ORACLE UNIVERSITY




# Oracle Database 12*c* R2: New Features for 12*c* R1 Administrators

Student Guide – Volume I
D93517GC10
Edition 1.0 | October 2016 | D98178

Learn more from Oracle University at **oracle.com/education/**

ORACLE

## Authors

Dominique Jeunot
Jean-François Verrier
Mark Fuller
James Spiller

## Technical Contributors and Reviewers

Krishnanjani Chitta
Randall Richeson
Gerlinde Frenzen
Sailaja Pasupuleti
Anita Mukundan
Harald van Breederode
Jim Stenoish
Branislav Valny

## Editors

Nikita Abraham
Chandrika Kennedy
Vijayalakshmi Narasimhan

## Graphic Editor

Seema Bopaiah

## Publishers

Pavithran Adka
Syed Ali
Giri Venugopal

# Contents

1

# Introduction

# Overview

- This course focuses on the features and enhancements of Oracle Database 12*c* Release 2 (12.2.0.1) that are applicable to database administrators.
- Previous experience with Oracle databases and particularly Oracle Database 12*c* Release 1 patchset 1 (12.1.0.2) is required for a full understanding of many of the new features and enhancements.
- Hands-on practices emphasize functionality rather than test knowledge.
- Hands-on practices complete knowledge that is not covered in lessons.


Refer to *Oracle Database New Features Guide 12c Release 2 (12.2)*.

