, therefore, the application common entities such as tables, users, and privileges.
- A proxy PDB is created in the application root in the local CDB to reference the application root replica in the remote CDB.

When you are connected to the `toys_root` application root and you query from a shared table of the application installed on the `toys_root` application root, the query fetches rows from the two local application PDBs, `robots` and `dolls`, and from the application proxy PDB, `px_app_rr`, executing the query in the remote root replica and, therefore, from the `doodles` application PDB.

# Creating a Proxy PDB

| CDB_PDBS           |
|--------------------|
| IS_PROXY_PDB = YES |
| FOREIGN_CDB_DBID   |
| FOREIGN_PDB_ID     |



A proxy PDB allows execution in a proxied PDB.

1. Switch the shared UNDO mode to local UNDO mode in both CDBs.
2. Set the ARCHIVELOG mode in both CDBs.
3. Connect to **CDB1** and create a database link (to **CDB2**).
4. Create the **PXPDB1** proxy PDB in **CDB1** as a view referencing the entire proxied **PDB1** in **CDB2**.

```
SQL> CONNECT sys@cdb1 AS SYSDBA
SQL> CREATE PLUGGABLE DATABASE pxpdb1 AS PROXY
 FROM pdb1@link_cdb2;
```

5. Execute all the statements in the **PXPDB1** proxy PDB context to have them executed in the proxied **PDB1** PDB in **CDB2**.

```
SQL> CONNECT sys@pxpdb1 AS SYSDBA
SQL> ALTER PLUGGABLE DATABASE pxpdb1 OPEN;
SQL> SELECT * FROM app.c;
```

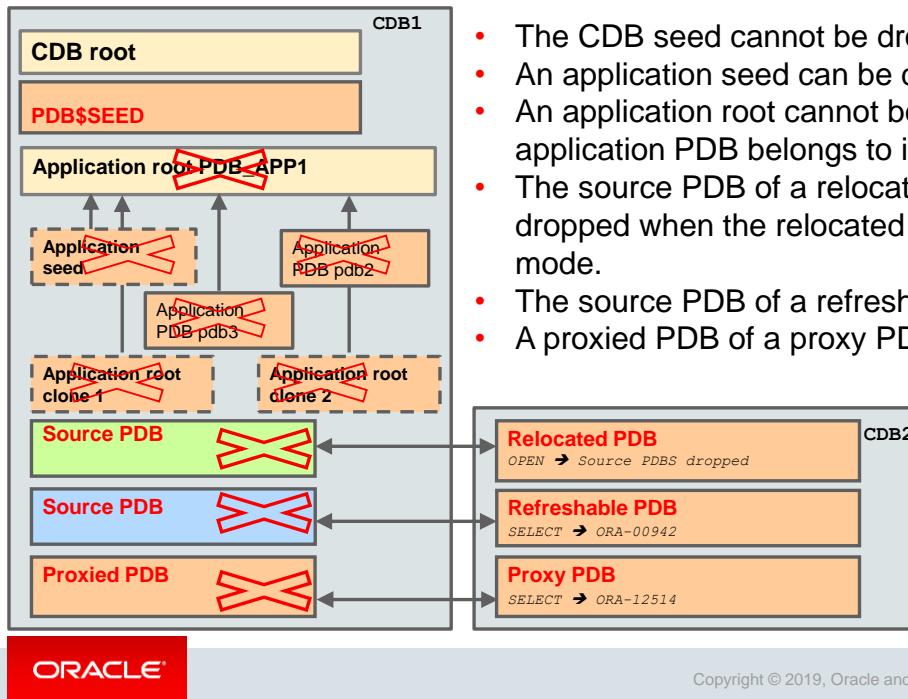
Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Creating a proxy PDB copies the datafiles of the SYSTEM, SYSAUX, and UNDO tablespaces of the proxied PDB. A proxy PDB can be created, altered, and dropped from the CDB root like any regular PDB.

The database link must be created in the CDB root that will contain the proxy PDB, and the database link connects either to the remote CDB root or to the remote application container or to the remote application PDB.

Any `ALTER PLUGGABLE DATABASE` statement issued from within the proxy PDB when the PDB is opened is executed in the proxied PDB. The new `AS PROXY` clause is used to create a PDB as a proxy PDB. The new `IS_PROXY_PDB` column in `CDB_PDBS` displays if a PDB is a proxy PDB.

## Dropping PDBs



- The CDB seed cannot be dropped.
- An application seed can be dropped.
- An application root cannot be dropped as long as an application PDB belongs to it.
- The source PDB of a relocated PDB is automatically dropped when the relocated PDB is opened in RW mode.
- The source PDB of a refreshable PDB can be dropped.
- A proxied PDB of a proxy PDB can be dropped.

The **DROP** operation updates controlfiles:

1. Removes PDB datafiles
2. Retain datafiles (default)

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

When you no longer need the data in a PDB, you can drop the PDB. There is only one PDB that cannot be dropped: It is the CDB seed.

- You cannot drop an application root as long as there are still application PDBs associated with it. First drop the application seed and then the application PDBs.
- When the relocation of a PDB is finished, opening the new PDB automatically drops the source PDB.
- Dropping the source PDB of a refreshable PDB does not drop the refreshable PDB. Queries on the source data are no longer possible.
- Dropping the proxied PDB of a proxy PDB does not drop the proxy PDB. Queries on the proxied PDB are no longer possible because the database link to the proxied PDB is no longer valid.

## Summary

In this lesson, you should have learned how to:

- Clone a regular PDB
- Clone an application container
- Unplug and plug or clone a non-CDB
- Unplug and plug a regular PDB
- Unplug and plug an application container
- Convert regular PDBs to application PDBs
- Configure and use the local UNDO mode
- Perform hot cloning
- Perform near-zero downtime PDB relocation
- Create and use a proxy PDB
- Drop PDBs



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Practice 4: Overview

- 4-1: Cloning remote regular PDBs in hot mode
- 4-2: Cloning an application container
- 4-3: Unplugging and plugging application containers
- 4-4: Converting a regular PDB to an application PDB
- 4-5: Relocating PDBs
- 4-6: Querying data across CDBs by using proxy PDBs
- 4-7: Dropping unnecessary PDBs



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

# CDB and PDB Management

The ORACLE logo, featuring the word "ORACLE" in white capital letters inside a red rectangular box.

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

# Objectives

After completing this lesson, you should be able to:

- Establish connections to a CDB / PDB
- Avoid service name conflicts
- Start PDB service
- Start up and shut down a CDB
- Open and close PDBs
- Change the different modes and settings of PDBs
- Evaluate the impact of parameter value changes
- Configure host name and port number per PDB



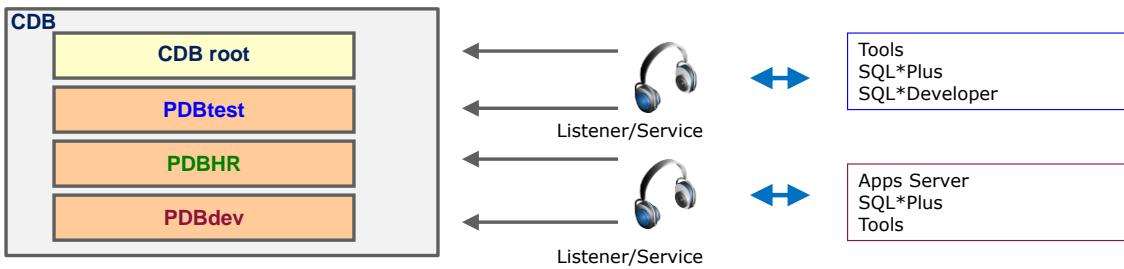
ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

For a complete understanding of the PDB management procedures, refer to the following guide in the Oracle documentation:

- *Oracle Multitenant Administrator's Guide 18c*

## Connection



1. Every PDB has a default service.

```
SQL> SELECT name, pdb FROM cdb_services;
```

2. Service name has to be unique across CDBs.

```
SQL> CONNECT / AS SYSDBA
SQL> CONNECT sys@CDB1 AS SYSDBA
SQL> CONNECT sys@PDBtest AS SYSDBA
SQL> CONNECT local_user1@hostname1:1525/PDBHR
SQL> CONNECT common_user2@PDBdev
SQL> SHOW CON_NAME
```

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Before performing any maintenance and management operation in PDBs, consider how you connect to a CDB and to a particular PDB. Any container in a CDB owns a service name.

- The CDB root service name is the CDB name given at the CDB creation concatenated with the domain name.
- Each new PDB is assigned a service name: the service name is the PDB name given at PDB creation concatenated with the domain name. If you create or plug a PDBtest PDB, its service name is PDBtest concatenated with the domain name. The container service names must be unique within a CDB, and even across CDBs that register with the same listener.
- You can find service names maintained in CDB and PDBs in CDB\_SERVICES or V\$SERVICES views. The PDB column shows the PDB to which the services are linked.  
If your database is being managed by Oracle Restart or Oracle Clusterware RAC, you can view the services names using Server Control utility as follows:

```
srvctl config database -db cdb1
```

To connect to the CDB root, use local OS authentication or the CDB root service name. For example, if you set the ORACLE\_SID to the CDB instance name and use the command CONNECT / AS SYSDBA, you are connected to the CDB root under common SYS user granted system privileges to manage and maintain all PDBs.

With the service name, you can use the EasyConnect syntax or the alias from tnsnames.ora.

Using EasyConnect, you would enter the following connect string:

```
SQL> CONNECT username@hostname:portnumber/service_name
SQL> CONNECT username@localhost:portnumber/service_name
```

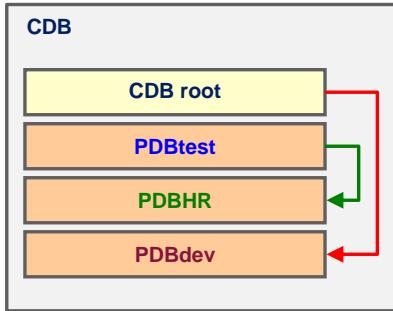
Using the tnsnames.ora file:

```
SQL> CONNECT username@net_service_name
```

# Switching Connection

Two possible ways to switch connection between containers within a CDB:

- Reconnect: Allows connection under common or local user



```

SQL> CONNECT / AS SYSDBA
SQL> CONNECT local_user1@PDBdev

```

- Use ALTER SESSION SET CONTAINER statement:

```

SQL> CONNECT sys@PDBtest AS SYSDBA
SQL> ALTER SESSION SET CONTAINER=PDBHR;
SQL> SHOW CON_NAME
SQL> ALTER SESSION SET CONTAINER=CDB$ROOT;

```

Allows connection under common user only who is granted new system privilege SET CONTAINER.

- AFTER LOGON triggers do not fire.
- Transactions are still pending after switching containers.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

A CDB administrator can connect to any container in the CDB using either CONNECT or ALTER SESSION SET CONTAINER to switch between containers. For example, the CDB administrator can connect to the CDB root in one session and then in the same session switch to the PDBHR container, the requirement here being a common user known in both containers, and a system privilege SET CONTAINER.

- Using the command CONNECT allows connections under common or local users.
- Using ALTER SESSION SET CONTAINER only allows connections under a common user who is granted the new system privilege SET CONTAINER.
  - Be aware that AFTER LOGON triggers do not fire.
  - Transactions that are not committed or rolled back in the original container are still in a pending state while switching to another container and when switching back to the original container.

# Creating Services

- Using the **DBMS\_SERVICE** package in an environment without Oracle Restart:

```
SQL> EXEC DBMS_SERVICE.CREATE_SERVICE('hrpdb', 'hrpdb')
```

```
SQL> EXEC DBMS_SERVICE.START_SERVICE('hrpdb')
```

- Using the **SRVCTL** utility in a Grid Infrastructure environment with Oracle Restart:

```
$ srvctl add service -db mycdb -service hrpdb -pdb hrpdb
```

```
$ srvctl start service -db mycdb -service hrpdb
```

- Oracle Restart configuration automatically updated:

| Create operations and the Oracle Restart configuration     | Automatically added to configuration? |
|------------------------------------------------------------|---------------------------------------|
| Create a database service with SRVCTL                      | YES                                   |
| Create a database service with DBMS_SERVICE.CREATE_SERVICE | NO                                    |



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Creating Additional Services by Using the **DBMS\_SERVICE** Package

If your database is not being managed by Oracle Restart, you can create or modify additional services for each PDB using the **DBMS\_SERVICE** package.

```
SQL> CONNECT system@salespdb
SQL> EXEC DBMS_SERVICE.CREATE_SERVICE('hrpdb', 'hrpdb')
SQL> EXEC DBMS_SERVICE.START_SERVICE('hrpdb')
```

## Creating Additional Services by Using the **SRVCTL** Utility

If your database is being managed by Oracle Restart, you can create or modify additional services for each PDB using the Server Control utility as follows:

```
$ srvctl add service -db mycdb -service hrpdb -pdb hrpdb
```

This example adds the `hrpdb` additional service for the `hrpdb` PDB in the `mycdb` CDB. If the `pdb` option is set to an empty string, then the additional service is associated with the CDB root.

Oracle Restart maintains a list of all the components that it manages and maintains configuration information for each component. When Oracle Restart is installed, many operations that create Oracle components using Oracle utilities will automatically add the components to the Oracle Restart configuration. If a component is created manually, then `SRVCTL` commands can be used to add it to the Oracle Restart configuration if desired. The table in the above slide shows which `CREATE` operation automatically adds the component to the Oracle Restart configuration and which operations do not.

# Renaming Services

- Renaming PDB service to avoid name conflicts

```
SQL> CREATE PLUGGABLE DATABASE pdb1 ... FROM pdb1@link_node1
 SERVICE_NAME_CONVERT = ('pdb1_node1', 'pdb1_node2');
```

- Starting a PDB service at PDB opening

```
SQL> ALTER PLUGGABLE DATABASE pdb1 OPEN
 SERVICES = ('pdb1_node2');
```



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The `SERVICE_NAME_CONVERT` clause can be used to avoid conflicts with the names of existing services when cloning from another PDB or plugging a PDB because the new PDB inherits services from the source. This works only on managed services and not the default service. Then the `SERVICES` clause can be used when opening the new PDB to automatically start the new service. By default `SERVICES` is set to `NONE`.

## Starting Up a CDB Instance

```
SQL> CONNECT sys@CDB1 AS SYSDBA
SQL> STARTUP NOMOUNT
```

```
SQL> SELECT name, open_mode FROM v$pdbs;

no rows selected
```

NOMOUNT

CDB instance started

SHUTDOWN



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

In a non-RAC environment, a CDB runs with a single instance. The instance is started exactly the same way as for non-CDB databases, using a `STARTUP NOMOUNT` statement. You need to be connected to the CDB root as `SYSDBA` to start the instance up.

Use the `V$PDBS` view to view the open mode of PDBs.

## Mounting a CDB

```
SQL> CONNECT sys@CDB1 AS SYSDBA
SQL> STARTUP MOUNT
```

```
SQL> SELECT name, open_mode
 FROM v$pdbs;
```

| NAME      | OPEN MODE |
|-----------|-----------|
| PDB\$SEED | MOUNTED   |
| PDB1      | MOUNTED   |
| PDB2      | MOUNTED   |

NOMOUNT

Instance started

SHUTDOWN

MOUNT

- CDB control files opened for the instance
- CDB root mounted
- PDBs mounted

ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

A CDB may be started up with the `MOUNT` option. The command used to mount a CDB is the same as for non-CDB databases, using a `STARTUP MOUNT` statement. Again, you need to be connected to the CDB root as `SYSDBA` to perform this operation.

When a CDB is mounted, the CDB root is mounted, which means that the control files are opened, as well as the PDBs.

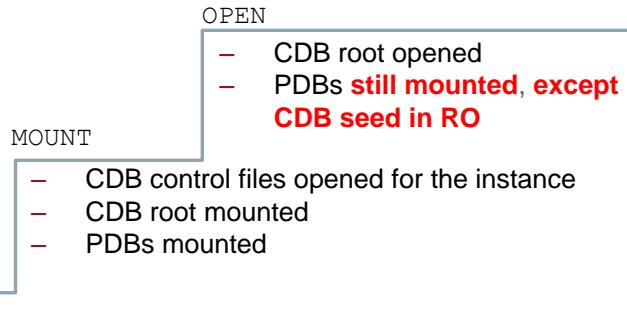
Use the `open_mode` column from the `V$PDBS` view to verify that all PDBs are mounted.

# Opening a CDB

```
SQL> STARTUP
```

```
SQL> SELECT name, open_mode
 FROM v$pdbs;
```

| NAME      | OPEN_MODE |
|-----------|-----------|
| PDB\$SEED | READ ONLY |
| PDB1      | MOUNTED   |
| PDB2      | MOUNTED   |



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

When a CDB is opened, the CDB root is opened, which means that all redo log files and CDB root datafiles are opened while all PDBs are still only mounted. Only connections to the CDB root allow operations. Separate statements need to be issued to open PDBs, unless triggers automatically open PDBs after `STARTUP DATABASE`.

Use the `open_mode` column from the `V$PDBS` view to verify that all PDBs are still mounted, except the CDB seed being in read only mode. This allows creating new PDBs from it.

## Opening a PDB

```
SQL> ALTER PLUGGABLE DATABASE pdb2 OPEN;
```

```
SQL> ALTER PLUGGABLE DATABASE ALL OPEN;
```

```
SQL> SELECT name, open_mode
 FROM v$pdbs;
```

| NAME      | OPEN_MODE  |
|-----------|------------|
| PDB\$SEED | READ ONLY  |
| PDB1      | READ WRITE |
| PDB2      | READ WRITE |

PDB OPEN

PDBs opened RW,  
except CDB seed in RO

OPEN

- CDB root opened
- PDBs still mounted, **except CDB seed in RO**

MOUNT

- CDB control files opened for the instance
- CDB root mounted
- PDBs mounted

NOMOUNT

Instance  
started

SHUTDOWN

ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

To open a PDB or some of them or all of them, connect to the CDB root as **SYSOPER** or **SYSDBA** and issue an **ALTER PLUGGABLE DATABASE OPEN** statement, specifying the PDB or PDBs names or **ALL EXCEPT** or **ALL**.

This operation opens the datafiles of the PDBs opened and provides availability to users.

Use the **open\_mode** column from the **V\$PDBS** view to verify that all PDBs are all in **READ WRITE** open mode except the CDB seed being still in **READ ONLY** open mode.

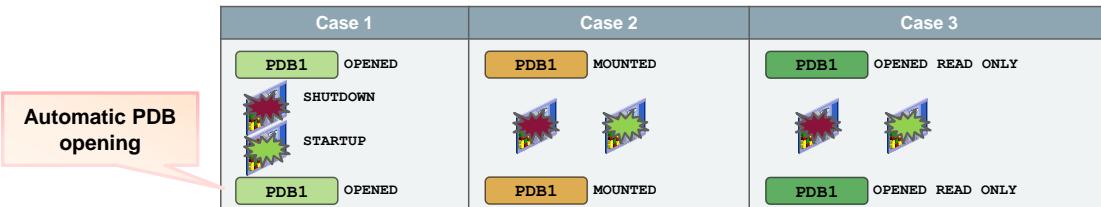
You can also open a PDB while connected as **SYSDBA** within the PDB. In this case, it is not necessary to name the PDB to open.

**Note:** Opening an application PDB requires that the application root it is associated to is already opened.

# Automatic PDB Opening

- Automatically keep PDBs state after CDB STARTUP:

```
SQL> ALTER PLUGGABLE DATABASE pdb1 SAVE STATE;
```



- Automatically discard PDBs state after CDB STARTUP:

```
SQL> ALTER PLUGGABLE DATABASE pdb1 DISCARD STATE;
```



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

After restarting a CDB instance, the PDBs are by default kept in mounted mode. If you want the PDBs to automatically open whenever the CDB restarts, use the `SAVE STATE` clause of the `ALTER PLUGGABLE DATABASE` command to preserve a PDB's open mode across CDB restart. The `SAVE STATE` clause saves the last open state of the PDB. So, the PDB will open after the CDB restart only if the PDB was in the open state when the `SAVE STATE` clause was used to save the last state. To revert back to the default behavior, use the `DISCARD STATE` clause.

## Closing a PDB

```
SQL> CONNECT / AS SYSDBA
SQL> ALTER PLUGGABLE DATABASE pdb1 CLOSE IMMEDIATE;
SQL> ALTER PLUGGABLE DATABASE ALL EXCEPT pdb1, pdb2 CLOSE;
SQL> ALTER PLUGGABLE DATABASE ALL CLOSE;
```

PDB CLOSE

PDBs closed

```
SQL> CONNECT sys@pdb1 AS SYSDBA
SQL> ALTER PLUGGABLE DATABASE CLOSE;
Or
SQL> SHUTDOWN IMMEDIATE;
```

CDB OPEN

MOUNT

NOMOUNT

SHUTDOWN

Instance started

ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

To close a PDB or some of them or all of them, connect to the CDB root as `SYSOPER` or `SYSDBA` and issue an `ALTER PLUGGABLE DATABASE CLOSE` statement, specifying the PDB(s) name(s) or `ALL EXCEPT` or `ALL`. If you use the clause `CLOSE IMMEDIATE`, the transactions in the selected PDBs are roll-backed and the sessions disconnected. If you omit the `IMMEDIATE` clause, the statement waits until all sessions are disconnected.

This operation closes the datafiles of the closed PDBs and prevents availability to users. Though all PDBs are closed, it is still possible to perform operations from the CDB root, such as dropping PDBs or creating new PDBs from the CDB seed.

The statement `SHUTDOWN IMMEDIATE` when connected to a PDB is equivalent to `ALTER PLUGGABLE DATABASE CLOSE`. It closes the PDB.

**Note:** Though `SHUTDOWN IMMEDIATE` issues the traditional message `ORACLE instance shut down`, this does not mean that the instance is down. Understand that the PDB is closed.

If the PDB is already closed, then the message explains the situation clearly with:

```
SQL> shutdown immediate
ORA-65020: Pluggable database already closed
```

**Note:** Closing an application root automatically closes the application PDBs associated to it.

## Shutting Down a CDB Instance

```
SQL> CONNECT sys@CDB1 AS SYSDBA
SQL> SHUTDOWN IMMEDIATE
```

- All PDBs closed (no new specific message)
- CDB closed
- CDB dismounted
- Instance shut down

```
SQL> CONNECT sys@PDB1 AS SYSDBA
SQL> SHUTDOWN IMMEDIATE
```

- PDB closed



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

When a CDB is shut down, the datafiles of the CDB root and all PDBs are closed; then all the control files are closed, and in the last step the instance is shut down.

When a PDB is shut down, this means that the datafiles of the PDB are closed.

# Changing PDB Mode

After closing a PDB, open in:

- Restricted read-write

```
SQL> CONNECT sys@pdb1 AS SYSDBA
SQL> ALTER PLUGGABLE DATABASE CLOSE;
```

```
SQL> ALTER PLUGGABLE DATABASE OPEN RESTRICTED;
```

```
SQL> SELECT name, open_mode FROM v$pdbs;
```

| NAME | OPEN MODE  | RES |
|------|------------|-----|
| PDB1 | READ WRITE | YES |

- Read only mode

```
SQL> CONNECT / AS SYSDBA
SQL> ALTER PLUGGABLE DATABASE ALL OPEN READ ONLY;
```

- Read write



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

You can change the mode of each PDB to perform specific administration operations.

The first example opens the PDB in the RESTRICTED READ WRITE mode. This allows only users with the RESTRICTED SESSION privilege to connect. This allows the local administrator of the PDB to manage files movement, backups preventing sessions from accessing the data.

Use the V\$PDBS view to verify that the PDB is in RESTRICTED READ WRITE open mode.

The second example opens the PDB in a READ ONLY mode. Any session connected to the PDB can perform read-only transactions only.

To change the open mode, first close the PDB. You can apply the same open mode to all PDBs or to some of them.

## Modifying PDB Settings

- Bring a PDB datafile online
- Change the PDB default tablespace
- Change the PDB default temporary tablespace
- Set the PDB storage limit
- Change the global name

```
SQL> CONNECT sys@pdb1 AS SYSDBA
SQL> ALTER PLUGGABLE DATABASE DATAFILE '/u03/pdb1_01.dbf' ONLINE;
```

```
SQL> ALTER PLUGGABLE DATABASE DEFAULT TABLESPACE pdb1_tbs;
```

```
SQL> ALTER PLUGGABLE DATABASE DEFAULT TEMPORARY TABLESPACE temp_tbs;
```

```
SQL> ALTER PLUGGABLE DATABASE STORAGE (MAXSIZE 2G);
```

```
SQL> ALTER PLUGGABLE DATABASE RENAME GLOBAL_NAME TO pdbAPP1;
```



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

You can modify settings of each PDB without necessarily changing the mode of the PDB. You have to be connected in the PDB to perform the settings changes.

The first example uses a `DATAFILE` clause to bring the datafile online.

The second example sets the default permanent tablespace to `pdb1_tbs` for the PDB.

The third example sets the default temporary tablespace to `temp_tbs` for the PDB.

The fourth example sets the storage limit for all tablespaces that belong to the PDB to two gigabytes.

The fifth example changes the global database name of the PDB to `pdbAPP1`. The new global database name for this PDB must be different from that of any container in the CDB, and this operation can be done only in restricted mode.

# Instance Parameter Change Impact

- A single SPFILE per CDB
- PDB values change:
  - Loaded in memory after PDB close
  - Stored in dictionary after CDB shutdown
  - Only for parameter `ISPDB_MODIFIABLE=TRUE`

```
SQL> CONNECT sys@pdb1 AS SYSDBA
Connected.
SQL> ALTER SYSTEM SET ddl_lock_timeout=10;
System altered.
SQL> SHOW PARAMETER ddl_lock_timeout

NAME TYPE VALUE
----- boolean 10
```



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

There is a single SPFILE per CDB to store parameters. Values of parameters are associated with the CDB root and apply to the CDB root and serve as default values for all other containers.

You can set different values in PDBs for the parameters where the column `ISPDB_MODIFIABLE` in `V$PARAMETER` is `TRUE`. These are set in the scope of a PDB; then they are remembered properly across PDB close/open and across bouncing the CDB instance. They also travel with clone and unplug/plug operations. Other initialization parameters can be set for the CDB root only.

Connect to the CDB root to view all containers' specific values of parameters.

```
SQL> select DB_UNIQ_NAME, PDB_UID, NAME, VALUE$ from pdb_spfile$;
```

| DB_UNIQ_NAME | PDB_UID    | NAME             | VALUE\$ |
|--------------|------------|------------------|---------|
| cdb2         | 3072231663 | ddl_lock_timeout | 10      |
| cdb2         | 4030283986 | ddl_lock_timeout | 20      |
| cdb2         | 3485283967 | ddl_lock_timeout | 30      |

## Instance Parameter Change Impact: Example

```
SQL> CONNECT sys@pdb2 AS SYSDBA

SQL> ALTER SYSTEM SET ddl_lock_timeout=20 SCOPE=BOTH;

SQL> ALTER PLUGGABLE DATABASE CLOSE;
SQL> ALTER PLUGGABLE DATABASE OPEN;
```

```
SQL> CONNECT / AS SYSDBA
SQL> SELECT value, ispdbs_modifiable, con_id FROM v$system_parameter
 WHERE name = 'ddl_lock_timeout';

VALUE ISPDB CON_ID
----- ----- -----
0 TRUE 0
10 TRUE 3
20 TRUE 4
```



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

In this example, a different value of the `DDL_LOCK_TIMEOUT` parameter is set in `pdb2`. The value changes are remembered after the PDB closes and opens. The new column `CON_ID` in the `V$SYSTEM_PARAMETER` view shows the `DDL_LOCK_TIMEOUT` value in each container, the CDB root, `pdb1`, and `pdb2`.

## Using ALTER SYSTEM Statement on PDB

- Some statements change the way a PDB operates:

| ALTER SYSTEM Affecting the PDB only            | Objects Impacted              |
|------------------------------------------------|-------------------------------|
| ALTER SYSTEM FLUSH SHARED_POOL                 | Only for objects of the PDB   |
| ALTER SYSTEM FLUSH BUFFER_CACHE                | Only for buffers of the PDB   |
| ALTER SYSTEM ENABLE/DISABLE RESTRICTED SESSION | Only for sessions of the PDB  |
| ALTER SYSTEM KILL SESSION                      | Only for sessions of the PDB  |
| ALTER SYSTEM SET <i>parameter</i>              | Only for parameter of the PDB |

- Some ALTER SYSTEM statements can be executed in a PDB but affect the whole CDB:
- |                             |                                                            |
|-----------------------------|------------------------------------------------------------|
| ALTER SYSTEM CHECKPOINT     | Affects all datafiles except those in read only or offline |
| ALTER SYSTEM SWITCH LOGFILE | Operation not allowed from within a pluggable database     |



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

You can use an ALTER SYSTEM statement to change the way a PDB operates. When the current container is a PDB, you can run the following ALTER SYSTEM statements:

```
ALTER SYSTEM FLUSH SHARED_POOL / BUFFER_CACHE
ALTER SYSTEM ENABLE / DISABLE RESTRICTED SESSION
ALTER SYSTEM SET USE_STORED_OUTLINES
ALTER SYSTEM SUSPEND / RESUME
ALTER SYSTEM CHECK DATAFILES
ALTER SYSTEM REGISTER
ALTER SYSTEM KILL SESSION
ALTER SYSTEM DISCONNECT SESSION
ALTER SYSTEM SET initialization_parameter
```

Some ALTER SYSTEM statements can be run from within a PDB but still affect the whole CDB such as ALTER SYSTEM CHECKPOINT.

Other ALTER SYSTEM statements affect the entire CDB and must be run by a common user in the CDB root such as ALTER SYSTEM SWITCH LOGFILE, unless you set the NONCDB\_COMPATIBLE parameter to TRUE, which enables you to get a behavior similar to a non-CDB when issuing SQL commands inside a PDB if you are using a single PDB in your CDB configuration. This parameter influences ALTER DATABASE statements behavior in the same way, like ALTER DATABASE BACKUP CONTROLFILE TO TRACE statement.

## Configuring Host Name and Port Number per PDB

```
DATABASE_PROPERTIES
CONTAINERS_HOST=host1
CONTAINERS_PORT=1522
```

- The host name and port number settings for a PDB are important only if proxy PDBs will reference the PDB.

```
SQL> ALTER PLUGGABLE DATABASE CONTAINERS HOST = <host_name>;
SQL> ALTER PLUGGABLE DATABASE CONTAINERS PORT = <port_nb>;
```

- The host name and port number can be reset to their default:

```
SQL> ALTER PLUGGABLE DATABASE CONTAINERS HOST RESET;
SQL> ALTER PLUGGABLE DATABASE CONTAINERS PORT RESET;
```



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

During PDB creation, the `HOST=<host_name_string>` and `PORT=<port_number>` clauses can be used to indicate the host name and port number to be used by internal database links from proxy PDBs that reference the new PDB. By default, altering the host name and port number for a PDB does not have any effect on internal database links that already have been created to point to this PDB. Only subsequently created internal database links will be affected by the new host name and port number. A proxy PDB will have to be re-created in order for it to pick up the new port number for its target PDB.

## Summary

In this lesson, you should have learned how to:

- Establish connections to a CDB / PDB
- Avoid service name conflicts
- Start PDB service
- Start up and shut down a CDB
- Open and close PDBs
- Change the different modes and settings of PDBs
- Evaluate the impact of parameter value changes
- Configure host name and port number per PDB



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Practice 5: Overview

- 5-1: Starting up and shutting down a CDB
- 5-2: Opening and closing PDBs
- 5-3: Renaming a PDB
- 5-4: Setting parameter values for PDBs
- 5-5: Renaming PDB services



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.



# Storage

ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

# Objectives

After completing this lesson, you should be able to manage:

- Permanent tablespaces in CDB and PDBs
- Temporary tablespaces in CDB and PDBs
- UNDO tablespaces in CDB root and PDB



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

For a complete understanding of the PDB storage procedures, refer to the following guide in the Oracle documentation:

- *Oracle Multitenant Administrator's Guide 18c*

## Objects in Tablespaces

- A tablespace in a PDB can contain objects associated with exactly one PDB.
- An object can be common to multiple PDBs.
  - Regular PDBs:
    - Common objects exist in the CDB root only.
    - Data is stored in the CDB root only.
    - Or each PDB has a private data copy.
    - CANNOT be created by users
    - Can be created by Oracle-supplied users
  - Application PDBs: Common objects exist in the application root
    - The object definition exists in the application root only.
    - Data is stored in the application root only.
    - Or data is stored in both the application root and the application PDB.
    - Or data is stored in the application PDB.
    - Can be created by users



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

In a non-CDB, all the tablespaces belong to one database.

In a CDB, one set of tablespaces belong to the CDB root, and each PDB has its own set of tablespaces.

Common objects, introduced with the Multitenant architecture, avoid having to store redundant representations of data and metadata across a CDB and simplify the process of upgrading a CDB. Common objects exist in Oracle-supplied schemas.

- Common objects at the CDB level are created in the CDB root and their data stored in a tablespace in the CDB root. The common object is visible in the PDBs through links.
  - Metadata-linked objects store metadata about dictionary objects only in the CDB root. Each PDB has a private data copy of an object pointing to a metadata link stored in the CDB root.
  - A data-linked object and its data reside in the CDB root only and are shared by all PDBs.
  - You can find the type of sharing in the SHARING column of DBA\_OBJECTS.
- Application common objects at the application root level are created in the application root.
  - A metadata-linked object stores the object definition in an application root only. Each application PDB has a private data copy pointing to a metadata-linked object that is stored in an application root.
  - A data-linked object and its data reside in an application root only and are shared by all application PDBs.
  - An extended data-linked object combines data found in a table in an application PDB with data from a corresponding table in the application root.

The common objects are covered in the lesson titled “Security.”

## Tablespaces Created During PDB Creation

Define the default tablespace of a new PDB:

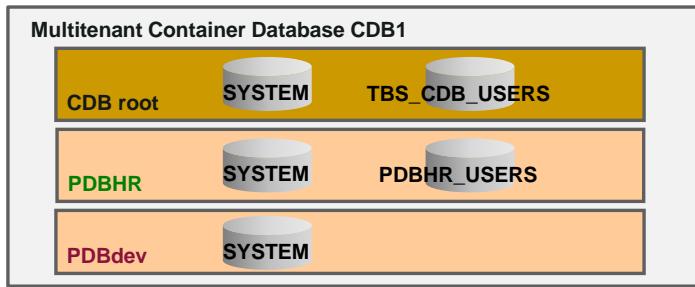
- Created from the CDB seed
- During PDB cloning
- Cloning with `USER_TABLESPACES` and `COMPATIBLE=12.2`



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

- You can define a default tablespace at PDB creation from another PDB through a cloning operation, provided that the tablespace in the source PDB already exists.
- The `USER_TABLESPACES` clause is extended to allow users to specify a creation mode for a subset of user tablespaces, which could be different from that specified outside of this clause in the `CREATE PLUGGABLE DATABASE` statement and would then apply only to Oracle tablespaces (for example, `SYSTEM, SYSAUX`): `USER_TABLESPACES=(<ts_list>) | NONE | ALL [EXCEPT (<ts_list>)]`  
If `COMPATIBLE` is set to 12.2 or later, skipped tablespaces are still created as offline, but have no datafiles associated (which is different from the pre-12.2 behavior where the control file would include datafiles belonging to such tablespaces, marking them as `UNNAMED`).

## Defining Default PermanentTablespaces



- In the CDB:

```
SQL> CONNECT system@cdb1
SQL> ALTER DATABASE DEFAULT TABLESPACE tbs_CDB_users;
```

- In the PDB:

```
SQL> CONNECT pdb1_admin@pdbhr
SQL> ALTER PLUGGABLE DATABASE DEFAULT TABLESPACE pdbhr_users;
```



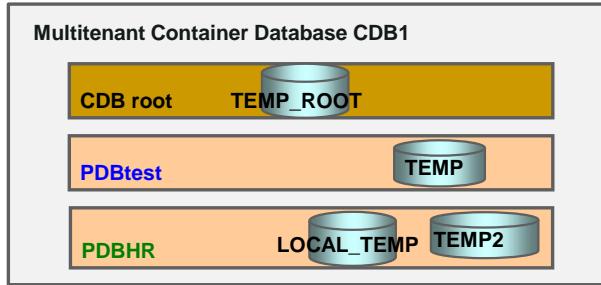
Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The default tablespace for a database is a database property. To change the default tablespace for a CDB root container, you must connect to the CDB root container as a user with the proper privileges and issue the `ALTER DATABASE` command. This operation does not change the default permanent tablespace of PDBs.

To change the default tablespace for a PDB, you must connect to the PDB as a user with proper permissions and issue the `ALTER PLUGGABLE DATABASE` command. When connected to the PDB, the `ALTER DATABASE` and `ALTER PLUGGABLE DATABASE` commands perform the same modifications to the PDB. The `ALTER DATABASE` command is allowed for backward compatibility.

## Temporary Tablespaces

- Only one default temporary tablespace or tablespace group is allowed per CDB or PDB.
- Each PDB can have temporary tablespaces or tablespace groups.



- Define the default temporary tablespace in a PDB:

```
SQL> CONNECT pdb1_admin@pdbhr
SQL> ALTER DATABASE DEFAULT TEMPORARY TABLESPACE local_temp;
```

ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

A CDB can have only one default temporary tablespace or tablespace group. As with a non-CDB, there can be other temporary tablespaces to which users can be assigned.

PDBs must have their own temporary tablespace (or tablespace group) for use by users in the PDB. These temporary tablespaces will be transported with the PDB when it is unplugged.

Creating a local user requires a default temporary tablespace in the PDB to be assigned.

Creating a common user requires that the default temporary tablespace exists in the container where it is replicated.

The default temporary tablespace for the CDB is set at the CDB root level. There may be multiple temporary tablespaces, but only one can be the default.

When you create a user, you can specify a temporary tablespace to be used by that user. If a temporary tablespace is not specified, the default tablespace for the PDB is used.

The amount of space a PDB can use in the shared temporary tablespace can be limited :

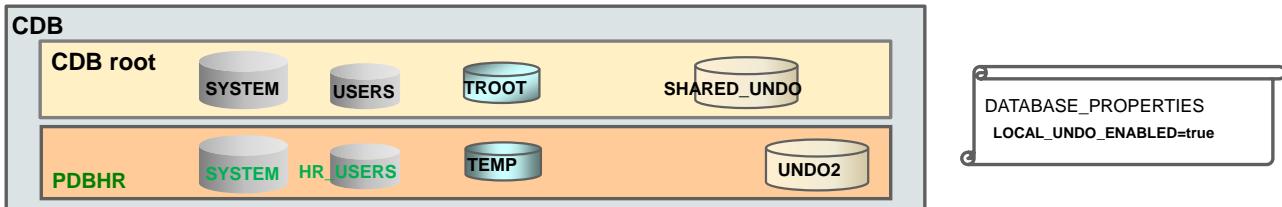
```
ALTER PLUGGABLE DATABASE STORAGE (MAX_SHARED_TEMP_SIZE 500M);
```

In this example, if the value used by sessions that are connected to the PDB is greater than 500M, then no additional storage in the shared temporary tablespace will be available for sessions connected to the PDB until the amount of storage used by them becomes smaller than 500M.

# UNDO Tablespaces

Two UNDO modes: SHARED versus LOCAL

- There is only one shared UNDO tablespace (in CDB root).
- There can be a local UNDO tablespace in each PDB.



When is local UNDO mode required?

- Hot cloning
- Near-zero downtime PDB relocation

```
SQL> STARTUP UPGRADE
SQL> ALTER DATABASE LOCAL UNDO ON;
```



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Using the local UNDO mode is required when cloning a PDB in hot mode or performing a near-zero downtime PDB relocation or refreshing PDBs or using proxy PDBs. UNDO mode is the default mode.

You can set a CDB in shared or local UNDO mode either at CDB creation or by altering the CDB property.

- When the database property **LOCAL\_UNDO\_ENABLED** is **FALSE**, there is only one UNDO tablespace that is created in the CDB root, and that is shared by all PDBs.
- When **LOCAL\_UNDO\_ENABLED** is **TRUE**, every container in the CDB uses local undo, and each PDB must have its own local UNDO tablespace. To maintain ease of management and provisioning, UNDO tablespace creation happens automatically and does not require any action from the user. When a PDB is opened and an UNDO tablespace is not available, it is automatically created.

## Summary

In this lesson, you should have learned how to manage:

- Permanent tablespaces in CDB and PDBs
- Temporary tablespaces in CDB and PDBs
- UNDO tablespaces in CDB root and PDB



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Practice 6: Overview

- 6-1: Managing permanent and temporary tablespaces
- 6-2: Managing UNDO tablespaces



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.



# Security

ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

# Objectives

After completing this lesson, you should be able to:

- Manage common and local users, roles, privileges, and profiles in PDBs
- Manage common and local objects in application containers
- Enable common users to access data in PDBs
- Manage PDB lockdown profiles
- Audit users in CDB and PDBs
- Manage other types of policies in application containers
- Protect data with Database Vault policies in CDB and PDBs
- Encrypt data in PDBs
- Configure isolated PDB keystores
- Unplug and plug an encrypted PDB in a one-step operation
- Allow per-PDB wallets for certificates



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

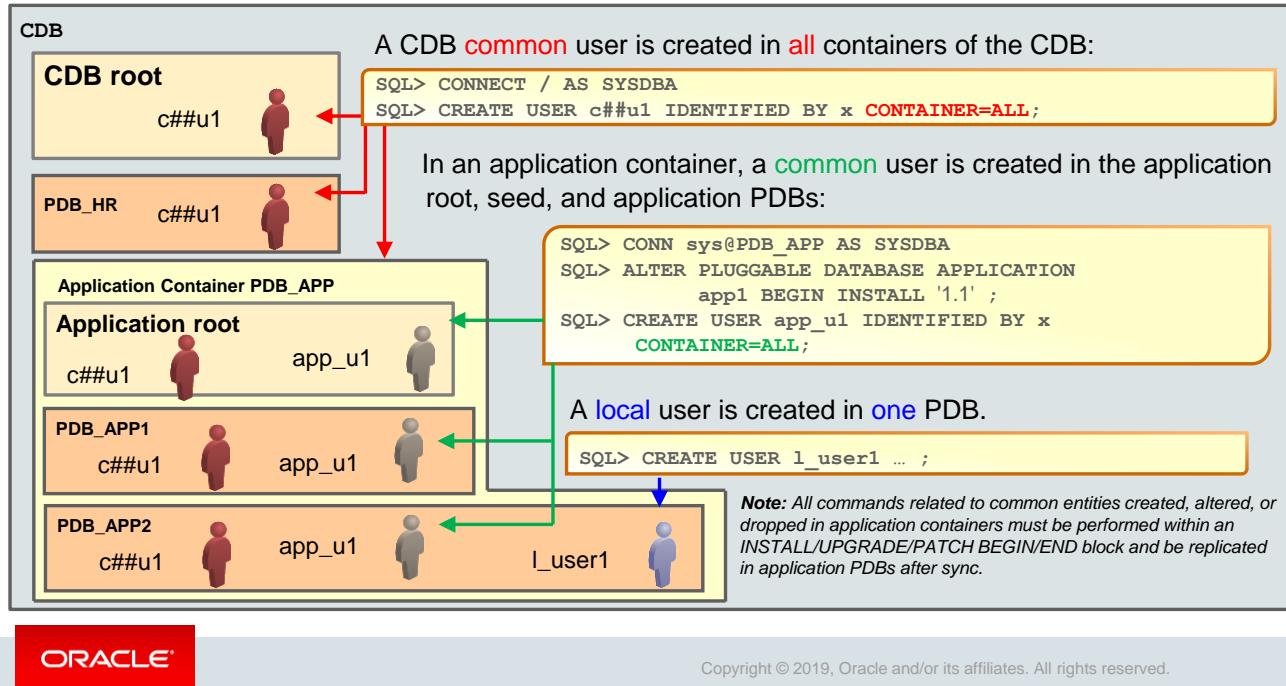
For a complete understanding of the CDB and PDB security procedures, refer to the following Oracle Database 18c documentation:

- “Using Oracle Database Vault with a CDB” in *Oracle Multitenant Administrator’s Guide*
- *Oracle Database Security Guide*
- *Oracle Database Advanced Security Guide*
- *Oracle Database Vault Administrator’s Guide*

Refer to other sources of information available under Oracle Learning Library:

- *Oracle By Example (OBE)*:
  - *creating\_and\_using\_dynamic\_lockdown\_profiles* in *Learning Path: 18c New Features for Multitenant*
  - *creating\_and\_using\_lockdown\_profiles\_at\_different\_levels\_in\_cdb* in *Learning Path: 18c New Features for Multitenant*
  - *managing\_pdb\_keystores* in *Learning path: 18c New Features for Security*

## Creating Common Users in the CDB and PDBs



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

A common user is a user that has the same user name and authentication credentials across multiple PDBs of the CDB or an application container, unlike a local user that exists in only one PDB.