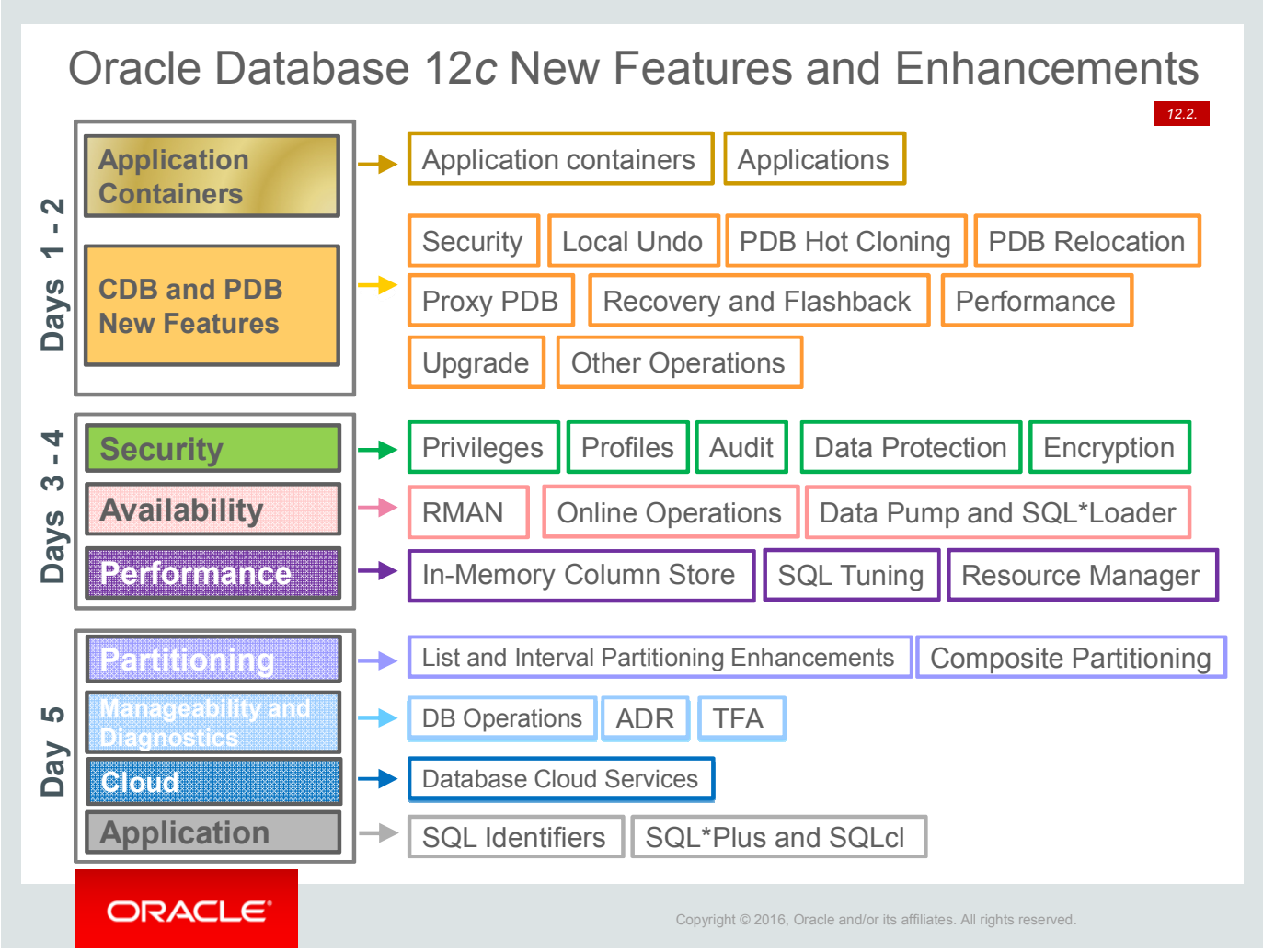
ORACLE





This course is designed to introduce you to the new features of Oracle Database 12*c* Release 2 *(12.2)* that are applicable to the work that is usually performed by database administrators and related personnel. The course covers all features and enhancements. It does not attempt to provide every detail about a feature or cover aspects of a feature that were available in previous releases (except when defining the context for a new feature or comparing past behavior with current behavior). Consequently, the course is most useful if you have administered Oracle Database 12*c*. Even with this background, you should not expect to be able to implement all the features discussed in the course without supplemental reading, especially the Oracle Database 12*c* Release 2 *(12.2)* documentation.

The course consists of instructor-led lessons, in addition to many hands-on practices that enable you to see by yourself how certain new features behave. As with the course content in general, these practices are designed to introduce you to the fundamental aspects of a feature. They are not intended to test your knowledge of unfamiliar syntax or provide an opportunity for you to examine every nuance of a new feature. The length of this course precludes such activity. Consequently, you are strongly encouraged to use the provided scripts to complete the practices rather than struggle with unfamiliar syntax.




## Features, Enhancements, and Schedule

**Oracle Multitenant:** Oracle Multitenant is an option introduced with Oracle Database 12*c* (12.1.0.1). Although the non-CDB architecture is deprecated in Oracle Database 12*c*, and may be desupported and unavailable in a release after Oracle Database 12*c* Release 2, it is still supported for the lifetime of Oracle Database 12*c*. However, Oracle recommends use of the CDB architecture.

Refer to *"Oracle Database Online Documentation 12c Release 1 (12.1) / Installing and Upgrading - Deprecated and Desupported Features for Oracle Database 12c."*

Oracle Database 12*c* Release 2 (12.2) introduces a new type of container in multitenant container databases (CDBs), the application container. Named applications can be installed on top of application containers, allowing versioned set of common data and metadata to be stored in an application container independently from another application container. Commonality in applications refers to users, roles, granted privileges, profiles, objects, and data that are common and shared within an application.

Oracle Database 12*c* Release 1 (12.1.0.1) provided only one method for pluggable database (PDB) cloning. Oracle Database 12*c* Release 2 (12.2) introduces hot PDB cloning, PDB relocation, and proxy PDB. Some of these PDB cloning methods require the local UNDO mode to be configured.

Resource management between PDBs with Resource Manager is enhanced with new performance profiles. Furthermore, Oracle Database 12*c* Release 2 (12.2) introduces PGA memory limits in PDBs. Heat map and automatic data optimization (ADO) policies are supported in CDBs and PDBs.

In Oracle Database 12*c* Release 1 (12.1.0.1), flashback operations could be performed at the CDB level, whereas Oracle Database 12*c* Release 2 (12.2) introduces PDB-level flashback.

12.1.0.1 and 12.1.0.2 CDBs and PDBs can upgrade to 12.2 in a very short time.

**Security**

Oracle Database 12*c* Release 2 (12.2) enables Unified Auditing to audit users that are granted roles and to capture Virtual Private Database (VPD) generated predicates in Unified Audit Trail.

System administrative privileges integrate a new administrative privilege for RAC.

Oracle Database 12*c* Release 2 (12.2) allows Privilege Analysis to make a distinction between captures and runs, and to compare two runs.

Oracle Database 12*c* Release 2 (12.2) includes a redaction format library as part of Data Redaction.

Transparent Sensitive Data Protection (TSDP) is now enhanced to use Unified Auditing, Fine-Grained Auditing (FGA), and Transparent Data Encryption (TDE).

**Availability**

RMAN involves enhancements in table recovery and transport data across platforms. There are enhancements related to online operations, and ongoing enhancements with Oracle Data Pump and SQL*Loader.