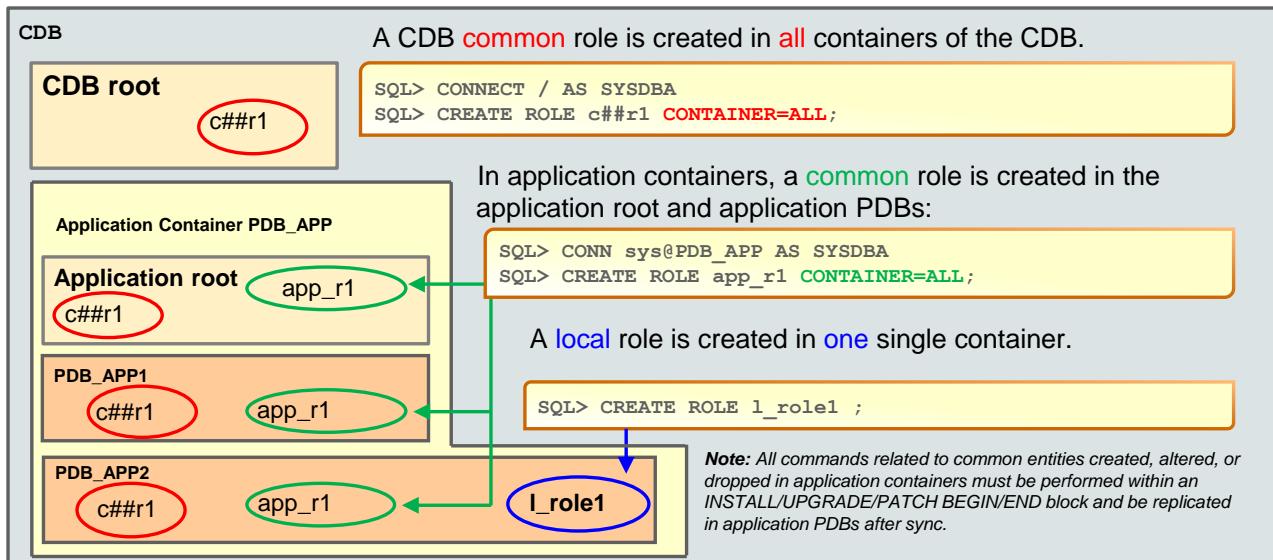
A common user cannot have the same name as any local user across all the PDBs. A common user can be created in the CDB root or in an application root: a common user is a database user that has the same identity in the CDB root and in every existing and future PDBs in the CDB or in an application root and in every existing and future application PDBs in the application container. An application common user does not require a prefix like a CDB common user.

To create an application common user, you must be logged in to the application root. The application common user is replicated in all application PDBs when the application PDBs are synchronized with the application root.

A local user can be created in a specific PDB and cannot be created in the CDB root or in an application root. A local user cannot create a common user.

**Note:** If an application PDB is closed, the CDB common users, application common users, and local users of the application PDB are not visible because the metadata is retrieved from the PDB SYSTEM tablespace.

## Creating Common Roles in the CDB and PDBs



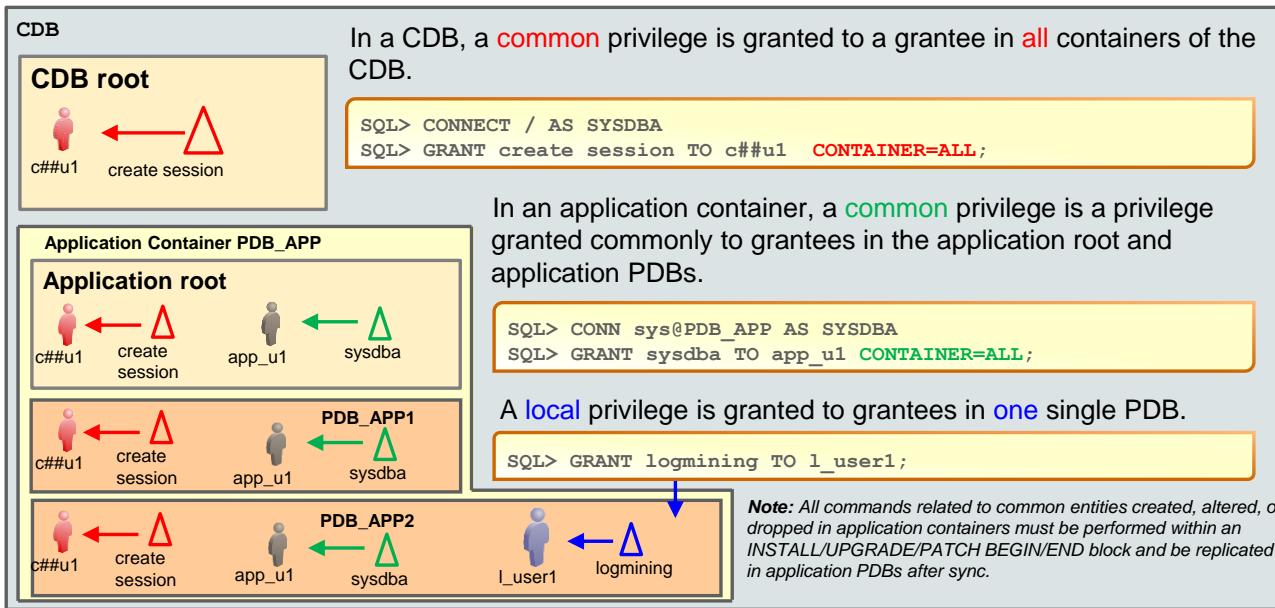
Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

A CDB common role is created from the CDB root and replicated across all PDBs in the CDB. In the example in the slide, the `c##r1` role is created commonly by using the `CONTAINER=ALL` clause. The create operation is replicated in all containers of the CDB. Consequently, the same role `c##r1` is created in each container. A CDB role requires a prefix, in this case, `c##`.

When you are in the application root and use the `CONTAINER=ALL` clause in the `CREATE ROLE` statement, an application role (such as `app_r1` in the graphic) is common to the application container. This role is replicated in all application PDBs of the application container when the DBA completes the synchronization of the application PDBs with the application root. An application common role does not require a prefix.

A role that is created in a specific PDB is local. In the example in the slide, the `l_role1` role is created locally. The create operation cannot be replicated in all containers. Consequently, the `l_role1` role is created in the `PDB_APP2` container only.

## Granting Privileges Commonly in the CDB and PDBs



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

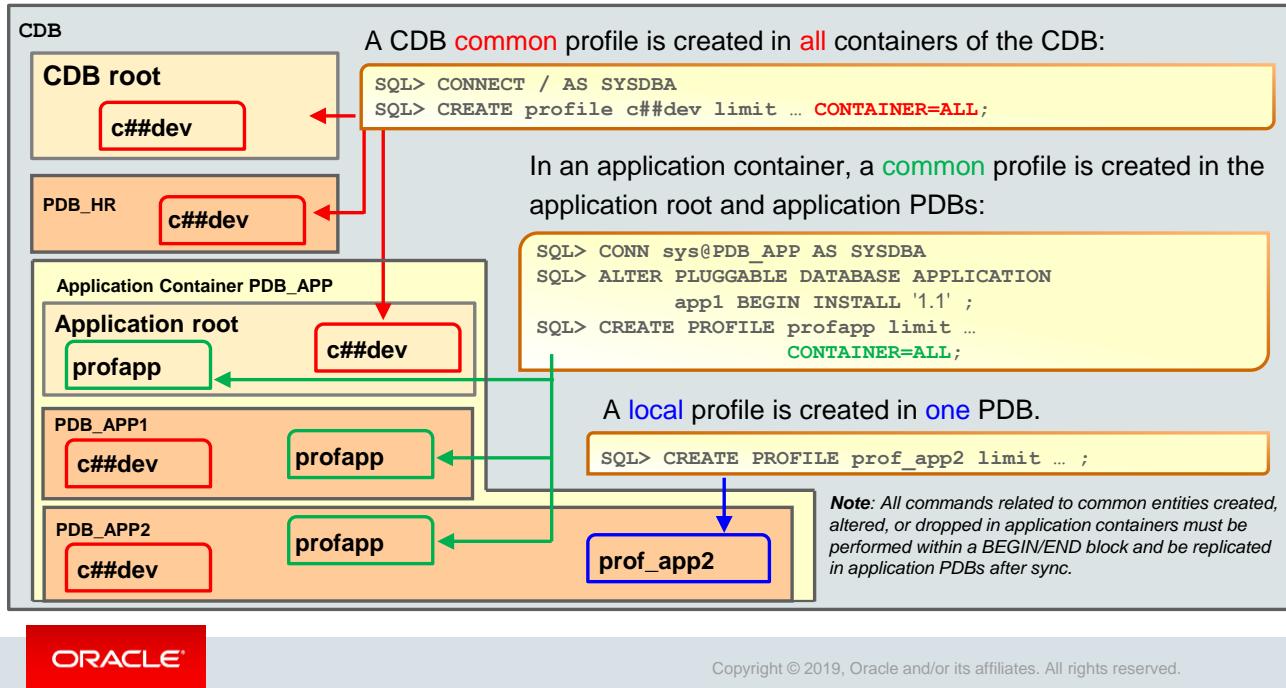
In a CDB, a privilege granted across all containers is a common privilege. In the example in the slide, the CREATE SESSION privilege is granted commonly to the `c##u1` user by using the `CONTAINER=ALL` clause. The grant operation is replicated in all containers, including the CDB root where it is initially granted. Consequently, the same user `c##u1` is granted the same privilege in each container.

If the privilege is granted commonly from an application root, then it is granted commonly to the common user in all the application PDBs of the application container by using the `CONTAINER=ALL` clause and when the synchronization of the application PDBs with the application root is completed. In the example in the slide, the administrative SYSDBA privilege is granted commonly to the `app_u1` user by using the `CONTAINER=ALL` clause from the application root. The grant operation is replicated in all application PDBs in the application container when the synchronization of the application PDBs with the application root is completed. Consequently, the same user `app_u1` is granted the same SYSDBA privilege in each application PDB of the application container. `V$PWFILE_USERS` displays the application common grants of administrative privileges when queried within an application root or PDB.

A privilege that is granted in a specific PDB is local. In the example in the slide, the LOGMINING privilege is granted locally to the `l_user1` user in `PDB_APP2`. Consequently, the `l_user1` user is granted the privilege in the `PDB_APP2` container only.

The preceding principles apply to whichever grantee the privilege is granted to: a user or a role.

## Creating Common Profiles in the CDB and PDBs



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

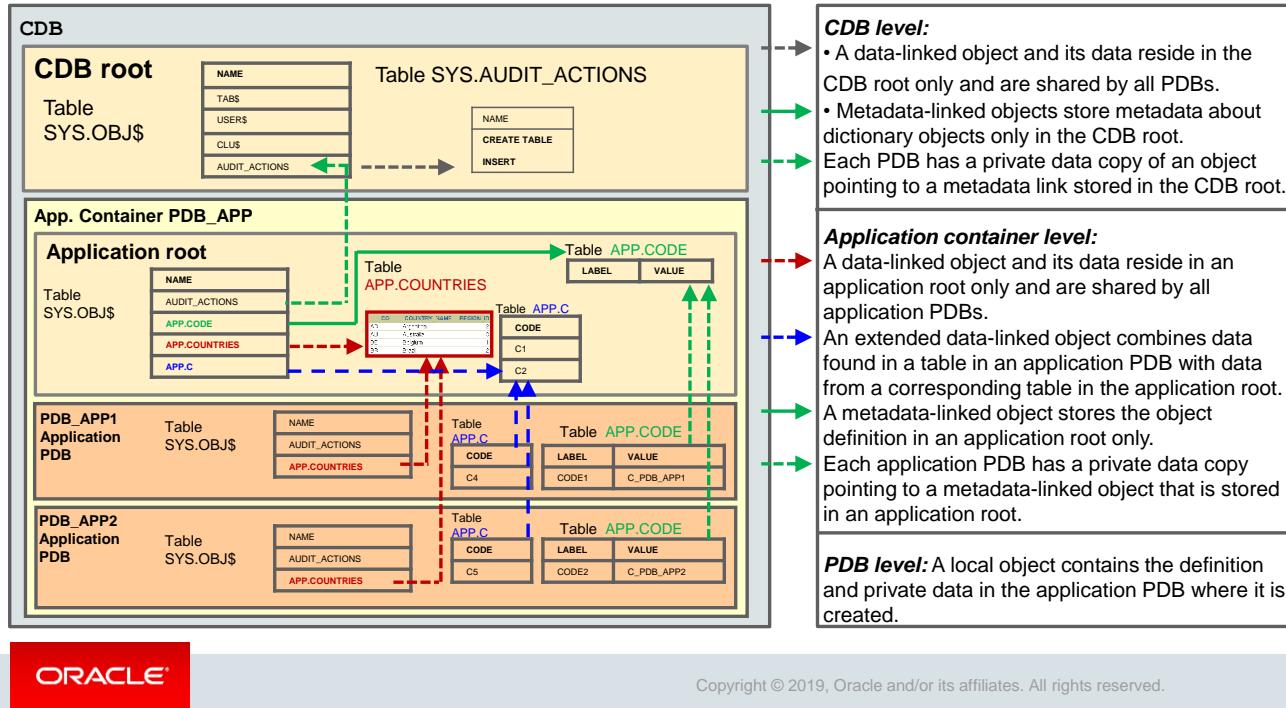
A profile that is created across all containers of the CDB is a common profile of the CDB. In the example in the slide, the `c##dev` profile is created commonly at the CDB level. The `CREATE` operation is replicated in all containers, including the CDB root where it is initially created. Consequently, the same profile `c##dev` is created in each container of the CDB.

A profile that is created across all containers of an application container is a common profile of the application container. In the example in the slide, the `profapp` profile is created commonly at the application container level. The `CREATE` operation is replicated in all application PDBs, including the application root where it is initially created. Consequently, the same profile `profapp` is created in each container of the `PDB_APP` application container.

A profile that is created in a specific PDB is local. In the example in the slide, the `prof_app2` profile is created locally in the `PDB_APP2` application container. Consequently, the `prof_app2` profile is created in the `PDB_APP2` application container only.

Common profiles can be assigned locally to any user in any PDB of the CDB (regular or application PDB) or commonly to all regular or application PDBs of any application container. Common application profiles can be assigned locally to any user in any application PDB of the application container or commonly to any common application user that exists in the same PDB.

# Common Objects in Application Containers



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Common objects, introduced in the multitenant option, avoid having to store redundant representations of data and metadata across a CDB and simplify the process of upgrading a CDB.

Common objects exist in Oracle-supplied schemas and, therefore, cannot be user-defined.

Application common objects can be created by users in an application container by issuing `CREATE` statements while being connected to the application root. This avoids having to store redundant representations of data and metadata across the application PDBs of an application container and, therefore, simplifies the process of upgrading an application in an application container. Specify an appropriate `SHARING` value in the `CREATE TABLE` statement.

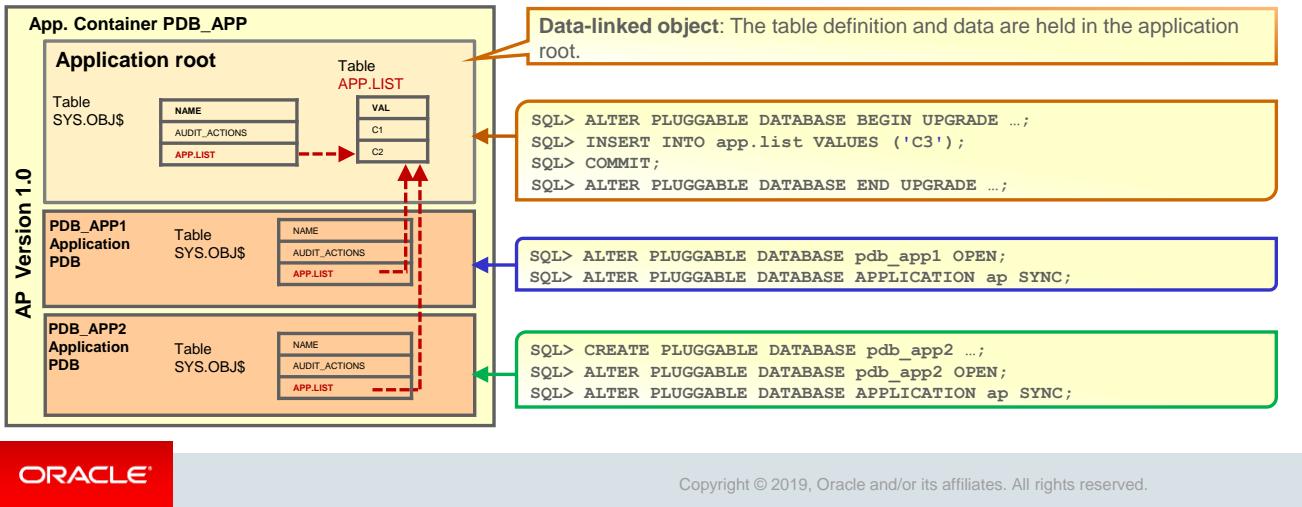
- Metadata-linked objects share the customer-created object definitions that are created in the application root, but data is stored in application PDBs.
- Data-linked objects share the data that is stored in the application root accessible across all application PDBs. Both the definition and data reside in the application root.
- Extended-data-linked objects share data that is stored in the application root accessible across all application PDBs and also contain PDB-specific data. This type of object supports combining data that is found in a table in an application PDB with data from a corresponding table in the application root. Such tables contain both common and local data.

You can also set the `DEFAULT_SHARING` parameter, which minimizes the need to change existing customer installation scripts.

# Operations on Data-Linked Objects

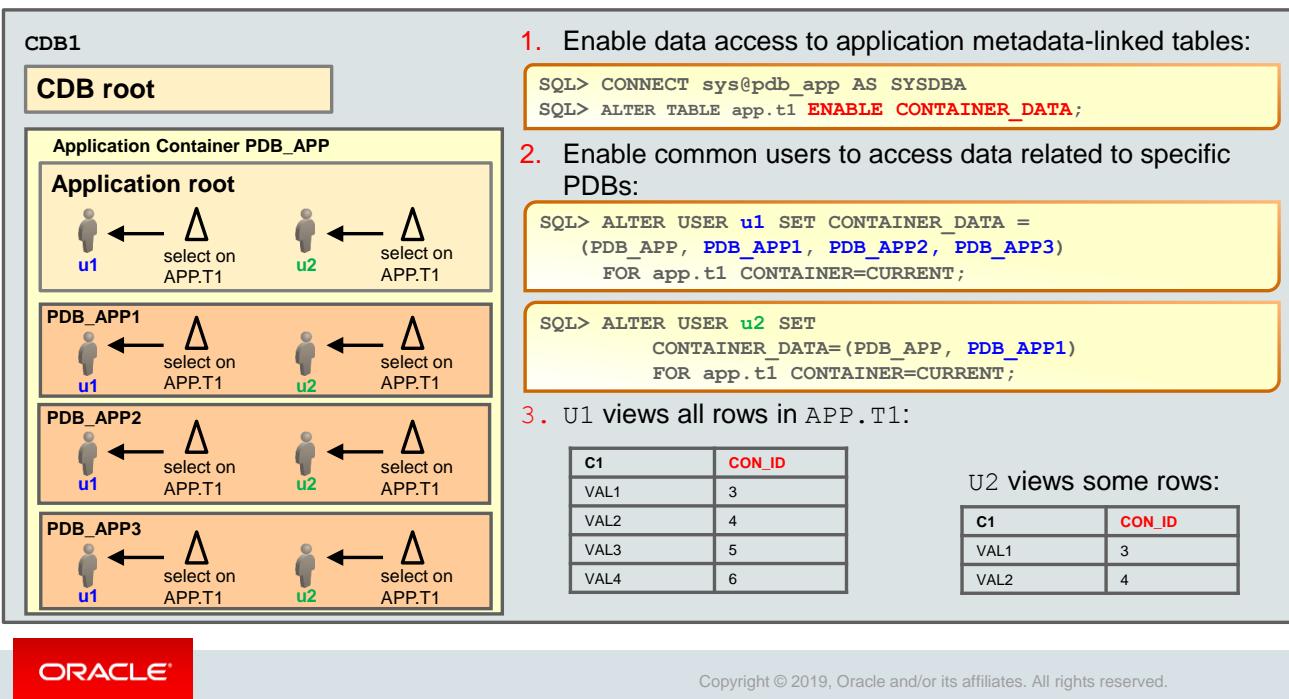
Apply **recorded DDL or DML** statements at synchronization:

- To **new** application PDBs
- To PDBs that were **closed** when the DDL or DML statements were issued



The DDL statements that apply to Oracle-supplied objects or common objects in an application root and the DML statements that alter the contents of data-linked tables are recorded so that they can be applied to new application PDBs and to PDBs that were closed when such statements were issued. The pending operations are executed in application PDBs when the application PDBs are created or opened and synchronized with the application root.

# Enabling Common Users to Access Data in PDBs



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Common users can view information about the CDB root and about the entire CDB by querying a set of Oracle-supplied views that are created as container data objects. A common user that is connected to the CDB root can view data from the metadata-linked objects pertaining to the PDBs by way of the container data objects (CDB views and V\$ views) in the CDB root, provided that the common user has been granted the privileges required to access these views and his or her CONTAINER\_DATA attribute has been set to allow viewing data related to various PDBs.

By default, common users cannot view information about specific PDBs. You can control common users' ability to see data pertaining to specific PDBs. This is useful in cases where you do not want to expose sensitive information about other PDBs.

The same CONTAINER\_DATA clause can be enabled or disabled on common metadata-linked tables and views in an application root. You can therefore control application common users' ability to see data pertaining to common metadata-linked tables in specific application PDBs, provided that the table contains a CON\_ID column. This is useful in cases where you do not want to expose sensitive information about other application PDBs.

In the example in the slide, the common application user, U2, although granted the SELECT privilege on the common APP.T1 table cannot view all rows pertaining to other application PDBs, other than the application root and PDB\_APP1 application. Alternatively, the common application user, U1, can view all rows of the same table, rows pertaining to the application root, and all application PDBs.

## Finding Information About CONTAINER\_DATA Attributes

Find information about the default (user-level) and object-specific CONTAINER\_DATA attributes that are explicitly set to a value other than DEFAULT.

```
SQL> SELECT username, default_attr, object_name, allContainers, container_name,
 con_id
 FROM cdb_container_data ORDER BY object_name;
```

| USERNAME  | DEFAULT | OBJECT_NAME | ALL_CONTAINER_ | CON_ID |
|-----------|---------|-------------|----------------|--------|
| C##JIM    | N       | V\$SESSION  | N PDB_HR       | 1      |
| C##JIM    | N       | V\$SESSION  | N CDB\$ROOT    | 1      |
| C##JIM    | N       | V\$SESSION  | N PDB2_2       | 1      |
| SYSTEM    | Y       |             | Y              | 1      |
| DBSNMP    | Y       |             | Y              | 1      |
| SYSBACKUP | Y       |             | Y              | 1      |
| SYS       | Y       |             | Y              | 1      |



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

To find information about the default (user-level) and object-specific CONTAINER\_DATA attributes that are explicitly set to a value other than DEFAULT, you can query the DBA\_CONTAINER\_DATA data dictionary view where:

- USERNAME is the name of the user whose attribute is described by this row.
- DEFAULT\_ATTR is an indicator of whether the attribute is a default attribute.
- OWNER is the name of the object owner if the attribute is object specific.
- OBJECT\_NAME is the name of the object if the attribute is object specific.
- ALL\_CONTAINERS is an indicator of whether this attribute applies to all containers.
- CONTAINER\_NAME is the name of a container included in this attribute if it does not apply to all containers.

## Restricting Operations with PDB Lockdown Profiles

- A potential for elevation of privileges exists where identity is shared between PDBs.
- You can restrict operations, features, and options used by users connected to a given PDB by using three `ALTER SYSTEM` clauses.

| STATEMENT                                                             | FEATURE                                                                                         | OPTION                    |
|-----------------------------------------------------------------------|-------------------------------------------------------------------------------------------------|---------------------------|
| ALTER SYSTEM<br>FLUSH SHARED_POOL, CHECKPOINT,<br>SWITCH LOGFILE, SET | NETWORK_ACCESS<br>UTL_TCP, UTL_SMTP, UTL_HTTP,<br>UTL_INADDR, XDB_PROTOCOLS,<br>DBMS_DEBUG_JDWP | Partitioning              |
|                                                                       | COMMON_SCHEMA_ACCESS                                                                            | Advanced Queuing          |
|                                                                       | OS_ACCESS<br>UTL_FILE, JAVA_OS_ACCESS,<br>EXTERNAL PROCEDURES                                   | Real Application Clusters |
|                                                                       | XDB_PROTOCOLS                                                                                   | Oracle Data Guard         |
|                                                                       | JAVA, JAVA_RUNTIME                                                                              |                           |



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

A PDB lockdown profile is a mechanism to restrict users connected to a given PDB or in all PDBs from completing operations such as setting instance parameters or using certain features related to network access or common schema access or using options such as partitioning.

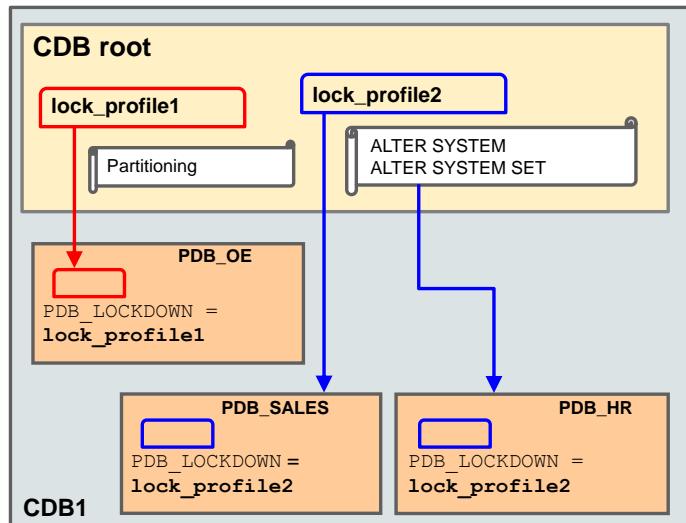
- **Network access:** Includes operations that use the network to communicate outside the PDB. These features can be controlled as the `NETWORK_ACCESS` group or individually through `UTL_TCP`, `UTL_HTTP`, `UTL_SMTP`, `UTL_INADDR`, and others.
- **Common schema access:** Includes operations where a local user in the PDB can proxy through common user or access objects in a common schema. These operations include:
  - Adding or replacing objects in common schemas
  - Accessing common directory objects
  - Granting inherit privileges to a common user

This feature enforces restrictions on creation or access of common objects by the local users in a PDB through the ANY type privileges. A local user can access or create common objects when the user has the `SYSDBA` or a specific object privilege or the specific ANY privilege. The ANY privilege behaves differently. An error is thrown if the local user accesses or creates common user's objects. There is no restriction, however, if the local user accesses another local user's schema through the ANY privilege within the PDB.

- **Options:** Includes operations that are used to administer database options such as partitioning.

# Restricting Operations in a PDB Lockdown Profile

CDB\_LOCKDOWN\_PROFILES



1. Create PDB lockdown profiles.
2. Define enabled and disabled:
  - Statement and clauses
  - Feature
  - Option
3. Set the **PDB\_LOCKDOWN** parameter to a PDB lockdown profile for all PDBs.
4. Optionally set the **PDB\_LOCKDOWN** parameter to another PDB lockdown profile for a PDB.

ORACLE®

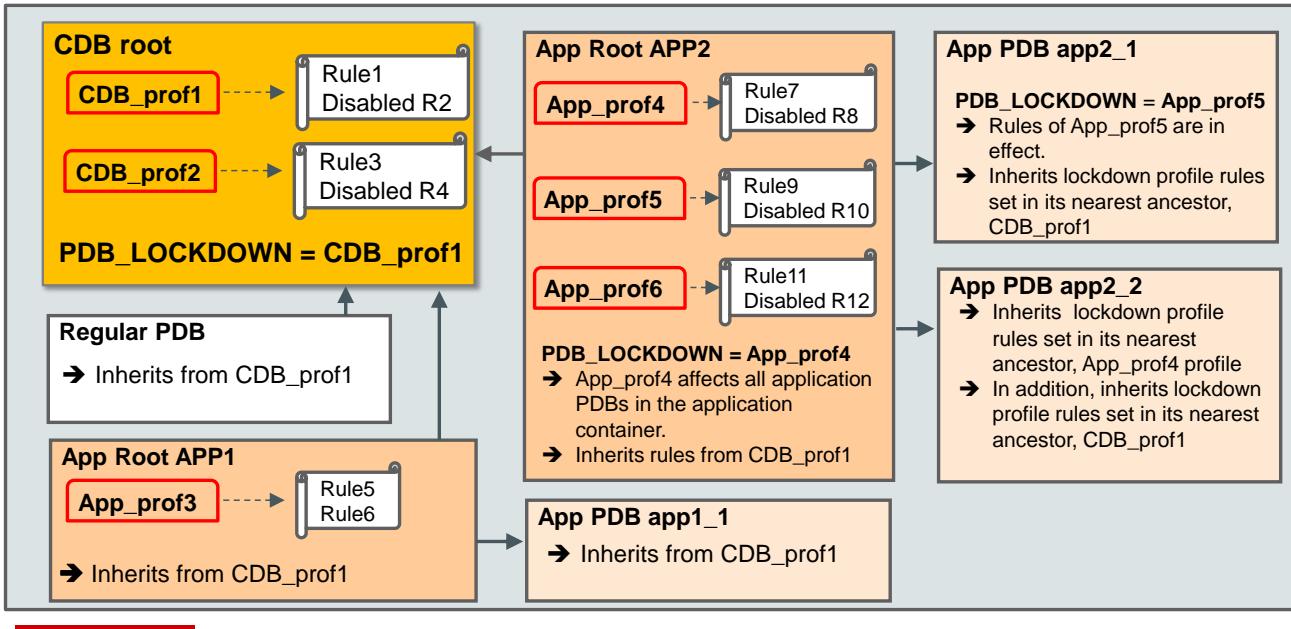
Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

This graphic illustrates how PDB lockdown profiles can be managed from the CDB root.

PDB lockdown profiles can be created by users who are granted the **CREATE LOCKDOWN PROFILE** system privilege.

The PDB lockdown profile whose name is stored in the **PDB\_LOCKDOWN** parameter determines the operations that may be performed in a given PDB. If the **PDB\_LOCKDOWN** parameter is set to a PDB lockdown profile at the CDB root level, and no **PDB\_LOCKDOWN** parameter is set at the PDB level, then the PDB lockdown profile that was defined at the CDB root level determines the operations that may be performed in all the PDBs.

# PDB Lockdown Profiles Inheritance



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

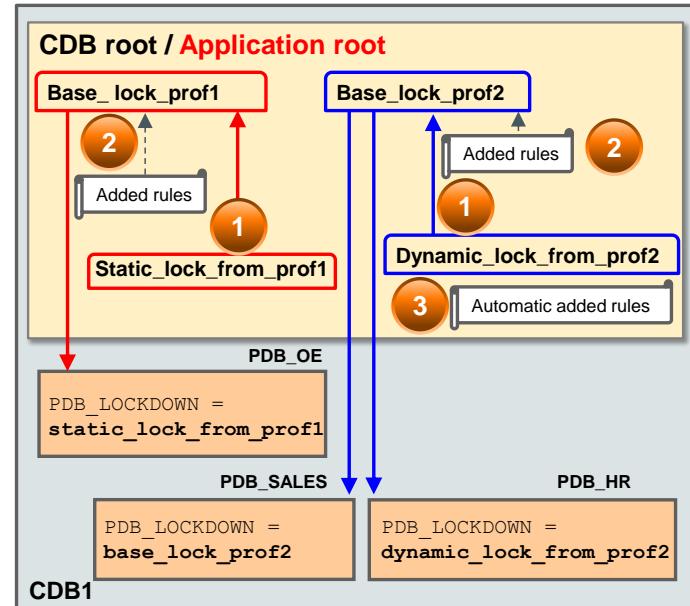
In Oracle Database 18c, you can create PDB lockdown profiles in the application root, as well as the CDB root.

In Oracle Database 18c, if the `PDB_LOCKDOWN` parameter in a PDB is set to a name of a lockdown profile that is different from that in its ancestor container, then the CDB root or application root for application PDBs, the following rules govern the interaction between restrictions imposed by these lockdown profiles in the CDB root.

In the last bulleted item, lower case "Application":

- If the `PDB_LOCKDOWN` parameter in a regular or application PDB is set to a CDB lockdown profile, lockdown profiles specified by the `PDB_LOCKDOWN` parameter, respectively, in the CDB root or application root are ignored.
- If the `PDB_LOCKDOWN` parameter in an application PDB is set to an application lockdown profile while the `PDB_LOCKDOWN` parameter in the application root or CDB root is set to a lockdown profile, in addition to rules stipulated in the application lockdown profile, the PDB lockdown profile inherits the `DISABLE` rules from the lockdown profile set in its nearest ancestor, the CDB root.
- If there are conflicts between rules comprising the CDB lockdown profile and the Application lockdown profile, the rules in CDB lockdown profile take precedence. For example, an `OPTION_VALUE` clause of a CDB lockdown profile takes precedence over `OPTION_VALUE` clause of an Application lockdown profile.

# Static and Dynamic PDB Lockdown Profiles



There are two ways to create lockdown profiles by using an existing profile:

- Static lockdown profiles:

```
SQL> CREATE LOCKDOWN PROFILE prof3
 FROM base_lock_prof1;
```

- Dynamic lockdown profiles:

```
SQL> CREATE LOCKDOWN PROFILE prof4
 INCLUDING base_lock_prof2;
```



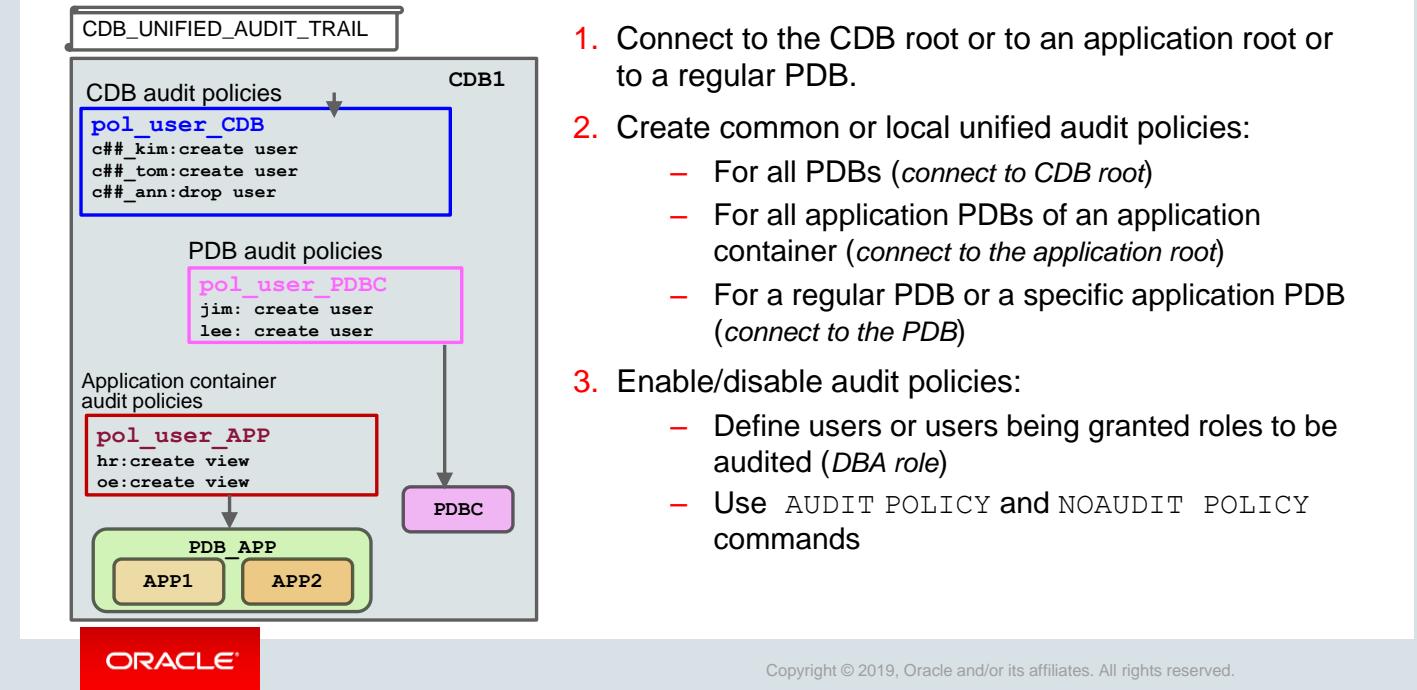
Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

When a PDB lockdown profile is created, it can derive rules from a “base” lockdown profile.

There are two ways of creating lockdown profiles by using existing profiles:

- **Static lockdown profiles:** When the lockdown profile is created with the `FROM` clause, rules comprising the base profile at the time are copied to the static lockdown profile. Any subsequent changes to the base profile do not affect the newly created static lockdown profile.
- **Dynamic lockdown profiles:** When the lockdown profile is created with the `INCLUDING` clause, the dynamic lockdown profile inherits disabled rules comprising base profile as well as any subsequent changes to the base profile. If rules explicitly added to the newly created dynamic lockdown profile come into conflict with rules comprising the base profile, then the latter takes precedence.

## Auditing Actions in the CDB and PDBs



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

You can create a unified audit policy commonly at the CDB level for the whole CDB. If you create the unified audit policy in the application root, it applies to all application PDBs that belong to the application container. If you create the unified audit policy locally, it applies to the local container, whether it is an application PDB or regular PDB.

Enabled in the CDB root, in the application root, or in a PDB, the unified audit policy provides the ability to audit the system privileges used or actions performed in all PDBs in the CDB or the object privileges used on common objects by application common users in all application PDBs in an application container or exclusively in a specific PDB.

Application common unified audit policies need NOT be explicitly synchronized in application PDBs with the application root. An implicit application BEGIN-END block is added for application common unified audit policies when the end user does not create them inside an explicit application BEGIN-END block. In this case, the creation of application common unified audit policies does not require explicit application BEGIN-END block statements. However, application common unified audit policies, when created within an explicit application BEGIN statement, would require an explicit application END statement.

- In the example in the slide, the `pol_user_CDB` unified audit policy that is created in the CDB root audits any `CREATE` or `DROP USER` statement that is performed by specific users (`c##_kim`, `c##_tom`, and `c##_ann`).
- The `pol_user_APP` unified audit policy that is created in the `PDB_APP` application root audits any `CREATE VIEW` operation performed by application common users (`hr` and `oe`).
- The `pol_user_PDB` unified audit policy that is created in the regular `PDB` PDB audits any `CREATE` or `DROP USER` performed by local users (`jim`, `lee`, and `bob`).

The audit records are still generated in the container's own audit trail and are generated in the container where the action was executed.

A local unified audit policy can be enabled on local and common roles and becomes effective for users to whom the local or common role is granted directly. A common unified audit policy, alternatively, can be enabled only on common roles and becomes effective only for common users to whom the common role is granted. The clause used is `BY USERS WITH GRANTED ROLES role_list`.

A good example of using this capability is the predefined role called `DBA`, which contains most of the system privileges, granted to special privileged users, which might be considered for auditing.

The audit-administrator does not have to enable the unified audit policies on all the individual users explicitly. Over a period of time, there could be new users with the `DBA` role granted. Some of the earlier `DBA` users might no longer have the `DBA` role. The audit-administrator does not have to keep track of such changing auditing requirements and enable the unified audit policies appropriately for a new set of `DBA` users. Similarly the users, who no longer have the `DBA` role granted, will automatically be excluded from auditing and thus avoid generating unnecessary audit records.

## Managing Other Types of Security Policies in Application Containers

| Policy Type               | Compatible in Application Containers | Created in Install / Upgrade / Patch BEGIN-END block | Automatic synchronization in application PDBs |
|---------------------------|--------------------------------------|------------------------------------------------------|-----------------------------------------------|
| Unified Audit             | Y                                    | Y (explicit or implicit)                             | Y (explicit or implicit)                      |
| FGA                       | Y                                    | Y                                                    | N                                             |
| Application Context & VPD | Y                                    | Y                                                    | N                                             |
| TSDP                      | Y                                    | N                                                    | n/a                                           |
| OLS                       | N                                    | n/a                                                  | n/a                                           |



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The application-common Fine-grained auditing (FGA) policies that are applied in an application root are not automatically propagated to application PDBs that belong to the application container. The application PDBs need to be synchronized with the application root. Otherwise, any access to an application common object from an application PDB would not be audited in the FGA audit trail in that application PDB.

The application-common “application contexts” and application-common Virtual Private Database (VPD) policies that protect the common objects created in an application root are not automatically implemented in all application PDBs. The application PDBs need to be synchronized with the application root. An application-common VPD policy can be created only in the application root and attached to an application common object.

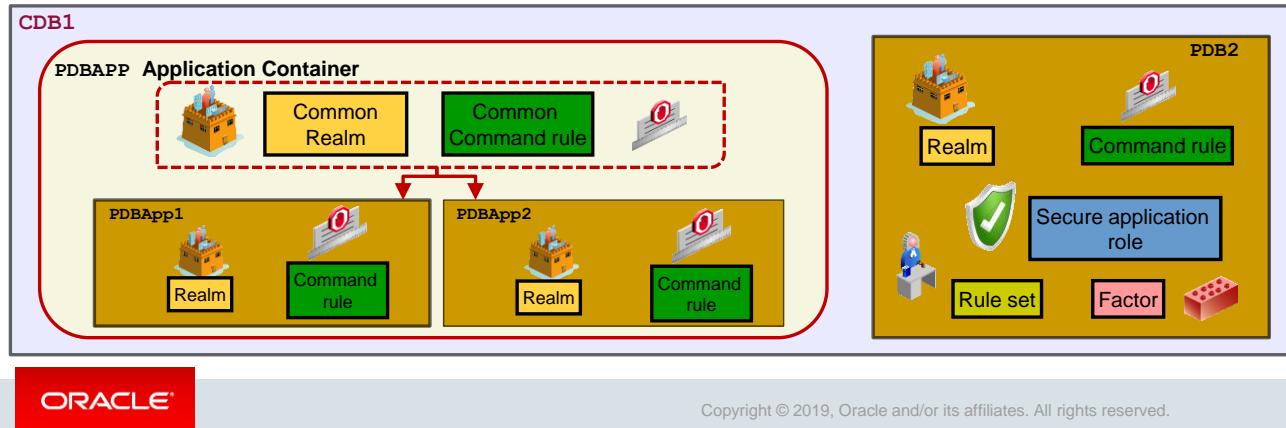
Application-common transparent sensitive data protection (TSDP) policies can be created in an application root but only outside the application install/upgrade/patch BEGIN-END block. Application-common TSDP policies can be applied on both application common objects and application root local objects. The application-common TSDP policies that are defined in the application root on local objects should behave as if they are local policies, meaning that they are effective in the application root only. TSDP operations are container-specific.

Application-common Oracle Label Security (OLS) policies cannot be created in an application root within or outside of the application install/patch BEGIN-END block. Therefore, these policies cannot be applied on common objects within application PDBs.

# Securing Data with Oracle Database Vault

DVSYS.DBA\_DV\_POLICY  
DVSYS.DBA\_DV\_POLICY\_OBJECT

- Each PDB has its own Database Vault metadata.
- Database Vault common protection can protect the common objects of an application container:
  - Database Vault common realm
  - Database Vault common command rule



Each PDB has its own Database Vault metadata. Database Vault constructs such as realms, command rules, and secure application roles are isolated within a PDB. If the common objects are required to be protected by Database Vault in every PDB, you can set a Database Vault common protection, which is designed to protect the common objects in application containers.

Instead of configuring and managing the same command rule and the same realm in every application PDB of an application container, common protection can be configured and managed in the application root. The common protection enforcement applies to all application PDBs in the application container.

You can use Database Vault policies to group realm and command rule definitions into one policy, which then can be collectively enabled or disabled, or partially enabled, setting the associated realms and command rules to take precedence over the policy. Or you can enable it in simulation mode, writing violations to a log table. You can view the contents of this table by querying Database Vault-specific views. Common realms can be added to Database Vault policies in application roots only. In application PDBs, you can add local realms to Database Vault policies.

The Database Vault common realm does not protect any local object owned by local users in PDBs. A common Database Vault administrator is a common user who is granted the DV\_OWNER or DV\_ADMIN role with CONTAINER=ALL in an application root. The Database Vault common realm can be created, updated, and deleted only by the common Database Vault administrator in an application root.

The Database Vault common realm has two types of authorization:

- Common authorization: Granted to common users or common roles in the application root
- Local authorization: Granted to common users, common roles, local users, and local roles in the PDB. The authorization takes effect only in that PDB.

The Database Vault common command rule protects commands on a specific common schema or a specific common object. The rule that is added to a rule set that is associated with the common command rule cannot involve any local object.

# Oracle Database Vault-Enabled Strict Mode

## Mixed mode:

Both Database Vault enabled and disabled PDBs can work together in the same application container.



Database Vault common protection does not protect the common objects in the Database Vault disabled PDBs.

## Strict mode:

The common protection must cover the common objects in every PDB in the same application container.



The Database Vault disabled PDBs are opened in restricted mode.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Common protection on objects pertaining to application PDBs can be Database Vault enabled in PDBs and Database Vault disabled in others.

By default, when Database Vault is enabled in the CDB root, the CDB supports mixed mode. Both the Database Vault enabled PDBs and Database Vault disabled PDBs plugged in the CDB function normally.

### Mixed Mode

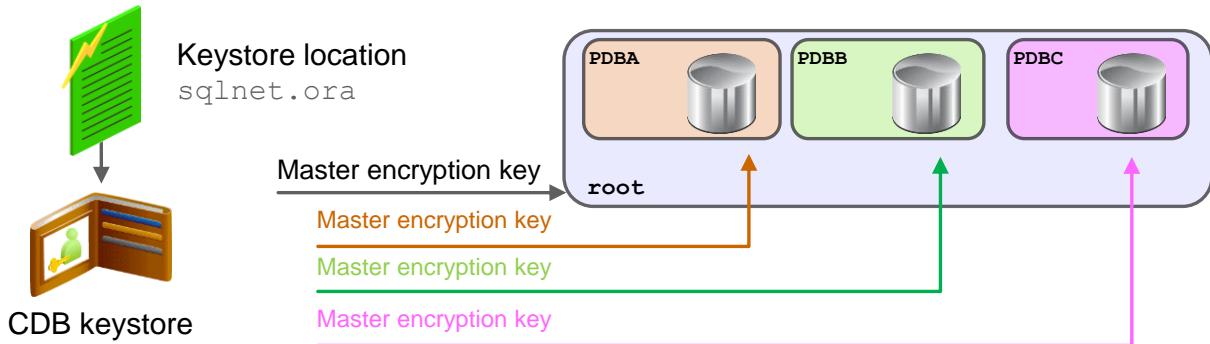
By default, when Database Vault is enabled in an application root, the application container supports mixed mode. Both the Database Vault enabled PDBs and Database Vault disabled PDBs plugged in the application container function normally. The Database Vault common protection cannot protect the common objects in Database Vault disabled PDBs because the Database Vault common protection requires that PDBs have Database Vault enabled.

### Strict Mode

The Database Vault strict mode is another behavior. It allows you to make the common protection cover common objects in every application PDB. When Database Vault is enabled in strict mode in an application root, the Database Vault disabled PDBs that are plugged in the application container are opened in restricted mode. To make the PDB open normally, Database Vault must be enabled in the PDB, and then the PDB must be restarted.

## Managing Keystore in the CDB and PDBs

- There is one TDE master encryption key per PDB to encrypt PDB data.
- The TDE master encryption key must be transported from the source database keystore to the target database keystore when a PDB is moved from one host to another.



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

In a CDB, the CDB root container and each PDB has its own TDE master encryption key used to encrypt data in the PDB. The TDE master encryption key must be transported from the source database keystore to the target database keystore when a PDB is moved from one host to another.

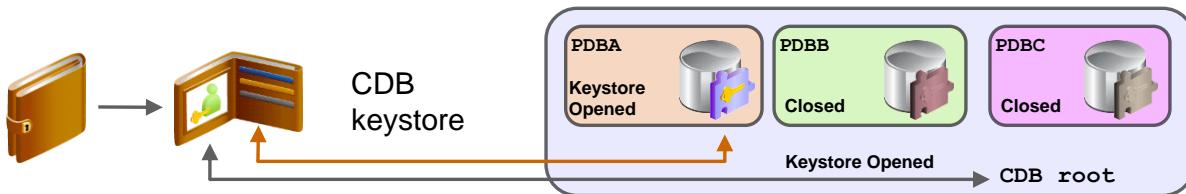
## Creating and Opening a Keystore

- Create the unique keystore in the CDB root.

```
SQL> ADMINISTER KEY MANAGEMENT CREATE KEYSTORE
 'u01/app/oracle/product/12.2.0/dbhome_1/wallet'
 IDENTIFIED BY k_password;
```

- Open the keystore in the CDB root and then for a specific PDB.

```
SQL> CONNECT john@PDBA AS SYSKM
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY k_password
CONTAINER = CURRENT;
```



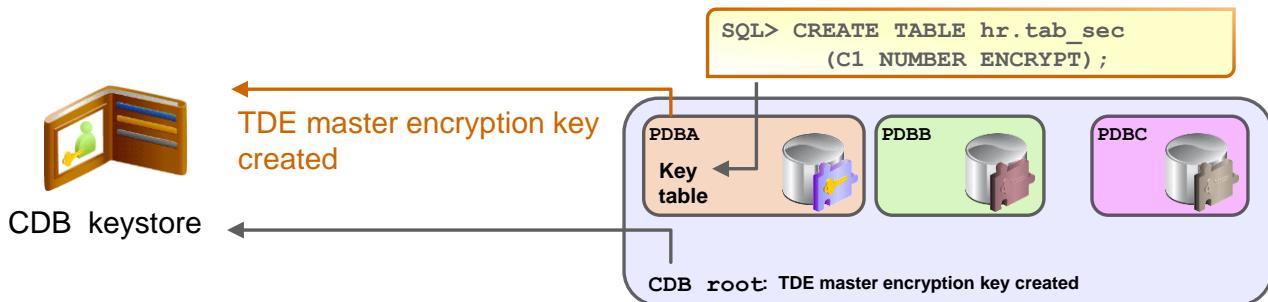
Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

- Create the keystore from the root for the CDB using the first statement in the slide.
- Open the keystore:
  - First for the root: Use CONTAINER=ALL to open the keystore for all PDBs including for the root container or without the CONTAINER clause to open the keystore at least for the root.
  - Then for a specific PDB: Use CONTAINER=CURRENT. CONTAINER=CURRENT as the default when you are in the PDB.

## Setting TDE Master Encryption Keys

- Set the TDE master encryption key for a PDB.

```
SQL> CONNECT john@PDBA AS SYSKM
SQL> ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY k_password WITH BACKUP
CONTAINER = CURRENT;
```



You can now encrypt data in tablespaces and tables.



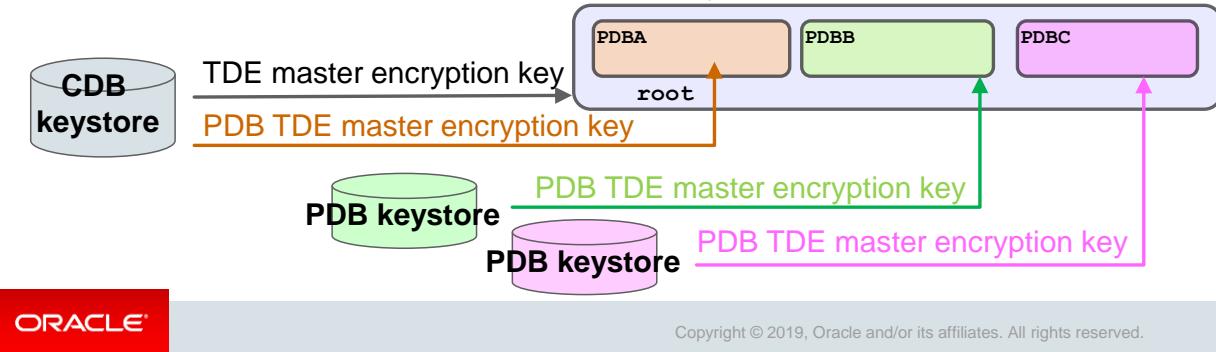
Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

- Set the TDE master encryption key:

- First for the CDB root: Use CONTAINER=ALL to set a TDE master encryption key for each PDB and for the CDB root container or without the CONTAINER clause to set a TDE master encryption key at least for the CDB root.
- Then for a specific PDB: Use CONTAINER=CURRENT after a connection to the PDB under a user with SYSKM administrative privilege.

## Managing Keystore in the CDB and PDBs

- There is still one single keystore for the CDB and optionally one keystore per PDB.
- There is still one TDE master encryption key per PDB to encrypt PDB data, stored in the PDB keystore.
- Modes of operation
  - United mode: PDB keys are stored in the unique CDB root keystore.
  - Isolated mode: PDBs keys are stored in their own keystore.
  - Mixed mode: Some PDBs use united mode, some use isolated mode.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

In an Oracle Database 12c CDB, the CDB root and each PDB have its own TDE master encryption key used to encrypt data in the PDB, all of them stored in the common single keystore. The TDE master encryption key must be transported from the source database keystore to the target database keystore when a PDB is moved from one host to another.

The TDE master encryption keys are stored in a PKCS#12 software keystore or a PKCS#11-based HSM, outside the database. For the database to use TDE, the keystore must exist in a directory defined by the `ENCRYPTION_WALLET_LOCATION` location in the `sqlnet.ora` file.

In Oracle Database 12c, the multitenant architecture was mainly focused on providing support for database consolidation.

In Oracle Database 18c, the multitenant architecture continues to provide support for database consolidation; however, focus is on independent, isolated PDB administration. To support independent, isolated PDB administration, support for separate keystores for each PDB is now provided. Providing the PDBs with their own keystore is called the “isolated mode.” Having independent keystores allows PDBs to be managed independently of each other. The shared keystore mode provided with Oracle Database 12c is now called “united mode”. Both modes can be used at the same time, in a single multitenant environment, with some PDBs sharing a common keystore in united mode and some having their own independent keystores in isolated mode.

This feature allows each PDB running in isolated mode within a CDB to manage its own keystore. Isolated mode allows a tenant to manage its TDE keys independently and supports the requirement for a PDB to be able to use its own independent keystore password. The project aims to allow the customer to decide how the keys of a given PDB are protected, either with the independent password of an isolated keystore or with the password of the united keystore.

# Keystore Management Changes for PDBs

```
V$ENCRYPTION_WALLET
ENCRYPTION_MODE = NONE
```

PDBs can optionally have their own keystore, allowing tenants to manage their own keys.

- Define the shared location for the CDB root and PDB keystores:

```
SQL> ALTER SYSTEM SET wallet_root = /u01/app/oracle/admin/ORCL/tde_wallet;
```

- Define the default PDB keystore type for each future isolated PDB and then define a different file type in each isolated PDB if necessary:

```
SQL> ALTER SYSTEM SET tde_configuration = 'KEYSTORE_CONFIGURATION=FILE';
```

- United: → **WALLET\_ROOT/component/ewallet.p12**

**CDB root** and **PDBA** /u01/app/oracle/admin/ORCL/tde\_wallet/tde/ewallet.p12

- Isolated: → **WALLET\_ROOT/pdb\_guid/component/ewallet.p12**

**PDBB** /u01/app/oracle/admin/ORCL/tde\_wallet/51FE2A4899472AE6/tde/ewallet.p12

**PDBC** /u01/app/oracle/admin/ORCL/tde\_wallet/7893AB8994724ZC8/tde/ewallet.p12



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

In Oracle Database 12c, the `ENCRYPTION_WALLET_LOCATION` parameter in the `$ORACLE_HOME/network/admin/sqlnet.ora` file defines the path to the united keystore.

In Oracle Database 18c, no isolated keystore can be used unless the new initialization `WALLET_ROOT` parameter is set, replacing `ENCRYPTION_WALLET_LOCATION`. The `WALLET_ROOT` initialization parameter specifies the path to the root of a directory tree containing a subdirectory for each PDB GUID, under which a directory structure is used to store the various keystores associated with features such as TDE, EUS, and SSL. A new column, `KEYSTORE_MODE`, is added to the `V$ENCRYPTION_WALLET` view with values `NONE`, `ISOLATED`, and `UNITED`. If all PDBs use united mode, then you can still create the keystore for the CDB in the CDB root by using the `ALTER KEY MANAGEMENT` statement without needing the `WALLET_ROOT` parameter to be set.

```
SQL> ADMINISTER KEY MANAGEMENT CREATE KEYSTORE
 '/u01/app/oracle/admin/ORCL/tde_wallet/' IDENTIFIED BY password;
```

All of the existing statements allowed only in the CDB root until Oracle Database 12c are now supported in Oracle Database 18c at the PDB level, with the understanding that the `ADMINISTER KEY MANAGEMENT` privilege first needs to be granted to a newly created security officer for the PDB.

## Defining the Keystore Type

Values of keystore types allowed:

- FILE
- OKV (Oracle Key Vault)
- HSM (Hardware Security Module)
- FILE|OKV: Reverse-migration from OKV to FILE has occurred
- FILE|HSM: Reverse-migration from HSM to FILE has occurred
- OKV|FILE: Migration from FILE to OKV has occurred
- HSM|FILE: Migration from FILE to HSM has occurred

In isolated mode, when the CDB is in mounted state:

```
SQL> STARTUP MOUNT
SQL> ALTER SYSTEM SET tde_configuration='CONTAINER=pdb1; KEYSTORE_CONFIGURATION=FILE';
```

ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

A per-PDB dynamic instance initialization parameter, TDE\_CONFIGURATION, is added, which takes an attribute-value list.

TDE\_CONFIGURATION has only two attributes:

- **KEYSTORE\_CONFIGURATION:** Can take the values FILE and those defined in the slide
- **CONTAINER:** Specifies the PDB. This attribute can be specified only when setting the parameter in the CDB root when performing crash recovery or media recovery and the database is in the MOUNTED state. If the control file is lost, it may be necessary to run an ALTER SYSTEM statement in the CDB root to set the TDE\_CONFIGURATION parameter appropriately for each PDB. Because this statement must be run in the CDB root, the PDB name is provided via the additional attribute CONTAINER, for example, as follows:

```
ALTER SYSTEM SET
TDE_CONFIGURATION='CONTAINER=CDB1_PDB1;KEYSTORE_CONFIGURATION=FILE'
SCOPE=MEMORY;
```

The command configures the CDB1\_PDB1 to run in isolated mode using its own keystore.

# Isolating a PDB Keystore

```
V$ENCRYPTION_WALLET
ENCRYPTION_MODE = ISOLATED
```

- Create / open the CDB root keystore:  

```
SQL> ADMINISTER KEY MANAGEMENT CREATE KEYSTORE
IDENTIFIED BY <united_keystore_pass> ;
```
- Connect as the PDB security admin to the newly created PDB to:  
1) Create the PDB keystore.



```
SQL> ADMINISTER KEY MANAGEMENT CREATE KEYSTORE
IDENTIFIED BY isolated_keystore_pass;
```

- 2) Open the PDB keystore.

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
IDENTIFIED BY isolated_keystore_pass;
```

- 3) Create the TDE PDB key in the PDB keystore.

```
SQL> ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY isolated_keystore_pass
WITH BACKUP;
```

ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

In the case of a newly created PDB, the ADMINISTER KEY MANAGEMENT privilege needs to be granted to a local user in the PDB, who acts as the security officer for the new PDB. It is assumed that this security officer is provided with the password of the united keystore, because that password is required to gain access to the TDE master key. Note that knowledge of this password does *not* allow the user to perform an ADMINISTER KEY MANAGEMENT UNITE KEYSTORE operation. Additional privilege scope is needed for the unite keystore operation.

The PDB security officer is then allowed to invoke the ADMINISTER KEY MANAGEMENT CREATE KEYSTORE statement, which creates an isolated keystore for the PDB and automatically configures the keystore of the PDB to run in isolated mode.

**Note:** Observe that the ADMINISTER KEY MANAGEMENT CREATE KEYSTORE command does not use the definition of the keystore location. The keystore location is defined in the `WALLET_ROOT` parameter.

In the `V$ENCRYPTION_WALLET` view, the `KEYSTORE_MODE` column shows `NONE` for the CDB root container. For the isolated PDB, the `KEYSTORE_MODE` column shows `ISOLATED`.

# Converting a PDB to Run in Isolated Mode

```
V$ENCRYPTION_WALLET
ENCRYPTION_MODE = ISOLATED
```

- In the CDB root:
  - Create a common user to act as the security officer
  - Grant the ADMINISTER KEY MANAGEMENT privilege commonly
- Connect as the security officer to the PDB and create the keystore in the PDB.

```
SQL> ADMINISTER KEY MANAGEMENT ISOLATE KEYSTORE
 IDENTIFIED BY isolated_keystore_password
 FROM ROOT KEYSTORE IDENTIFIED BY [EXTERNAL STORE | <united_keystore_password>]
 WITH BACKUP;
```



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

If you want to convert a PDB to run in isolated mode, the ADMINISTER KEY MANAGEMENT privilege needs to be commonly granted to a common user who will act as the security officer for the PDB. The security officer for each PDB will now be managing their own keystore.

Then after logging in to the PDB as the security officer, the ADMINISTER KEY MANAGEMENT ISOLATE KEYSTORE statement must be executed to isolate the key of the PDB into a separate isolated keystore. The isolated keystore is created by this command, with its own password.

All of the previously active (historical) master keys associated with the PDB are moved to the isolated keystore.

# Converting a PDB to Run in United Mode

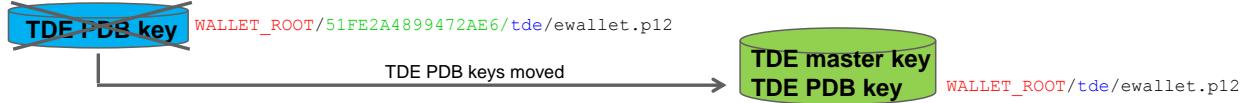
```
V$ENCRYPTION_WALLET
ENCRYPTION_MODE = UNITED
```

## 1. In the CDB root:

- The security officer of the CDB  exists.
- The security officer of the CDB  is granted the ADMINISTER KEY MANAGEMENT privilege commonly.

## 2. Connect as the security officer to the PDB and unite the TDE PDB key with those of the CDB root.

```
SQL> ADMINISTER KEY MANAGEMENT UNITE KEYSTORE
IDENTIFIED BY isolated_keystore_password
WITH ROOT KEYSTORE IDENTIFIED BY [EXTERNAL STORE | united_keystore_password]
[WITH BACKUP [USING backup_id]];
```



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

If a PDB no longer wants to manage its own separate keystore in isolated mode, the security officer can decide to unite its keystore with that of the CDB root and allow the security officer of the CDB root to administer its keys.

The PDB security officer who is a common user with the ADMINISTER KEY MANAGEMENT privilege granted commonly logs in to the PDB and issues the ADMINISTER KEY MANAGEMENT UNITE KEYSTORE statement to unite the keys of the PDB with those of the CDB root.

When the keystore of a PDB is being united with that of the CDB root, all of the previously active (historical) master keys associated with the PDB are also moved to the keystore of the CDB root.

When V\$ENCRYPTION\_WALLET is queried from the united PDB, the PDB being configured to use the CDB root keystore, in this case the KEYSTORE\_MODE column, shows UNITED.

## Migrating a PDB Between Keystore Types

To migrate a PDB from using wallet as the keystore to using Oracle Key Vault if the PDB is running in isolated mode:

1. Upload the TDE encryption keys from the isolated keystore to Oracle Key Vault by using a utility.
2. Set the `TDE_CONFIGURATION` parameter of the PDB to the appropriate value:

```
SQL> ALTER SYSTEM SET tde_configuration = 'KEYSTORE_CONFIGURATION=OKV' ;
```



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

To migrate a PDB from using wallet as the keystore to using Oracle Key Vault if the PDB is running in isolated mode, you must upload the TDE encryption keys from the isolated keystore to Oracle Key Vault. You can do this by using the `okvutil` utility `upload` command to migrate the existing TDE wallet to Oracle Key Vault. Then you must change the `TDE_CONFIGURATION` parameter of the PDB to `KEYSTORE_CONFIGURATION=OKV`.

Refer to the following Oracle documentation:

- *Oracle Database Advanced Security Guide 18c* – Chapter *Managing the Keystore and the Master Encryption Key - Migration of Keystores to and from Oracle Key Vault*.
- *Oracle Key Vault Administrator's Guide 12c Release 2 (12.2)* – Chapter *Migrating an Existing TDE Wallet to Oracle Key Vault – Oracle Key Vault Use Case Scenarios*
- *Oracle Key Vault Administrator's Guide 12c Release 2 (12.2)* – Chapter *Enrolling Endpoints for Oracle Key Vault – okvutil upload Command*

## Unplugging and Plugging a PDB with Encrypted Data

1. Unplugging an encrypted PDB exports the master encryption key of the PDB.

```
SQL> ALTER PLUGGABLE DATABASE pdb1
 UNPLUG INTO '/tmp/pdb1.xml'
 ENCRYPT USING "tpwd1";
```

PDB wallet opened



2. Plugging the encrypted PDB imports the master encryption key of the PDB into the CDB keystore.

```
SQL> CREATE PLUGGABLE DATABASE pdb1
 USING '/tmp/pdb1.xml'
 KEYSTORE IDENTIFIED BY keystore_pwd1
 DECRYPT USING "tpwd1";
```

Target CDB wallet opened



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

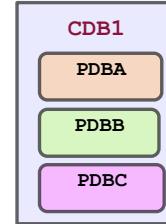
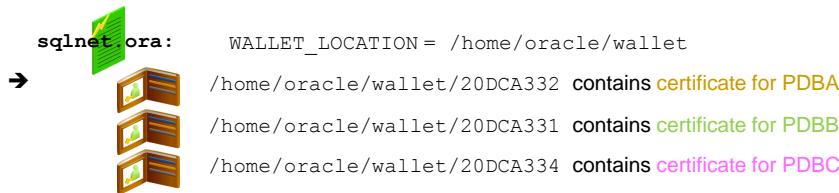
The unplugging operation of an encrypted PDB is a one-step operation and the plugging operation of the new encrypted PDB is also a one-step operation.

1. As part of the unplug command, the administrator must specify a temporary transport password to protect the implicitly exported master keys, and this can be achieved only if the wallet on the source database is already opened.
2. Trusted administrators must then be able to plug the encrypted PDB into a CDB without separately importing the master keys used by the PDB by specifying the transport password that was specified at unplug time to decrypt the implicitly exported master keys for import into the target database. This assumes that the “trusted administrator” is defined as someone who is authorized to have access to the wallet for the target CDB and has the privilege to plug the PDB into the CDB.

The preceding commands require the SYSKM administrative privilege as well as the privilege to unplug and plug the PDB.

## Per-PDB Wallet for PDB Certificates

- There is only one `sqlnet.ora` file and one `WALLET_LOCATION` parameter per CDB.
- Each PDB has its own keystore to store the TLS credentials and identity to communicate with other PDBs.



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Each PDB can have its own wallet with its own Transport Layer Security (TLS) credentials. This means that each PDB can have its own TLS identity and protect its communications from other PDBs on the same CDB.

Each PDB must be able to use its own wallet with its own certificates for TLS authentication. Because there are no per-PDB `sqlnet.ora` configuration files, the wallet must be placed in a subdirectory of the `WALLET_LOCATION` path where the name of the subdirectory is the GUID of the PDB that is using that wallet. More specifically:

If `WALLET_LOCATION` is:

```
(SOURCE= (METHOD=FILE) (METHOD_DATA= (DIRECTORY=/home/oracle/wallet)))
```

then each PDB's wallet location is effectively: `/home/oracle/wallet/pdb_guid`

If `WALLET_LOCATION` is not specified, the PDB wallet must be in a subdirectory of the default wallet path where the name of the subdirectory is the GUID of the PDB. More specifically:

If `ORACLE_BASE` is set:

```
$ORACLE_BASE/admin/db_unique_name/wallet/pdb_guid/
```

Else:

```
$ORACLE_HOME/admin/db_unique_name/wallet/pdb_guid/
```

## Summary

In this lesson, you should have learned how to:

- Manage common and local users, roles, privileges, and profiles in PDBs
- Manage common and local objects in application containers
- Enable common users to access data in PDBs
- Manage PDB lockdown profiles
- Audit users in CDB and PDBs
- Manage other types of policies in application containers
- Protect data with Database Vault policies in CDB and PDBs
- Encrypt data in PDBs
- Configure isolated PDB keystores
- Unplug and plug an encrypted PDB in a one-step operation
- Allow per-PDB wallets for certificates



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Practice 7: Overview

- 7-1: Managing common and local users, privileges, and roles
- 7-2: Managing common and local objects in application containers
- 7-3: Enabling common users to view information about PDB objects
- 7-4: Applying recorded statements in application PDBs
- 7-5: Managing PDB lockdown profiles
- 7-6: Auditing operations in PDBs
- 7-7: Protecting application common objects with Database Vault common realms
- 7-8: Managing PDB keystores
- 7-9: Unplugging and plugging encrypted PDBs



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.



# Backup and Duplicate

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

# Objectives

After completing this lesson, you should be able to:

- Back up a CDB
- Back up a PDB
- Duplicate an active PDB into an existing CDB
- Duplicate a CDB as encrypted
- Validate CDB and PDBs



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

For a complete understanding of the backup and duplicate procedures under the multitenant architecture, refer to the Oracle documentation:

- “Backing Up and Recovering CDBs and PDBs” in *Oracle Multitenant Administrator’s Guide 18c*
- “Duplicating CDBs and PDBs” in *Oracle Database Backup and Recovery User’s Guide 18c*

Refer to other sources of information available under Oracle Learning Library like *Oracle By Example (OBEs)*:

- [\*duplicating\\_cdb\\_as\\_encrypted\*](#)
  - [\*duplicating\\_active\\_pdbs\*](#)

## Goals

Back up CDB and PDBs independently:

- ARCHIVELOG mode at CDB level
- CDB backups and PDB backups: cold and hot backups

Recover CDB or PDBs:

- Instance failure: CDB level
- Complete media recovery:
  - CDB or PDB tempfile
  - Controlfile / redo log file / CDB root essential datafile: CDB mounted
  - PDB datafile: PDB opened if non-essential datafile / PDB mounted if essential datafile
- Incomplete media recovery: CDB mounted or PDB closed
- Flashback database: CDB mounted or PDB closed
- Flashback PDB using PDB snapshots



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Any database, non-CDB or CDB, needs to be backed up in case a potential recovery is required.

The ARCHIVELOG mode can be set only at CDB level.

With a non-CDB, you can perform whole and partial non-CDB backup and, on another level, hot tablespace or datafile backups.

With CDBs, you can still perform the same types of backups. The new level for backing up data is the PDB level.

The granularity of media recovery is very flexible and can be done for the entire CDB, for a PDB, for a tablespace, for a datafile, or even for a block.

You can perform different types of recovery when a failure occurs:

- What to do if an instance failure occurs?
- What to do if a PDB tempfile is missing?
- How to proceed if a controlfile is missing or corrupted?
- What to do if a CDB root essential datafile is missing or corrupted?
- How to proceed if a PDB essential or nonessential datafile is missing or corrupted?
- Do you use flashback database if a local schema is dropped or is there another type of flashback available at the PDB level?
- What if you do not have backups and enough flashback data to recover?

The recovery and flashback procedures are covered in the next lesson.

## Syntax and Clauses in RMAN

```
$ export ORACLE_SID=cdb1
$ rman TARGET / ←→ $ rman TARGET jim@pdb1
```

- DATABASE keyword operates on all PDBs and CDB root or on only one PDB.

```
RMAN> BACKUP DATABASE;
RMAN> RECOVER DATABASE;
```

- PLUGGABLE DATABASE keywords operate on individual PDBs.

```
RMAN> BACKUP PLUGGABLE DATABASE hr_pdb, sales_pdb;
RMAN> RECOVER PLUGGABLE DATABASE hr_pdb;
```

- Back up, restore, recover the CDB root using CDB\$ROOT keyword.

```
RMAN> BACKUP PLUGGABLE DATABASE "CDB$ROOT";
```

- Qualify tablespace of PDB with PDB name.

```
RMAN> BACKUP TABLESPACE sales_pdb:tbs2;
RMAN> RESTORE TABLESPACE system;
```



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

You can use Recovery Manager (RMAN) or Enterprise Manager to back up and recover entire CDBs, partial CDBs, individual whole PDBs, or partial PDBs such as tablespace/datafile of specific PDBs.

The CDB or PDBs are the possible targets for RMAN; an individual PDB is a valid RMAN TARGET database. Connect to the CDB root or a PDB as a user with SYSDBA or SYSBACKUP privilege .

The traditional syntax, such as BACKUP DATABASE, RESTORE DATABASE, and RECOVER DATABASE, operates on the CDB root and all its PDBs and, therefore, on the whole CDB or on a single PDB depending on the connection.

- If you are connected to the CDB root, the BACKUP DATABASE backs up the CDB root and all its PDBs, including the controlfile and the server parameter file (SPFILE).
- If you are connected to a PDB, the BACKUP DATABASE backs up the PDB datafiles.

The PLUGGABLE DATABASE syntax allows backup of, restore, and recover single or several PDBs.

To back up, restore, or recover, the CDB root, CDB\$ROOT, must be used and therefore must be at least mounted.

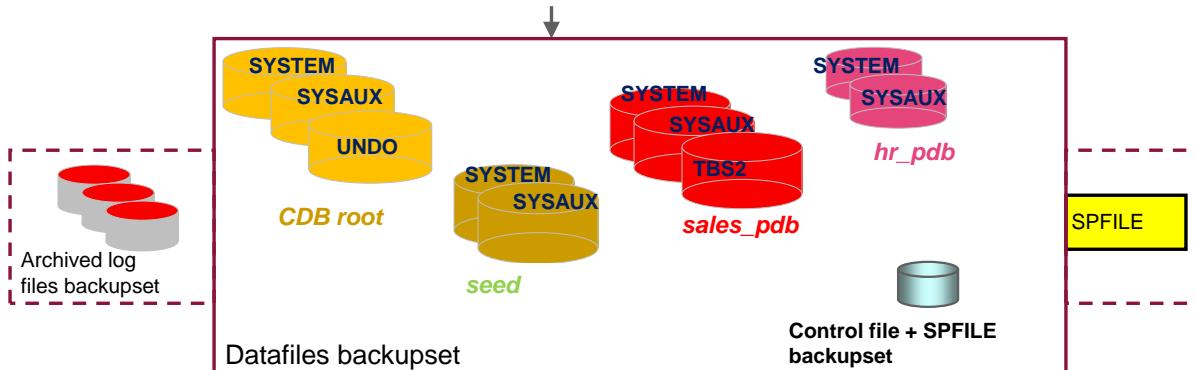
The RMAN TABLESPACE syntax can be qualified with the PDB name so that a user can specify the name of the tablespace as it is known to PDB during backup, restore, and recover commands. If no PDB qualifier is used, then the CDB root is used by default. To list the tablespaces and their associated PDB, use the REPORT SCHEMA syntax.

The same new clauses can be applied to DUPLICATE, SWITCH, REPORT, CONVERT, CHANGE, LIST, DELETE.

## CDB Backup: Whole CDB Backup

- Back up all PDBs datafiles and CDB root files.

```
$ rman TARGET /
RMAN> CONFIGURE CONTROLFILE AUTOBACKUP ON;
RMAN> BACKUP DATABASE PLUS ARCHIVELOG;
```



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

RMAN can back up the entire CDB and contained or individual PDBs. In addition, individual tablespaces or datafiles can be backed up from a specific PDB.

A whole database backup of a CDB can be, in a similar way to non-CDBs, either backup sets or image copies of the entire set of datafiles, namely, the root datafiles and all PDBs datafiles, and the control file. You can optionally include the SPFILE and archived redo log files.

Using RMAN to make an image copy of all the CDB files simply requires mounting or opening the CDB, starting RMAN, connecting to the root with SYSDBA or SYSBACKUP privilege, and entering the BACKUP command shown in the slide. Optionally, you can supply the DELETE INPUT option when backing up archivelog files.

You can also create a backup (either a backup set or image copies) of previous image copies of all datafiles and control files in the CDB by using the following command:

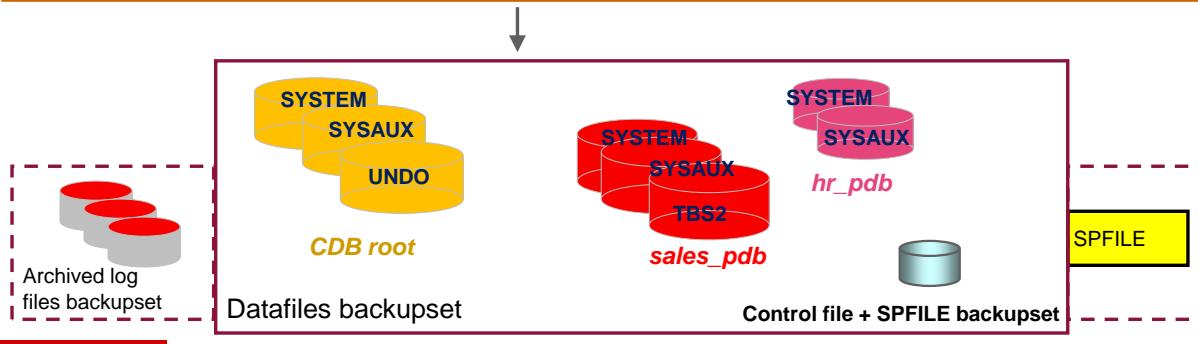
```
RMAN> BACKUP COPY OF DATABASE;
```

## CDB Backup: Partial CDB Backup

- Back up the CDB root and/or individual PDBs.

```
$ rman TARGET /
RMAN> BACKUP PLUGGABLE DATABASE "CDB$ROOT", sales_pdb;
RMAN> BACKUP PLUGGABLE DATABASE hr_pdb PLUS ARCHIVELOG;
```

```
$ rman TARGET sys@hr_pdb
RMAN> BACKUP DATABASE;
```



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

A partial CDB backup backs up the entire set of datafiles of the CDB root, all datafiles of defined PDBs, and the control file and SPFILE because it has been configured to be backed up automatically after each backup.

The command `BACKUP PDB "CDB$ROOT", sales_pdb`, backs up all datafiles of the root container, namely, the SYSTEM, SYSAUX, and UNDO datafiles and then all datafiles of the `sales_pdb` PDB, namely, the SYSTEM, SYSAUX, and TBS2 datafiles.

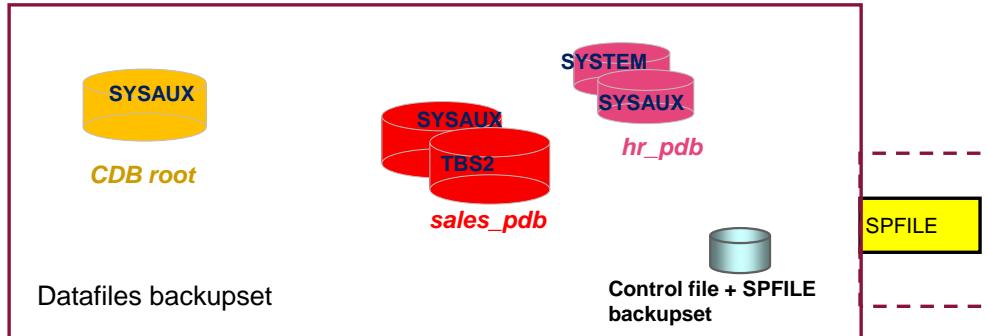
If you connect to the PDB with RMAN like in the second example, some options become invalid such as `PLUS ARCHIVELOG`.

```
RMAN> BACKUP DATABASE PLUS ARCHIVELOG;
...
piece
handle=/u03/app/oracle/fast_recovery_area/ORCL/6587334BFE4E3465E0532133960A01
2c/backupset/2018_02_23/o1_mf_nnndf_TAG20180223T021454_f8yy8z9n_.bkp
tag=TAG20180223T021454 comment=NONE
channel ORA_DISK_1: backup set complete, elapsed time: 00:00:16
Finished backup at 23-FEB-18
```

```
Starting backup at 23-FEB-18
using channel ORA_DISK_1
skipping archived logs when connected to a PDB
backup cancelled because there are no files to backup
Finished backup at 23-FEB-18
```

## PDB Backup: Partial PDB Backup

```
$ rman TARGET /
RMAN> REPORT SCHEMA;
RMAN> BACKUP TABLESPACE sales_pdb:tbs2;
RMAN> BACKUP TABLESPACE hr_pdb:system, sales_pdb:sysaux;
RMAN> BACKUP TABLESPACE sysaux, hr_pdb:sysaux;
```



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

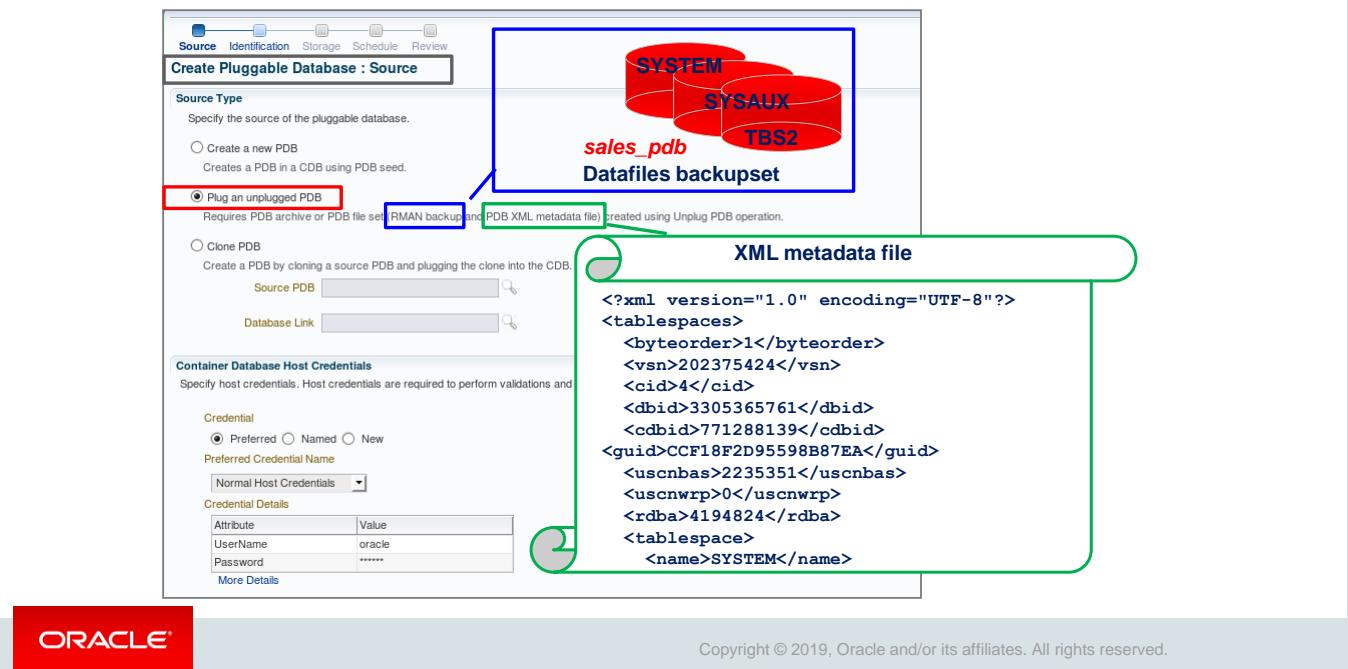
A partial PDB backup backs up the datafiles named tablespaces of individual PDBs and the control file and SPFILE because it has been configured to be backed up automatically at each backup performed.

The example uses the command `BACKUP TABLESPACE` to back up all datafiles of tablespace `TBS2` of `sales_pdb` PDB. To find the names of tablespaces within PDBs, use the `REPORT SCHEMA` command.

The second backup uses the same command to back up all datafiles of tablespace `SYSTEM` of `hr_pdb` PDB and all datafiles of tablespace `SYSAUX` of `sales_pdb` PDB.

The third backup uses the same command to back up all datafiles of tablespace `SYSAUX` of the CDB root and all datafiles of tablespace `SYSAUX` of `hr_pdb` PDB.

# Using RMAN Backup to Plug an Unplugged PDB



Using an RMAN PDB backup, you can create a new pluggable database in a CDB after having unplugged a PDB.

To plug in an unplugged PDB, follow these steps in EM Cloud Control:

- From the Administration menu of the Database page, choose Pluggable Database and then choose Create Pluggable Database.
- On the Source page of the Create Pluggable Database Wizard, select “Plug an unplugged PDB.” This requires:

Either

- A PDB archive, which is a compressed TAR file consisting of PDB XML metadata file and all datafiles that belong to PDB. This method is used for transporting the PDB when both the source and target CDB are using a file system for storage. It is not supported for a PDB using Oracle ASM for storage.

Or

- A PDB file set that consists of PDB XML metadata file and RMAN backup of PDB. This method is recommended for transporting the PDB when the source or target CDB is using Oracle ASM for storage.

using the Unplug PDB operation.

- Enter the Host Credentials for the host. You can choose Preferred, Named, or New credentials and then click Next. Enterprise Manager displays the Destination page of the wizard.

## Duplicating Pluggable Databases

- A single pluggable database

```
RMAN> DUPLICATE DATABASE TO cdb1 PLUGGABLE DATABASE pdb1;
```

- Several pluggable databases

```
RMAN> DUPLICATE DATABASE TO cdb1 PLUGGABLE DATABASE pdb1, pdb3;
```

- All pluggable databases except one

```
RMAN> DUPLICATE DATABASE TO cdb1 SKIP PLUGGABLE DATABASE pdb3;
```

- A PDB and tablespaces of other PDBs

```
RMAN> DUPLICATE DATABASE TO cdb1
 PLUGGABLE DATABASE pdb1 TABLESPACE pdb2:users;
```



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

RMAN enables you to duplicate PDBs by using the DUPLICATE command.

In Oracle Database 12c, to duplicate PDBs, you must create the auxiliary instance as a CDB. To do so, start the instance with the declaration `enable_pluggable_database=TRUE` in the initialization parameter file. When you duplicate one or more PDBs, RMAN also duplicates the CDB root (`CDB$ROOT`) and the CDB seed (`PDB$SEED`). The resulting duplicate database is a fully new, functional CDB that contains the CDB root, the CDB seed, and the duplicated PDBs.

The first example shows how to duplicate a single PDB, the second one how to duplicate a set of PDBs, the third one how to duplicate all the databases in the CDB, except a PDB, and the last one how to duplicate a set of tablespaces within a PDB.

You must be logged in to the CDB root as a user who is granted the `SYSDBA` or `SYSBACKUP` role.

Find the whole procedure in *Oracle Database Backup and Recovery User's Guide*

*12c Release 2 - Chapter Duplicating a Database*

## Cloning Active PDB into an Existing CDB

Duplicate a PDB or PDB tablespaces in active mode to an existing opened CDB.

- Set the COMPATIBLE initialization parameter to 18.1.
- Clone only one PDB at a time.
  - Set the destination CDB in RW mode.
  - Set the REMOTE\_RECOVERY\_FILE\_DEST initialization parameter in the destination CDB to the location where to restore foreign archive log files.

```
RMAN> DUPLICATE PLUGGABLE DATABASE pdb1 AS pdb2 FROM ACTIVE DATABASE
 DB_FILE_NAME_CONVERT ('cdb1', 'cdb2');
```



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

In Oracle Database 18c, the destination instance acts as the auxiliary instance.

- An active PDB can be duplicated directly into an open CDB.
- The passwords for target and auxiliary connections must be the same when using active duplicate.
- In the auxiliary instance, define the location where to restore the foreign archive log files via the new initialization parameter, REMOTE\_RECOVERY\_FILE\_DEST.

RMAN should be connected to the CDB root of the target and auxiliary instances.

### Limitations

- Non-CDB to PDB duplication is not supported.
- Encryption is not supported for PDB cloning.
- SPFILE, NO STANDBY, FARSYNC STANDBY, LOG\_FILE\_NAME\_CONVERT keywords are not supported.
- NORESUME, DB\_FILE\_NAME\_CONVERT, SECTION SIZE, USING COMPRESSED BACKUPSET keywords are supported.

## Example: 1

To duplicate pdb1 from CDB1 into CDB2:

1. Set the REMOTE\_RECOVERY\_FILE\_DEST initialization parameter in CDB2.

```
SQL> ALTER SYSTEM SET REMOTE_RECOVERY_FILE_DEST='/dir_to_restore_archive_log_files';
```

2. Connect to the source (TARGET for DUPLICATE command): CDB1
3. Connect to the existing CDB2 that acts as the auxiliary instance:

```
RMAN> CONNECT TARGET "sys/oracle_4U@cdb1 AS SYSDBA"
RMAN> CONNECT AUXILIARY "sys/oracle_4U@cdb2 AS SYSDBA"
```



4. Start duplicate.

```
RMAN> DUPLICATE PLUGGABLE DATABASE pdb1 TO cdb2 FROM ACTIVE DATABASE;
```

ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The example shows a duplication of pdb1 from cdb1 into the existing cdb2 as pdb1.

To perform this operation, connections to the source (TARGET) cdb1 and to the destination (AUXILIARY) cdb2 are required.

The location where to restore the foreign archive log files in the auxiliary instance is defined via the new initialization parameter, REMOTE\_RECOVERY\_FILE\_DEST.

Then the DUPLICATE command defines that the operation is performed while the source pdb1 is opened.

- cdb2 needs to be opened in read/write.

## Example: 2

To duplicate pdb1 from CDB1 into CDB2 as pdb2:

1. Set the REMOTE\_RECOVERY\_FILE\_DEST initialization parameter in CDB2.

```
SQL> ALTER SYSTEM SET REMOTE_RECOVERY_FILE_DEST='/dir_to_restore_archive_log_files';
```

2. Connect to the source (TARGET for DUPLICATE command): CDB1
3. Connect to the existing CDB2 that acts as the auxiliary instance:

```
rman TARGET sys@cdb1 AUXILIARY sys@cdb2
```



4. Start duplicate.