**Performance** of the databases and query execution is an ongoing subject for improvement: Oracle Database 12*c* Release 2 (12.2) introduces the Optimizer Statistics Advisor and SQL JOIN processing, and enhancements in SQL Tuning Sets and SQL Performance Analyzer.

In-Memory Column Store is an option introduced in Oracle Database 12*c* Release 1 PS 1 (12.1.0.2). Major enhancements appear in Oracle Database 12*c* Release 2 (12.2) such as dynamic in-memory area resizing, in-memory FastStart, in-memory expressions and virtual columns, and objects eviction with integration of heat map statistics and ADO policies.

**Partitioning:** Oracle Database 12*c* Release 2 (12.2) introduces enhancements in list and interval partitioning methods, as well as new composite partitioning methods.

**Manageability:** You will discover enhancements in Database Operations, Real-Time ADDM, Automatic Diagnostic Repository (ADR) with automatic diagnostic file management, and finally Trace File Analyzer (TFA) Collector and TFA Web.

**Cloud Services:** You will discover the Oracle Database Public Cloud Services.

**Application:** Oracle Database 12*c* Release 2 (12.2) allows new SQL identifier lengths. SQL*Plus commands are now stored in a history SQL list, allowing you to run commands from the History list. A new tool, SQLcl, introduces new commands and features that were missing from SQL*Plus itself.

## Configuration for On-Premises Practices

- Pre-created `ORCL` database and instance with one PDB
- 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`



With the Oracle Database 12.2 new release, the configuration of the environment used for the practices of the course matches the configuration used on the Oracle Cloud Virtual Machine and for the pre-created `ORCL` database of the Database as a Service (DBaaS) instance.

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

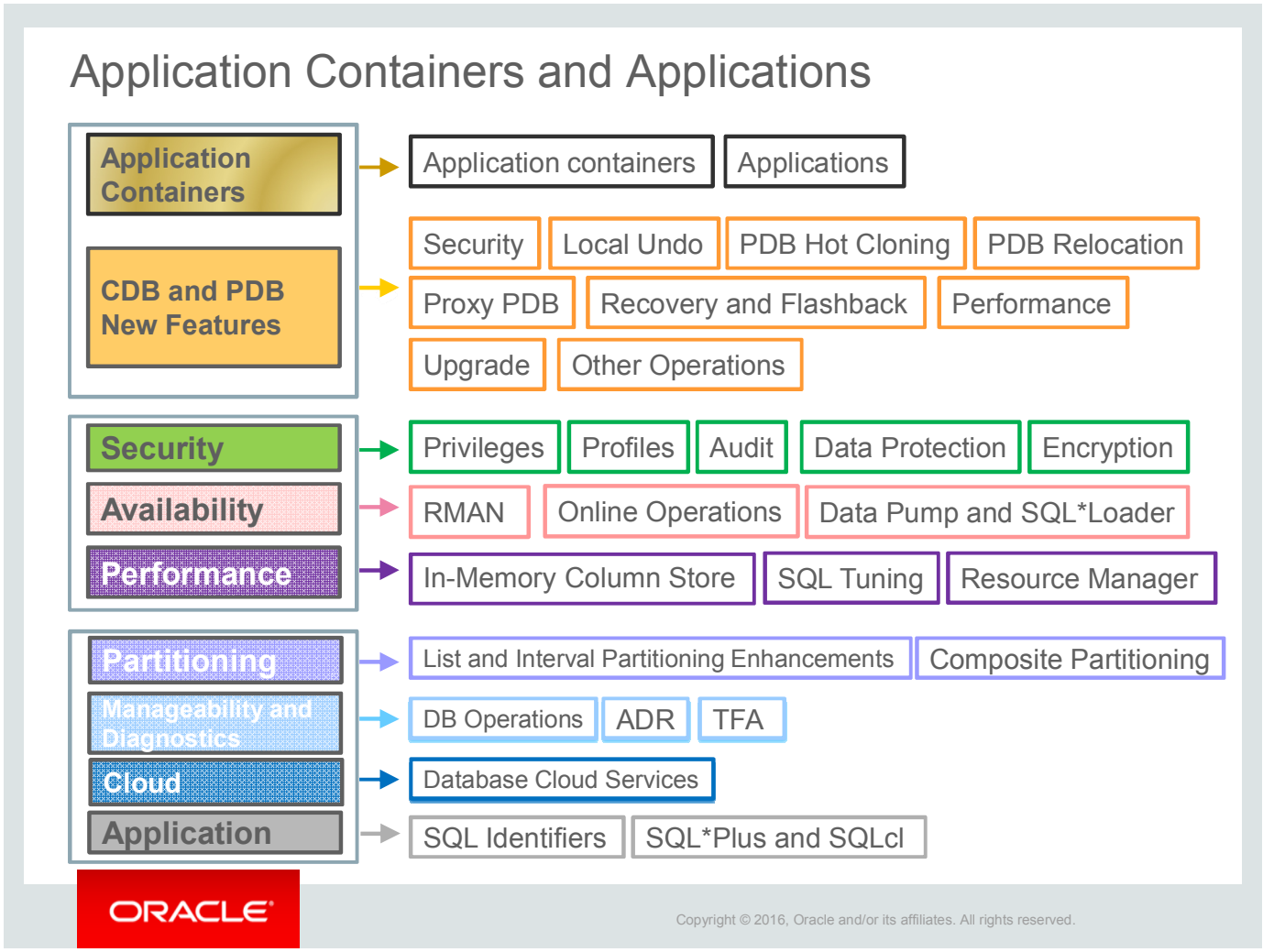
The configuration that is used on the Oracle Cloud Virtual Machine and the pre-created database of the Database as a Service (DBaaS) instance will be covered in detail in the lesson titled “Oracle Database Cloud Services.”