```
RMAN> DUPLICATE PLUGGABLE DATABASE pdb1 AS pdb2 TO cdb2 FROM ACTIVE DATABASE;
```

**ORACLE®**

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The example shows a duplication of pdb1 from cdb1 into the existing cdb2 as pdb2.

To perform this operation, connections to the source (TARGET) cdb1 and to the destination (AUXILIARY) cdb2 are required.

The location where to restore the foreign archive log files in the auxiliary instance is still defined via the new initialization parameter, REMOTE\_RECOVERY\_FILE\_DEST.

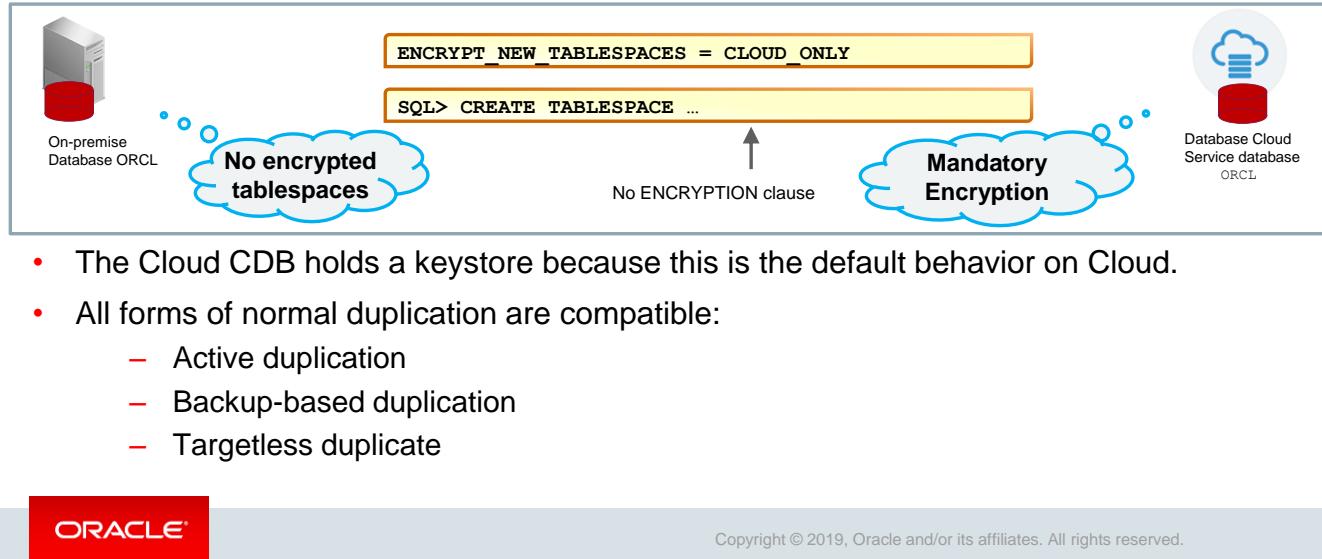
Then the DUPLICATE command defines that the operation is performed while the source pdb1 is opened.

- cdb2 needs to be opened read/write.

# Duplicating On-Premise CDB as Cloud Encrypted CDB

Duplicating an on-premise CDB to the Cloud:

- Any newly created tablespace is encrypted in the Cloud CDB.



If you decide to migrate an on-premise CDB to the Cloud, any tablespace created in the Cloud CDB will be encrypted, even if no encryption clause is declared.

Since Oracle Database 12c, encryption of new user-defined tablespaces is allowed via a new `ENCRYPT_NEW_TABLESPACES` instance parameter.

- A user-defined tablespace that is created in a CDB in the Cloud is transparently encrypted with Advanced Encryption Standard 128 (AES 128) even if the `ENCRYPTION` clause for the `SQL CREATE TABLESPACE` statement is not specified and the `ENCRYPT_NEW_TABLESPACES` instance parameter is being set to `CLOUD_ONLY` by default.
- A user-defined tablespace that is created in an on-premise database is not transparently encrypted. Only the `ENCRYPTION` clause of the `CREATE TABLESPACE` statement determines if the tablespace is encrypted.

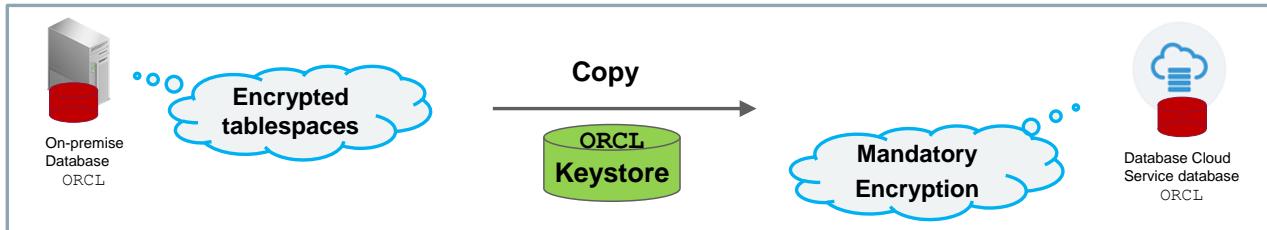
All forms of duplication are compatible except for standby duplicate.

- Active duplication connects as `TARGET` to the source database and as `AUXILIARY` to the Cloud instance.
- Backup-based duplication without a target connection connects as `CATALOG` to the recovery catalog database and as `AUXILIARY` to the Cloud instance. RMAN uses the metadata in the recovery catalog to determine which backups or copies are required to perform the duplication.
- Targetless duplication without connections to the target or to the recovery catalog database instance connects only to the Cloud instance and uses backups or copies of the source database that are stored in a disk location on the destination host. RMAN obtains metadata about where the backups and copies reside from the `BACKUP LOCATION` clause of the `DUPLICATE` command. A disk backup location containing all the backups or copies required for duplication must be available to the destination host, the compute node for the Cloud CDB.

# Duplicating On-Premise Encrypted CDB as Cloud Encrypted CDB

Duplicating an on-premise CDB with encrypted tablespaces to the Cloud:

1. Tablespaces of the source CDB need to be decrypted.



2. Restored tablespaces are re-encrypted in the Cloud CDB.

- Requires the master TDE key from the source CDB keystore
- Requires the source keystore to be copied and opened at the destination CDB

```
RMAN> SET DECRYPTION WALLET OPEN IDENTIFIED BY password;
RMAN> DUPLICATE DATABASE TO orcl FROM ACTIVE DATABASE AS ENCRYPTED;
```

ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

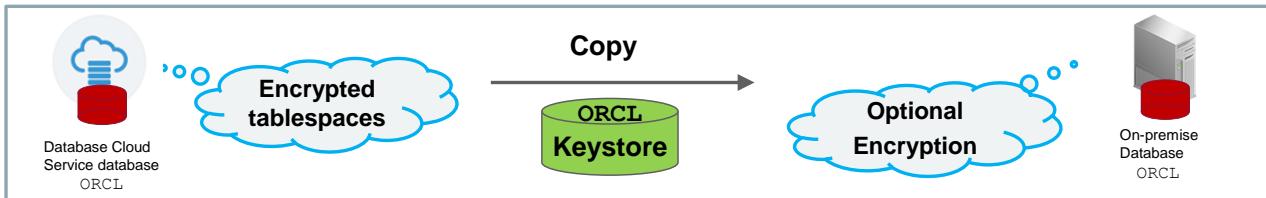
If the source database already contains encrypted tablespaces, the DUPLICATE must have access to the TDE master key of the source (TARGET) database because the clone instance needs to decrypt the datafiles before re-encrypting them during the restore operation. In this case, the keystore has to be copied from the on-premise CDB to the clone instance before starting the DUPLICATE and must be opened.

The DUPLICATE command allows the new “AS ENCRYPTED” clause to restore the CDB with encryption.

Refer to *Oracle Database Backup and Recovery User's Guide 18c – Chapter Duplicating Databases to Oracle Cloud* and more particularly [RMAN Duplicate from an Active Database](#) for more details about duplicating databases to Oracle Cloud Infrastructure.

# Migrating Cloud Encrypted CDB as On-Premise CDB

- Tablespaces of the source CDB are necessarily encrypted.



- Restored tablespaces need to be decrypted to be created:
  - Requires the TDE master key from the source CDB keystore
  - Requires the source keystore to be copied and opened at the destination CDB

```
RMAN> SET DECRYPTION WALLET OPEN IDENTIFIED BY password;
RMAN> DUPLICATE DATABASE TO orcl FROM ACTIVE DATABASE AS DECRYPTED;
```



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The source database already contains encrypted tablespaces; therefore, DUPLICATE must have access to the master key of the source (TARGET) database because the clone instance needs to decrypt the datafiles before the restore operation. In this case, the keystore has to be copied from the Cloud CDB to the clone instance before starting DUPLICATE and must be opened by using the SET DECRYPTION WALLET OPEN IDENTIFIED BY "password" command.

The DUPLICATE command uses the "AS DECRYPTED" clause to restore the CDB without encryption.

If the user does not have Advanced Security Option (ASO) license on on-premise side, the on-premise database cannot have TDE encrypted tablespaces. The DUPLICATE command using the "AS DECRYPTED" clause provides a way to get encrypted tablespaces/databases from Cloud to on-premise servers. It is important to note that it will not decrypt tablespaces that were explicitly created with encryption, using the ENCRYPTION USING clause.

Refer to *Oracle Database Backup and Recovery User's Guide 18c – Chapter Duplicating an Oracle Cloud Database as an On-premise Database*.

# Checking for Block Corruption



Invoking proactive health check of the database and its components using RMAN `VALIDATE` command:

- Scans the specified files and verifies their contents
  - CDB: All datafiles of the CDB root and PDBs

```
RMAN> VALIDATE DATABASE;
```

- CDB root: All datafiles of the CDB root only

```
RMAN> VALIDATE DATABASE ROOT;
```

- PDB: All datafiles of the listed PDBs

```
RMAN> VALIDATE PLUGGABLE DATABASE pdb1, pdb2;
```

- Confirms that the datafiles exist and are in the correct location
- Checks for corrupt data blocks

ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

For very important databases, you may want to execute proactive checks (possibly daily during low peak interval periods). You can schedule periodic health checks by using the RMAN `VALIDATE` command to validate any datafile of a whole CDB, of the CDB root only, of a PDB, and of individual backup sets and data blocks.

- When connected to the CDB root, validate:
  - The whole CDB using the `VALIDATE DATABASE` command
  - The CDB root only using the `VALIDATE DATABASE ROOT` command
  - A PDB or several PDBs using the `VALIDATE PLUGGABLE DATABASE` command
- When connected to the PDB, validate the PDB with the `VALIDATE DATABASE` command.

Any problem detected during validation is displayed to you. If a failure is detected, it is logged into ADR as a finding.

Similarly, you can validate the restore of the CDB, the CDB root only, or PDBs using the `RESTORE DATABASE VALIDATE`, `RESTORE DATABASE ROOT VALIDATE`, and `RESTORE PLUGGABLE DATABASE pdb1 VALIDATE` commands, respectively.

For more detailed information about logical and physical corruption, refer to the Oracle University course *Oracle Database 12c: Backup and Recovery Workshop*.

## Summary

In this lesson, you should have learned how to:

- Back up a CDB
- Back up a PDB
- Duplicate an active PDB into an existing CDB
- Duplicate a CDB as encrypted
- Validate CDB and PDBs



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Practice 8: Overview

- 8-1: RMAN whole CDB backup
- 8-2: RMAN PDB backup
- 8-3: Duplicating a PDB into an existing CDB
- 8-4: Duplicating an on-premises CDB for Cloud



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

# Recovery and Flashback

The ORACLE logo, featuring the word "ORACLE" in white capital letters inside a red rectangular box.

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

# Objectives

After completing this lesson, you should be able to:

- Recover a PDB from essential file damage
- Recover a PDB from nonessential file damage
- Reuse preplugin backups after conversion of a non-CDB to a PDB
- Reuse preplugin backups after plugging/relocating a PDB into another CDB
- Perform CDB flashback
- Perform PDB flashback
- Use clean restore points to complete PDB flashback
- Manage PDB snapshots
- Switch over a refreshable cloned PDB



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

For a complete understanding of the recovery and flashback procedures under the multitenant architecture, refer to the Oracle documentation:

- *Oracle Database Backup and Recovery User's Guide 18c*
- “Backing Up and Recovering CDBs and PDBs” in *Oracle Multitenant Administrator’s Guide 18c*
- “Administering a PDB Snapshot Carousel” in *Oracle Multitenant Administrator’s Guide 18c*

Refer to other sources of information available under Oracle Learning Library like *Oracle By Example (OBEs)*:

- *recovering\_pdbs\_using\_preplugin\_backups*
  - *recovering\_plugged\_noncdb\_using\_preplugin\_backups*
  - *managing\_and\_using\_pdb\_snapshots\_carousel*
  - *switching\_over\_refreshable\_pdb*

## Goals

Recover CDB or PDBs:

- Instance failure: CDB level
- Complete media recovery:
  - CDB or PDB tempfile
  - Controlfile / redo log file / CDB root essential datafile: CDB mounted
  - PDB datafile: PDB opened if nonessential datafile / PDB mounted if essential datafile
- Incomplete media recovery: CDB mounted or PDB closed
- Flashback database: CDB mounted or PDB closed



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Any database, non-CDB or CDB, needs to be backed up in case a potential recovery is required.

The ARCHIVELOG mode can be set only at CDB level.

With a non-CDB, you can perform whole and partial non-CDB backup and, on another level, hot tablespace or datafile backups.

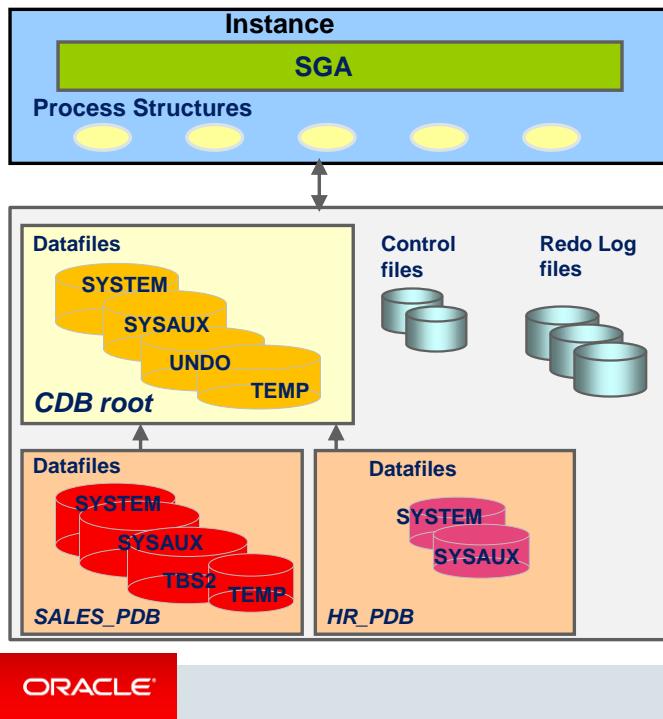
With CDBs, you can still perform the same types of backups. The new level for backing up data is the PDB level.

The granularity of media recovery is very flexible and can be done for the entire CDB, for a PDB, for a tablespace, for a datafile, or even for a block.

You can perform different types of recovery when a failure occurs:

- What to do if an instance failure occurs?
- What to do if a PDB tempfile is missing?
- How to proceed if a controlfile is missing or corrupted?
- What to do if a CDB root essential datafile is missing or corrupted?
- How to proceed if a PDB essential or nonessential datafile is missing or corrupted?
- Do you use flashback database if a local schema is dropped or is there another type of flashback available at the PDB level?

## Instance Failure and Instance Recovery



PDB instance recovery is **impossible**.

After instance failure:

- Connect to the CDB root.
- Open the CDB root.
- Open all PDBs.

```
SQL> STARTUP
SQL> ALTER PLUGGABLE DATABASE ALL OPEN;
```

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Crash/instance recovery is supported for a CDB as a whole, because there is one single instance for the root and all its PDBs.

Redo log files required to perform the instance recovery are stored in the unique set of redo log files of the CDB. There is a single shared redo stream for the CDB root and all the PDBs used to perform instance recovery. The instance recovery is performed while opening the CDB root.

There is no way to perform a PDB instance recovery.

For an instance to open a datafile, the system change number (SCN) contained in the datafile's header must match the current SCN that is stored in the control files.

If the numbers do not match, the instance applies redo data from the online redo logs, sequentially “redoing” transactions until the datafiles are up-to-date. After all datafiles have been synchronized with the control files, the CDB root is opened, and by default all PDBs are still in mount state.

When redo logs are applied, all transactions are applied to bring the CDB up to the state as of the time of failure. This usually includes transactions that are in progress but have not yet been committed. After the CDB root has been opened, the uncommitted transactions are rolled back on the CDB root datafiles. After the PDBs are opened, the uncommitted transactions are rolled back on PDBs' datafiles. When opening a CDB, by default all PDBs remain in mounted state. Either triggers are automatically executed to open the PDBs or the open saved state for the PDBs have been previously kept or you execute the `ALTER PLUGGABLE DATABASE ALL OPEN` command to open them all.

## NOARCHIVELOG Mode

If the database is in NOARCHIVELOG mode, and a datafile is lost, perform the following tasks:

- Shut down the instance if it is not already down.
- Restore the entire CDB including all datafiles and control files.
- Start up the instance and open the CDB and all PDBs.

Users must reenter all changes made since the last backup.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The loss of any datafile from a CDB in NOARCHIVELOG mode requires complete restoration of the CDB, including control files and all datafiles of the CDB root and all PDBs. If you have incremental backups, then you need to perform the restore and recover operations on the CDB.

With the database in NOARCHIVELOG mode, recovery is possible only up to the time of the last backup. So users must reenter all changes made since that backup.

For this type of recovery, proceed with the following steps:

- Shut down the instance if it is not already down.
- Restore the entire CDB, including all data and controlfiles from the last backup.
- Start up the instance and open the root and PDBs.

In the following slides, you consider that the CDB is in ARCHIVELOG mode.

## PDB Tempfile Recovery

SQL statements that require temporary space to execute may fail if one of the tempfiles is missing.

```
SQL> CONNECT local_user@HR_PDB
SQL> select * from my_table order by 1,2,3,4,5,6,7,8,9,10,11,12,13;
 select * from my_table order by 1,2,3,4,5,6,7,8,9,10,11,12,13
 *
ERROR at line 1:
ORA-01565: error in identifying file
'/u01/app/oracle/oradata/CDB1/HR_PDB/temp2_01.dbf'
ORA-27037: unable to obtain file status
Linux Error: 2: No such file or directory
```

- Automatic re-creation of temporary files at PDB opening
- Manual re-creation also possible



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

If a tempfile belonging to a PDB temporary tablespace is lost or damaged, and the user issuing the statement uses it, an error occurs during the execution of SQL statements that require that temporary space for sorting.

The SQL statement shown in the slide has a long list of columns to order by, which results in the need for temporary space from the PDB temporary tablespace. The missing file error is encountered when this statement requiring a sort is executed.

The PDB can open with a missing temporary file. If any of the temporary files do not exist when the PDB is opened, they are re-created automatically at PDB startup.

You can perform a manual re-creation instead, while connected to the PDB:

```
SQL> ALTER TABLESPACE temp ADD TEMPFILE
 '/u01/app/oracle/oradata/CDB1/HR_PDB/temp2_02.dbf'
 SIZE 20M;
SQL> ALTER TABLESPACE temp DROP TEMPFILE
 '/u01/app/oracle/oradata/CDB1/HR_PDB/temp2_01.dbf';
```

## PDB SYSTEM or UNDO Tablespace Recovery

The CDB and all other PDBs can be left opened.

1. Connect to the PDB.
2. “Shutdown abort” the PDB if it is not automatically done.

```
$ sqlplus sys@sales_pdb as sysdba
SQL> SHUTDOWN ABORT
```

or

```
SQL> ALTER PLUGGABLE DATABASE CLOSE ABORT;
```

3. Restore and recover the PDB or the missing tablespace or the damaged datafile:

```
$ rman target sys@sales_pdb
RMAN> RESTORE DATABASE;
RMAN> RECOVER DATABASE;
RMAN> ALTER PLUGGABLE DATABASE sales_pdb OPEN;
```



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Closing a PDB with the ABORT mode forcefully closes it without bringing down the entire CDB instance. It is successful regardless of the states of the actual PDB datafiles.

This operation requires you to be connected to the PDB, have the ALTER PLUGGABLE DATABASE system privilege, and have the CDB in ARCHIVELOG mode.

If the datafile that is missing or corrupted belongs to a PDB and more specifically to the SYSTEM or UNDO tablespace, the PDB must be shut down in ABORT mode.

The health checker can automatically detect that there is a severe failure and aborts the PDB.

```
Checker run found 1 new persistent data failures
2015-10-07T01:17:39.950612+00:00
SALES_PDB(5):ALTER PLUGGABLE DATABASE CLOSE ABORT
2015-10-07T01:17:39.978484+00:00
SALES_PDB(5):KILL SESSION for sid=(254, 7528):
SALES_PDB(5): Reason = PDB close abort
SALES_PDB(5): Mode = KILL HARD FORCE -/-
```

A PDB, tablespace, or datafile media recovery is required before the PDB can be reopened.

In case the PDB is the application root of an application container, other datafiles in the application PDBs may have to be restored and recovered as well.

## PDB non-SYSTEM Tablespace Recovery

Similar to non-CDBs: Perform the recovery within the PDB

- Connect to the PDB.
- Put the tablespace OFFLINE.
- Other PDBs are not impacted.

```
SQL> CONNECT system@sales_pdb
SQL> ALTER TABLESPACE tbs2 OFFLINE IMMEDIATE;
RMAN> CONNECT TARGET /
RMAN> RESTORE TABLESPACE sales_pdb:tbs2;
RMAN> RECOVER TABLESPACE sales_pdb:tbs2;
SQL> ALTER TABLESPACE tbs2 ONLINE;
```

**Note:** You can also use the REPAIR command.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

If the datafile that is missing or corrupted belongs to a PDB and more specifically to any tablespace other than SYSTEM tablespace, the PDB need not be closed. The datafile with the error is taken OFFLINE IMMEDIATE. Media recovery for that datafile is required and performed in a similar way to media recovery of a set of tablespaces. During that recovery time, users can work with other PDB tablespaces and within other PDBs.

RMAN RESTORE and RECOVER commands are frequently run consecutively. The RMAN REPAIR command accepts DATAFILE, TABLESPACE, PLUGGABLE DATABASE, and DATABASE options and performs both RESTORE and RECOVER commands. Where possible, REPAIR automatically takes a file offline, restores and recovers it, and then brings it back online again:

1. Automatically offlines files (if applicable)
2. Restores designated files
3. Recovers designated files
4. Automatically onlines any files offlined in step 1

# PITR

- PDB PITR

```
RMAN> ALTER PLUGGABLE DATABASE pdb1 CLOSE;
RMAN> RUN {
 SET UNTIL SCN = 1851648 ;
 RESTORE pluggable DATABASE pdb1;
 RECOVER pluggable DATABASE pdb1
 AUXILIARY DESTINATION='/u01/app/oracle/oradata';
 ALTER PLUGGABLE DATABASE pdb1 OPEN RESETLOGS;
 }
```

- PDB Tablespace PITR

```
RMAN> RECOVER TABLESPACE pdb1:test_tbs
 UNTIL SCN 832972
 AUXILIARY DESTINATION '/tmp/CDB1/reco';
RMAN> ALTER TABLESPACE pdb1:test_tbs ONLINE;
```



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

If you need to recover a PDB database to a point in time in the past beyond flashback retention, then in this case, flashback is not possible; therefore, a point-in-time recovery is necessary.

Recovering a PDB to a point in time does not affect all parts of the CDB—the whole CDB is still opened and, therefore, all other PDBs are opened. After recovering a PDB to a specified point in time, when you open the PDB using the `RESETLOGS` option, a new incarnation of the PDB is created. The PDB `RESETLOGS` does not perform a `RESETLOGS` for the CDB.

- A PDB record in the controlfile is updated.
- Each redo log record carries PDB ID in the redo header. This is how recovery knows which redo applies to which PDB. Redo logs are shared by all PDBs—redo from each PDB is written to a single set of redo logs.

Conceptually, a PDB resetlogs is similar to a database resetlogs.

After recovery, the old backup of the PDB remains valid and can be used if a media failure occurs.

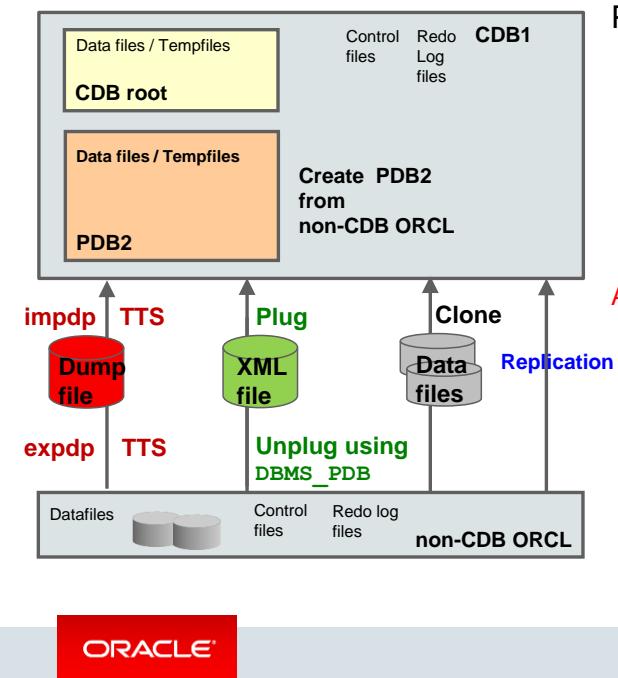
A PDB incarnation is a subincarnation of the CDB. For example, if the CDB is incarnation 5, and a PDB is incarnation 3, then the fully specified incarnation number of the PDB is (5, 3). The initial incarnation of a PDB is 0. To view the incarnation of a PDB, query the `V$PDB_INCARNATION` view.

If you do not use a fast recovery area, you must specify the temporary location of the auxiliary set files by using the `AUXILIARY DESTINATION` clause (example in the slide).

Each PDB has its own set of tablespaces. TSPITR can be used to recover a tablespace to an earlier point in time. Perform the TSPITR as described in the second example in the slide, specifying the full tablespace name including the PDB name.

Recovering a tablespace of a PDB to a point in time does not affect all parts of the CDB —the whole CDB is still opened and, therefore, all PDBs are opened. After recovering a tablespace back in time, put the tablespace back online.

# Migrating a Non-CDB to a CDB



## Possible methods:

- Data Pump (TTS or TDB or full export/import)
- Plugging (XML file definition with `DBMS_PDB`)
- Cloning
- Replication

## After conversion:

- Is it possible to recover the PDB back in time before the non-CDB was converted?
- Are the non-CDB backups transported with the non-CDB?



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

In Oracle Database 12c, there are different possible methods to migrate a non-CDB to a CDB.

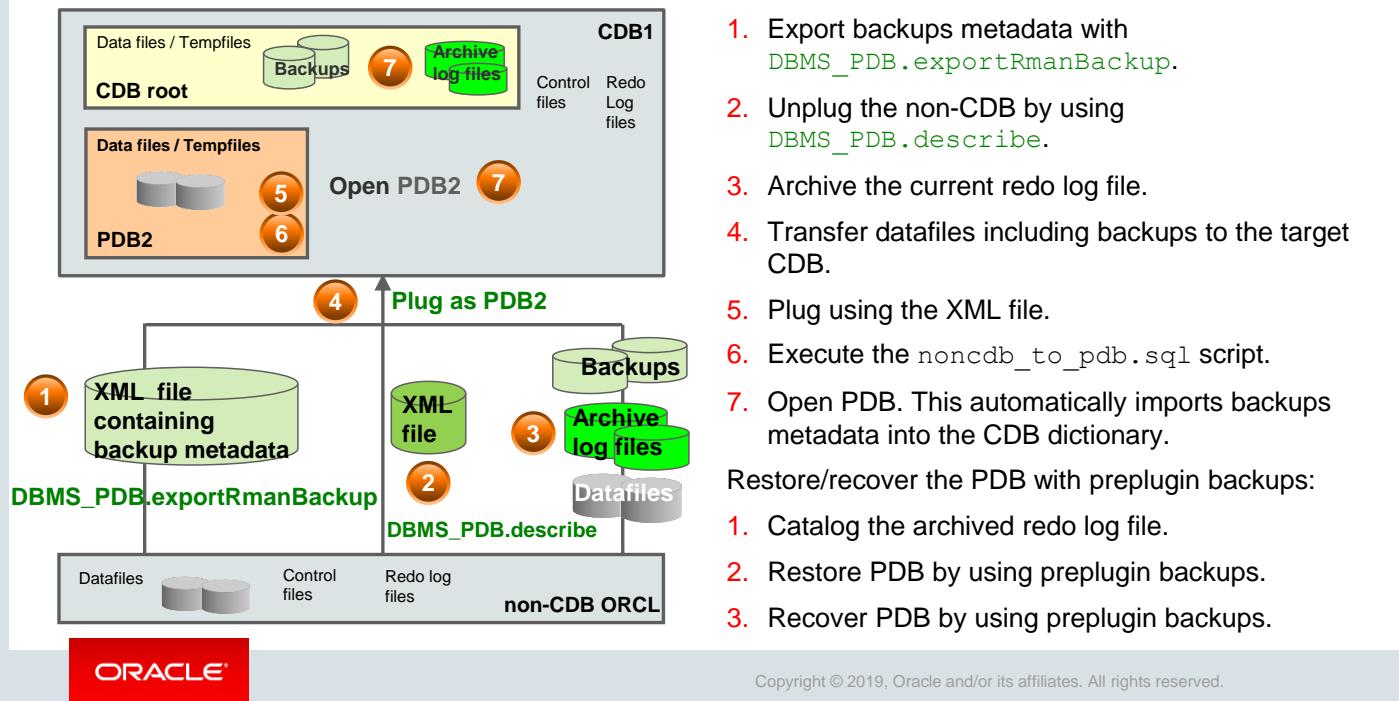
Whichever method is used, are the non-CDB backups transported with the non-CDB during the migration?

- Does Oracle Data Pump transport the non-CDB backups?
  - You use either transportable tablespace (TTS) or full conventional export/import or full transportable database (TDB) provided that in the last one any user-defined object resides in a single user-defined tablespace.
  - Using any of these Data Pump methods, Data Pump transports objects definition and data, but not backups
- Does the plugging method transport the non-CDB backups? Generating an XML metadata file from the non-CDB to use it during the plugging step into the CDB only describes the non-CDB datafiles of the non-CDB, but it does not describe the list of backups associated to the non-CDB.
- Does the cloning method transport the non-CDB backups? Cloning non-CDBs in a CDB requires copying the files of the non-CDB to a new location. It does not copy the backups to a recovery location.
- Does replication transport the non-CDB backups? The replication method replicates the data from a non-CDB to a PDB. When the PDB catches up with the non-CDB, you fail over to the PDB. Backups are not associated with the replication.

Because there are no backups transported with the non-CDB into the target CDB, no restore or recovery using the old backups is possible. Even if the non-CDB backups were manually transported/copied to the target CDB, users cannot perform restore/recover operations using these backups. You had to create a new baseline backup for the CDB converted as a PDB.

# Migrating a Non-CDB and Transporting Non-CDB Backups to a CDB

18c



In Oracle Database 18c, you can transport the existing backups and backed-up archive log files of the non-CDB and reuse them to restore and recover the new PDB.

The backups transported from the non-CDB into the PDB are called preplugin backups.

Transporting the backups and backed-up archive log files associated to a non-CDB before migration requires the following steps:

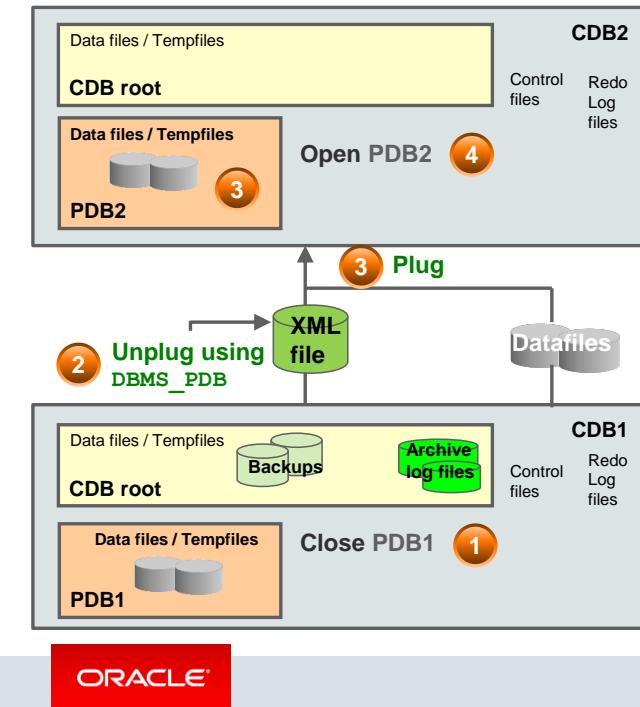
1. The following new `DBMS_PDB.exportRmanBackup` procedure must be executed in the non-CDB opened in read/write mode. This is a mandatory step for non-CDB migration:  

```
SQL> EXEC dbms_pdb.exportRmanBackup()
```

The procedure exports all RMAN backup metadata that belongs to the non-CDB into its own dictionary. The metadata is transported along with the non-CDB during the migration.
2. Use `dbms_pdb.describe` to generate an XML metadata file from the non-CDB describing the structure of the non-CDB with the list of datafiles.
3. Archive the current redo log file required for a potential restore/recover using preplugin backups.
4. Transfer the datafiles, backups, and archive log files to the target CDB.
5. Use the XML metadata file during the plugging step to create the new PDB into the CDB.
6. Run the `ORACLE_HOME/rdbms/admin/noncdb_to_pdb.sql` script to delete unnecessary metadata from the PDB `SYSTEM` tablespace.
7. Open the PDB. When the PDB is open, the exported backup metadata is automatically copied into the CDB dictionary, except the current redo log file archived in step 3. Catalog the archived redo log file as one of the preplugin archived logs.

Because the backups for the PDB are now available in the CDB, they can be reused to recover the PDB.

## Relocating/Plugging a PDB into Another CDB



After relocating/plugging the PDB into another CDB:

- Is it possible to recover the PDB back in time before it was relocated/unplugged?
- Are the PDB backups transported with the relocated/unplugged PDB?



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

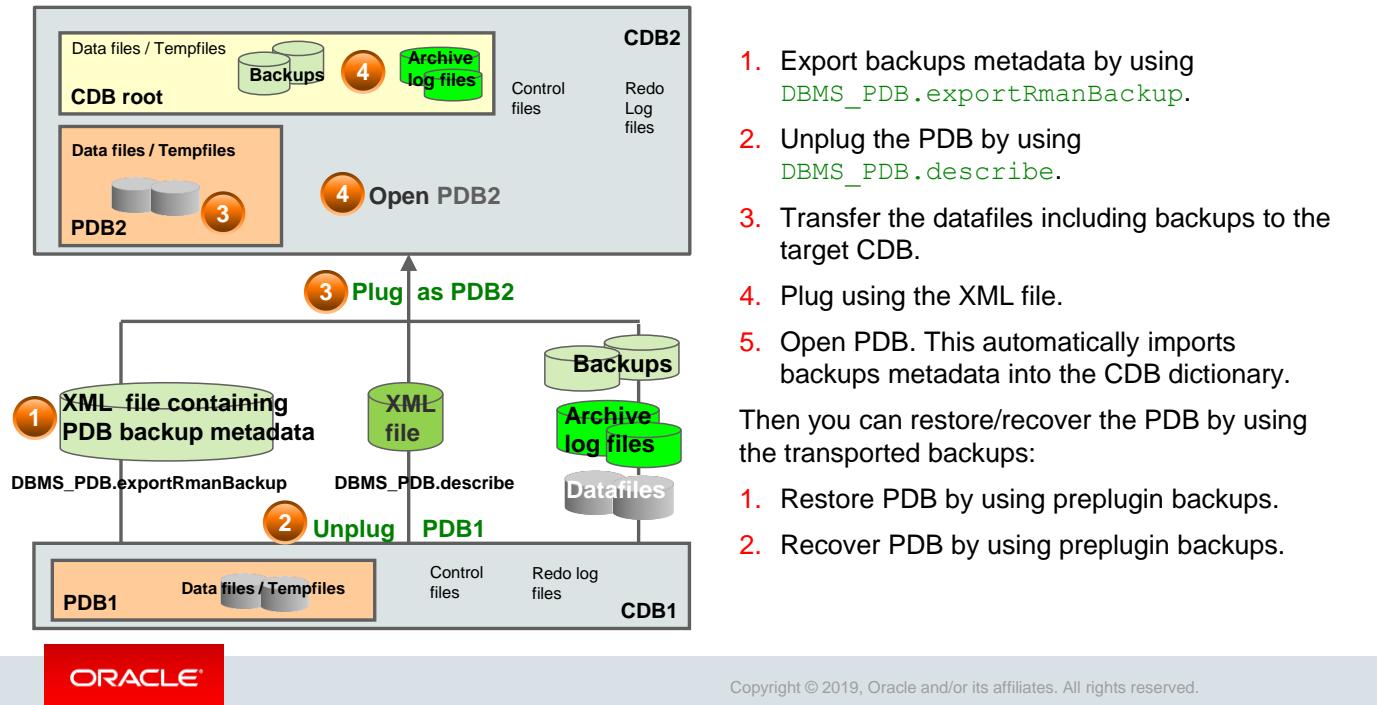
In Oracle Database 12c, PDBs can be hot-cloned from one CDB into another CDB by using local UNDO tablespaces.

Are the PDB backups transported with the PDB during the cloning?

- Cloning a PDB into another CDB requires copying the files of the PDB to a new location. It does not copy the backups to a recovery location.

If there are no backups transported with the PDB into the target CDB, no restore or recovery using the old backups is possible. Even if the PDB backups were manually transported/copied to the target CDB, users cannot perform restore/recover operations using these backups. You had to create a new baseline backup for PDBs relocated or plugged.

# Plugging a PDB and Transporting PDB Backups to a CDB - 1



In Oracle Database 18c, you can transport the existing backups and backed-up archive log files of the PDB and reuse them to restore and recover the new plugged PDB.

To transport the backups and backed-up archive log files associated to a PDB before replugging the PDB, perform the following steps:

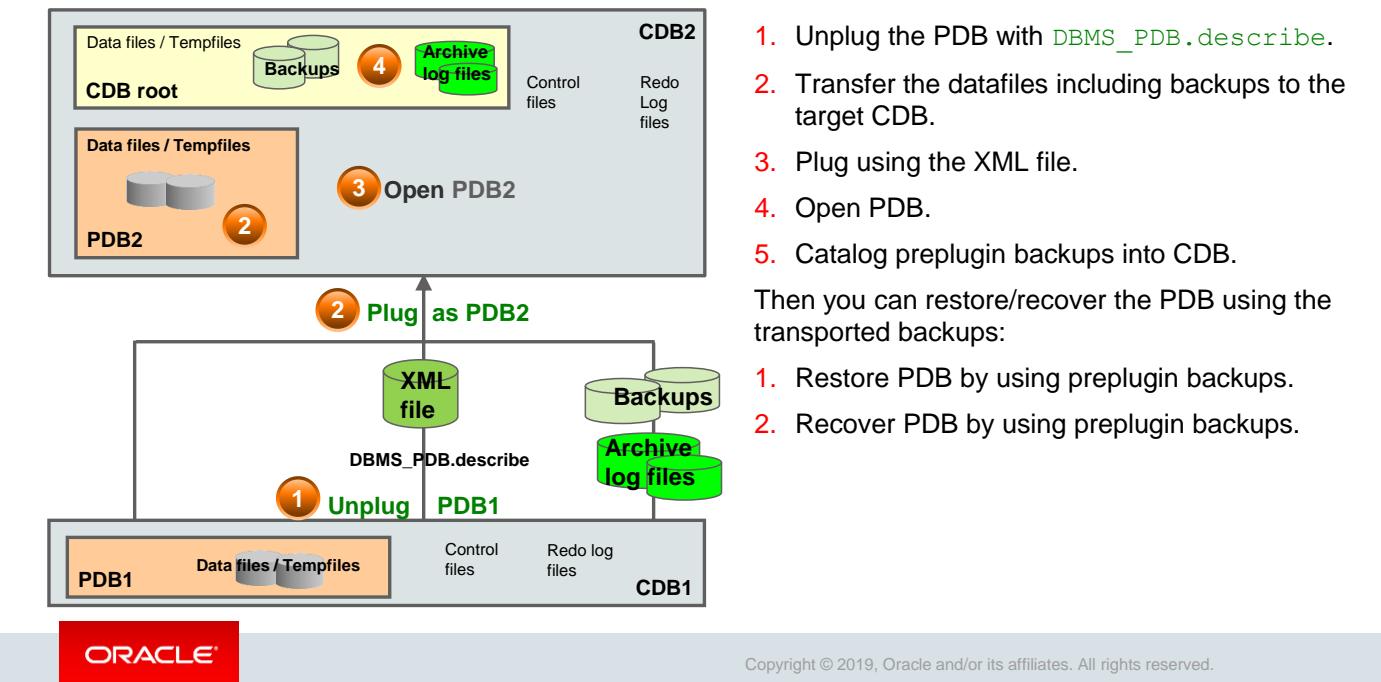
1. The following new `DBMS_PDB.exportRmanBackup` procedure can be executed in the PDB opened in read/write mode. This is not a mandatory step before unplugging the PDB:  

```
SQL> EXEC dbms_pdb.exportRmanBackup ('< pdb_name >')
```

The procedure exports all RMAN backup metadata that belongs to the PDB into its own dictionary. The metadata is transported along with the PDB during the unplug operation.
2. Unplug the PDB.
3. Transfer the datafiles, backups, and archive log files to the target CDB.
4. Plug the PDB with the `COPY` clause to copy the datafiles, backups, and backed-up archive log files of the source PDB into a new directory.
5. Open the new PDB. When the PDB is open, the exported backup metadata is automatically copied into the CDB dictionary.

Because the backups for the PDB are now available in the CDB, they can be reused to recover the PDB.

## Plugging a PDB and Transporting PDB Backups to a CDB - 2



If you forgot to execute the `DBMS_PDB.exportRmanBackup` procedure before unplugging the PDB, you can still catalog the existing backups and backed-up archive log files of the plugged PDB after the plugging operation and reuse them to restore and recover the plugged PDB.

If preplugin backups and archive log files are moved or new backups and archive log files were created on the source CDB after the PDB was transported, then the target CDB does know about them. You can catalog those preplugin files:

```
RMAN> SET PREPLUGIN CONTAINER=< pdb_name >;
RMAN> CATALOG PREPLUGIN ARCHIVELOG '< archivelog >';
RMAN> CATALOG PREPLUGIN BACKUP '< backup_name >';
```

## Using PrePlugin Backups

Use the `PrePlugin` option to perform RMAN operations using preplugin backups.

- Restore a PDB from its preplugin backups cataloged in the target CDB.

```
RMAN> RESTORE PLUGGABLE DATABASE pdb_noncdb FROM PREPLUGIN;
```

- Recover a PDB from its preplugin backups until the datafile was plugged in.

```
RMAN> RECOVER PLUGGABLE DATABASE pdb_noncdb FROM PREPLUGIN;
```

- Check whether preplugin backups and archive log files are cataloged in the target CDB.

```
RMAN> SET PREPLUGIN CONTAINER pdb1;
RMAN> LIST PREPLUGIN BACKUP;
RMAN> LIST PREPLUGIN ARCHIVELOG ALL;
RMAN> LIST PREPLUGIN COPY;
```

- Verify that cataloged preplugin backups are available on disk.

```
RMAN> CROSSCHECK PREPLUGIN BACKUP;
RMAN> DELETE PREPLUGIN BACKUP;
```

**ORACLE®**

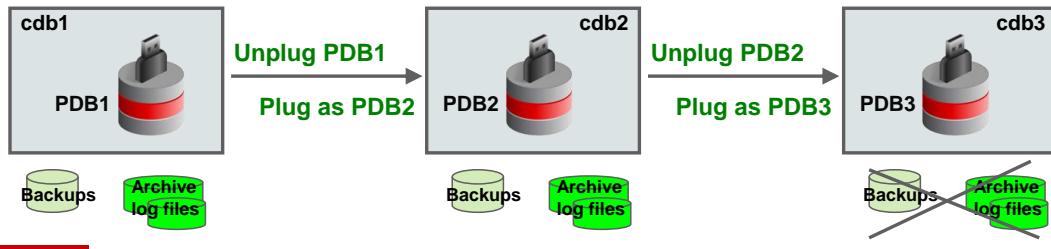
Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

- A restore operation from preplugin backups restores the datafiles taken before the PDB was plugged in.
- A recover operation using preplugin backups use preplugin incremental backup and archivelogs to recover the datafile until the datafile was plugged in. At the end of the recover operation, the datafile is checkpointed as of plugin SCN.  
The preplugin archivelogs are restored to the target archivelog destination by default as long as the target archivelog destination is not a fast recovery area (FRA). If the target archivelog destination is the FRA, then the user has to provide an explicit archivelog destination using the `SET ARCHIVELOG DESTINATION` command before executing `RECOVER FROM PREPLUGIN`.
- If there are preplugin metadata that belongs to more than one PDB, a command that does not clarify the PDB the syntax refers to errors out, indicating that the user should scope the PDB. The scoping of PDB can be done by using the `SET PREPLUGIN CONTAINER` command. Scoping is not necessary if the user has connected to PDB as the target CDB. The `SET PREPLUGIN CONTAINER` command is necessary if you connected to the target CDB.
- `CROSSCHECK`, `DELETE`, and `CHANGE` commands can use the `PREPLUGIN` option. The `CROSSCHECK` command can validate the existence of preplugin backups, archived log files, and image copies. The `DELETE` command can delete preplugin backups, archived log files and image copies, and also expired preplugin backups.

```
RMAN> DELETE EXPIRED PREPLUGIN BACKUP;
RMAN> CHANGE PREPLUGIN archivelog all unavailable;
RMAN> CHANGE PREPLUGIN backup available;
RMAN> CHANGE PREPLUGIN copy unavailable;
```

## To Be Aware Of

- The source and destination CDBs must have COMPATIBLE set to 18.1 or higher to create/restore/recover preplugin backups.
- In case of plugging in a non-CDB, the non-CDB must use ARCHIVELOG mode.
- The target CDB does not manage preplugin backups.
  - Use CROSSCHECK and DELETE commands to manage the preplugin backups.
- A RESTORE using preplugin backups can restore datafiles from one PDB only.
- Backups taken by the source `cdb1` are visible in target `cdb2` only.



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

- The target CDB does not manage the source database backups. However, there are commands to delete and crosscheck the source database backups.
- In one RMAN command, you cannot specify datafiles belonging to different PDBs when using preplugin backups.
- The CDB root must be opened to make use of preplugin backups.
- Backups taken by the source `cdb1` are visible in target `cdb2` only. For instance, a PDB can migrate from `cdb1` to `cdb2` and then to `cdb3`. The backups of the PDB taken at `cdb1` are accessible by `cdb2`. They are not accessible by `cdb3`. The `cdb3` can only see backups of the PDB taken by `cdb2`.

## Example

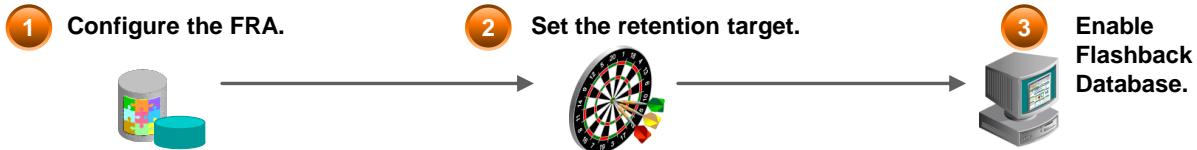
```
RMAN> SET PREPLUGIN CONTAINER pdb1;
RMAN> CATALOG PREPLUGIN ARCHIVELOG '/u03/app/.../o1_mf_1_8_dngwm59v_.arc';
RMAN> RUN { RESTORE PLUGGABLE DATABASE pdb1 FROM PREPLUGIN;
 RECOVER PLUGGABLE DATABASE pdb1 FROM PREPLUGIN;
}
RMAN> RECOVER PLUGGABLE DATABASE pdb1;
```



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

In this example, you first avoid any ambiguity about which PDB the backups belong to by scoping the PDB. Then you catalog the last archive log file created after the PDB was unplugged and the metadata exported. Then you restore and recover the PDB by using preplugin backups. And finally, you run a normal media recovery after recovering from preplugin backups.

# CDB and PDB Flashback



```
SQL> STARTUP MOUNT;
SQL> ALTER DATABASE ATTACHIVELOG;
SQL> ALTER DATABASE OPEN;
SQL> ALTER SYSTEM SET DB_FLASHBACK_RETENTION_TARGET=2880 SCOPE=BOTH;
SQL> ALTER DATABASE FLASHBACK ON;
SQL> ALTER DATABASE OPEN;
```

- No flashback of CDB root without flashing back the whole CDB
- PDB flashback similar to CDB flashback

```
RMAN> CONN sys@pdb1
RMAN> ALTER PLUGGABLE DATABASE CLOSE;
RMAN> FLASHBACK PLUGGABLE DATABASE pdb1 TO SCN 411010;
RMAN> ALTER PLUGGABLE DATABASE pdb1 OPEN RESETLOGS;
```

ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## CDB Flashback

You can configure Flashback Database for the CDB as you would for any non-CDB. You cannot flashback the CDB root without flashing back the entire CDB. You can flashback a PDB.

## PDB Flashback

All datafiles belonging to a PDB can be flashed back and recovered in place. All undo application that is needed to make a PDB consistent after flashback is applied in place. After a PDB flashback operation, the old backup of the PDB is still valid.

In the second example in the slide, OPEN RESETLOGS creates a new incarnation for the PDB, similar to a database OPEN RESETLOGS, which creates a new incarnation for the CDB. The CDB root must be opened when the PDB OPEN RESETLOGS is performed, except that recovering a PDB to a point in time does not affect all parts of the CDB. The whole CDB and all other PDBs are opened. After recovering a PDB to a specified point in time, when you open the PDB by using the RESETLOGS option, a new incarnation of the PDB is created.

The PDB RESETLOGS option does not perform a RESETLOGS for the CDB. A PDB record in the control file is updated. Each redo log record carries the PDB ID in the redo header. This is how recovery knows which redo applies to which PDB. Redo logs are shared by all PDBs; redo from each PDB is written to a single set of redo logs. Conceptually, a PDB OPEN RESETLOGS is similar to a database OPEN RESETLOGS. Query V\$PDB\_INCARNATION for details.

This is very useful when a user error has been issued, for example, a DROP USER. Instead of restoring and recovering the PDB, flashback to the time before the user error had been issued is easy.

# PDB Flashback and Clean Restore Point

- Clean PDB restore points can be created after a PDB is closed and ONLY in shared undo mode.
- The benefits of clean PDB restore points include:
  - Faster than other types of PDB flashback
  - No restore of any backup
  - No clone instance created
  - No need to take a new backup

V\$RESTORE\_POINT

```
SQL> CONNECT / AS SYSDBA
SQL> ALTER PLUGGABLE DATABASE pdb1 CLOSE;
SQL> CREATE CLEAN RESTORE POINT start_step1 FOR PLUGGABLE DATABASE pdb1
 GUARANTEE FLASHBACK DATABASE;
SQL> ALTER PLUGGABLE DATABASE pdb1 OPEN;
SQL> @script_patch_step1
SQL> ALTER PLUGGABLE DATABASE pdb1 CLOSE;
```

```
$ rman target /
RMAN> FLASHBACK PLUGGABLE DATABASE pdb1 TO RESTORE POINT start_step1;
RMAN> ALTER PLUGGABLE DATABASE pdb1 OPEN RESETLOGS;
```



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Normal and Guaranteed Restore Points

PDB normal and guaranteed restore points are restore points that pertain to only a specific PDB and are visible within the PDB only.

- A normal PDB restore point is essentially a bookmark for a past point for that PDB.
- A guaranteed PDB restore point guarantees a PDB flashback to that restore point.

PDB flashback uses the Oracle Database 12.1.0.1 mechanism, the PDB point-in-time recovery, to recover the shared undo. RMAN automatically restores the shared undo and certain tablespaces in a clone instance and recovers the data to the same point in time.

## Clean Restore Points

A clean PDB restore point can be created after a PDB is closed with no outstanding transactions and ONLY in shared undo mode. Thus, flashing back a PDB to a clean restore point is faster than other types of flashback because it does not require restoring backups or creating a clone instance. For example, a DBA expecting to roll back an application upgrade should consider creating a clean PDB guaranteed restore point before the application upgrade.

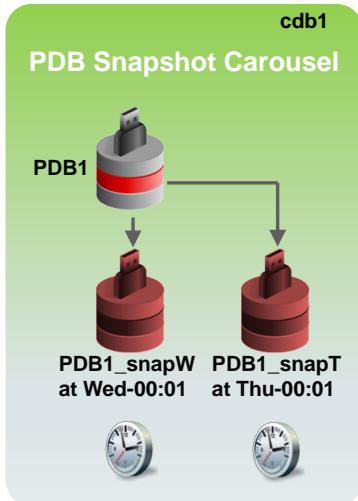
A restore point that is created while being connected to the CDB root without specifying the new FOR PLUGGABLE DATABASE clause is called a CDB restore point.

In case the PDB is the application root of an application container, all restore points are “clean” with local undo, because the availability of local undo means that the effects of active transactions at the time of the restore point can be undone.

**Caution:** PDB guaranteed restore points can potentially result in Oracle running out of space in the FRA. The DBA should remove PDB guaranteed restore points when the PDB flashback operations are completed.

## PDB Snapshot Carousel

A PDB snapshot is a named copy of a PDB at a specific point in time.



- Recovery extended beyond flashback retention period
- Reporting on historical data kept in snapshots
- Storage-efficient snapshot clones taken on periodic basis
- Maximum of eight snapshots for CDB and each PDB

Example:

On Friday, need to recover back to Wednesday.

- Restore `PDB1_snapW`.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

In Oracle Database 18c, a newly created PDB is enabled for PDB snapshots by default, though you can specify whether it is enabled for PDB snapshots or not. A PDB snapshot is an archive file ( .pdb) containing the contents of the copy of the PDB at snapshot creation.

PDB snapshots allow the recovery of PDBs back to the oldest PDB snapshot available for a PDB. This feature extends the recovery beyond the flashback retention period that necessitates database flashback enabled.

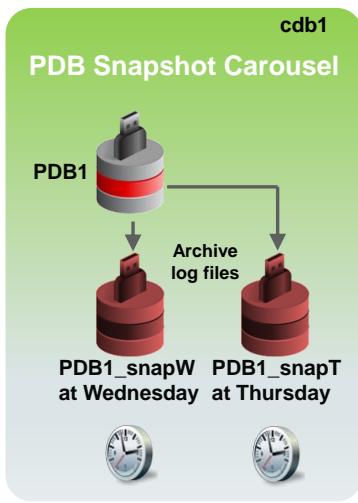
The example in the slide shows a situation where you have to restore `PDB1` back to Wednesday.

A use case of PDB snapshots is reporting on historical data. You might create a snapshot of a `sales` PDB at the end of the financial quarter. You could then create a PDB based on this snapshot so as to generate reports from the historical data.

Every PDB snapshot is associated with a snapshot name and the SCN and timestamp at snapshot creation. The `MAX_PDB_SNAPSHOTS` database property sets the maximum number of PDB snapshots for each PDB. The default and allowed maximum is 8. When the maximum number is reached for a PDB, and an attempt is made to create a new PDB snapshot, the oldest PDB snapshot is purged. If the oldest PDB snapshot cannot be dropped because it is open, an error is raised. You can decrease this limit for a given PDB by issuing an `ALTER DATABASE` statement specifying a max number of snapshots. If you want to drop all PDB snapshots, you can set the limit to 0.

# Creating PDB Snapshot

To create PDB snapshots for a PDB:



1. Enable a PDB for PDB snapshots.

```
SQL> CREATE PLUGGABLE DATABASE pdb1 ...
 SNAPSHOT MODE MANUAL;
```

```
SQL> ALTER PLUGGABLE DATABASE pdb1
 SNAPSHOT MODE EVERY 24 HOURS;
```

2. You can create multiple manual PDB snapshots of a PDB.

```
SQL> ALTER PLUGGABLE DATABASE pdb1
 SNAPSHOT pdb1_first_snap;
SQL> ALTER PLUGGABLE DATABASE pdb1
 SNAPSHOT pdb1_second_snap;
```

3. Disable snapshot creation for a PDB.

```
SQL> ALTER PLUGGABLE DATABASE pdb1 SNAPSHOT MODE NONE;
```

|                                   |
|-----------------------------------|
| DATABASE_PROPERTIES               |
| PROPERTY_NAME = MAX_PDB_SNAPSHOTS |
| PROPERTY_VALUE = 8                |

|                   |
|-------------------|
| DBA_PDB_SNAPSHOTS |
| DBA_PDBS          |
| SNAPSHOT_MODE     |

ORACLE®

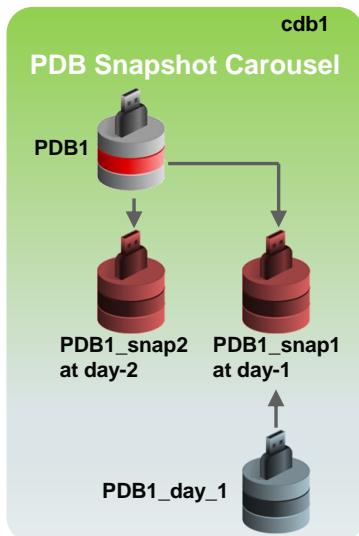
Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

By default, a PDB is enabled for PDB snapshots. There are two ways to define PDBs enabled for PDB snapshot creation:

- **Manually:** The first example in the slide uses the `SNAPSHOT MODE MANUAL` clause of the `CREATE PLUGGABLE DATABASE` or `ALTER PLUGGABLE DATABASE` statement. No clause gives the same result.
- **Automatically after a given interval of time:** The second example in the slide uses the `SNAPSHOT MODE EVERY <snapshot_interval> [MINUTES | HOURS]` clause of the `CREATE PLUGGABLE DATABASE` or `ALTER PLUGGABLE DATABASE` statement. When the amount of time is expressed in minutes, it must be less than 3000. When the amount of time is expressed in hours, it must be less than 2000. In the second example in the slide, the `SNAPSHOT MODE` clause specifies that a PDB snapshot is created automatically every 24 hours.

Every PDB snapshot is associated with a snapshot name and the SCN and timestamp at snapshot creation. You can specify a name for a PDB snapshot. The third and fourth examples in the slide show how to create PDB snapshots manually, even if the PDB is set to have PDB snapshots created automatically. If PDB snapshots are created automatically, the system generates a name.

## Creating PDBs Using PDB Snapshots



After a PDB snapshot is created, you can create a new PDB from it:

```
SQL> CREATE PLUGGABLE DATABASE pdb1_day_1 FROM pdb1
 USING SNAPSHOT <snapshot_name>;
```

```
SQL> CREATE PLUGGABLE DATABASE pdb1_day_2 FROM pdb1
 USING SNAPSHOT AT SCN <snapshot_SCN>;
```



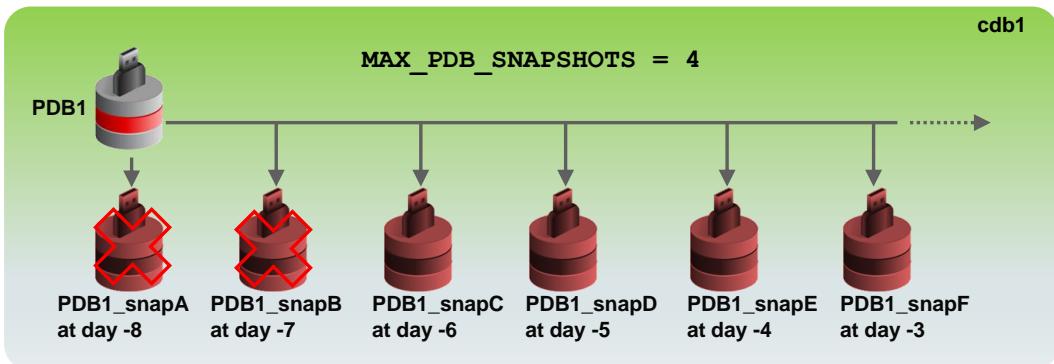
Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

You can create a new PDB from an existing PDB snapshot by using the **USING SNAPSHOT** clause. Provide any of the following:

- The snapshot name
- The snapshot SCN at which the snapshot was created
- The snapshot time at which the snapshot was created

## Dropping PDB Snapshots

- Automatic PDB snapshot deletion when MAX\_PDB\_SNAPSHOTS limit is reached:



- Manual PDB snapshot deletion:

```
SQL> ALTER PLUGGABLE DATABASE pdb1 DROP SNAPSHOT pdb1_first_snap;
```

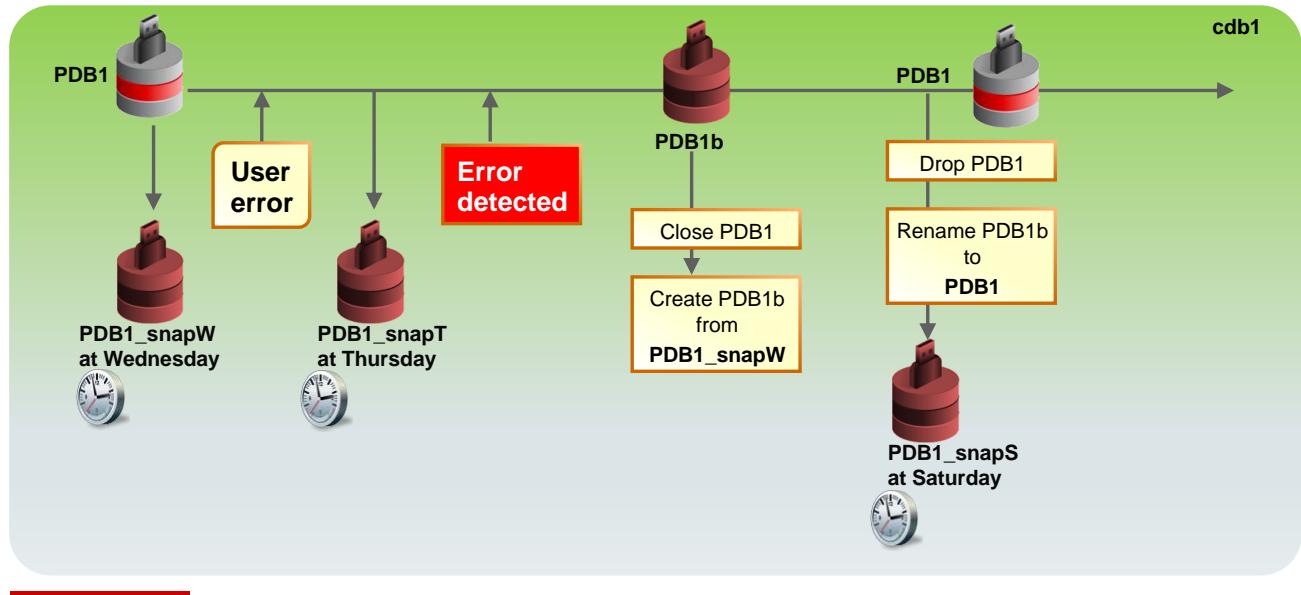
ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

When the carousel reaches eight PDB snapshots or the maximum number of PDB snapshots defined, the oldest PDB snapshot is deleted automatically, whether or not it is in use. There is no need to materialize a PDB snapshot in carousel, because PDB snapshots are all full clone. Be aware that if the SNAPSHOT COPY clause is used with the USING SNAPSHOT clause, the SNAPSHOT COPY clause is simply ignored.

You can manually drop the PDB snapshots by altering the PDB for which the PDB snapshots were created and using the DROP SNAPSHOT clause.

# Flashback PDBs Using PDB Snapshots



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

PDB snapshots enable the recovery of PDBs back to the oldest PDB snapshot available for a PDB.

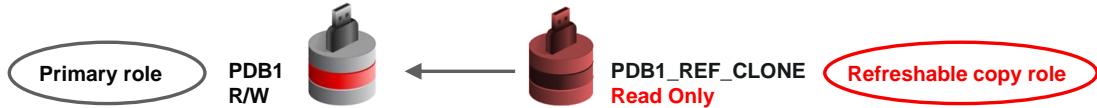
The example in the slide shows a situation where you detect an error that happened between PDB1\_SNAPW and PDB1\_SNAPT creation. To recover the situation, perform the following steps:

1. Close PDB1.
2. Create PDB1b from the PDB1\_SNAPW snapshot created before the user error.
3. Drop PDB1.
4. Rename PDB1b to PDB1.
5. Open PDB1 and create a new snapshot.

# Switching Over a Refreshable Cloned PDB

Switchover at the PDB level:

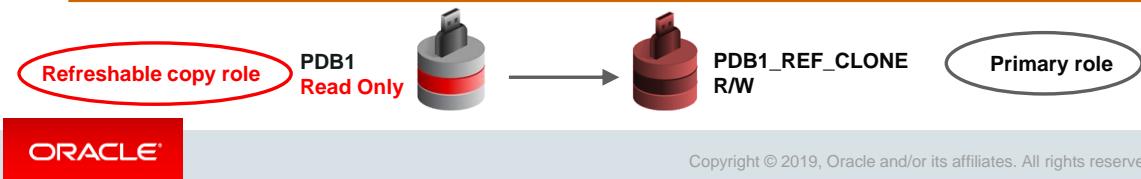
1. A user creates a refreshable clone of a PDB.



2. The roles can be reversed: the refreshable clone can be made the primary PDB.

- The new primary PDB can be opened in read/write mode.
- The primary PDB becomes the refreshable clone.

```
SQL> CONNECT sys@PDB1 AS SYSDBA
SQL> ALTER PLUGGABLE DATABASE REFRESH MODE EVERY 6 HOURS
 FROM pdb1_ref_clone@link_cdb_source_for_clone SWITCHOVER;
```



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.



In Oracle Database 18c, after a user creates a refreshable clone of a PDB, the roles can be reversed. The refreshable clone can be made the primary PDB, which can be opened in read/write mode while the primary PDB becomes the refreshable clone.

The `ALTER PLUGGABLE DATABASE` command with the `SWITCHOVER` clause must be executed from the primary PDB. The refresh mode can be either `MANUAL` or `EVERY <refresh interval> [MINUTES | HOURS]`. `REFRESH MODE NONE` cannot be specified when issuing this statement.

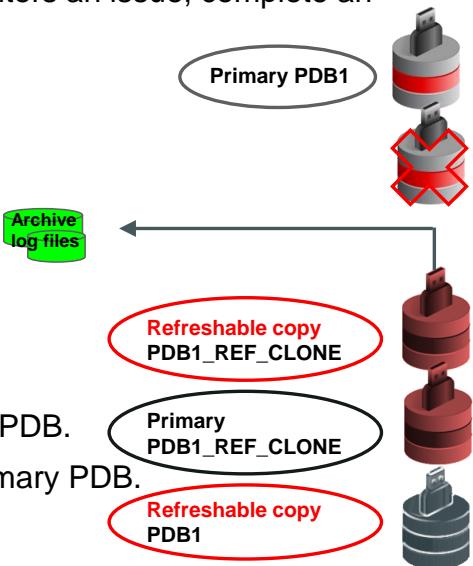
After the switchover operation, the primary PDB becomes the refreshable clone and can only be opened in `READ ONLY` mode. During the operation, the source is quiesced, and any redo generated from the time of the last refresh is applied to the destination to bring it current.

The database link user also has to exist in the primary PDB if the refreshable clone exists in another CDB.

# Unplanned Switchover

When a PDB with an associated refreshable clone encounters an issue, complete an unplanned switchover:

1. Close the primary PDB.
2. Archive the current redo log file. 
3. Drop the primary PDB.
4. Copy the archive redo log files to a new folder.
5. Set the destination for the archive redo log files.
6. Refresh the refreshable clone PDB.
  
7. Disable the refresh mode of the refreshable clone PDB.
8. Open the refreshed PDB that became the new primary PDB.
9. Optionally, create a new refreshable clone.



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

In the example in the previous slide, the roles can be reversed at any time because none of the primary and refreshable cloned PDBs are damaged.

In the example of an unplanned switchover, the primary encounters an issue. Then the way to refresh the refreshable cloned PDB is to first archive the current redo log file and copy the archive log files to a new directory that you define as the `REMOTE_RECOVERY_FILE_DEST`. Then once the PDB is refreshed, you disable the refreshable capability on the former refreshable cloned PDB that becomes the primary PDB and open it. You can finally re-create a new refreshable cloned PDB from the former refreshable cloned PDB. Because the original primary PDB is dropped, you can give the new refreshable cloned PDB the same name of the former primary PDB.

## Summary

In this lesson, you should have learned how to:

- Recover a PDB from essential file damage
- Recover a PDB from nonessential file damage
- Reuse preplugin backups after conversion of a non-CDB to a PDB
- Reuse preplugin backups after plugging/relocating a PDB into another CDB
- Perform CDB flashback
- Perform PDB flashback
- Use clean restore points to complete PDB flashback
- Manage PDB snapshots
- Switch over a refreshable cloned PDB



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Practice 9: Overview

- 9-1: RMAN recovery from SYSTEM PDB datafile loss
- 9-2: RMAN recovery from nonessential PDB datafile loss
- 9-3: PDB PITR
- 9-4: Recovering a plugged non-CDB by using preplugin backups
- 9-5: Recovering a plugged PDB by using preplugin backups
- 9-6: Flashing back an application upgrade by using restore points
- 9-7: Managing and using PDB snapshots
- 9-8: Switching over refreshable cloned PDBs



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

# Performance

ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

# Objectives

After completing this lesson, you should be able to:

- Monitor performance in a CDB and PDBs
- Manage SGA and PGA limits at the PDB level
- Manage AWR snapshots at the CDB and PDB levels
- Run ADDM tasks for CDB and PDB recommendations
- Manage application shared object statistics
- Control query DOP involving the `containers()` construct
- Manage Heat Map and ADO policy declaration in a PDB
- Manage a CDB fleet
- Describe use cases for Consolidated Database Replay



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

For a complete understanding of the CDB and PDB performance and monitoring, refer to the following guide in the Oracle documentation:

- “Monitoring CDBs and PDBs” in *Oracle Multitenant Administrator’s Guide 18c*
- *Oracle Database Performance Tuning Guide, 18c*
- *Oracle Database Testing Guide 18c*

Refer to other sources of information available under Oracle Learning Library:

- *Oracle By Example (OBE): Learning Path: 18c New Features for Multitenant*
  - *managing\_cdb\_fleets*

# Tuning a CDB

Basic rules:

- The PDB appears to applications exactly the same as a non-CDB.
- Some initialization parameters can be set at the PDB level.
- SQL statements are tuned on a per PDB basis.
  - Common objects statistics are gathered in the application root of the common object.
  - Local objects statistics are gathered in the PDB of the local object.
- AWR tools run at the CDB and PDB level: ASH / ADDM
- AWR snapshots can be taken at CDB or PDB level.
- The cursors in the shared pool are identified by PDB.
- Instance-wide information is kept in the CDB root container.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The tuning methodology for a multitenant container database (CDB) with many pluggable databases is basically the same as for a single instance non-CDB. Oracle Corporation has developed a tuning methodology based on years of experience. The methodology presented in this course is also presented in the *Oracle Database Performance Tuning Guide*. This methodology is applied independent of the tools that you use. The ADDM tool follows this methodology automatically.

The rules relating to the behavior of the PDB are fairly simple, but have some complex implications.

The first rule is that in a PDB servicing an application behaves in exactly the same way a non-CDB with the same data would with regard to the SQL and PL/SQL issued by the application. The implications of this statement with regard to tuning include that some initialization parameters can be set at the PDB level, and some of these parameters affect SQL performance.

SQL statements are tuned at the PDB level. The SQL Tuning Advisor runs in a specific PDB. SQL Profiles are applied at the PDB level. These require that cursors are identified by PDB in the shared pool. The objects statistics are collected with the `DBMS_STATS` package in the PDB where the object resides.

The Automatic Workload Repository (AWR) snapshots and Active Session History (ASH) data are kept in the CDB root container when created in the CDB root. This information is used to provide instance-wide tuning advice using the Automatic Database Diagnostic Monitor (ADDM), AWR, and ASH tools.

Oracle Database 12c introduces PDB level snapshots—collection of statistics that matters at the PDB level, created when connected to the PDB. PDB level automatic snapshot is off by default. A PDB has the ability to take its PDB-specific AWR data when it unplugs from a CDB. The AWR data captured at the PDB level is stored in the `SYSAUX` tablespace of the PDB.

# Sizing the CDB

Areas of concern:

- Memory (SGA and PGA)
  - Buffer Cache
  - Shared Pool
  - Program Global Area
- CPU over allocation
- SQL Tuning



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

When considering non-CDBs' consolidation into a single CDB, several questions arise:

- How much memory will I need?
- How much CPU will I need for peak loads?
- Can I tune the applications independently?

An estimate of memory and CPU requirements can be obtained by gathering statistics as in an AWR report over a period of time for each non-CDB instance to estimate and then take a sum of non-CDBs' buffer cache, shared pool, and PGA in the current instances. These estimates should also include period of peak usage and the times of peak usage for each non-CDB to be consolidated.

Reducing the CPU resources allocated to non-CDBs, which are usually based on peak usage, is one of the main reasons for consolidation. But a strategy for handling peak usage should be determined. Do multiple non-CDBs experience peak usage at the same time? Can the loads be time-shifted? Can the applications tolerate being throttled by Resource Manager?

Application tuning is often the most effective tuning. If the application is well tuned on a non-CDB, it behaves well on a PDB. Determining which PDB is handling the poorly behaving SQL is clear. SQL statements flagged at the CDB level have a tag that identifies the PDB of origin.

## Testing the Estimates

Consolidated Database Replay tests:

- Consolidation of servers
- Scale-up
- Peak load capacity



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

After performing the initial sizing estimates for the multitenant container database, you can use Consolidated Database Replay to test the estimates by combining captured workloads from each of the individual non-CDB databases into a scheduled replay in a single CDB. This replay schedule can be modified to adjust peak loads and test scenarios where additional workloads are expected to be added. Details on Consolidated Database Replay are covered in "*Appendix C: Consolidated Database Replay Procedures*".

# Managing SGA for PDBs

## CDB Instance

`SGA_TARGET=10TB`



`SGA_TARGET=10TB`



- `SGA_TARGET` set at the PDB level enforces a hard limit for the PDB's SGA.
- `SGA_TARGET` at the PDB level provides more SGA for other containers.
- `SGA_MIN_SIZE` set for a PDB guarantees SGA space for the PDB.
- Parameters at PDB level are:
  - `DB_CACHE_SIZE`
  - `SHARED_POOL_SIZE`
- (PDB minimums) cannot be > 50 percent of memory.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

In a CDB, there is one SGA allocation for the instance, which is shared by all containers, the CDB root, and all PDBs. Most of the SGA is a cache that favors frequently accessed objects in the buffer cache, the shared pool, and the In-Memory column store.

Active PDBs may dominate the space in the SGA cache. In the example in the slide on the left side, the SUPPORT PDB has an active, memory-intensive workload. It monopolizes the SGA. In the example, the MARKETING PDB needs very little SGA. The performance of the SALES PDB depends on critical buffer cache data and parsed cursors. The SUPPORT PDB is more active and is evicting its data.

In the example in the slide on the right side, SGA and PGA memory management is configured at the PDB level.

- Setting an `SGA_TARGET` for a PDB enforces a hard limit for the PDB's SGA and provides more SGA for the other containers within the CDB. The sum of all PDBs' `SGA_TARGET` does not necessarily need to be less than the instance `SGA_TARGET`, but each PDB `SGA_TARGET` cannot exceed the instance `SGA_TARGET` or `SGA_MAX_SIZE`. `SGA_TARGET` for PDBs works only if the CDB's `SGA_TARGET` is set.
- Setting `DB_CACHE_SIZE` and `SHARED_POOL_SIZE` guarantees minimum sizes for the PDB.
- Setting `SGA_MIN_SIZE` for a PDB guarantees the SGA space for the PDB.

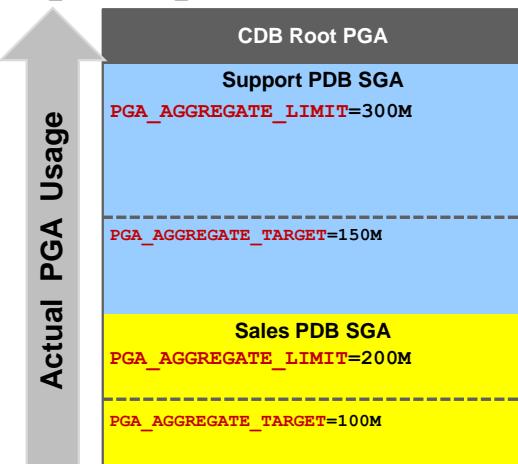
`SGA_TARGET`, `DB_CACHE_SIZE`, and `SHARED_POOL_SIZE` do not work if the CDB's `MEMORY_TARGET` is set.

No more than 50 percent of the memory can be set aside for the PDB minimums: `SGA_MIN_SIZE`, `DB_CACHE_SIZE`, and `SHARED_POOL_SIZE`.

# Managing PGA for PDBs

## CDB Instance

PGA\_AGGREGATE\_LIMIT=1TB  
PGA\_AGGREGATE\_TARGET=500GB



## Instance PGA\_AGGREGATE\_LIMIT

- No more PGA can be allocated.
- Call or session of the largest PGA users is terminated.

## Instance PGA\_AGGREGATE\_TARGET

- All sessions must use TEMP rather than PGA.
- PDB PGA\_AGGREGATE\_LIMIT
- PDB PGA\_AGGREGATE\_TARGET
- These parameters set the same behavior at the PDB level.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

PGA\_AGGREGATE\_TARGET specifies the target aggregate PGA memory that is available to all server processes attached to the instance. To set a hard limit for aggregate PGA memory, use the PGA\_AGGREGATE\_LIMIT parameter. PGA\_AGGREGATE\_LIMIT is set by default to the greater of 2 GB, 200 percent of PGA\_AGGREGATE\_TARGET, and 3 MB times the PROCESSES parameter. It is set below 200 percent of PGA\_AGGREGATE\_TARGET if it is larger than 90 percent of the physical memory size minus the total SGA size, but not below 100 percent of PGA\_AGGREGATE\_TARGET.

The parameters can be configured at the PDB level.

## Monitoring PDB Memory Usage

- Monitor memory usage before and after configuring PDB memory parameters.

```
V$RSRCPDBMETRIC /
V$RSRCPDBMETRIC_HISTORY /
V$RSRC_PDB
 SGA_BYTES
 PGA_BYTES
 SHARED_POOL_BYTES
 BUFFER_CACHE_BYTES
```

- Monitor per PDB history statistics. **V\$RSRC\_PDB\_HISTORY**



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Several views have been added to allow you to monitor the memory and Resource Manager operations at the PDB level.

Statistics are collected every minute into `V$RSRCPDBMETRIC` and `V$RSRCPDBMETRIC_HISTORY`, showing PDB memory and Resource Manager statistics rolled up to the PDB level. This is different from the `V$RSRCMGRMETRIC` view that shows statistics at the consumer group level.

All these views include information that allows you to monitor different aspects of SGA and PGA memory allocations by PDB.

## AWR and ADDM Behavior

- AWR snapshots are created to collect statistics:

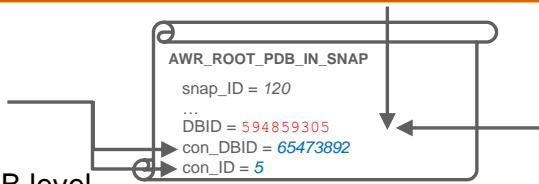
```
SQL> CONNECT / as sysdba
SQL> exec DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT (FLUSH_LEVEL => 'TYPICAL', DBID => 594859305)
```

- Collects statistics at the PDB level
- Collects statistics for each PDB opened

- ADDM runs at the CDB level only:

- Recommendations at the CDB level and PDB level

```
SQL> CONNECT / AS SYSDBA
SQL> var task_name VARCHAR2(60)
SQL> DECLARE
 taskid NUMBER;
 BEGIN
 dbms_advisor.create_task('ADDM',taskid,:task_name);
 dbms_advisor.set_task_parameter(:task_name, 'START_SNAPSHOT', 97);
 dbms_advisor.set_task_parameter(:task_name, 'END_SNAPSHOT', 119);
 dbms_advisor.set_task_parameter(:task_name, 'DB_ID', 594859305);
 dbms_advisor.execute_task(:task_name);
 END;
/
```



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

PDB-level snapshots—collection of statistics that matter at the PDB level are created when the PDB is opened. Even if the snapshot is created while being connected to the CDB root, the snapshot contains statistics that are collected for each opened PDB. A PDB-level report contains information that is specific only to a PDB.

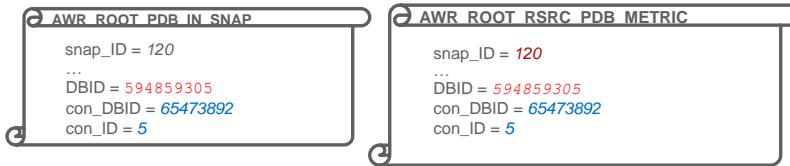
A new view, AWR\_ROOT\_PDB\_IN\_SNAP, captures the list of opened PDBs at the time of the Automatic Workload Repository (AWR) snapshot creation. The CON\_ID column displays the ID of the PDB to which the data pertains. The same information is kept in the DBA\_HIST\_PDB\_IN\_SNAP view.

```
SQL> select * from AWR_ROOT_PDB_IN_SNAP;
 SNAP_ID DBID INSTANCE_NUMBER CON_DBID FLAG CON_ID
----- ----- ----- ----- ----- -----
 120 594859305 1 216956870 0 4
 120 594859305 1 65473892 0 5
 121 594859305 1 216956870 0 4
 121 594859305 1 65473892 0 5
```

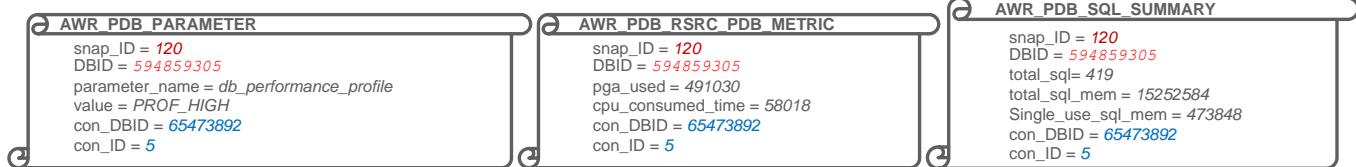
Thus, ADDM tasks can provide recommendations for the CDB and for PDBs as well.

# PDB-Level Snapshot Views

- CDB-level AWR views



- PDB-level AWR views



- List of all PDB-level AWR views

```
SQL> SELECT view_name FROM dba_views WHERE view_name LIKE '%AWR%_PDB%';
```



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The AWR data that is captured at the PDB level is stored in the SYSAUX tablespace of the CDB root.

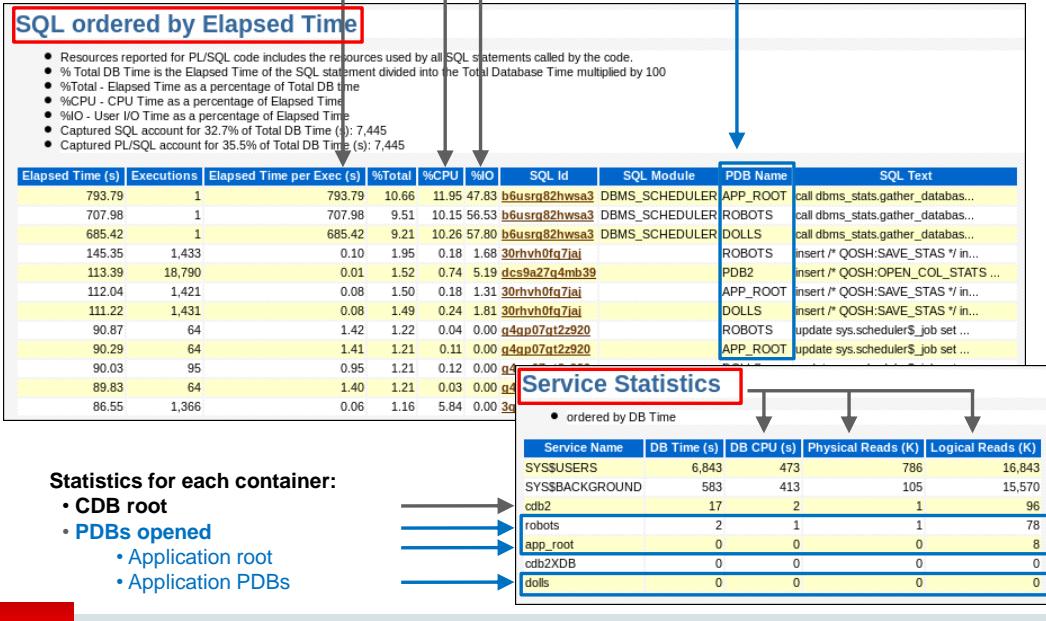
- You can view AWR CDB-level data in AWR\_ROOT\_xxx views.
- You can view AWR PDB-level data for all opened PDBs at the time of snapshot capture in AWR\_PDB\_xxx views.

A PDB has the ability to take its AWR PDB-specific data when it unplugs from a CDB. The target CDB differentiates between the old AWR data that is carried along with the PDB and the new AWR data that is captured after the plugging. The AWR report will not be different from a report that would have been generated in the source CDB.

# AWR Report

## Statistics for each container:

- CDB root
- PDBs opened



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.



AWR reporting is performed at the CDB level.

However, an AWR report contains the following AWR data:

- AWR CDB-level data
- AWR PDB-level data for all opened PDBs at the time of snapshot capture, including application roots and application PDBs

## ADDM Tasks: At the CDB Level Only

The screenshot shows two views of the ADDM interface for a container database (cdb2). The top view shows a message: "A snapshot is being taken which will automatically result in an ADDM run". The bottom view shows recommendations for SQL statements. A red box highlights the "SQL statements executed in PDB and CDB root" section, and a red arrow points from the top message to this section. The Oracle logo is at the bottom left, and copyright information is at the bottom right.

**ADDM Tasks: At the CDB Level Only**

**Processing: Run ADDM Now**

A snapshot is being taken which will automatically result in an ADDM run

**Impact (Active Sessions)** .07  
**Percentage of Finding's Impact (%)** 64.5  
 Period Start Time Nov 14, 2012 6:00:27 AM  
 End Time Nov 14, 2012 7:00:39 AM  
 Filtered No **Filters**

**Recommendations**

**SQL statements executed in PDB and CDB root**

Action: Investigate the ALTER PLUGGABLE DATABASE statement with SQL ID "411k08d873avg" for possible performance improvements.  
 You can supplement the information given here  
 SQL Text  
 SQL ID 411k08d873avg

Rationale: The SQL statement executed in container PDB2 with database ID 3...  
 Rationale: The SQL Tuning Advisor cannot operate on A...  
 Rationale: Database time for this SQL was divided as fo...  
 Rationale: Java execution.  
 Rationale: Waiting for event "opishd" in wait class "Othe...  
 Rationale: The SQL statement executed in the root container.  
 Rationale: The SQL spent 100% of its database time on CPU, I/O and Cluster waits. This part of database time may be improved by the SQL Tuning Advisor.

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

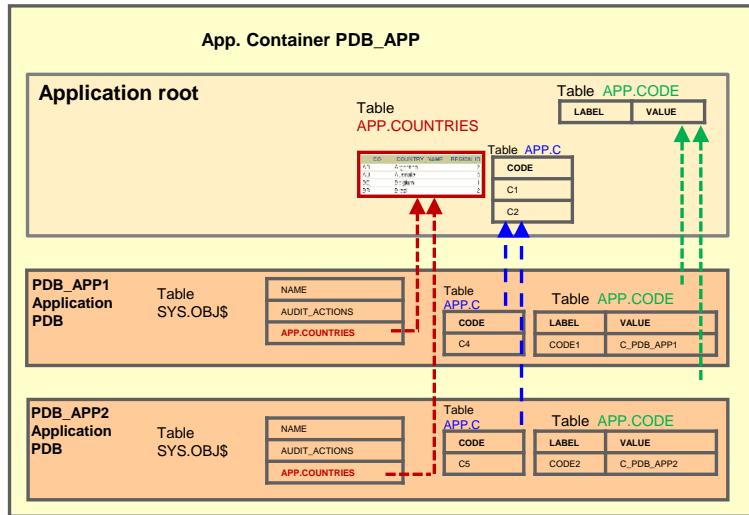
All ADDM runs must be performed in the CDB root, and all ADDM results are stored in the CDB root.

However, ADDM tasks provide recommendations on statements executed in any container—in the CDB root, in application roots, in regular PDBs, and in application PDBs. However, ADDM results cannot be viewed when the current container is a PDB.

ADDM analyzes activity in a PDB within the context of the current analysis target, but ADDM does not analyze only one PDB at a time.

ADDM results related to a PDB are not included if the PDB is unplugged.

## Basic Rules: Statistics for Common Objects



- Statistics for common data-linked objects are gathered in the application root.
- Statistics for common metadata-linked objects are gathered in the application PDB.
- Statistics for common extended data-linked objects are gathered both in the application root and application PDB.
- Statistics for local objects are gathered in the application PDB.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The rules relating to the behavior of a PDB are fairly simple, but have some complex implications.

The first rule is that a PDB that services an application behaves in exactly the same way that a non-CDB with the same data would with regard to the SQL and PL/SQL issued by the application. The implications of a statement with regard to tuning include that some initialization parameters can be set at the PDB level, some of these parameters affecting SQL performance.

SQL statements are tuned at the PDB level. The SQL Tuning Advisor runs in a specific PDB. A container database (CDB) administrator can tune the same query in many PDBs at the same time, whereas a PDB administrator can only tune a single PDB. SQL Profiles are applied at the PDB level. Object statistics are collected with the `DBMS_STATS` package in the PDB where the object resides.

Statistics for user-defined application common objects are gathered in the application root and/or the application PDB depending on the type of common object, whereas those for local objects in an application PDB are gathered in the application PDB.

## Controlling the Degree of Parallelism of Queries

```
SQL> CONNECT toys_app@toys_root
SQL> SELECT sum(revenue), year
 FROM CONTAINERS(sales_data)
 WHERE year = 2014 GROUP BY year;
```



→ Queries using the `CONTAINERS()` construct execute in parallel by default.

→ The query DOP used is 4: sum (app. root + opened application PDBs).

```
SQL> ALTER SESSION SET containers_parallel_degree = 12;
SQL> SELECT sum(revenue), year FROM CONTAINERS(sales_data)
 WHERE year = 2014 GROUP BY year;
```

→ The query DOP that is used for each statement by using the `CONTAINERS()` construct is now 12.