# Practice 1: Overview

- 1-1: Starting Enterprise Manager Database Express
- 1-2: Configuring Enterprise Manager Cloud Control targets
- 1-3: Creating and testing new named credentials

2

# Application Containers and Applications




This lesson explains the new concept of application containers and applications. The lesson also describes many use cases of the new concept of application containers.

# Objectives

After completing this lesson, you should be able to:

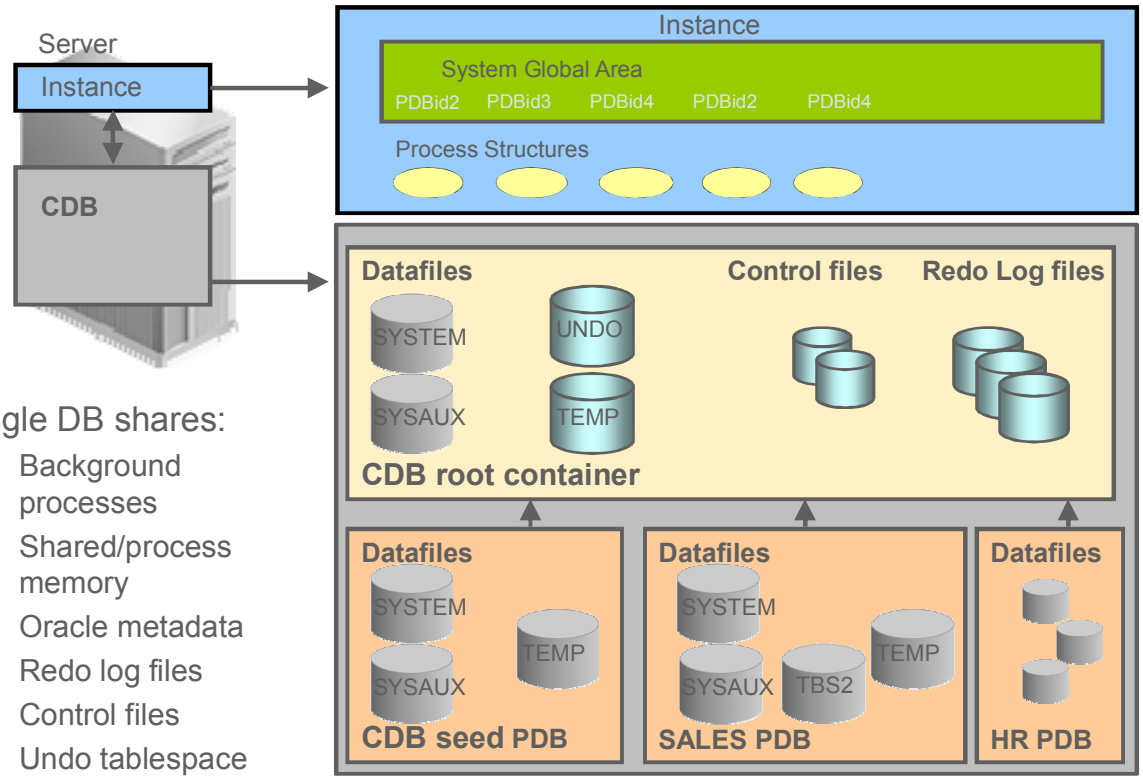
- Describe regular and application containers in CDBs
- Explain the purpose of application root and application seed
- Define application PDBs
- Create application PDBs
- Explain application installation on top of application containers
- Install an application
- Upgrade and patch applications
- Describe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