```
SQL> SELECT /*+ CONTAINERS(DEFAULT_PDB_HINT=PARALLEL 8*)/ sum(revenue), year
 FROM CONTAINERS(sales_data)
 WHERE year = 2014 GROUP BY year;
```

→ The query DOP that is used for the statement is now 8.

**ORACLE®**

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The `CONTAINERS_PARALLEL_DEGREE` instance parameter controls the Degree of Parallelism of queries, which involves a `CONTAINERS()` construct.

By default, a query involving a `CONTAINERS()` construct uses a DOP that is equal to the following value:

- In the case of a query in the CDB root, the value is (1 + number of open PDBs).
- In the case of a query in an application root, the value is (1 + number of open application PDBs).

If the value of this parameter is less than 65535, the value is used as the DOP for all queries involving `CONTAINERS()`:

- In the session if the value was set at the session level
- In the instance if the value was set at the system level

## Heat Map and ADO Support

Oracle Database CDBs support ADO and Heat Map statistics.

- ADO policies automatically compress data in objects in PDBs.
- ADO policies automatically move segments in PDBs to other tablespaces in the same PDB when necessary.
- ADO is dependent on Heat Map statistics collection and does not work unless Heat Map is enabled.

HEAT\_MAP = ON



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Automatic Data Optimization (ADO) policies automatically execute actions under predefined conditions:

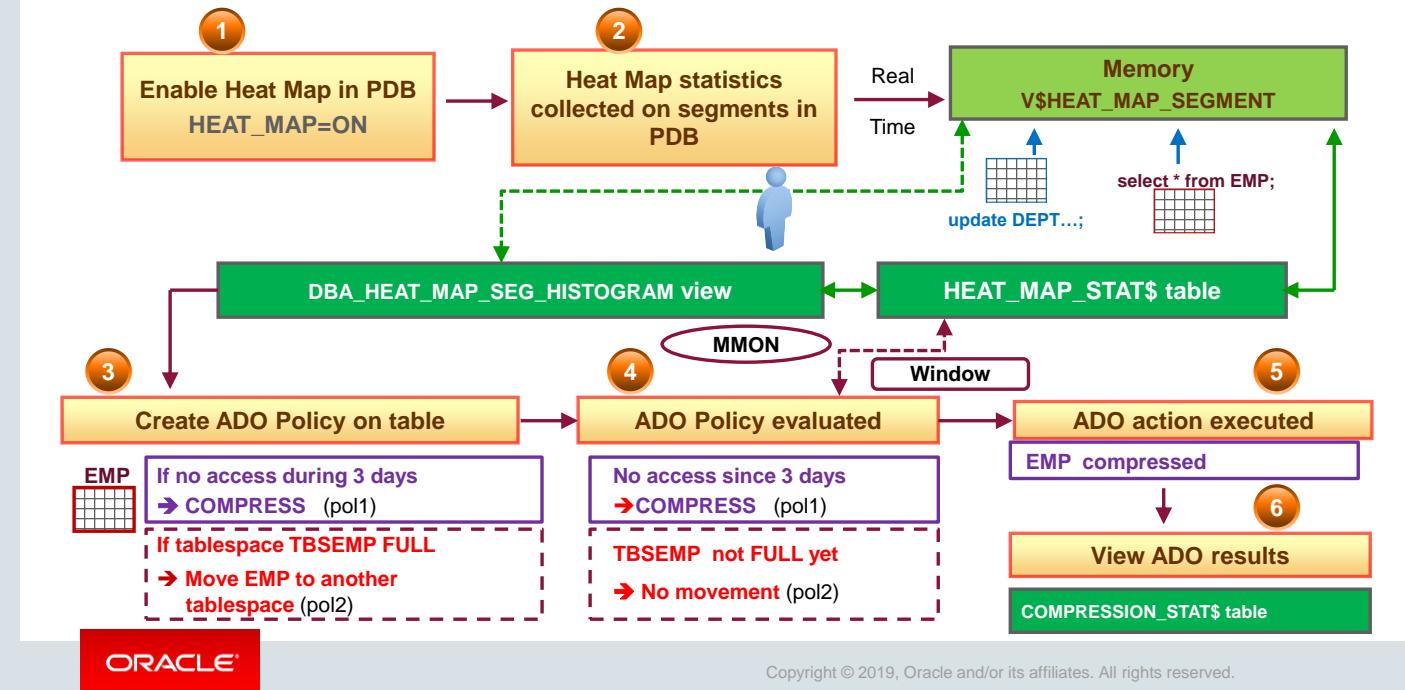
- Compress data only
- Move segments to other storage tiers under space pressure

ADO can execute compression and data movement only if Heat Map is enabled. After being enabled, Heat Map automatically collects statistics to execute ADO actions after the statistics evaluation is completed.

Automatic Data Optimization requires the Advanced Compression option.

Since Oracle Database 12.2, ADO, and Heat Map statistics are supported in CDBs and more precisely on objects in PDBs.

## Managing Heat Map and ADO Policies in PDB



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

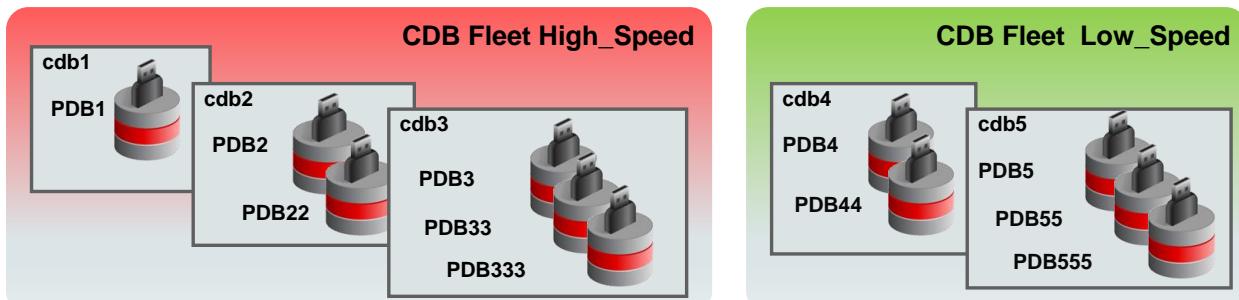
The slide shows how to set up the different steps between Heat Map and ADO to automate the movement of a segment to another tablespace and/or the compression of blocks or a segment depending on certain conditions defined in ADO policies, in a PDB.

1. The first operation for the DBA is to enable Heat Map at the PDB level, tracking activity on blocks and segments. Heat Map activates system-generated statistics collection, such as segment access and row and segment modification.
2. Real-time statistics are collected in memory (`V$HEAT_MAP_SEGMENT` view) and regularly flushed by scheduled `DBMS_SCHEDULER` jobs to the persistent table `HEAT_MAP_STAT$`. Persistent data is visible by using the `DBA_HEAT_MAP_SEG_HISTOGRAM` view.
3. The next step is to create ADO policies in the PDB on segments or groups of segments or as default ADO behavior on tablespaces.
4. The next step is to schedule when ADO policy evaluation must happen if the default scheduling does not match business requirements. ADO policy evaluation relies on Heat Map statistics. MMON evaluates row-level policies periodically and starts jobs to compress whichever blocks qualify. Segment-level policies are evaluated and executed only during the maintenance window.
5. The DBA can then view ADO execution results by using the `DBA_ILM_EVALUATIONDETAILS` and `DBA_ILMRESULTS` views in the PDB.
6. Finally, the DBA can verify if the segment in the PDB is moved and stored on the tablespace that is defined in the ADO policy and/or if blocks or the segment was compressed, by viewing the `COMPRESSION_STAT$` table.

## CDB Fleet

A CDB fleet is a collection of different CDBs that can be managed as one logical CDB:

- To provide the underlying infrastructure for massive scalability and centralized management of many CDBs
- To provision more than the maximum number of PDBs for an application



- To manage appropriate server resources for PDBs, such as CPU, memory, I/O rate, and storage systems



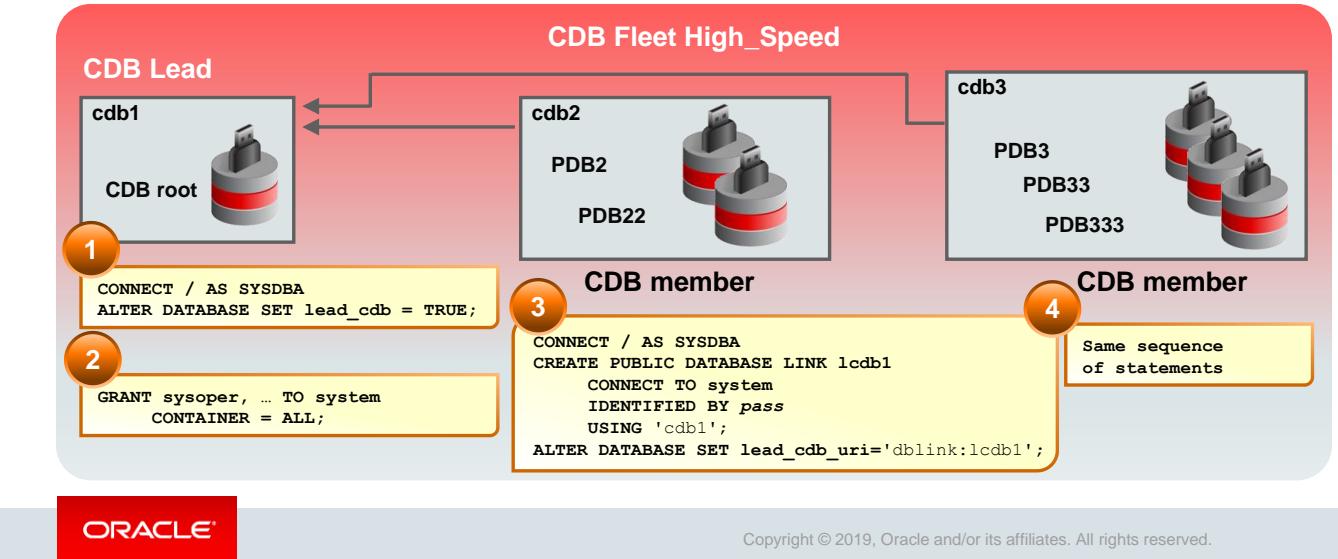
Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Oracle Database 18c introduces the CDB Fleet feature. CDB Fleet aims to provide the underlying infrastructure for massive scalability and centralized management of many CDBs.

- The maximum number of PDBs in a CDB is 4096 PDBs. A CDB fleet can hold more than 4096 PDBs.
- Different PDBs in a single configuration require different types of servers to function optimally. Some PDBs might process a large transaction load, whereas other PDBs are used mainly for monitoring. You want the appropriate server resources for these PDBs, such as CPU, memory, I/O rate, and storage systems.
- Each CDB can use all the usual database features for high availability, scalability, and recovery of the PDBs in the CDB, such as Real Application Clusters (RAC), Data Guard, RMAN, PITR, and Flashback.
- PDB names must be unique across all CDBs in the fleet. PDBs can be created in any CDB in the fleet, but can be opened only in the CDB where they physically exist.

## CDB Lead and CDB Members

- The CDB lead in a fleet is the CDB from which you perform operations across the fleet.
- The CDB members of the fleet link to the CDB lead through a database link.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

A CDB fleet contains a CDB lead and CDB members. PDB information from the individual CDBs is synchronized with the CDB lead.

The CDB lead is, from the CDB root, able to:

- Monitor all PDBs of all CDBs in the fleet.
- Report information and collect diagnostic information from all PDBs of all CDBs in the fleet through a cross-container query.
- Query Oracle-supplied objects from all PDBs of all CDBs in the fleet.

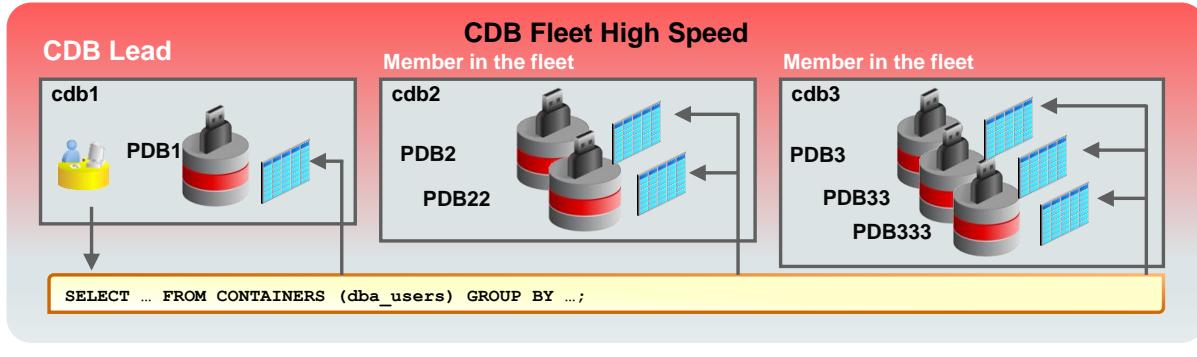
To configure a CDB fleet, define the lead and then the members.

- To define a CDB as the CDB lead in a CDB fleet, from the CDB root, set the `LEAD_CDB` database property to `TRUE`.
- Still in the CDB root of the CDB lead, use a common user and grant appropriate privileges.
- To define other CDBs as members of the CDB fleet:
  - Connect to the CDB root of another CDB.
  - Use a common user identical to the common user used in the lead CDB because you have to create a public database link using a fixed user.
  - Set the `LEAD_CDB_URI` database property to the name of the database link to the CDB lead.

It assumes that the network is configured so that the current CDB can connect to the CDB lead by using the connect descriptor defined in the link.

## Use Cases

- Monitoring and collecting diagnostic information across CDBs from the lead CDB
- Querying Oracle-supplied objects, such as DBA views, in different PDBs across the CDB fleet



- Serving as a central location where you can view information about and the status of all the PDBs across multiple CDBs

ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

- The CDB lead in the CDB fleet can monitor PDBs across the CDBs in the CDB fleet. You can install a monitoring application in one container and use CDB views and GV\$ views to monitor and process diagnostic data for the entire CDB fleet. A cross-container query issued in the lead CDB can automatically execute in all PDBs across the CDB fleet through the Oracle-supplied objects.
- Using Oracle-supplied or even common application schema objects in different PDBs (or application PDBs) across the CDB fleet, you can use the `CONTAINERS` clause or `CONTAINER_MAP` to run queries across all of the PDBs of the multiple CDBs in the fleet. This enables the aggregation of data from PDBs in different CDBs across the fleet. The application can be installed in an application root, and each CDB in the fleet can have an application root clone to enable the common application schema across the CDBs.
- The CDB lead can serve as a central location where you can view information about and the status of all the PDBs across multiple CDBs.

# Consolidated Database Replay Use Cases

Perform realistic load testing for scenarios including:

- Consolidation of servers
- Workload scale-up
- Test peak load capacity



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

In many enterprises, mission-critical business functions rely on databases. The changing technology environment forces businesses to adopt the latest technology to stay competitive. The challenge is to keep the transition from disrupting business.

Several questions need to be answered to be assured of a smooth transition:

Can the new server handle all the projected consolidate databases?

How much additional workload can the new server handle?

What if the current workload increases?

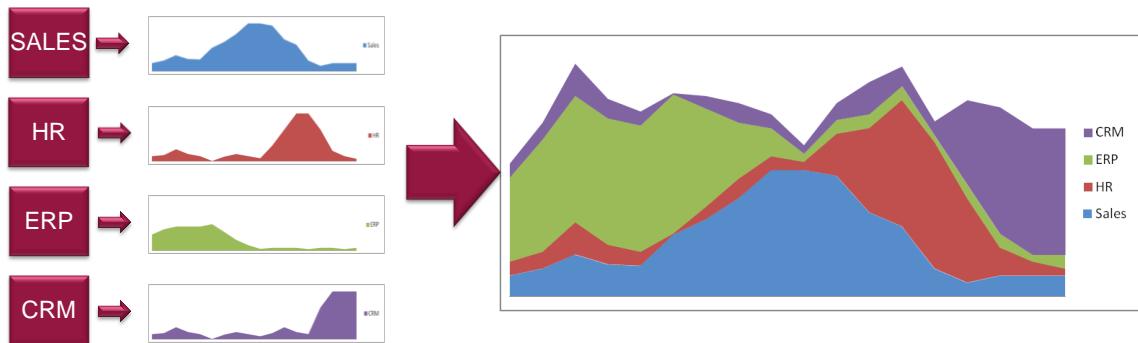
Consolidated Database Replay allows you to test with real production workloads to answer these questions. With Consolidated Database Replay, multiple captured workloads can be replayed concurrently. A captured workload can be divided into subsets covering particular time periods, and multiple captured workloads can be scheduled to run at specified times.

Because a subset of a captured workload is a self-contained captured workload, a subset can be replicated, time-shifted, and replayed with other subsets.

These features allow you to test scenarios such as consolidation by capturing multiple workloads on different servers and replaying on one server, scale-up by replaying multiple subsets concurrently, and test peak load conditions by replaying captures together with varying schedules.

## Use Cases: Source Workloads

**Workload:** Each application has distinct peaks at different times of the workday.



- Each workload captured on different databases and being consolidated is independent of the other.
- Allows multiple workloads captured from different non-CDBs or PDBs to be replayed concurrently in a single CDB.

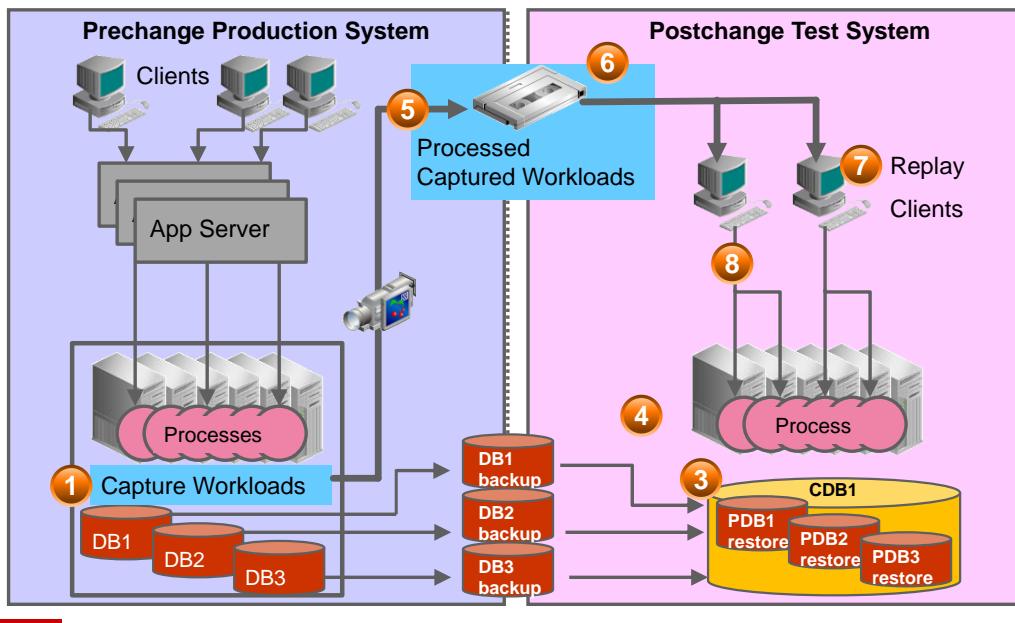


Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The ability to capture workloads from more than one non-CDB and then replay all of the workloads as a single replay against a single CDB is a valuable tool for those wishing to do database consolidation. This assumes that a PDB exists in the CDB for each of the non-CDB capture workloads. It is a simple matter to then remap the connections to the PDB service names in the CDB.

The tool also offers the ability to capture workloads from more than one PDB of different CDBs and then replay all of the workloads as a single replay against a single CDB. This is a valuable tool for those wishing to merge dispersed PDBs into a single CDB.

## The Big Picture



ORACLE®

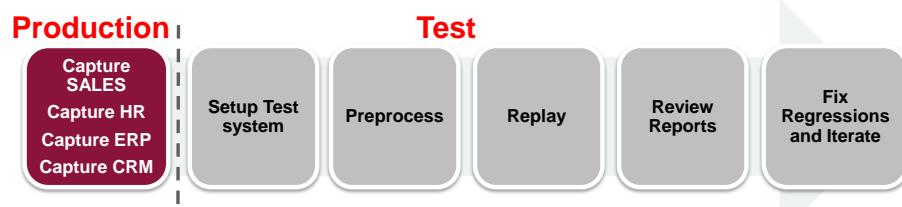
Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The following are the typical steps to perform Database Replay. Note that not all steps are shown in the slide.

1. Capture the workload on a database.
2. Optionally, export the AWR data.
3. Restore the replay database on a test system to match the capture database at the start of the workload capture. The DBA determines the details of this step. It depends on the backup strategy used to return the test system to the starting SCN of the capture recording. With Consolidated Database Replay in Oracle Database 18c, this may mean restoring multiple non-CDBs as PDBs (export / import) or PDBs (cloning) in a single CDB to the start conditions for each capture being replayed.
4. Make changes (such as performing an upgrade or simply testing performance when all non-CDBs are consolidated within a single CDB) to the test system as required.
5. Copy the generated workload files to the replay system.
6. Preprocess the captured workload on the test system or on a system with the same version of database that exists on the test system.
7. Configure the test system for the replay. (Calibrate the `wrc` processes.)
8. Replay the workload on the restored PDBs in the single CDB.

## Step 1

Consolidate multiple non-CDBs or PDBs on a single CDB.



- Capture a typical eight-hour workday in each non-DB or PDB.
- Export performance data from each DB (AWR, SQL Tuning Sets).



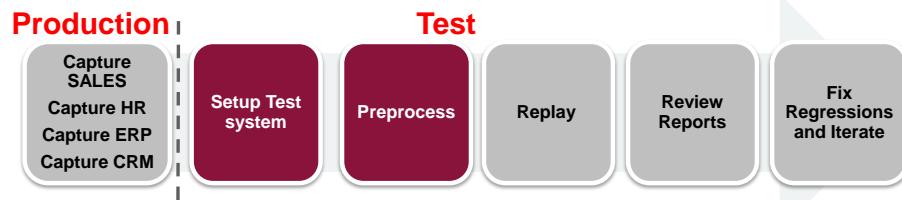
Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The first step consists of capturing the workloads from the multiple non-CDBs that you consider as good candidates for consolidation into a single CDB and/or from PDBs of distinct CDBs that you also consider as good candidates for consolidation into another CDB.

The captured files are generated in separate directories.

You may export the AWR and SQL Tuning Sets to import into the CDB before replay.

## Step 2



- Move all capture files from production to staging system.
- Restore non-CDBs as PDBs or PDBs to the start conditions for each capture being replayed on the single CDB.
- Process each workload (needed once).
- Replay each workload in isolation on new hardware to obtain baseline and verify suitability.
- Flashback/restore.

ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

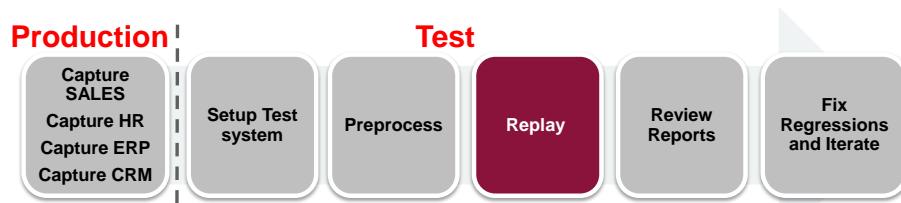
Set a replay directory on the test system to place all capture workloads in the same directory.

Then into the CDB on the test system, import each non-CDB captured (if any) into its dedicated precreated PDB so that the data correspond to what they were at the capture start. Clone each PDB captured (if any) into the test CDB so that the data correspond to what they were at the capture start. Then process all capture workloads once. This preprocessing must be executed once. The same preprocessed files can then be reused for repetitive replays.

Think about replaying each capture separately on the new CDB to verify if on this new server, within a CDB, you've already got new performance results.

Then flashback the whole CDB to the time of the capture start.

## Step 3



- Configure the test system for the replay.
  - Add each workload by using `ADD_CAPTURE()`.
  - Initialize consolidated replay and remap all connections.
    - Optionally, remap schema users.
    - Synchronize on “Object\_ID”.
  - Calibrate the `wrc` processes.
- Replay the workload on the restored database.
  - Two objects with the same name in different PDBs will be different Object IDs and will not collide during replay.

ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Before starting the replay itself, first create a replay schedule that sets a directory containing multiple workload captures as the current replay directory. Add all the capture workloads by capture directory name. The schedule is now saved and associated with the replay directory and can be used for a replay.

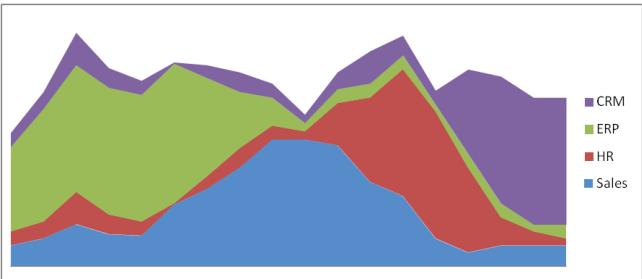
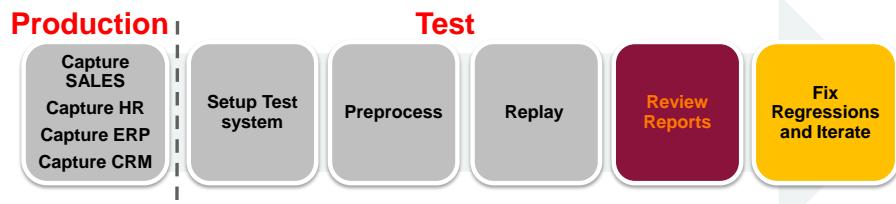
Then initialize the replay. This step reads the information from the workloads in the replay directory and populates the `DBA_WORKLOAD_CONNECTION_MAP` table. Query this table to find the connections that need to be remapped.

Each capture workload in a schedule can remap each connection in the workload to a different connect for replay. This allows each capture to be mapped to a different PDB in a CDB.

Then specify the parameters of synchronization during the replay process of multiple capture workloads. `Object_id` synchronization offers more finer grain synchronization and, therefore, more replay concurrency. The capture process tracks “`object_ids`” used by each user call, and, therefore, the replay process can minimize object collision. Two objects with the same name in different PDBs will be different Object IDs and will not collide during the replay.

Now the replay in the CDB is ready to accept workload replay client (WRC) connections to replay the multiple captures at the same time.

## Step 4



- Verify:
  - Quality of service metrics as perceived by application user per workload
  - Overall capacity
- Only replay can provide definitive answers.

ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Enterprise Manager Cloud Control can provide UI to perform all steps and generate the final report. The report will provide a comparison of the performance before (after the capture) and after (after the replay).

For more detailed information about the set of steps for using Consolidated Replay, refer to Appendix C of the course.

## Summary

In this lesson, you should have learned how to:

- Monitor performance in a CDB and PDBs
- Manage SGA and PGA limits at the PDB level
- Manage AWR snapshots at the CDB and PDB levels
- Run ADDM tasks for CDB and PDB recommendations
- Manage application shared object statistics
- Control query DOP involving the `containers()` construct
- Manage Heat Map and ADO policy declaration in a PDB
- Manage a CDB fleet
- Describe use cases for Consolidated Database Replay



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Practice 10: Overview

- 10-1: Monitoring performance at the CDB and PDB levels
- 10-2: Getting performance ADDM recommendations at CDB and PDB levels
- 10-3: Monitoring and tuning SQL executions at the PDB level
- 10-4: Configuring a CDB fleet with its CDB lead and CDB members



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

# Resources Allocation

The Oracle logo, consisting of the word "ORACLE" in white capital letters inside a red rectangular box.

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

# Objectives

After completing this lesson, you should be able to:

- Manage resource allocation between PDBs and within a PDB
- Control PDB IO rate limit
- Enable parallel statement queuing at PDB level
- Avoid excessive session PGA
- Manage PDB performance profiles



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

For a complete understanding of the PDB resources allocation procedures, refer to the following guide in the Oracle documentation:

- “*Using Oracle Resource Manager for PDBs*” in *Oracle Multitenant Administrator’s Guide 18c*

# Allocating Resources in the CDB

Choose a strategy:

- Allow all PDBs to use all the resources.
  - Gives maximum flexibility for each PDB
  - Allows any PDB to consume all available resources
- Assign a minimum allocation to each PDB.
  - Ensures all PDBs get a specific share of the resources
  - Allows any PDB to consume any unused resources
- Assign a maximum allocation to each PDB.
  - Prevents a PDB from taking more than the maximum value assigned
  - May result in unused capacity



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

A valid concern in a CDB is that one or more PDBs will consume so much resource that other PDBs will suffer performance degradation. The consolidated resource plan allows you to choose a strategy for allocating resources among the various PDBs in a CDB. The resource plan sets limits on the CPU, IO, and parallel processes that may be used by a PDB. The three main strategies are:

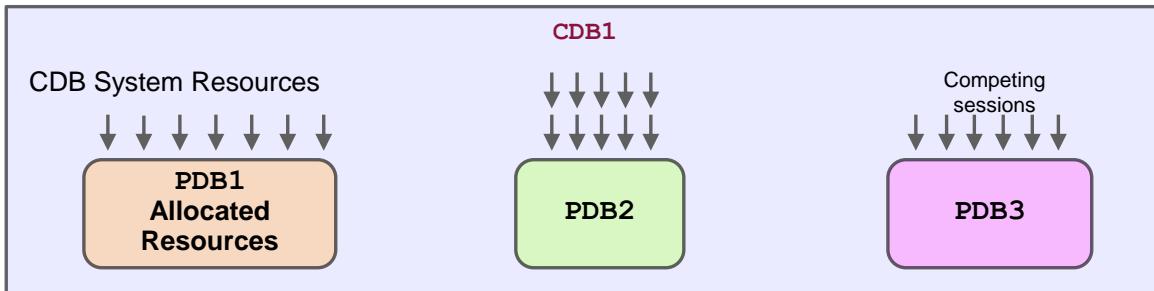
- **No resource plan:** All PDBs have equal access to all the resources. There is no arbitration. If one PDB starts a large parallel query, it could consume all the resources in the CDB instance. This is the default configuration.
- **Minimum allocation:** Set a resource plan to give each PDB a minimum resource share, for example, one share each. In this case, each PDB is guaranteed a share of the resource. That share is equal to  $1/(\text{the number of shares allocated})$ . When all the CDB resources are being used, the resource manager will force PDB processes using more than their minimum share to wait, allowing all PDBs to get a minimum share.
- **Limited allocation:** Using a resource plan, each PDB is assigned shares to set its minimum allocation and a percent limit to set the maximum amount of a resource such as CPU used by the PDB. When a PDB attempts to exceed the maximum value, the resource manager forces the processes to wait.

Details on consolidated resource plan are covered in the next part of the lesson.

# Resource Manager and Pluggable Databases

In a CDB, the Resource Manager manages resources:

- Between PDBs
- Within each PDB



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

In a non-CDB, you can use the Resource Manager to manage multiple workloads that are contending for system and database resources. However, in a CDB, you can have multiple workloads within multiple PDBs competing for system and CDB resources.

In a CDB, the Resource Manager can manage resources at two basic levels:

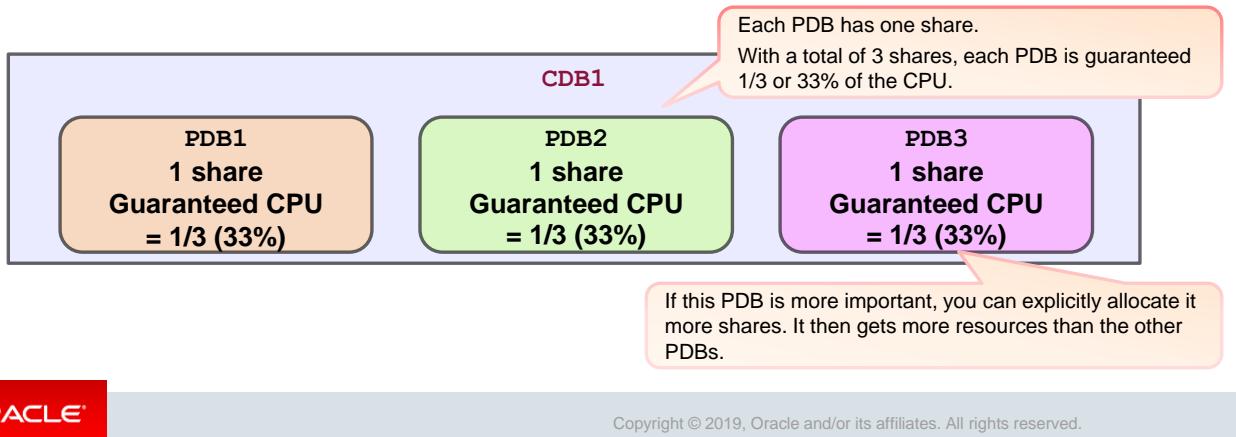
- **CDB level:** The Resource Manager can manage workloads for multiple PDBs that are contending for system and CDB resources. You can specify how resources are allocated to PDBs, and you can limit the resource utilization of specific PDBs.
- **PDB level:** The Resource Manager can manage workloads within each PDB.

The Resource Manager allocates resources in two steps:

1. It allocates a portion of the system's resources to each PDB.
2. In a specific PDB, it allocates a portion of the system resources obtained in Step 1 to each session that is connected to the PDB.

## Managing Resources Between PDBs

- PDBs compete for resources: CPU, Exadata I/O, and parallel servers
  - System shares are used to allocate resources for each PDB.
  - Limits are used to cap resource utilization of each PDB.
- When a new PDB is plugged in, the CDB DBA can specify a default or an explicit allocation.



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

In a CDB with multiple PDBs, some PDBs typically are more important than others. The Resource Manager enables you to prioritize the resource (CPU and Exadata I/O, as well as allocation of parallel execution slaves in the context of parallel statement queuing) usage of specific PDBs. This is done by granting different PDBs different shares of the system resources so that more resources are allocated to the more important PDBs.

In addition, limits can be used to restrain the system resource usage of specific PDBs.

To allocate resources among PDBs, you assign a share value to each PDB. A higher share value results in more resources for a PDB. In this example, each existing PDB is assigned one share. With a total of three shares, each PDB is guaranteed to get at least 33 percent of the system resources. When a PDB is plugged into a CDB and no directive is defined for it, the PDB uses the default directive for PDBs. As the CDB DBA, you can control this default, and you can also create a specific directive for each new PDB. In this case, a new PDB gets one share. With a total of four shares, each PDB is guaranteed to get 25 percent of the system resources.

Depending on the workload, each PDB can get up to 100 percent of the system resources but may actually use much less than the guaranteed level of these resources.

**Note:** Each CDB resource plan gets a default directive added to it. You can change this default directive if its default value is not suitable for your plan.

## CDB Resource Plan Basics: Limits

- Four limits can be defined for each PDB:
  - Utilization limit for CPU, Exadata I/Os, and parallel servers
  - Parallel server limit to override the utilization limit
  - Memory\_min
  - Memory\_limit
- You can change default values.

PDB1

|                                         |             |                                       |
|-----------------------------------------|-------------|---------------------------------------|
| <code>max_utilization_limit</code>      | replaced by | <code>utilization_limit = 30</code>   |
| <code>parallel_target_percentage</code> | replaced by | <code>parallel_server_limit=50</code> |
|                                         |             | <code>memory_min = 50</code>          |
|                                         |             | <code>memory_limit = 80</code>        |

ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

A limit restrains the system resource usage of a specific PDB. You can specify limits for CPU, Exadata I/Os, and parallel execution servers.

To limit the CPU, Exadata I/Os, and parallel servers' (`PARALLEL_SERVER_TARGET` initialization parameter) usage of each PDB, you can create a plan directive using the `UTILIZATION_LIMIT` parameter expressed as a percentage of the system resources the PDB can use. Resource Manager throttles the PDB sessions so that the CPU, Exadata I/Os, and parallel servers' utilization for the PDB does not exceed the utilization limit. In the slide example, PDB1 gets a maximum of 30 percent of the system CPU, Exadata I/Os bandwidth, and available parallel servers for the CDB instance.

For parallel execution servers, you can override the value defined by the `UTILIZATION_LIMIT` by creating a plan directive that uses the `PARALLEL_SERVER_LIMIT` parameter. The `PARALLEL_SERVER_LIMIT` value corresponds to a percentage of `PARALLEL_SERVERS_TARGET`. For example, if the `PARALLEL_SERVERS_TARGET` initialization parameter is set to 200 and the `PARALLEL_SERVER_LIMIT` is set to 50 percent for a PDB, then the maximum number of parallel servers the PDB can use is 100 ( $200 \times .50$ ).

Shares provide a basic mechanism to provide a soft maximum PGA, buffer cache, and shared pool allocation. If the PDB is not using the maximum amount of memory, the memory may be used by other PDBs. The minimum is calculated by the number of shares over the total shares times the total memory.

A more advanced method uses the `MEMORY_LIMIT` parameter and the `MEMORY_MIN` parameter. Both of these parameters are a percentage of PGA limits, buffer cache, and shared pool sizes.

The `MEMORY_MIN` is set to a percentage of memory. If the sum of the minimums is greater than 100 percent, the percentages are scaled to 100 percent. Though the mechanisms vary for PGA, buffer cache, and shared pool, the goal of each is to guarantee each PDB that requests it at least the minimum memory. If a PDB is using more than its minimum memory, it will be preferred for releasing memory. A PDB that is not using its minimum memory will not necessarily retain a minimum allocation, but will be preferred when it requests memory until it reaches its minimum.

### CDB Resource Plan Basics: Limits

`MEMORY_LIMIT` parameter sets a hard maximum amount of memory the PDB is allowed to use. When a PDB is using its maximum, it can only allocate memory that it has released. Other PDBs not using their maximum may allocate memory that is freed from any PDB. This may lead to more free buffer waits for PDBs at their maximum allocation.

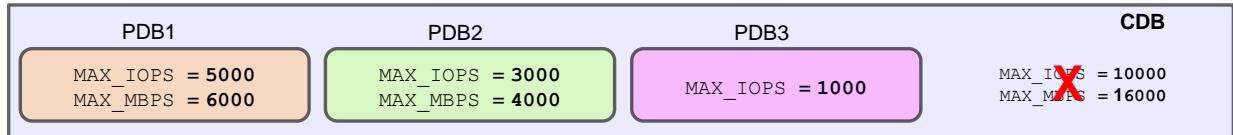
Even when the minimum and maximum memory limits are set, the shares are considered to maintain fairness. The default value for `MEMORY_LIMIT` parameter is 100 percent, and the default for `MEMORY_MIN` parameter is 0 percent.

When you do not explicitly define limits for a PDB, the PDB uses the default limits of the active plan or default plan.

**Note:** You can also change the default directive attribute values for PDBs by using the `UPDATE_CDB_DEFAULT_DIRECTIVE` procedure in the `DBMS_RESOURCE_MANAGER` package.

## PDB IO Rate Limit

- MAX\_IOPS: Number of IOs issued per second
- MAX\_MBPS: MB of IO issued per second
- Set to 0 by default (*no value at the CDB root level*) → no limit
- Stored in the PDB dictionary
- Migrated with the PDB on an unplug or a plug into a new CDB



```
SQL> ALTER SESSION SET CONTAINER = PDB1;
SQL> ALTER SYSTEM SET MAX_IOPS = 5000 SCOPE=SPFILE;
SQL> ALTER SYSTEM SET MAX_MBPS = 6000 SCOPE=SPFILE;
```

**DBA\_HIST\_RSRC\_PDB\_METRIC**  
IO\_REQUESTS, IO\_MEGABYTES,  
IOPS, IOMBPS,  
AVG\_IO\_THROTTLE



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Managing the IO issued per PDB can be important in consolidation environments. The PDB IO rate limits set limits on the IOs that are issued in a CDB on a per PDB basis. Two dynamic parameters are introduced in Oracle Database 12.2 that can be set from within a PDB and can be changed dynamically as required:

- MAX\_IOPS: The parameter enforces a limit on IO rate and is specified as the number of IOs issued per second.
- MAX\_MBPS: The parameter enforces a limit on IO throughput and is specified as the MB of IO issued per second.

One of them or both may be enabled to throttle PDB IOs. DBWR I/Os, control file I/Os, password file I/Os, and other critical I/Os are exempted from the rate limits, but their IOs and MB are accounted for while throttling.

It is enabled only for CDBs, not for non-CDBs. Databases working with Exadata have a more sophisticated I/O Resource Manager implementation on the storage cells. This explains why the feature is enabled only for CDBs and does not work with Exadata.

Several views have been added to allow you to monitor Resource Manager operations at the PDB level.

## CDB Resource Plan: Full Example

| PDB/Directive Name    | Shares | Utilization Limit | Parallel Server Limit | Memory Limit | Memory Minimum |
|-----------------------|--------|-------------------|-----------------------|--------------|----------------|
| (Default Allocation)  | (1)    | (100%)            | (100%)                | (100%)       |                |
| (Autotask Allocation) | (-1)   | (90%)             | (100%)                | (100%)       |                |
| PDB1                  | 1      | 30%               | 50%                   | 50%          | 30%            |
| PDB2                  | 1      | 30%               | 80%                   | 40%          | 10%            |
| PDB3                  | 1      | 30%               | 30%                   | 30%          | 10%            |
| PDB4                  | 2      | 100%              | 100%                  | 100%         | 50%            |

PDB1 is:

- Guaranteed 1/5 (20%) of CPU and Exadata disk bandwidth
- Limited to 30% of CPU and Exadata disk bandwidth
- Limited to 50% of the parallel servers

Limits usage of parallel servers

Sets minimum of memory

Limits usage of memory



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

This example shows you a possible CDB resource plan created in the CDB root container.

The default allocation directive applies to PDBs for which specific directives have not been defined.

Default allocation is one share, and both corresponding limits are set to their default of 100 percent. These are the default values for the default allocation directive.

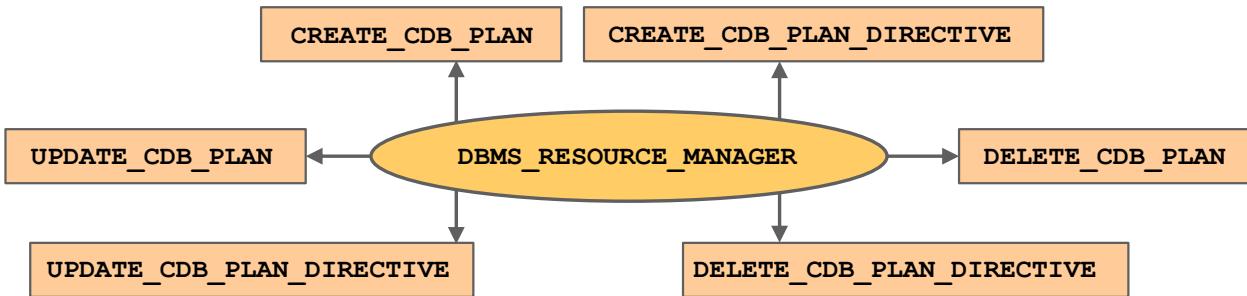
The autotask allocation directive applies to automatic maintenance tasks that are run in the CDB root or in PDBs.

The autotask allocation is -1 share, which means that the automated maintenance tasks get an allocation of 20 percent of the system resources. You should always change this default value. By default, the autotask allocation gets a utilization limit of 90 percent and 100 percent for its parallel server limit.

PDB1 is allocated 1 share, which represents 1/5 or 20 percent of the CPU, Exadata disk bandwidth, and queued parallel queries will be selected 1/5th of the time compared to the other PDBs. In addition, a utilization limit of 30 percent of the resources is set as well as a 50 percent limit of the available parallel servers.

# Maintaining a CDB Resource Plan

```
SQL> CONNECT / AS SYSDBA
SQL> EXEC DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
```



```
SQL> EXEC DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA()
SQL> EXEC DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA()
```



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

1. You can update a CDB resource plan to change its comment using the UPDATE\_CDB\_PLAN procedure.
2. When you create a PDB in a CDB, you can create a CDB resource plan directive for the PDB using the CREATE\_CDB\_PLAN\_DIRECTIVE procedure. The directive specifies how resources are allocated to the new PDB.
3. You can delete the CDB resource plan directive for a PDB using the DELETE\_CDB\_PLAN\_DIRECTIVE procedure. You might delete the directive for a PDB if you unplug or drop the PDB. However, you can retain the directive, and if the PDB is plugged into the CDB in the future, the existing directive applies to the PDB.
4. You can update the CDB resource plan directive for a PDB using the UPDATE\_CDB\_PLAN\_DIRECTIVE procedure. The directive specifies how resources are allocated to the PDB.
5. You can delete a CDB resource plan using the DELETE\_CDB\_PLAN procedure. You might delete a CDB resource plan if the plan is no longer needed. You can enable a different CDB resource plan, or you can disable Resource Manager for the CDB. If you delete an active CDB resource plan, then some directives in PDB resource plans become disabled.

## Managing Resources Within a PDB

- In a non-CDB database, workloads within a database are managed with resource plans.
- In a PDB, workloads are also managed with resource plans, also called PDB resource plans.
- The functionality is similar except for the following differences:

V\$RSRC\_PLAN

| Non-CDB Database          | PDB Database                     |
|---------------------------|----------------------------------|
| Multilevel resource plans | Single-level resource plans only |
| Up to 32 consumer groups  | Up to eight consumer groups      |
| Subplans                  | No subplans                      |



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

A CDB resource plan determines the amount of resources allocated to each PDB. A PDB resource plan determines how the resources allocated to a specific PDB are allocated to consumer groups within that PDB. A PDB resource plan is similar to a resource plan for a non-CDB. Specifically, a PDB resource plan allocates resource among the consumer groups within a PDB.

In a CDB, the following restrictions apply to PDB resource plans:

- PDB resource plans cannot have a multiple-level scheduling policy.
- PDB resource plans can have a maximum of eight consumer groups.
- PDB resource plans cannot have subplans.

**Note:** If you try to create a PDB plan in the CDB root, you get an error.

## Putting It Together

- How do CDB and PDB resource plans work together?
  - Resources allocated to a PDB, based on CDB resource plan
  - Resource allocated to a consumer group based on the PDB resource plan

| CDB Plan |        |                   | PDB3 Plan      |        |                   |
|----------|--------|-------------------|----------------|--------|-------------------|
| PDB      | Shares | Utilization Limit | Consumer Group | Shares | Utilization Limit |
| PDB1     | 1      | 50%               | OLTP           | 3      | 100%              |
| PDB2     | 1      | 50%               | REPORTS        | 1      | 50%               |
| PDB3     | 2      | 100%              | OTHER          | 1      | 50%               |

- What does this mean for PDB3 Reports?
  - Guaranteed  $50\% \times 2/4 \times 20\% = 10\%$  of the resources
  - Limited to  $100\% \times 50\% = 50\%$  of the resources



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Resources are allocated to each PDB based on the CDB resource plan in action. Each PDB receives a certain percentage of the system resources, and Resource Manager distributes that lot to each consumer group based on that PDB's resource plan in action.

An example is shown in the slide. The example just use SHARES and UTILIZATION LIMIT for simplicity.

As you can see, the REPORTS consumer group is guaranteed to receive 10 percent of the total system resources available. This is because PDB3 is allocated two shares out of four in the CDB resource plan, which represents 50 percent of the resources, and out of this 50 percent, the REPORTS consumer group receives one share out of five, which represents 20 percent of the given 50 percent. Similarly, the REPORTS consumer group in PDB3 is limited to 50 percent of the total system resources.

## Considerations

- Setting a PDB resource plan is optional. If not specified, all sessions within the PDB are treated equally.
- A non-CDB is plugged into a CDB with a plan:
  - The plan runs exactly the same on the PDB if:
    - Consumer groups  $\leq$  8
    - There are no subplans
    - All allocations are on level 1
  - The plan is converted.
    - The original plan is stored in the dictionary with the LEGACY status.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

**Note:** V\$RSRC\_PLAN exists in previous releases of the Oracle Database. The resource plan with con\_id=1 is the active CDB resource plan.

If a non-CDB with a plan is plugged into a CDB as a new PDB, the plan acts exactly the same way in the PDB if it does not violate any of the PDBs restrictions:

- The number of consumer groups does not exceed 8.
- There are no subplans.
- All allocations are on level 1.

However, if it violates any of these restrictions, the plan is converted to something equivalent. If the plan is enabled, the converted version is used. If the converted plan does not meet your expectations, you can still use the original plan, stored in the dictionary with the LEGACY status.

## PDB-Level Parallel Statement Queuing

- Possible issues of parallel statements queuing in a PDB:

```
PARALLEL_DEGREE_POLICY = AUTO | ADAPTIVE
```

- Not sufficient parallel servers available
- Parallel statements queued for a long time

- A PDB DBA can make parallel statement queuing work just as it does in a non-CDB.

- Disable parallel statement queuing at CDB level: PARALLEL\_SERVERS\_TARGET = 0.
- Set the PARALLEL\_SERVERS\_TARGET initialization parameter for individual PDBs.
- Kill a runaway SQL operation:

```
SQL> ALTER SYSTEM CANCEL SQL '272,31460';
```

- Dequeue a parallel statement:

```
SQL> EXEC dbms_resource_manager.dequeue_parallel_statement();
```

- Define the action when dequeuing: PQ\_TIMEOUT\_ACTION plan directive



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

- In Oracle Database 12c, a DBA can set the PARALLEL\_SERVERS\_TARGET initialization parameter at the CDB level.
- In Oracle Database 18c, the DBA can set PARALLEL\_SERVERS\_TARGET at the PDB level once parallel statement queuing is disabled at the CDB level. When setting PARALLEL\_SERVERS\_TARGET at the CDB level to 0, parallel statement queues at the PDB level, based on the number of active parallel servers used by that PDB and the PDB's PARALLEL\_SERVERS\_TARGET.
- In Oracle Database 18c, the DBA can use the ALTER SYSTEM CANCEL SQL command to kill a SQL operation in another session that is consuming excessive resources, including parallel servers, either in the CDB root or PDB.
  - Sid, Serial:** The session ID and serial number of the session that runs the SQL statement
  - Inst\_id:** The instance ID if the session is connected to an instance of a RAC database
  - Sql id:** Optionally the SQL ID

The session that consumed excessive resources receives an ORA-01013: user requested cancel of current operation message.
- In Oracle Database 12c, if a parallel statement has been queued for a long time, the DBA can dequeue the statement by using the DBMS\_RESOURCE\_MANAGER.DEQUEUE\_PARALLEL\_STATEMENT procedure. The DBA can also set the PARALLEL\_STMT\_CRITICAL plan directive to BYPASS\_QUEUE. Parallel statements from this consumer group are not queued. The default is FALSE, which means that parallel statements are eligible for queuing.

- In Oracle Database 18c, you can specify the timeout behavior by using the `PQ_TIMEOUT_ACTION` resource manager plan directive. The values are:
  - `ERROR`
  - `RUN`
  - `DOWNGRADE`: In this case, a downgrade percentage can also be specified.

## PDB-Level Parallel Statement Queuing: CPU\_COUNT

- If `CPU_COUNT` is set at the PDB level, the maximum DOP generated by AutoDOP queries are the PDB's `CPU_COUNT`.
- Similarly, the default values for `PARALLEL_MAX_SERVERS` and `PARALLEL_SERVERS_TARGET` are computed based on the PDB's `CPU_COUNT`.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

If `CPU_COUNT` is set at the PDB level, the maximum DOP generated by AutoDOP queries are the PDB's `CPU_COUNT`. Similarly, the default values for `PARALLEL_MAX_SERVERS` and `PARALLEL_SERVERS_TARGET` are computed based on the PDB's `CPU_COUNT`.

## Session PGA Limit

For security purposes:

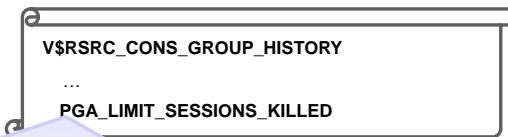
- Avoid excessive usage of PGA memory
- Set the PGA limit that a session can use before it hits an error

**DAY plan**

| Consumer Group | Session_PGA_limit |
|----------------|-------------------|
| OLTP           | 350               |
| REPORTS        | 100               |
| OTHER          | 50                |

**NIGHT plan**

| Consumer Group | Session_PGA_limit |
|----------------|-------------------|
| OLTP           | 50                |
| REPORTS        | 400               |
| OTHER          | 50                |



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

This functionality is important from a security perspective. There are numerous ways to write PL/SQL to use large amounts of PGA memory:

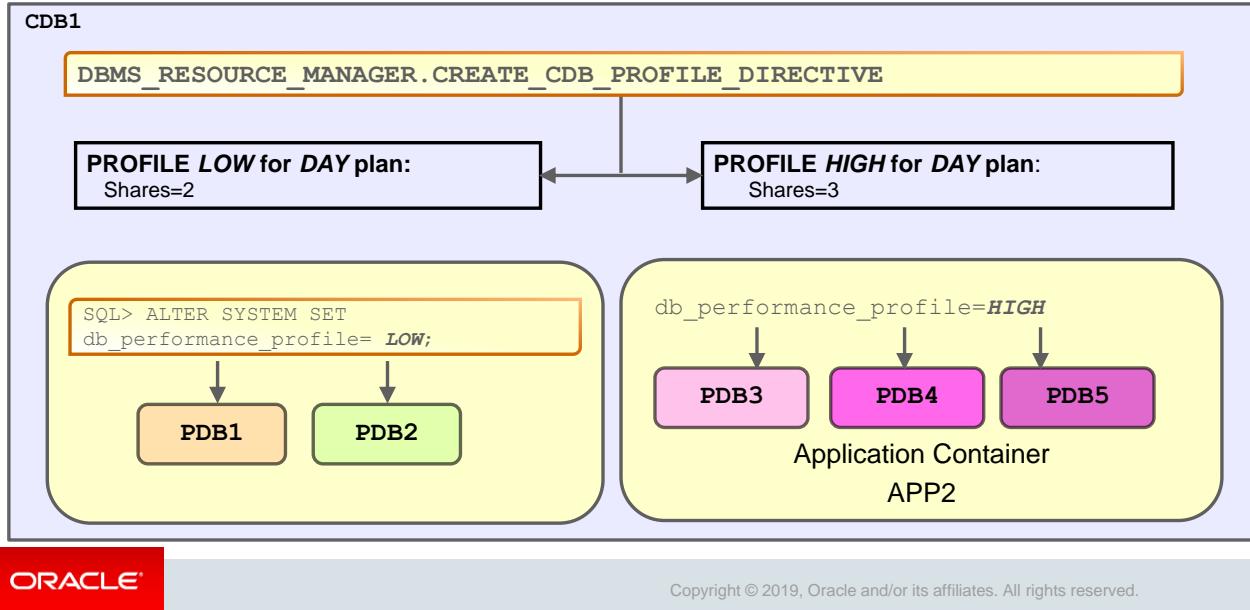
- A user can create a temporary LOB and append to it.
- A user can create a large PL/SQL varray.
- A user can write a PL/SQL statement that does a large recursion using memory at each level.

This is of particular concern in consolidated environments like DB Cloud.

Oracle Database 12.2 allows the DBA to define a per-session PGA limit at the consumer-group level by using the new `SESSION_PGA_LIMIT` argument of the `DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE` procedure. Set the value to the number of Mb of PGA that a session can use before it hits the PGA limit and gets an error. If a session exceeds this limit, it receives an error that aborts the current call. If aborting the current call does not free up enough PGA, the system may kill the session. Resource Manager tracks how many times a consumer group hits the PGA limit and presents this information in `V$RSRC_CONSUMER_GROUP` and `V$RSRC_CONS_GROUP_HISTORY` views.

# Performance Profiles

```
DBA_CDB_RSRC_PLAN_DIRECTIVES
PLUGGABLE_DATABASE = PDB1
PROFILE = LOW
```



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

How do you fairly dispatch CPU and other resources to each application PDB within an application container? Oracle Database 12.2 introduces the new concept of performance profiles in the Resource Manager.

By creating a Resource Manager performance profile, you can set the following resource limits for a plan:

- SHARES
- UTILIZATION\_LIMIT
- PARALLEL\_SERVER\_LIMIT

After creating performance profiles for different plans, set the `DB_PERFORMANCE_PROFILE` instance parameter value to the appropriate RM performance profile name for each application PDB in an application container or for all application PDBs in an application container. Thus each PDB or all application PDBs within an application container with `DB_PERFORMANCE_PROFILE=HIGH` can get three shares.

## Summary

In this lesson, you should have learned how to:

- Manage resource allocation between PDBs and within a PDB
- Control PDB IO rate limit
- Enable parallel statement queuing at PDB level
- Avoid excessive session PGA
- Manage PDB performance profiles



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Practice 11: Overview

- 11-1: Managing PDB performance profiles
- 11-2: Managing resource allocation between PDBs
- 11-3: Avoiding excessive session PGA memory usage in PDBs



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

# Data Movement

The Oracle logo, consisting of the word "ORACLE" in white capital letters inside a red rectangular box.

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

# Objectives

After completing this lesson, you should be able to:

- Export from a non-CDB and import into a PDB
- Export from a PDB and import into a PDB
- Export from a PDB and import into a non-CDB
- Use SQL\*Loader to load data into a PDB

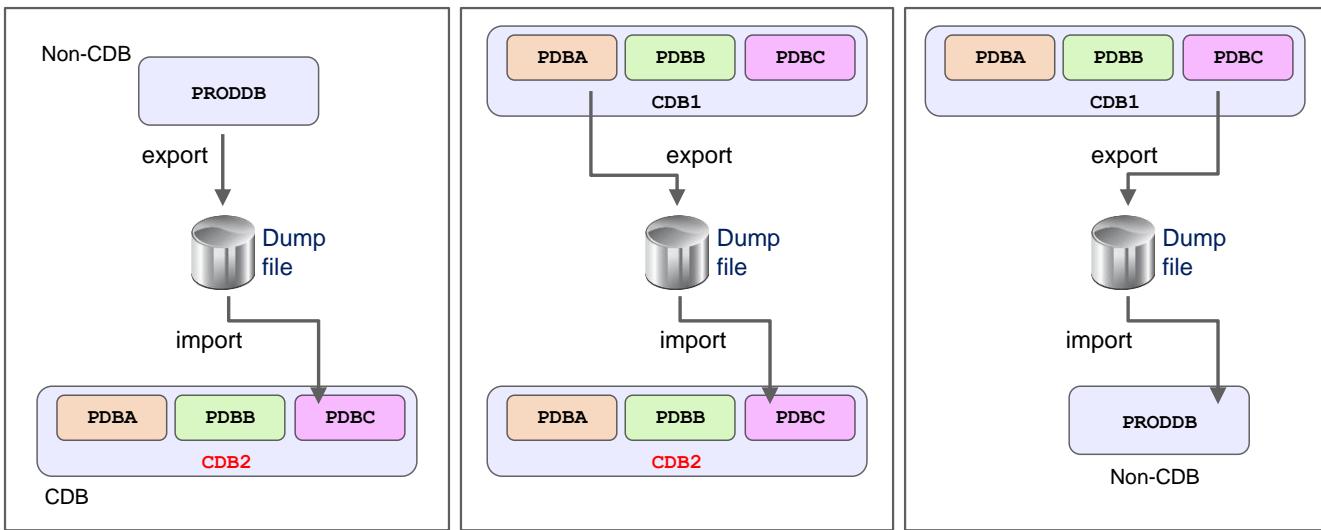


ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

For more information about data transportation between CDBs and PDBs using Data Pump, refer to “*Transporting Data*” in *Oracle Database Administrator’s Guide 18c* and also to *Oracle Database Database Utilities 18c*.

## Using Oracle Data Pump with PDBs



Use the PDB service name to export from or import into a PDB.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

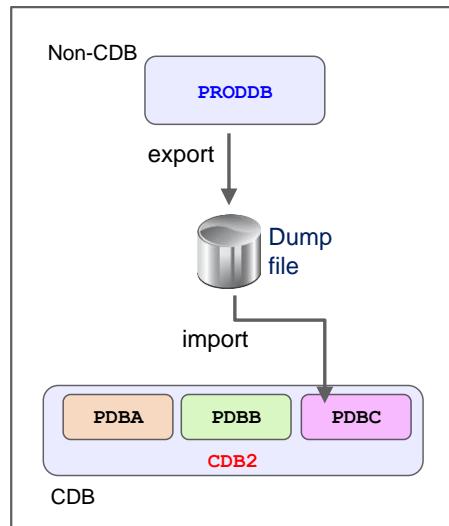
All types of export and import operations are possible between non-CDBs and PDBs. There are no supported CDB-wide Data Pump export or import operations, but only per-PDB Data Pump export or import operations, specifying the service name in the `USERID` clause.

- Export data from a non-CDB to import it into a PDB of a CDB.
- Export data from a PDB to import it into another PDB within the same CDB.
- Export data from a PDB to import it into a PDB of another CDB.
- Export data from a PDB to import it into a non-CDB.

Different types of Data Pump export and import are possible:

- Conventional export and import to export a full database (non-CDB or PDB) to import it into another database (non-CDB or PDB)
- Full transportable export and import to transport a full database (non-CDB or PDB) to import it into another database (non-CDB or PDB)
- Conventional or transportable tablespace export and import to export a full tablespace of a non-CDB or PDB to import it into another tablespace of a non-CDB or PDB
- Schema export and import to export a full schema of a non-CDB or PDB to import it into another tablespace of a non-CDB or PDB
- Table export and import to export a table of a non-CDB or PDB to import it into a non-CDB or PDB

# Exporting from non-CDB and Importing into PDB



1. Export **PROddb** with **FULL** clause:  

```
$ expdp system@PROddb FULL=Y DUMPFILE=proddb.dmp
```
2. If **PDBC** does not exist in **CDB2**, create **PDBC** in **CDB2**:  

```
SQL> CONNECT sys@CDB1
SQL> CREATE PLUGGABLE DATABASE PDBC ...;
```
3. Open **PDBC**.
4. Create a Data Pump directory in **PDBC**.
5. Copy the dumpfile to the Data Pump directory.
6. Create same **PROddb** tablespaces in **PDBC** for new local users' objects.
7. Import into **PDBC** with **FULL** and **REMAP** clauses:  

```
$ impdp system@PDBC FULL=Y DUMPFILE=proddb.dmp
```

ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

To export data from a non-CDB and import it into a PDB of a CDB, use the steps as described in the slide.

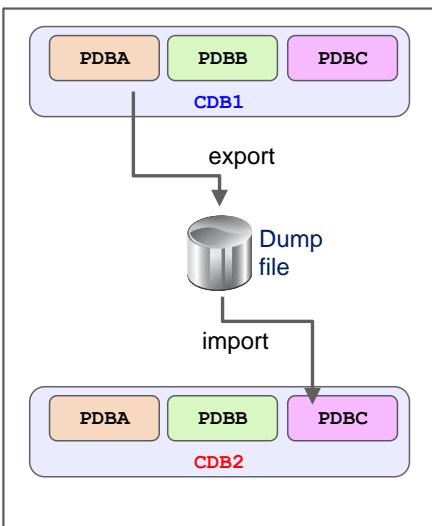
The choice made in the slide is to perform a conventional full database export from the non-CDB and a conventional full database import into the PDB. You can also perform a full transportable or a tablespace or schema or table-level export and import.

The tablespace export and import can be of either types: conventional or transportable.

The users exported from the non-CDB are re-created as local users in the PDB.

The tablespaces for the new local users and objects need to be created in the PDB before the import.

# Exporting and Importing Between PDBs



1. Export **PDBA** from **CDB1** with **FULL** clause:  

```
$ expdp system@PDBA FULL=Y ...
```
2. If **PDBC** does not exist in **CDB2**, create **PDBC** in **CDB2**:  

```
SQL> CONNECT sys@CDB2
SQL> CREATE PLUGGABLE DATABASE PDBC ...;
```
3. Open **PDBC**.
4. Create a Data Pump directory in **PDBC**.
5. Copy the dumpfile to the directory.
6. Create same **PDBA** tablespaces in **PDBC** for new local users objects.
7. Import into **PDBC** of **CDB2** with **FULL** and **REMAP** clauses:  

```
$ impdp system@PDBC FULL=Y REMAP_SCHEMA=c##u:lu...
```

ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

To export data from a PDB and import it into a PDB of the same or another CDB, use the steps as described in the slide.

The choice made in the slide is to perform a full database export from the PDB and a full database import into a PDB of another CDB. You can also perform a full transportable or a tablespace or schema or table-level export and import.

The tablespace export and import can be of either types: conventional or transportable.

The local users exported from the PDB are re-created as local users in the target PDB.

The common users are not re-created because their names prefixed with C## imply that a common user should be created. The statement fails with the following error message:

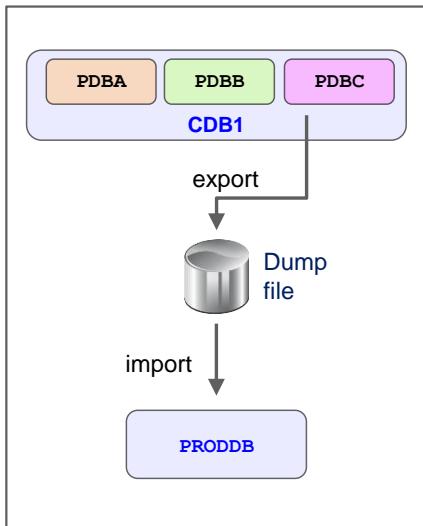
ORA-65094:invalid local user or role name

The only way to have common users re-created as local users is to use the clause

`REMAP_SCHEMA=C##xxx:local_user_name.`

You can also create a common user in the CDB root.

# Exporting from PDB and Importing into non-CDB



1. Export **PDBC** of **CDB1** with **FULL** clause:

```
$ expdp system@PDBC FULL=Y ...
```

2. Import into **PRODDB** with **FULL** and **REMAP** clauses:

```
$ impdp system@PRODDB FULL=Y
REMAP_SCHEMA=c##u:local_u
```



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

To export data from a PDB and import it into a non-CDB, use the steps as described in the slide.

The choice made in the slide is to perform a full database export from the PDB and a full database import into a non-CDB. You can also perform a full transportable or tablespace or schema or table-level export and import.

The tablespace export and import can be of either type: conventional or transportable.

The local users exported from the PDB are re-created as conventional users in the target non-CDB.

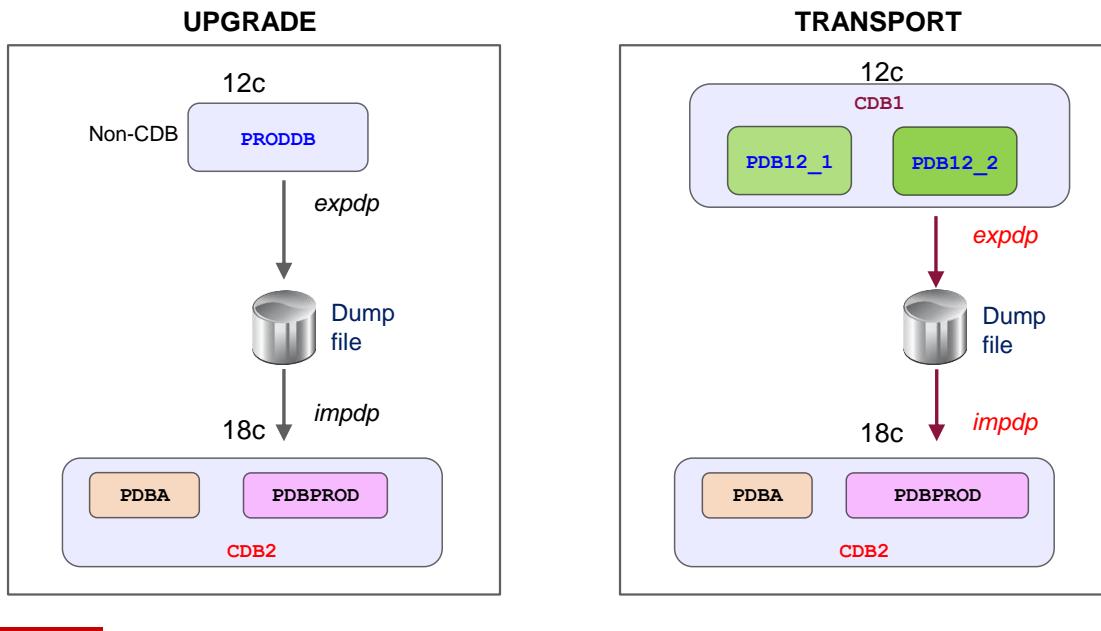
The common users are not re-created because their names prefixed with `C##` imply that a common user should be created. Type of users are not recognized in a non-CDB. The statement fails with the following error message:

```
ORA-65094:invalid local user or role name
```

The only way to have common users re-created as conventional users is to use the clause

`REMAP_SCHEMA=C##xxx:user_name`.

## Full Transportable Export/Import: Overview



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The full transportable export/import feature is useful in several scenarios:

- Upgrading to a new release of Oracle Database: You can use full transportable export/import to upgrade a non-CDB or PDB from 12c to Oracle Database 18c. To do so:
  1. Install Oracle Database 18c, create an empty CDB and a PDB.
  2. Use full transportable export/import to transport the 12c non-CDB or PDB into the Oracle Database 18c PDB.
- Transporting an Oracle Database 12c non-CDB or PDB into a Oracle Database 18c CDB: Transported into a CDB, the transported database becomes a PDB associated with the CDB.

## Full Transportable Export/Import: Usage

A full transportable export exports all objects and data necessary to create a complete copy of the database.

- TRANSPORTABLE=ALWAYS parameter
- FULL parameter

```
$ expdp user_name@pdb FULL=y DUMPFILE=expdat.dmp DIRECTORY=data_pump_dir
TRANSPORTABLE=always
```

A full transportable import imports a dump file only if it has been created using the transportable option during export.

- TRANSPORT\_DATAFILES
- If the NETWORK\_LINK is used, it requires TRANSPORTABLE=ALWAYS parameter.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

### Full Transportable Export

A full transportable export exports all objects and data necessary to create a complete copy of the database. A mix of data movement methods is used:

- Objects residing in transportable tablespaces have only their metadata unloaded into the dump file set. The data itself is moved when you copy the datafiles to the target database. The datafiles that must be copied are listed at the end of the log file for the export operation.
- Objects residing in nontransportable tablespaces (for example, SYSTEM and SYSAUX) have both their metadata and data unloaded into the dump file set, using direct path unload and external tables.

The example shows a full transportable export of a PDB.

Performing a full transportable export has the following restrictions:

- If the database being exported contains either encrypted tablespaces or tables with encrypted columns (either Transparent Data Encryption [TDE] columns or SecureFile LOB columns), then the ENCRYPTION\_PASSWORD parameter must also be supplied.
- The source and target databases must be on platforms with the same endianness if there are encrypted tablespaces in the source database.
- If the source platform and the target platform are of different endianness, you must convert the data being transported so that it is in the format of the target platform. Use either the DBMS\_FILE\_TRANSFER package or the RMAN CONVERT command.
- A full transportable export is not restartable.

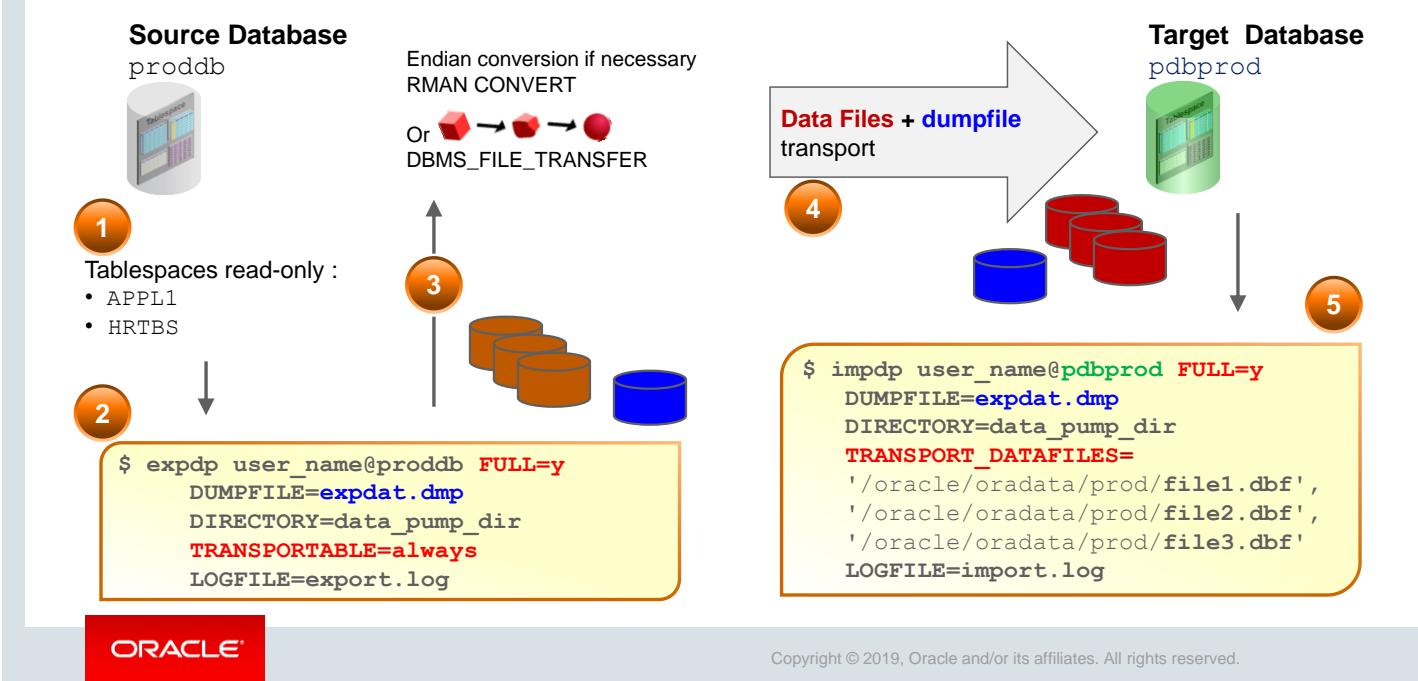
- All objects with storage that are selected for export must have all of their storage segments either entirely within administrative, non-transportable tablespaces (SYSTEM / SYSAUX) or entirely within user-defined, transportable tablespaces. Storage for a single object cannot straddle the two kinds of tablespaces.
- When transporting a database over the network using full transportable export, tables with LONG or LONG RAW columns that reside in administrative tablespaces (such as SYSTEM or SYSAUX) are not supported.
- When transporting a database over the network using full transportable export, auditing cannot be enabled for tables stored in an administrative tablespace (such as SYSTEM and SYSAUX) if the audit trail information itself is stored in a user-defined tablespace.

## Full Transportable Import

Performing a full transportable import has the following requirements:

- If the source platform and the target platform are of different endianness, then you must convert the data being transported so that it is in the format of the target platform. You can use the DBMS\_FILE\_TRANSFER package or the RMAN CONVERT command to convert the data.
- A full transportable import of encrypted tablespaces is not supported in network mode or dump file mode if the source and target platforms do not have the same endianness.
- When transporting a database over the network using full transportable import, tables with LONG or LONG RAW columns that reside in administrative tablespaces (such as SYSTEM or SYSAUX) are not supported.
- When transporting a database over the network using full transportable import, auditing cannot be enabled for tables stored in an administrative tablespace (such as SYSTEM and SYSAUX) if the audit trail information itself is stored in a user-defined tablespace.

## Full Transportable Export/Import: Example



To perform a full transportable operation, follow the steps as below:

1. Before the export, make all of the user-defined tablespaces in the database read-only.
2. Invoke the Oracle Data Pump export utility as a user with the DATAPUMP\_EXP\_FULL\_DATABASE role and specify the full transportable export options: `FULL=Y, TRANSPORTABLE=ALWAYS`. The `LOGFILE` parameter is important because it will contain the list of datafiles that need to be transported for the import operation.
3. Before the import, transport the dump file.
4. Transport the datafiles that you may have converted. If you are transporting the database to a platform different from the source platform, then determine if cross-platform database transport is supported for both the source and target platforms. If both platforms have the same endianness, no conversion is necessary. Otherwise you must do a conversion of each tablespace in the database either at the source or target database using either `DBMS_FILE_TRANSFER` or `RMAN CONVERT` command.
5. Invoke the Oracle Data Pump import utility as a user with the DATAPUMP\_IMP\_FULL\_DATABASE role and specify the full transportable import options: `FULL=Y, TRANSPORT_DATAFILES`.
6. Make the source tablespaces read-write. You can perform this operation before step 5.

## Transporting a Database Over the Network: Example

Transport a database over the network: perform an import using the NETWORK\_LINK parameter.

1. Create a database link in the target to the source database.
2. Make the user-defined tablespaces in the source database read-only.
3. Transport the datafiles for all of the user-defined tablespaces from the source to the target location.
4. Perform conversion of the datafiles if necessary.
5. Import in the target database.

```
$ impdp username@dbname full=Y network_link = sourcedb
 transportable = always
 transport_datafiles = '/oracle/oradata/prod/sales01.dbf',
 '/oracle/oradata/prod/cust01.dbf'
 logfile=import.log
```

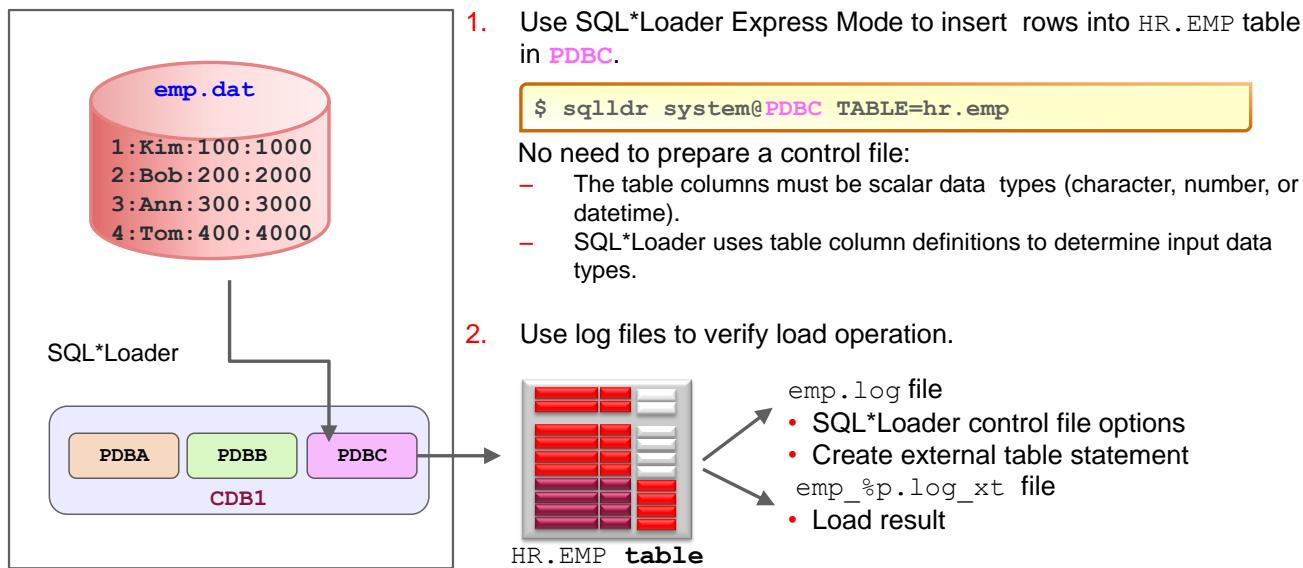


Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

To transport a database over the network, use import with the NETWORK\_LINK parameter. The import is performed using a database link, and there is no dump file involved.

- If the source or target database is a PDB, use the PDB service name in the USERID clause.
- The Oracle Data Pump network import copies the metadata for objects contained within the user-defined tablespaces and both the metadata and data for user-defined objects contained within the administrative tablespaces, such as SYSTEM and SYSAUX.
- When the import is complete, the user-defined tablespaces are in read-write mode.
- Make the user-defined tablespaces read-write again at the source database.

# Using SQL\*Loader with PDBs



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

To load records from a text file into a PDB, if you activate SQL\*Loader Express Mode, specifying only the username and the TABLE parameter, it uses default settings for a number of other parameters. You can override most of the defaults by specifying additional parameters on the command line.

SQL\*Loader express mode generates two files. The names of the log files come from the name of the table (by default).

- A log file that includes:
  - The control file output
  - A SQL script for creating the external table and performing the load using a SQL INSERT AS SELECT statement
  - Neither the control file nor the SQL script is used by SQL\*Loader express mode. They are made available to you in case you want to use them as a starting point to perform operations using regular SQL\*Loader or stand-alone external tables.
- A log file is similar to a SQL\*Loader log file that describes the result of the operation. The "%p" represents the process ID of the SQL\*Loader process.

## Summary

In this lesson, you should have learned how to:

- Export from a non-CDB and import into a PDB
- Export from a PDB and import into a PDB
- Export from a PDB and import into a non-CDB
- Use SQL\*Loader to load data into a PDB



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Practice 12: Overview

- 12-1: Performing a full transportable export/import from a 12c non-CDB into an 18c PDB
- 12-2: Performing a full transportable export/import from a 12c PDB into an 18c PDB



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

# Upgrade Methods

The ORACLE logo, featuring the word "ORACLE" in white capital letters inside a red rectangular box.

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

## Objectives

After completing this lesson, you should be able to:

- Upgrade CDBs from 12c to 18c
- Upgrade regular PDBs from 12c to 18c
- Plug in a remote PDB into a target CDB by using RMAN



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

For a complete understanding of the CDB and PDBs upgrade procedures, refer to the Oracle documentation:

- *Oracle Database Database Upgrade Guide 18c*

# Upgrading CDB and PDBs to 12c: Methods

- Data Pump Export / Import
  - Can provide better performance depending on data volume, metadata volume
  - Ensures support for new data types
- Database Upgrade Assistant (DBUA)
  - Interactively steps you through the upgrade process
  - Automatically fixes some configuration settings
  - Provides a list of items to fix manually
  - Upgrades the CDB, including all PDBs or a defined list of PDBs
- Manual upgrade
  - Provides finer control over the upgrade process
  - Provides a list of items to fix manually
  - Upgrades the CDB, including all PDBs or a defined list of PDBs



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

There are different methods to upgrade a CDB from Oracle Database 12c to Oracle Database 18c.

- Oracle Data Pump Export and Import
- Database Upgrade Assistant (DBUA)
- Manual upgrade by using scripts

# Upgrading a CDB Including PDBs from 12c to 18c

1. Install the 18c Oracle Database software.
2. Execute the Pre-Upgrade Information Tool in the 12c CDB.

```
$ cd /u01/app/oracle/product/18.1.0/dbhome_1/rdbms/admin
$ $ORACLE_HOME/jdk/bin/java -jar preupgrade.jar
```

3. Back up the CDB.
4. Execute the `preupgrade_fixups.sql` script on the 12c CDB.

```
$ORACLE_HOME/perl/bin/perl -I$ORACLE_HOME/perl/lib -I$ORACLE_HOME/rdbms/admin
$ORACLE_HOME/rdbms/admin/catcon.pl -l /u01/app/oracle/cfgtoollogs/cdb12/preupgrade/ -b
preup_cdb12 /u01/app/oracle/cfgtoollogs/cdb12/preupgrade/preupgrade_fixups.sql
```

5. Shut down the instance.
6. Copy the 12c instance spfile to the 18c `$ORACLE_HOME/dbs` directory.
7. Adjust the parameter file with the Oracle Database 18c parameters.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

To upgrade the CDB and PDBs from Oracle Database 12c to Oracle Database 18c, use the DBUA utility or proceed with manual steps as described in the slide.

In step 2, when running `preupgrade.jar` in a CDB, make sure that all the PDBs are opened. The fixup scripts and log files are generated in the pre-upgrade directory of the source database, in `$ORACLE_BASE/cfgtoollogs/SID/preupgrade`.

In step 4, use the `catcon.pl` script to execute the pre-upgrade SQL scripts in the CDB root and in specified PDBs in the correct order. The command in the slide executes the `preupgrade_fixups.sql` script on all containers in the CDB.

You can define a list of PDBs to upgrade either by using an inclusion list with the `-c` parameter or an exclusion list with the `-C` parameter.

The output of the progressing operation is displayed by default on `TERMINAL`. The output can be a `FILE` or a directory (`DIR`). Finally the type of the output is either `TEXT` by default or `XML`. It generates log files that you can view to confirm that the SQL script or SQL statement did not generate unexpected errors. It also starts multiple processes and assigns new scripts to them as they finish running scripts previously assigned to them.

In step 7, remove unsupported initialization parameters, adjust deprecated initialization parameters, and add new ones. Make sure that all path names in the parameter file are fully specified.

## Upgrading CDB Including PDBs from 12c to 18c

- Start the CDB and all PDBs in UPGRADE mode in the 18c environment.

```
SQL> STARTUP UPGRADE
SQL> ALTER PLUGGABLE DATABASE ALL OPEN UPGRADE;
```

- Execute the upgrade script on the CDB root and all PDBs.

```
$ cd $ORACLE_HOME/rdbms/admin
$./catctl.pl [-c 'PDB1 PDB2'] [-l /tmp] catupgrd.sql
```

- Open the CDB and upgraded PDBs in normal mode.

- Execute the `postupgrade_fixups.sql` script.