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

- *Oracle Database Administrator's Guide 12c Release 2 (12.2)*

# Multitenant Container Database Architecture




Single DB shares:

- Background processes
- Shared/process memory
- Oracle metadata
- Redo log files
- Control files
- Undo tablespace



The Oracle Database 12*c* Release 1 introduced the Oracle Multitenant architecture, allowing the consolidation of databases together in a multitenant container database (CDB). A database that is consolidated within a CDB is a pluggable database (PDB). 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.

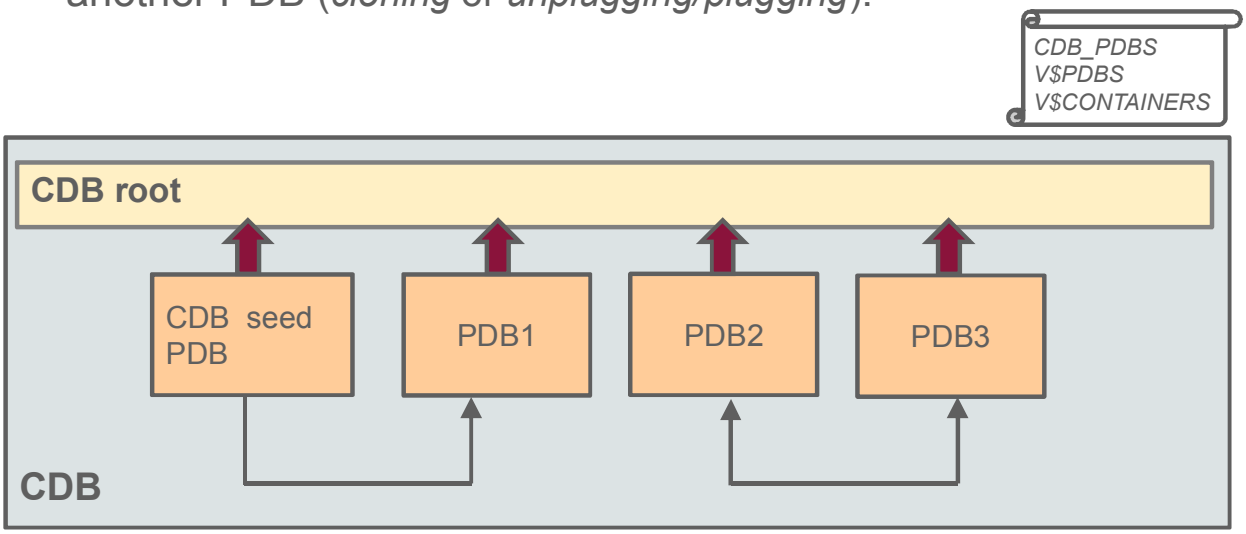
At the physical level, the CDB has a database instance and database files, just as a non-CDB does.

- Redo log files are common for the whole CDB. All the PDBs in a CDB share the ARCHIVELOG mode of the CDB.
- Control files are common for the whole CDB. The control files are updated to reflect any additional tablespaces and data files of the plugged PDBs.
- The UNDO tablespace is common for all containers. 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.
- Each container has its own data dictionary, which is stored in its proper `SYSTEM` tablespace, containing its own metadata, and a `SYSAUX` tablespace.
- Tablespaces can be created in PDBs according to application needs. Each datafile is associated with a specific container, named *`CON_ID`*.