```
$ cd /u01/app/oracle/product/18.1.0/dbhome_1/rdbms/admin
$ $ORACLE_HOME/perl/bin/perl catcon.pl -c PDB1 -b postupgrade
$ORACLE_BASE/cfgtoollogs/cdb12/preupgrade/postupgrade_fixups.sql
```

- Shut the instance down to update the `/etc/oratab` file and create the password file.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

In step 9, DBUA or `catctl.pl` automatically runs the Parallel Upgrade Utility.

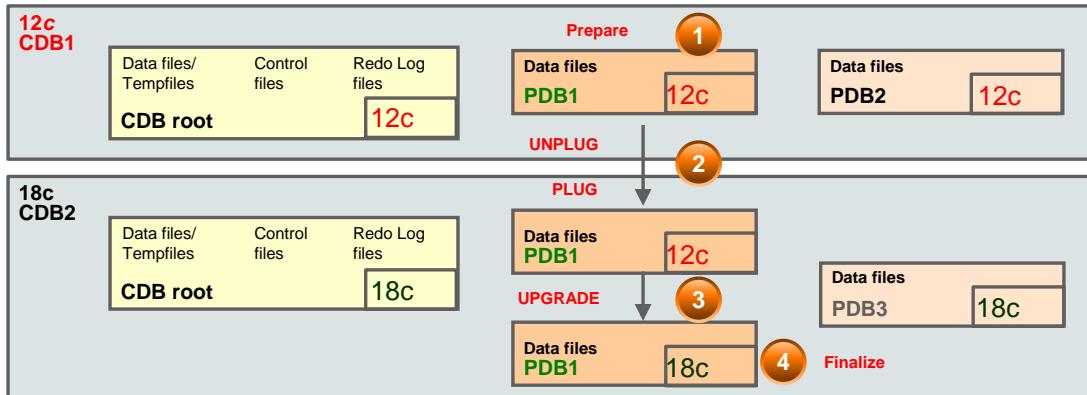
To upgrade a subset of PDBs within a CDB, you can specify either an inclusion list with the `-c` parameter or an exclusion list with the `-C` parameter. Use the `-l` parameter to specify the directory to use for the spool log files.

In step 11, use the `catcon.pl` script to execute the `postupgrade_fixups.sql` script.

**Note:** A prioritized list is used in any upgrade from 11.2.0.3.0, 11.2.0.4.0, 12.1.0.1.0, or 12.1.0.2.0, 12.2.0.1.0 to 18.1. Default priorities are set to containers, with the CDB root being always upgraded first. The DBA can alter the priority so that PDBs are upgraded according to the configured priority order.

In step 12, after shutting the instance down, update the `/etc/oratab` file to set the 18c `ORACLE_HOME` for the upgraded CDB and create the new password file, the 18c `ORACLE_HOME/dbs`.

## Upgrading a Single Regular PDB from 12c to 18c



1. Execute `preupgrade.jar` and then the generated `preupgrade_fixups.sql` script in the 12c PDB.
2. Unplug the PDB from the 12c CDB and plug the PDB into the 18c CDB.
3. Open the PDB in `UPGRADE` mode and upgrade the PDB.
4. Finalize by executing the `postupgrade_fixups.sql` script.

**ORACLE®**

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

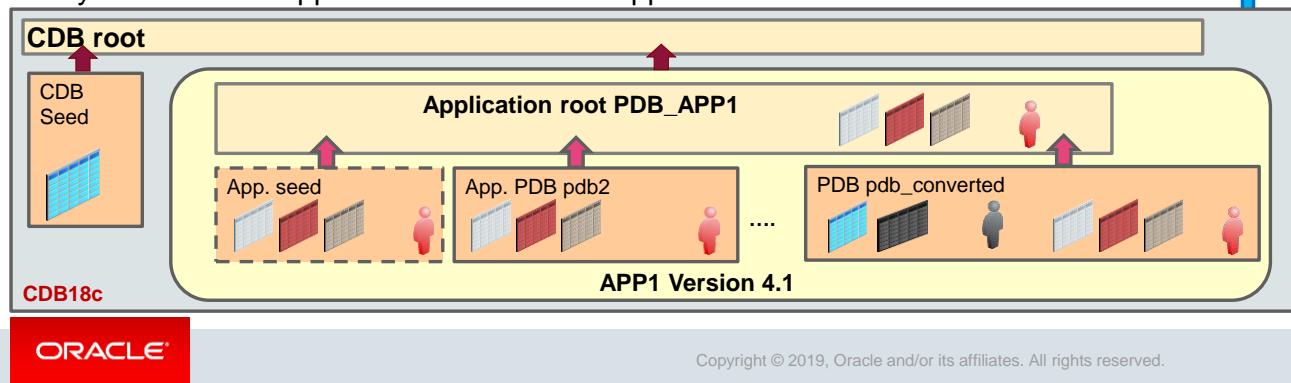
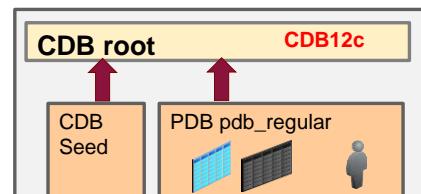
You can also upgrade a specific PDB without upgrading the whole CDB:

1. Prepare the PDB to be unplugged and upgraded in another CDB by executing the same jar file as in the previous slide but only on the PDB. The fixup scripts and log files are generated in the pre-upgrade directory of the source database, in the `$ORACLE_HOME/cfgtoollogs/SID/preupgrade` directory. Execute the `preupgrade_fixups.sql` script in the PDB.
  2. Close the PDB from the 12c CDB to unplug it. Then plug it into the target 18c CDB.
  3. Open the PDB in the target 18c CDB in upgrade mode and upgrade it to 18c.
- ```
$ cd $ORACLE_HOME/rdbms/admin
$ ./catctl.pl -c 'PDB12' -l /tmp/upgrade catupgrd.sql
```
4. Finally close and open the PDB in normal mode. Use the `catcon.pl` script to execute the `postupgrade_fixups.sql` script.

In case the PDB is migrated to an application root, shared objects can be marked as metadata-linked or data-linked by using the `DBMS_PDB.SET_METADATA_LINKED` or `DBMS_PDB.SET_DATA_LINKED` or `DBMS_PDB.SET_EXT_DATA_LINKED` procedures. The user responsible for the migration can be granted `EXECUTE` on the new `DBMS_PDB_ALTER_SHARING` package rather than on the `DBMS_PDB` package to avoid enabling the user to execute other `DBMS_PDB` procedures.

Converting and Upgrading Regular PDBs to Application PDBs

1. Unplug the 12c regular PDB.
2. Plug 12c PDB as a regular PDB in 18c CDB.
3. Upgrade the application PDB to 18c.
4. Unplug the regular PDB and plug it into an application root.
5. Execute the `pdb_to_appdb.sql` script in the application PDB.
6. Synchronize the application PDB with the application root.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Applications that are already installed in an Oracle Database 12c PDB can also take advantage of applications containers. Perform the steps listed in the slide.

Practice 13: Overview

- 13-1: Upgrading and converting a 12c regular PDB to a 18c application PDB

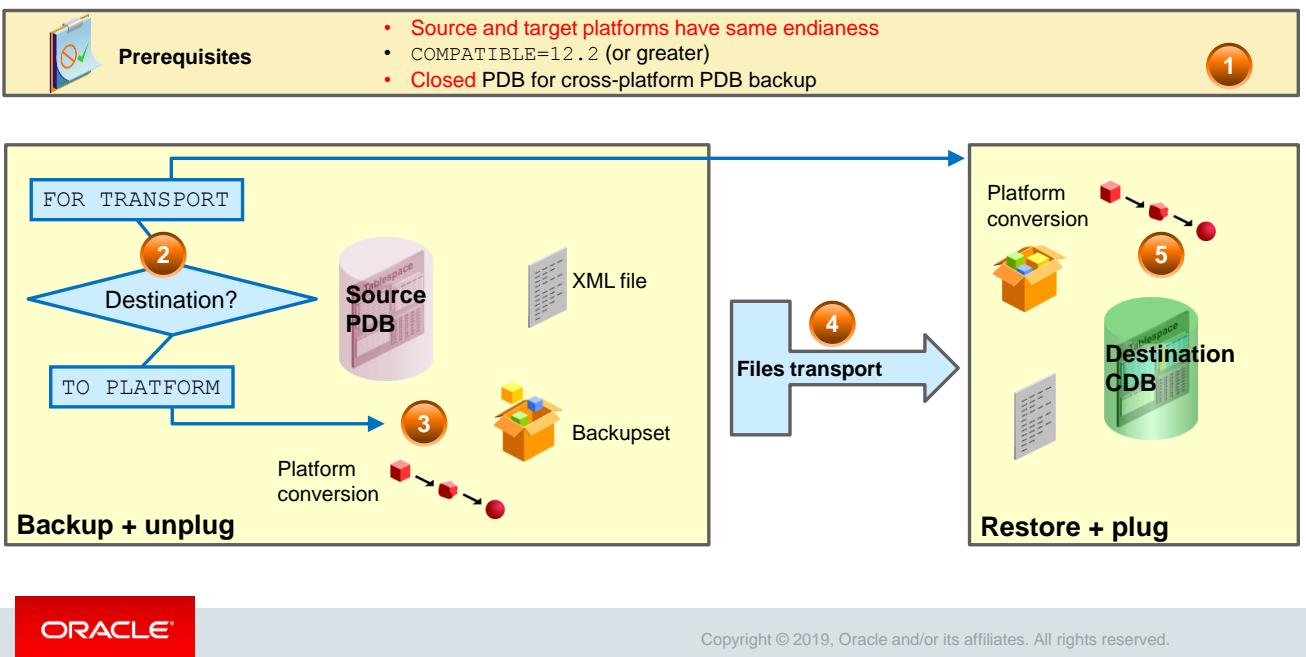
Note: While the PDB upgrade (practice 13-1) is taking place:

1. The end of this lesson and also lesson 14 can be taught.
2. Then practice 13-2 will be completed and practice 13-3 will be started.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Cross-Platform Transportable PDB



Oracle Database 12.1 allows you to transport database or tablespace backup sets with conversion at source or destination for same endian platforms.

Oracle Database 12.2 includes cross-platform PDB backup and restore into a CDB by unplugging at backup step and plugging at restore step as long as the source platform and destination platform have the same endian format. For a cross-endian migration, the tablespaces have to be exported and imported with Data Pump, using either the conventional expdp/impdp or the Full Transportable expdp/impdp.

1. Before backing the PDB, ensure that the prerequisites are satisfied.
2. Determine the location of the endian conversion.
 - The FOR TRANSPORT clause creates a cross-platform backup, indicating that the backup set can be transported to any destination database.
 - The TO PLATFORM clause indicates that the conversion performs on the source database for a specific platform and can be restored on that specific platform.
3. Use the BACKUP FOR TRANSPORT or TO PLATFORM command to create a cross-platform PDB backup set on the source host. The new UNPLUG INTO clause creates the XML file containing the metadata for the PDB.
4. Use any operating system utilities to transfer the created backup set and XML file.
5. Use the RESTORE FOREIGN command to restore the cross-platform PDB backup set on the destination host. The new USING clause uses the XML file to plug the appropriate files for the new PDB.
6. Open the PDB.

Cross-Platform PDB Transport: Phase 1

Source PDB



1. Verify the prerequisites:
 - COMPATIBLE: Greater or equal to 12.2
 - OPEN_MODE: MOUNTED
2. Start an RMAN session to connect to the CDB of the PDB.
3. Query the exact name of the destination platform from the V\$TRANSPORTABLE_PLATFORM view.
4. Back up the source PDB, including the XML file (*metadata*):
 - Conversion on the source host

```
RMAN> BACKUP TO PLATFORM 'Linux x86 64-bit'
      UNPLUG INTO '/tmp/pdb2.xml' PLUGGABLE DATABASE pdb1
      FORMAT '/bkp_dir/transport_%U';
```

- Conversion at the destination host

```
RMAN> BACKUP FOR TRANSPORT UNPLUG INTO '/tmp/pdb2.xml'
      PLUGGABLE DATABASE pdb1 FORMAT '/bkp_dir/transport_%U';
```

ORACLE

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

1. Verify the prerequisites: The source PDB must be closed in MOUNTED mode and the COMPATIBLE parameter must be set to 12.2 or higher.
2. Start an RMAN session and connect to the CDB instance of the PDB to be transported.
3. For performing cross-platform PDB transport, you may need the exact name of the destination platform to which you are transporting data and may need to verify that the destination platform is of the same endian format.

```
SQL> SELECT PLATFORM_ID, PLATFORM_NAME, ENDIAN_FORMAT
      FROM V$TRANSPORTABLE_PLATFORM
      WHERE UPPER(PLATFORM_NAME) LIKE '%LINUX%';
```

4. Back up the source PDB by using the BACKUP command with TO PLATFORM or FOR TRANSPORT.

The new UNPLUG INTO clause creates the XML file containing the metadata of the PDB— tablespaces list, datafiles list, options, and parameters values. The FORMAT clause indicates the directory where the backup sets containing the data required for cross-platform database transportation are stored on the source host.

In the first example in the slide, the conversion will take place on the source host, and the files stored in the /bkp_dir directory are converted for the Linux x86 64-bit platform. In the second example, the conversion will take place on the destination host during the restore command, and the files stored in /bkp_dir directory on the source host are not converted yet.

Cross-Platform PDB Transport: Phase 2

Destination CDB



5. Disconnect from the source CDB.
6. Move the backup sets and XML file to destination host.
7. Start an RMAN session to connect to the new target CDB.
8. Restore the full backup set to create the new PDB with the RESTORE command by using the XML file.
 - When the conversion occurs on the source host

```
RMAN> RESTORE USING '/tmp/pdb2.xml' FOREIGN PLUGGABLE DATABASE pdb1 TO NEW  
FROM BACKUPSET '/bkp_dir/transport_0gqoejqv_1_1';
```

- When the conversion occurs at the destination host

```
RMAN> ALTER SYSTEM SET DB_CREATE_FILE_DEST='/oradata/new_pdb';  
RMAN> RESTORE FROM PLATFORM 'Linux x86 64-bit' USING '/tmp/pdb2.xml'  
FOREIGN PLUGGABLE DATABASE pdb1 TO NEW  
FROM BACKUPSET '/bkp_dir/transport_0gqoejqv_1_1';
```



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

5. Disconnect from the source CDB.
6. Move the backup sets and the XML file created by the BACKUP command to the destination host. You can use operating system utilities to move the backup sets and the XML file from the source host to the destination host.
7. Connect to the destination CDB, to which the PDB must be transported. Ensure that the destination CDB is opened.
8. Use the RESTORE command to restore the files for the newly created PDB to the target location and plug the files into the new PDB by using the new USING clause.

Summary

In this lesson, you should have learned how to:

- Upgrade CDBs from 12c to 18c
- Upgrade regular PDBs from 12c to 18c
- Plug in a remote PDB into a target CDB by using RMAN



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Practice 13: Overview

- 13-1: Upgrading a 12.2 regular PDB to an 18c application PDB
- 13-2: Plugging remote PDBs through XTTs
- 13-3: Upgrading a 12.2 CDB to an 18c CDB



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Miscellaneous

ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Describe the limits of data replication
- Describe XStreams usage with PDB and CDB
- Describe Data Guard with CDB and PDB
- Instantiate a PDB on a standby
- Schedule operations in a PDB by using Oracle Scheduler
- Mine PDB statements using LogMiner



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

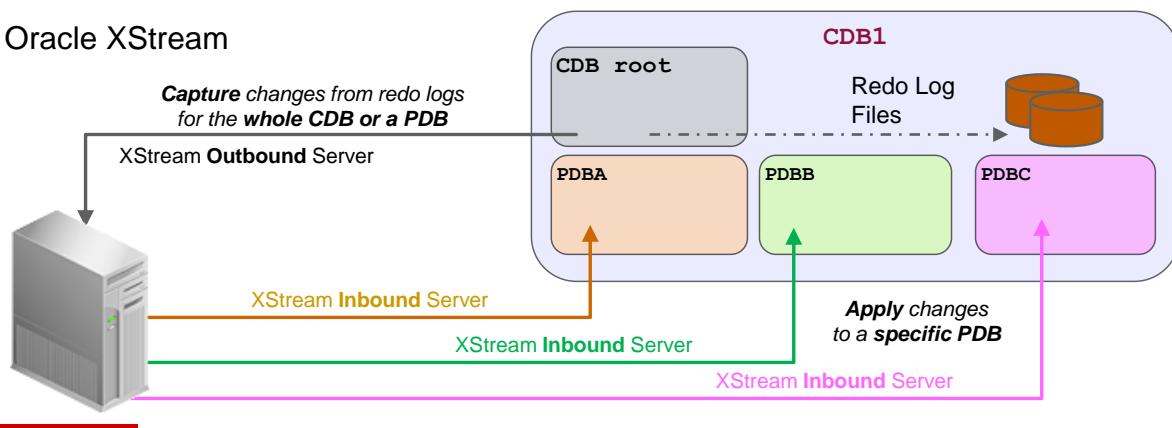
For a complete understanding of the multitenant architecture and usage, refer to the Oracle documentation:

- “*Using Oracle Features in a Multitenant Environment*” in *Oracle Multitenant Administrator’s Guide 18c*

Using Xstreams with a CDB and PDB

Replicate data:

- Oracle Streams supported in Oracle Database 12c
 - With non-CDBs
 - Not with CDBs
- Oracle XStream



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

- Oracle Streams continues to be supported in the Oracle Database 12c non-CDBs but is no longer enhanced for new database features or datatypes. Oracle Streams cannot be used with CDBs.
- XStream is a programmatic interface to allow a client application access to the changes in the database, known as XStream Outbound Server. XStream Inbound Server allows a client application to feed changes into the database and takes advantage of the apply process available in the database.
Oracle GoldenGate is the logical replication, and XStream is licensed via the Oracle GoldenGate license.
Capturing changes from the database must always be completed from the CDB root due to the single unified redo log for the CDB. The XStream outbound can be configured to capture changes from a PDB—regular PDB, application root PDB, application PDB, and/or the entire CDB.
Applying changes via Oracle GoldenGate is done per PDB. An XStream Inbound Server is configured to apply changes into a specific PDB and performs all of its work within the context of the PDB—regular PDB, application root PDB, application PDB, or CDB root.

Creating a Standby of a CDB

Oracle Data Guard at CDB level

- Create a standby of a CDB as you create a standby of a primary non-CDB.
- Create a PDB on a primary CDB.
 - From an XML file: Copy the datafiles specified in the XML file to the standby database.
 - As a clone from another PDB: Copy the datafiles belonging to the source PDB to the standby database.
- Remove or rename PDBs in a primary CDB.
 - UNPLUG and DROP operations on the PDB: The PDB must be closed on the primary as well as all standby databases.
 - RENAME operation on the PDB: The PDB must be in open restricted mode on the primary and closed on all standby databases.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Creating a Physical Standby of a CDB

The following are some of the behavioral differences to be aware of when you create and use a physical standby of a CDB:

- The database role is defined at the CDB level, not at the individual container level.
- If you execute a switchover or failover operation, the entire CDB undergoes the role change.
- Any DDL related to role changes must be executed in the CDB root because a role is associated with an entire CDB. Individual PDBs do not have their own roles.
- In a physical standby of a CDB, the syntax of SQL statements is generally the same as for non-CDBs. However, the effect of some statements, including the following, may be different:
 - ALTER DATABASE RECOVER MANAGED STANDBY functions only in the CDB root. It is not allowed in a PDB.
 - A role is associated with an entire CDB. Individual PDBs do not have their own roles. Therefore, the following role change DDL associated with physical standbys affects the entire CDB:

```
SQL> ALTER DATABASE SWITCHOVER TO target_db_name
SQL> ALTER DATABASE ACTIVATE PHYSICAL STANDBY
```

- The ALTER PLUGGABLE DATABASE [OPEN|CLOSE] SQL statement is supported on the standby, provided you have already opened the CDB root.
- The ALTER PLUGGABLE DATABASE RECOVER statement is not supported on the standby. (Standby recovery is always at the CDB level.)
- To administer a CDB environment, you must have the CDB_DBA role.
- Oracle recommends that the standby database have its own keystore.

Creating a Logical Standby of a CDB

Refer to the *Oracle Data Guard Concepts and Administration 18c - Creating a Logical Standby Database* chapter to get a full description of the topic.

Creating PDBs on a Primary CDB

You must first copy your datafiles to the standby. Do one of the following, as appropriate:

- If you plan to create a PDB from an XML file, copy the datafiles specified in the XML file to the standby database.
- If you plan to create a PDB as a clone from a different PDB, copy the datafiles belonging to the source PDB to the standby database.

The path name of the datafiles on the standby database must be the same as the path name that will result when you create the PDB on the primary in the next step, unless the `DB_FILE_NAME_CONVERT` database initialization parameter has been configured on the standby. In that case, the path name of the datafiles on the standby database should be the path name on the primary with `DB_FILE_NAME_CONVERT` applied.

Removing PDBs from the Primary CDB

The following restrictions apply when you are removing or renaming a PDB at the primary:

- If the primary database is a CDB, then to perform `DDL UNPLUG` and `DROP` operations on a PDB, the PDB must first be closed on the primary as well as all standby databases.
- If the primary database is a CDB, then to perform a `DDL RENAME` operation on a PDB, the PDB must first be put in open restricted mode on the primary and closed on all standby databases.
- If you do not close the PDB at the standby before removing it or renaming it at the primary database, then the standby stops the recovery process for all PDBs. You must close the dropped PDB at the standby and then restart recovery using the following SQL statement:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE;
```

Instantiating a PDB on a Standby

Creating a PDB on a primary CDB:

- 12c** From an XML file: Copy the datafiles specified in the XML file to the standby database.
- 18c** Use the `STANDBY_PDB_SOURCE_FILE_DIRECTORY` parameter to specify a directory location on the standby where source datafiles for instantiating the PDB may be found
→ Datafiles are automatically copied.
- 12c** As a clone from another PDB: Copy the datafiles belonging to the source PDB to the standby database.
- 18c** Use the `STANDBY_PDB_SOURCE_FILE_DBLINK` parameter to specify the name of a database link that is used to copy the datafiles from the source PDB to which the database link points.
→ The file copy is automatically done only if the database link points to the source PDB and the source PDB is open in read-only mode.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

In Oracle Database 12c, when you create a PDB in the primary CDB with a standby CDB, you must first copy your datafiles to the standby. Do one of the following, as appropriate:

- If you plan to create a PDB from an XML file, the datafiles on the standby are expected to be found in the PDB's OMF directory. If this is not the case, then copy the datafiles specified in the XML file to the standby database.
- If you plan to create a PDB as a clone, then copy the datafiles belonging to the source PDB to the standby database.

The path name of the datafiles on the standby database must be the same as the path name that will result when you create the PDB on the primary in the next step, unless the `DB_FILE_NAME_CONVERT` database initialization parameter has been configured on the standby. In that case, the path name of the datafiles on the standby database should be the path name on the primary with `DB_FILE_NAME_CONVERT` applied.

In Oracle Database 18c, you can use initialization parameters to automatically copy the datafiles to the standby.

- If you plan to create a PDB from an XML file, the `STANDBY_PDB_SOURCE_FILE_DIRECTORY` parameter can be used to specify a directory location on the standby where source datafiles for instantiating the PDB may be found. If they are not found there, files are still expected to be found in the PDB's OMF directory on the standby.
- If you plan to create a PDB as a clone, the `STANDBY_PDB_SOURCE_FILE_DBLINK` parameter specifies the name of a database link, which is used to copy the datafiles from the source PDB to which the database link points. The file copy is done only if the database link points to the source PDB, and the source PDB is open in read-only mode. Otherwise, the user is still responsible for copying datafiles to the OMF location on the standby.

Scheduling Operations in a PDB

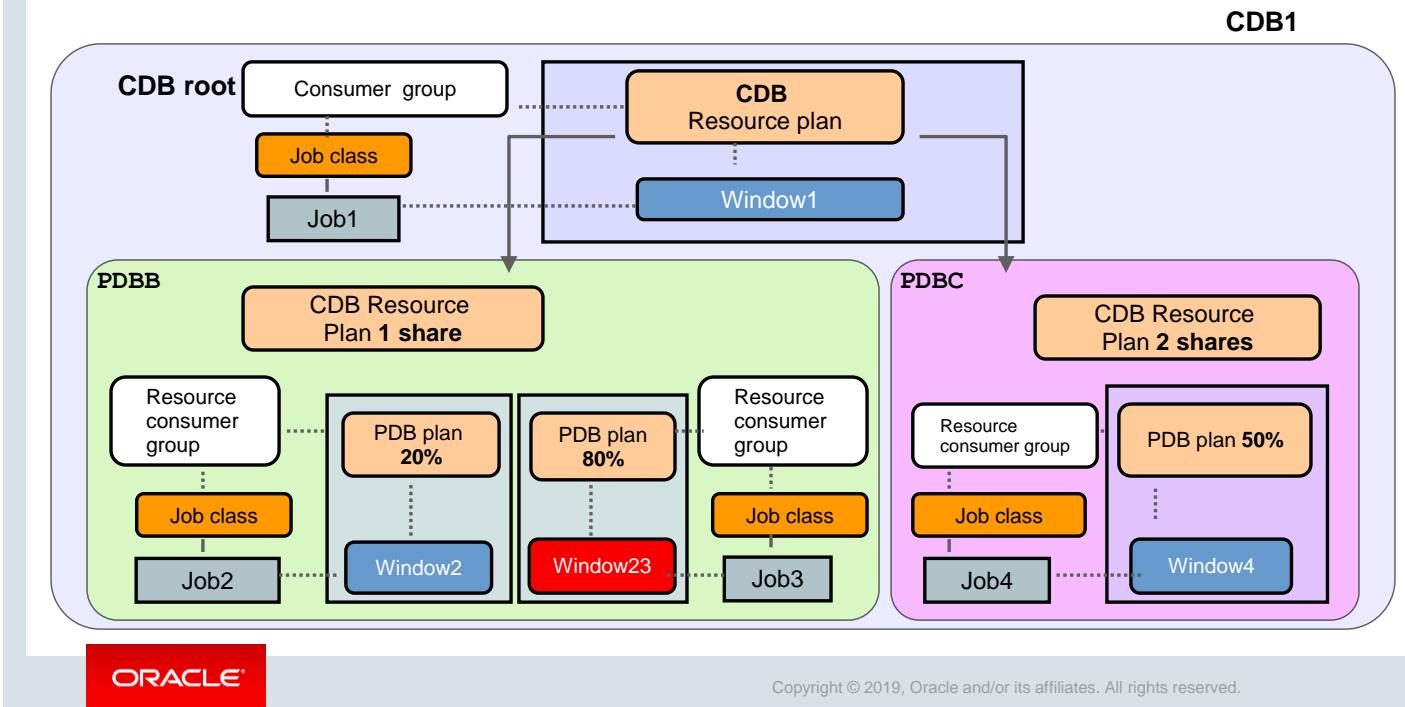
- A job defined in a PDB runs only if a PDB is open.
- User-created scheduler objects can be exported/imported into the PDB using Data Pump.
- Predefined scheduler objects are NOT exported.
 - Any changes made to these objects have to be made once again after the database has been imported into the PDB.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Scheduler objects created by the user can be exported/imported into the PDB using Oracle Data Pump. Predefined scheduler objects are not exported, and that means that any changes made to these objects by the user have to be made once again after the database has been imported into the PDB. A job defined in a PDB only runs if a PDB is open.

Jobs Coordinator and Resources



Scheduler Attribute Settings

Scheduler attribute settings is performed at the PDB level only. For example, if you set the `EMAIL_SENDER` attribute in the CDB root, it applies to the jobs that run in the CDB root, not the jobs running in a specific PDB. If you want to pick a new `EMAIL_SENDER` for a PDB, you must set the global attribute in that PDB.

Job Coordinator Process

The coordinator looks at the CDB root and all the PDBs to select jobs. The availability of resources to run a job depends on the consumer group of the job and the resource plan currently in effect in the container. The coordinator makes no attempt to be fair to every PDB. The only way to ensure that jobs from a PDB are not starved is to allocate enough resources to it.

Windows are open in the PDB and CDB root levels. In a non-CDB, only one window can be open at any given time. In a CDB, there are two levels of windows:

- At the PDB level, windows can be used to set resource plans that allocate resources among consumer groups belonging to that PDB.
- At the CDB root level, windows can be used to allocate resources to various different PDBs.

Therefore, at any time, there can be a window open in the CDB root and one in each PDB.

When a job slave executes a job, it switches to the PDB that the job belongs to.

Mining Statements of a PDB Using LogMiner

- Mining the CDB redo log files
- V\$LOGMNR_CONTENTS view
 - SRC_CON_NAME contains the pluggable database (PDB) name.
 - SRC_CON_ID contains the PDB ID.
 - SRC_CON_DBID contains the PDB identifier.
 - SRC_CON_GUID contains the GUID associated with the PDB.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The redo log files are common for the whole CDB. The information contained in redo log files is annotated with the identity of the PDB where a change occurs.

The V\$LOGMNR_CONTENTS view containing the log history information has new columns, the CON_ID, SRC_CON_ID, SRC_CON_NAME, SRC_CON_DBID, and SRC_CON_GUID.

The possible values include for CON_ID:

- 0: This value is used for rows containing data that pertain to the whole CDB. (This value is also used for rows in non-CDBs.)
- 1: This value is used for rows containing data that pertain to only the CDB root.
- n: It is the applicable container ID for the rows containing data.

The SRC_CON_NAME contains the PDB name. This information will only be available when mining with a current LogMiner dictionary.

The SRC_CON_ID contains the PDB ID (the ID column from the DBA_PDBS view). This information will be available with or without a current LogMiner dictionary.

The SRC_CON_DBID contains the PDB identifier (the DBID column from the DBA_PDBS view). This information will only be available when mining with a current LogMiner dictionary.

The SRC_CON_GUID contains the GUID associated with the PDB (the GUID column from the DBA_PDBS view). This information will only be available when mining with a current LogMiner dictionary.

Summary

In this lesson, you should have learned how to:

- Describe the limits of data replication
- Describe XStreams usage with PDB and CDB
- Describe Data Guard with CDB and PDB
- Instantiate a PDB on a standby
- Schedule operations in a PDB by using Oracle Scheduler
- Mine PDB statements using LogMiner



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Practice 14: Overview

- No practices



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.



A

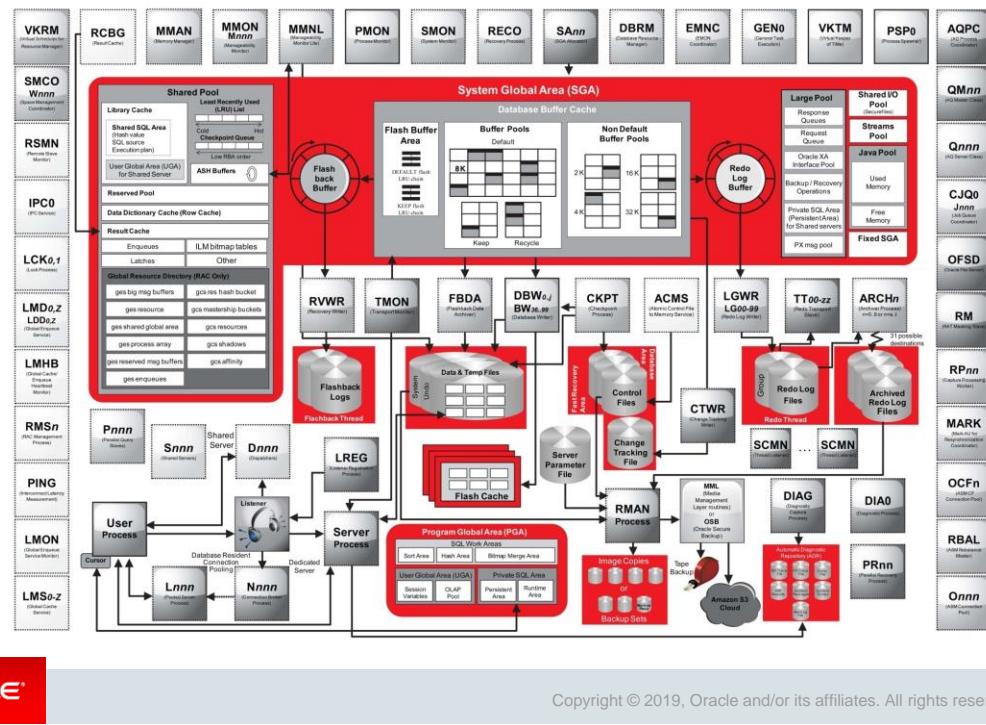
Processes, Views, Parameters, Packages, and Privileges



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Instance and Database



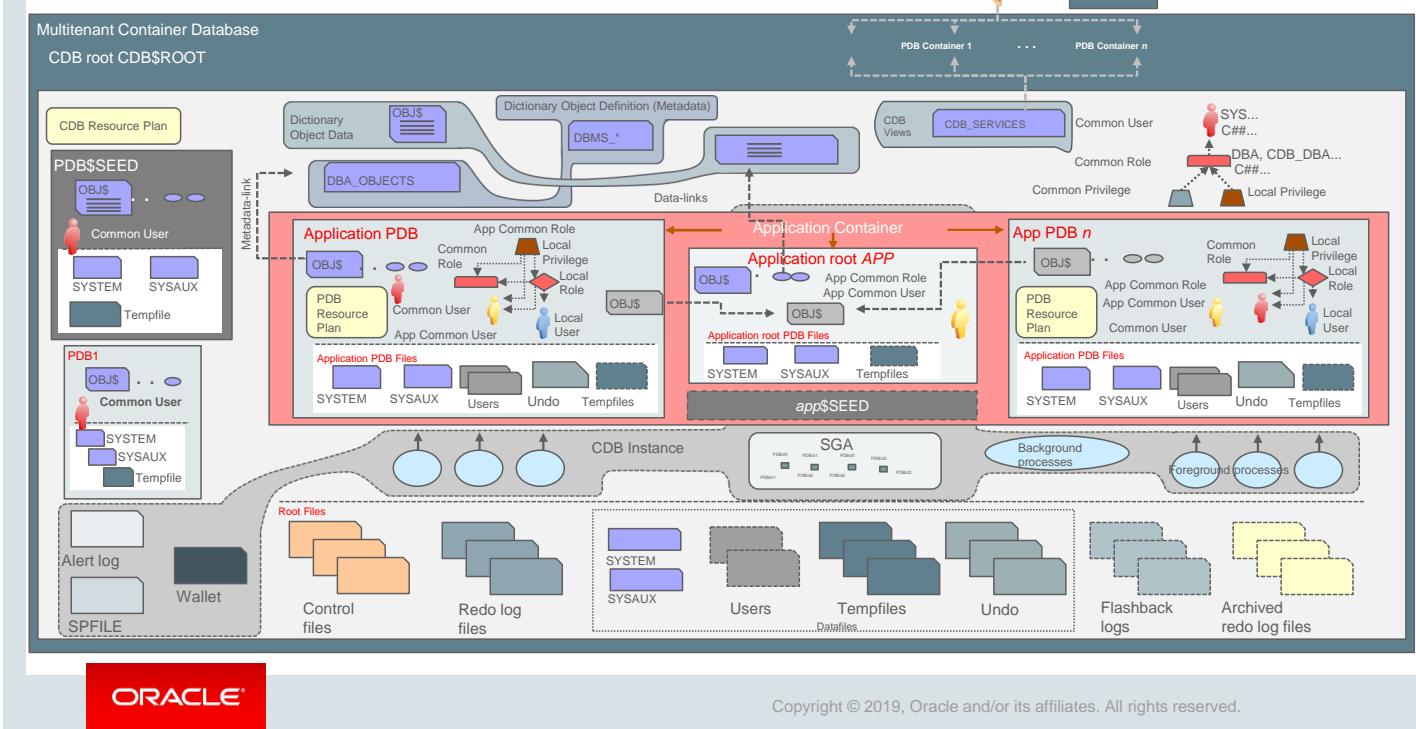
ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Captions:

1. Circle elements represent Oracle processes. If they are surrounded by dotted lines (equal dotted elements), they can run as either threads or OS processes. If they are surrounded by a solid line, they can run as only OS processes. The SCMN case is an exception. When using the multiprocess, multithreaded architecture, each OS process running more than one Oracle process also runs a special thread called SCMN, which is basically an internal listener thread. All thread creation is routed through this thread.
2. Darker circle elements are the new elements for DB 12c.
3. The two main different color nuances for circle elements are to make the distinction between RAC and non-RAC processes.
4. Files are represented by cylinders.
5. Storage location for those files is divided into three main areas: Fast Recover Area, Database Area, and Automatic Diagnostic Repository. Areas are designated by background rectangles by using three different colors. The exception is the server parameter file. If you find a file type part of two areas, it means that some corresponding files can be in both areas.
6. Inside one circle element, you may see two names. This is to indicate that the second (smaller letters) is a slave of the first.

Multitenant Architecture



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

CDB and PDB Views and Columns

- CDB_xxx: All of the objects in the CDB across all PDBs
- DBA_xxx: All of the objects in a container or PDB
- CDB_PDBS: All PDBs within CDB
- CDB_TABLESPACES: All tablespaces within CDB
- CDB_USERS: All users within CDB (common and local)
- CDB_PDB_SNAPSHOTS: All PDB snapshots associated with PDBs 18c
- V\$PDBS: Displays information about PDBs associated with the current instance
- V\$CONTAINERS: Displays information about PDBs and the root associated with the current instance
- PDB_PLUG_IN_VIOLATIONS: Displays information about PDB violations after compatibility check with CDB
- RC_PDBS: Recovery catalog view about PDB backups



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

CDB and PDB Views and Columns

- DBA_APPLICATIONS: Application created on top of application PDBs
- Other views related to applications:
 - DBA_APP_PATCHES
 - DBA_APP_STATEMENTS
 - DBA_APP_ERRORS
 - DBA_APP VERSIONS
- New columns in V\$CONTAINERS view:
 - APPLICATION_ROOT = YES|NO
 - APPLICATION_PDB = YES|NO
 - APPLICATION_SEED = YES|NO
 - APPLICATION_ROOT_CON_ID = <con_id>
 - APPLICATION_ROOT_CLONE = YES|NO
 - PROXY_PDB = YES|NO
 - PDB_COUNT = <nbr>
- New columns in V\$PDBS view:
 - PDB_COUNT = <nbr>



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

CDB and PDB Views and Columns

- New columns or column values in CDB_PDBS view:
 - UPGRADE_PRIORITY = <nbr>
 - APPLICATION_ROOT = YES | NO
 - APPLICATION_PDB = YES | NO
 - APPLICATION_SEED = YES | NO
 - APPLICATION_ROOT_CON_ID = <con_id>
 - IS_PROXY_PDB = YES | NO
 - STATUS = NORMAL | RELOCATING | RELOCATED | REFRESHING | STUB
 - SNAPSHOT_MODE = NONE | MANUAL | AUTO
- New columns in DBA_TABLES view:
 - CONTAINER_MAP_OBJECT = YES | NO
 - CONTAINERS_DEFAULT = YES | NO
 - CONTAINER_MAP = YES | NO



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

CDB and PDB Views and Columns

- New values in CDB_OBJECTS view, column SHARING = METADATA LINK | DATA LINK | EXTENDED DATA LINK | NONE
- New column INHERITED = YES | NO in views:
 - CDB_USERS
 - CDB_ROLES
 - CDB_TAB_PRIVS
 - CDB_SYS_PRIVS
 - CDB_ROLE_PRIVS
 - ROLE_ROLE_PRIVS
 - ROLE_SYS_PRIVS
 - ROLE_TAB_PRIVS
 - AUDIT_UNIFIED_POLICIES
 - DVSYS.DBA_DV_REALM
 - DVSYS.DBA_DV_REALM_OBJECT



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

CDB and PDB Views and Columns

- New values in PROPERTY_NAME, column in DATABASE_PROPERTIES view:
 - MAX_PDB_SNAPSHOTS = 8 18c
 - CONTAINER_MAP, LOCAL_UNDO_ENABLED , LEAD_CDB = TRUE | FALSE 18c
 - LEAD_CDB_URI = dblink:link_to_cdb_lead 18c
 - CONTAINERS_HOST = <hostname>
 - CONTAINERS_PORT = 1521
- New column in V\$ENCRYPTION_WALLET view:
 - KEYSTORE_MODE = NONE | UNITED | ISOLATED 18c
- CDB_LOCKDOWN_PROFILES
- V\$RESTORE_POINT



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

CDB and PDB Views and Columns

- AWR_ROOT_PDB_IN_SNAP
- AWR_PDB_PARAMETER
- AWR_PDB_RSRC_PDB_METRIC
- AWR_PDB_SQL_SUMMARY
- V\$RSRC_PDBMETRIC / V\$RSRC_PDBMETRIC_HISTORY / DBA_HIST_RSRC_PDB_METRIC
- V\$RSRC_PDB / V\$RSRC_PDB_HISTORY



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

CDB and PDB Parameters

- **ENABLE_PLUGGABLE_DATABASE:** Required to create a CDB at CDB instance startup
- **PDB_FILE_NAME_CONVERT:** Maps names of existing files to new file names when processing a CREATE PLUGGABLE DATABASE statement
- **CDB_COMPATIBLE:** Enables you to get behavior similar to a non-CDB
- **PDB_OS_CREDENTIAL:** Enables another OS user than `oracle` to connect to PDBs
- **DB_PERFORMANCE_PROFILE**
- **PDB_LOCKDOWN**
- **SGA_MIN_SIZE**
- **MAX_PDBS**
- **WALLET_ROOT:** Replaces the `ENCRYPTION_WALLET_LOCATION` parameter in the `sqlnet.ora` file, which is easier to use to set the location of keystores 18c
- **TDE_CONFIGURATION:** Sets the type of keystore to use 18c



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

CDB and PDB New Packages

- DBMS_PDB.DESCRIBE
- DBMS_PDB.CHECK_PLUG_COMPATIBILITY
- DVSYS.DBMS_MACADM.CREATE_REALM :
 - REALM_SCOPE =>
 - DBMS_MACUTL.G_SCOPE_COMMON
 - DBMS_MACUTL.G_SCOPE_LOCAL
 - ENABLED =>
 - DBMS_MACUTL.G_SIMULATION
 - DBMS_MACUTL.G_YES
 - DBMS_MACUTL.G_NO



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.



B

Consolidated Database Replay Procedures



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Consolidated Replay Steps

1. Create captures in separate directories.
2. Place all capture workloads in the same directory.
3. Process capture workloads for target.
4. Set replay directory.
5. Create replay schedule.
 - Add capture workloads.
 - Specify replay order of capture workloads.
6. The replay CDB is restored to start of capture state.
7. Initialize Consolidated Replay.
8. Remap Connections.
9. Prepare Consolidated Replay.
10. Calibrate and Start Workload Replay Clients (WRC).
11. Start Consolidated Replay.



ORACLE®

Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Shown here is a more detailed set of steps for using Consolidated Replay. In this procedure, it is assumed that you have already captured the workloads from different non-CDBs or PDBs of different CDBs. The slides in the rest of this lesson are ordered to follow this procedure.

The steps above are a general procedure for preparing to perform a consolidated replay.

In step 5, you must start with calling the `BEGIN_REPLAY_SCHEDULE` and end with calling `END_REPLAY_SCHEDULE`.

In step 6, the replay CDB is restored to the start of capture state. A variety of methods may be used to restore the replay CDB to the proper state, including full transportable export non-CDB/import PDB, cloning PDB, Data Guard snapshot standby, flashback database, and full database recovery.

In step 10, one or more `wrc` processes are started and optionally calibrated.

In step 11, the `START_CONSOLIDATED_REPLAY` procedure is executed. This procedure assumes sufficient number of `WRC` processes have been started.

Procedures for Steps 4 and 5

4. Set replay directory.
5. Create replay schedule.
 - Add capture workloads.
 - Specify replay order of capture workload.

```

DECLARE
    capture1 NUMBER;
    capture2      NUMBER;
BEGIN
    DBMS_WORKLOAD_REPLAY.SET_REPLAY_DIRECTORY('cap_root');
    DBMS_WORKLOAD_REPLAY.BEGIN_REPLAY_SCHEDULE('CONS_SCHEDULE');
    select DBMS_WORKLOAD_REPLAY.ADD_CAPTURE('CRM') into capture1 from dual;
    select DBMS_WORKLOAD_REPLAY.ADD_CAPTURE('SALES') into capture2 from dual;
    select DBMS_WORKLOAD_REPLAY.ADD_SCHEDULE_ORDERING(
        schedule_capture_id => capture2, waitfor_capture_id => capture1)
        from dual;
    DBMS_WORKLOAD_REPLAY.END_REPLAY_SCHEDULE;
END;
  
```



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

To create a replay schedule, first start the process with a call to `SET_REPLAY_DIRECTORY`. This procedure sets a directory that contains multiple workload captures as the current replay directory. `cap_root` is such a directory.

Then continue the process with a call to `BEGIN_REPLAY_SCHEDULE`. Only one replay schedule can be in creation mode at a time. Calling this again before `END_REPLAY_SCHEDULE` will cause an error.

Add all the capture workloads by capture directory name. Each `ADD_CAPTURE` function call returns a `capture_id`. With the `ADD_CAPTURE` function, you can also specify the capture replay attributes.

The `capture_id` is used to specify the workload ordering with the `ADD_SCHEDULE_ORDERING` function.

The last procedure `END_REPLAY_SCHEDULE` wraps up the creation of the current schedule. The schedule is now saved and associated with the replay directory and can be used for a replay.

Procedures for Steps 6 and 7

6. Restore replay database.

7. Initialize the replay.

```
DBMS_WORKLOAD_REPLAY.INITIALIZE_CONSOLIDATED_REPLY(REPLAY_NAME => 'CONS_REPLAY',
SCHEDULE_NAME => 'CONS_SCHEDULE');
```

- This procedure loads the connection information in the capture subsets into the DBA_WORKLOAD_CONNECTION_MAP view.
- Use the connection identifiers (conn_id) from this table to remap connections.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Restore the replay database to the start state of the capture using the method of your choice.

Initialize the replay with the INITIALIZE_CONSOLIDATED_REPLY procedure. This procedure reads the information from the workloads in the replay directory and populates the DBA_WORKLOAD_CONNECTION_MAP table. Query this table to find the connections that need to be remapped.

```
SQL> DESC dba_workload_connection_map
Name                           Null?    Type
-----                         -----
REPLAY_ID                      NOT NULL NUMBER
CONN_ID                        NOT NULL NUMBER
SCHEMA_CAP_ID                  NUMBER
CAPTURE_CONN                   NOT NULL VARCHAR2(4000)
REPLAY_CONN                    VARCHAR2(4000)
```

Procedure to Remap Connections with PDBs

8. Remap connections.

- For each capture in a schedule, map the service names for the connection.
- Each capture can have different connections even if they are identical captures.

```
DBMS_WORKLOAD_REPLAY.REMAP_CONNECTION(
    schedule_cap_id => 1
    connection_id     => 2,
    replay_connection => "oe/oe@pdb_oe.example.com")
```



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

Each capture workload in a schedule can remap each connection in the workload to a different connect for replay. This allows each capture to be mapped to a different pluggable database (PDB) in a multitenant container database (CDB).

This allows multiple copies of a capture to each map to a different PDB with each PDB matched to the start of the capture.

Procedure to Prepare the Replay

9. Prepare Consolidated Replay.
 - Specific type of synchronization

```
DBMS_WORKLOAD_REPLAY.PREPARE_CONSOLIDATED_REPLAY(  
    synchronization => 'OBJECT_ID')
```



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The next step is run the `PREPARE_CONSOLIDATED_REPLAY` procedure. The procedure `PREPARE_CONSOLIDATED_REPLAY` does similar work for consolidated replay as `PREPARE_REPLAY` does for the single replay. It sets up the replay options such as synchronization mode, connection time scale, and think time scale, and then puts the DB state in the REPLAY mode. One or more external replay clients (WRC) can be started once the `PREPARE_CONSOLIDATED_REPLAY` procedure has been executed.

Note: `PREPARE_CONSOLIDATED_REPLAY` requires that the preceding steps have been followed.

Modes of Synchronization

For single replays

- SCN synchronization is the default
- Recorded SCNs
 - Determine object dependencies
 - Replay call ordering

For Consolidated replay:

- Object ID synchronization:
 - It is recommended (SCN synchronization not supported).
 - It enables fine grain synchronization, providing more replay concurrency.
 - OBJECT_IDs are tracked by user call to minimize object collision on replay.
 - If collision happens, then replay orders calls. Otherwise, replay lets it run to get more concurrency.



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The synchronization parameter controls whether the `COMMIT` order in the captured workload will be preserved during replay.

If this parameter is set to `SCN`, the `COMMIT` order in the captured workload will be preserved during replay, and all replay actions will be executed only after all dependent `COMMIT` actions have completed. The SCN-based synchronization is not supported for consolidated replay.

Setting this parameter to `OBJECT_ID` allows for more concurrency during workload replays for `COMMIT` actions that do not reference the same database objects during workload capture. Independent transactions on the same database are likely to yield faster replay using Object ID synchronization.

With `OBJECT_ID`, each workload captured on a different database and being consolidated is independent of the other. Two objects with the same name in different PDBs will be different “object_ids” and will not collide during replay.

You can disable this option by setting the parameter to `OFF`. This option can yield a faster replay, but the replay will likely yield significant replay divergence. However, this may be desirable if the workload consists primarily of independent transactions and if divergence during unsynchronized replay is acceptable.

Note that consolidated replay only supports non-sync mode (`OFF`) and the `OBJECT_ID` synchronization.

Procedure to Start Replay

10. Calibrate and start the workload replay clients.

```
$ wrc REPLAYDIR=/home/oracle/solutions/dbreplay MODE = calibrate
```

```
$ wrc REPLAYDIR=/home/oracle/solutions/dbreplay MODE=replay USERID=system
```

```
PASSWORD=<password>
```

```
Workload Replay Client ...
```

```
Wait for the replay to start (21:47:01)
```

11. Execute the START_CONSOLIDATED_REPLAY procedure.

```
SQL> EXEC DBMS_WORKLOAD_REPLAY.START_CONSOLIDATED_REPLAY
```



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The replay database is ready to accept workload replay client (WRC) connections once the PREPARE_CONSOLIDATED_REPLAY procedure is executed.

As shown, `wrc` is an OS process and can be run from a client machine in the network. These machines must have the replay client software and a copy of the replay directory that contains the processed capture workloads.

Calibrate the workload clients with the calibrate mode. Because the client is a multithreaded process, it can open multiple sessions to the replay database. So one `wrc` process can simulate many user processes on the capture system. The output of the calibrate mode is an estimate of the number of `wrc` processes that will be needed.

The next step is to start the number of `wrc` clients recommended by the calibrate mode. These processes start, contact the replay database, and wait for the START_CONSOLIDATED_REPLAY command to be issued. After a workload replay is started, new replay clients will not be able to connect to the database. Only replay clients that were started before the START_CONSOLIDATED_REPLAY procedure is executed will be used to replay the multiple-capture.

Views

- DBA_WORKLOAD_REPLAY_SCHEDULES
- DBA_WORKLOAD_SCHEDULE_ORDERING



Copyright © 2019, Oracle and/or its affiliates. All rights reserved.

The views allow the workload schedules to be saved and reused. These tables exist in the root container of a CDB and in the PDBs.