# Regular PDBs

12.1

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

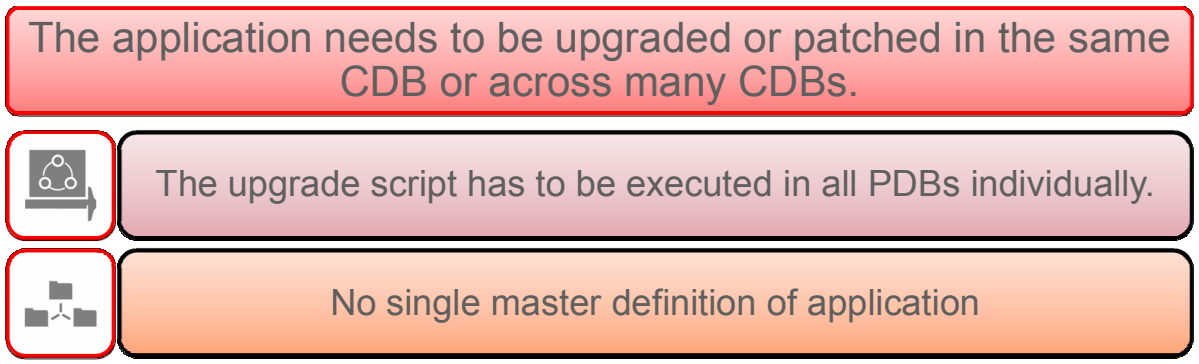





In an Oracle Database 12.1 CDB, a PDB stores data in objects independently from other PDBs. The PDBs are therefore regular PDBs, which can be created from the CDB seed or from another PDB (*cloning* or *unplugging/plugging*).

# PDBs and Applications

12.1






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

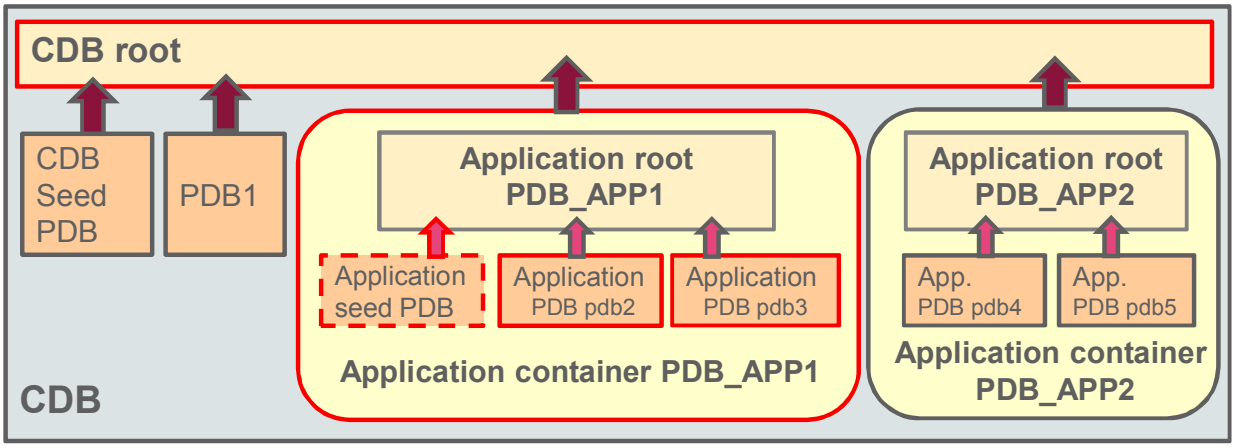
- There is no single master definition of an application on top of pluggable databases.
- The upgrade script has to be executed in all PDBs individually. This is time consuming.

# Application Containers

An application container is a collection of PDBs grouped together within a CDB; stores data for an application; consists of:

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

*New columns*
*CDB_PDBS*
*V$PDBS*
*V$CONTAINERS*








### Application Container

Oracle Database 12.2 introduces the concept of application containers, which 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.

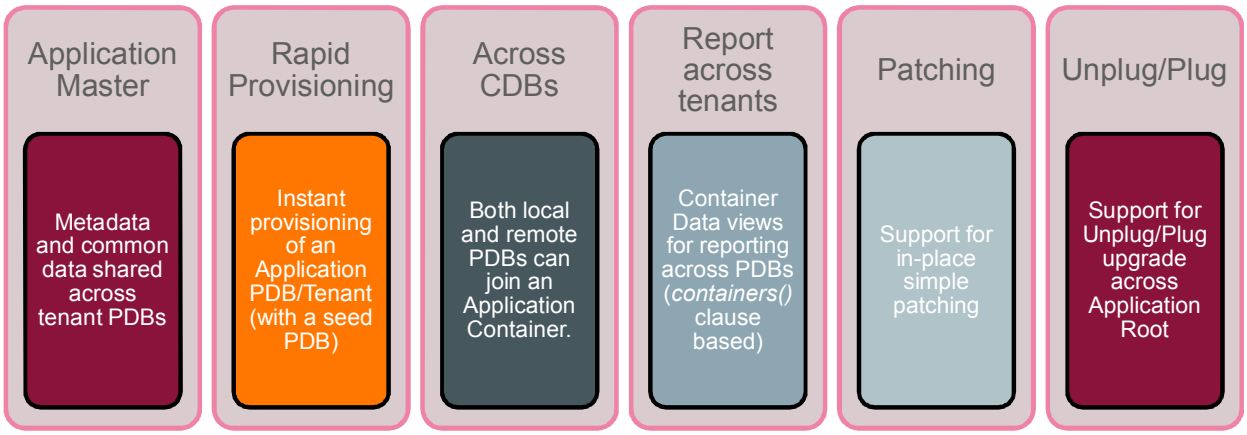
An application seed, which is tied to only one application root, can be created for instantaneous provisioning of application PDBs. Synchronization of the application code in the application seed must be completed before application PDB creation. Any application PDB that is created is a full clone of 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 unless the application PDBs were created from the application seed.

## Application Containers: Other Features






**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.

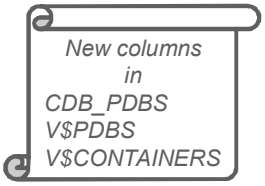
**Patching**

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$ROOT)
  - The first mandatory container created at CDB creation
  - Oracle system–supplied common objects and metadata
  - Oracle system–supplied common users and roles
- **Pluggable database containers** (PDBs)
  - The CDB seed PDB (PDB$SEED)
    - The second mandatory container created at CDB creation
    - Oracle system–supplied common entities for new PDBs
  - Regular containers (12.1 PDBs)
  - Application containers (12.2 PDBs)
    - Application root
    - Optional application seed (*application_container_root_name*$SEED)
    - Application PDBs

ORACLE





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.

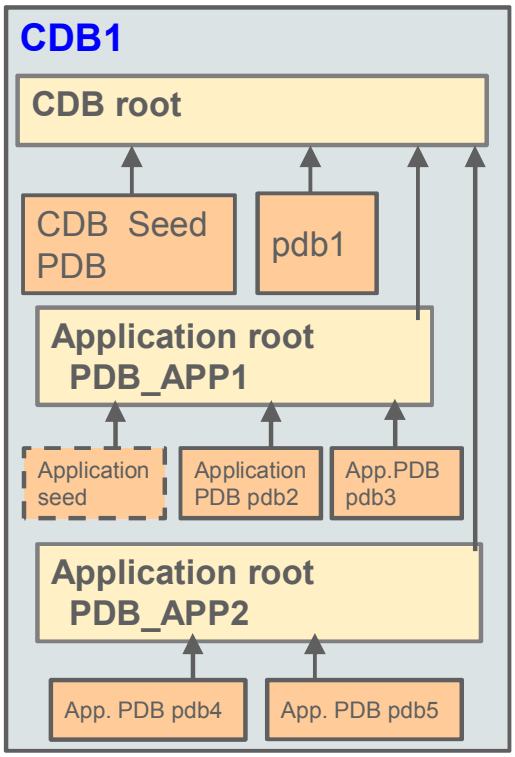
In a CDB, there is only one CDB root, and one CDB seed PDB. The CDB seed PDB is a system-supplied template that is used to create new PDBs.

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, with 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. Optionally, create the application seed for the application PDBs in the application container.
4. Connect to the **PDB_APP1** application root.
5. Create the **PDB2** PDB as an application PDB within the **PDB_APP1** application container.



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. 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.

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`.

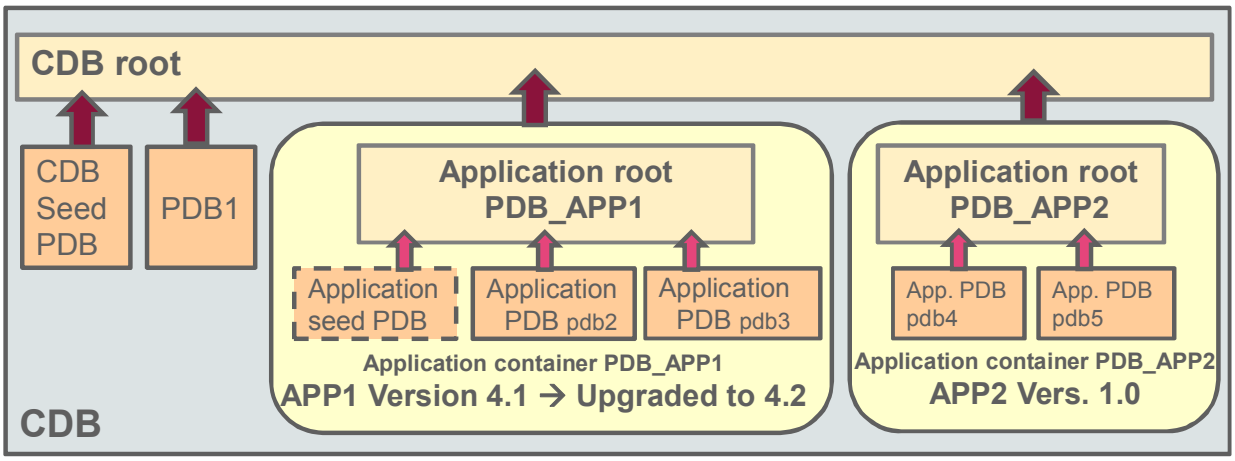
## Application Name and Version

An application container can be tagged with:

- An application name
- An application version number

An application can be patched, upgraded, or uninstalled.

*DBA_APPLICATIONS*
*DBA_APP_VERSIONS*
*DBA_APP_PATCHES*
*DBA_APP_ERRORS*
*DBA_APP_STATEMENTS*






### 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 number that is used for future upgrade and patching operations.

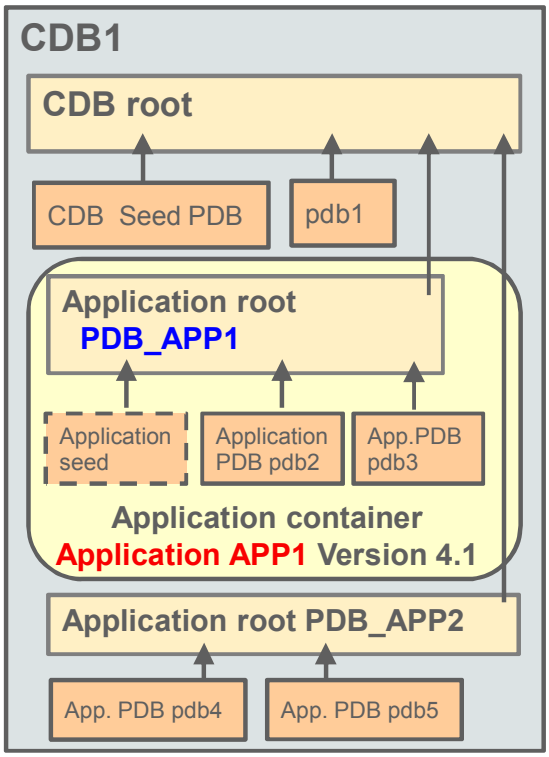
After the application installation is completed in the application root, the application needs to be propagated to the application PDBs. The application PDBs need to be synchronized with the application root.

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 version number of the application. 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 number 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.

ORACLE

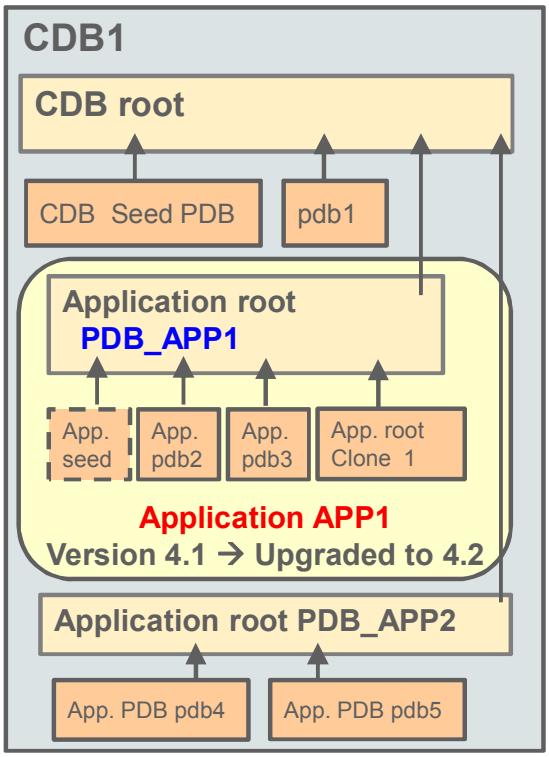


### Application Installation

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 number. This value is the application version number 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 or locally granted to the common or local 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 number of the **APP1** application before starting the upgrade.
3. Start the application upgrade to a higher version number.

```
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.



### Application Patching

Applying a patch on an application or upgrading an application requires that you know the current number of the application version, which is retrievable from the `DBA_APPLICATIONS` view.

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 Upgrade

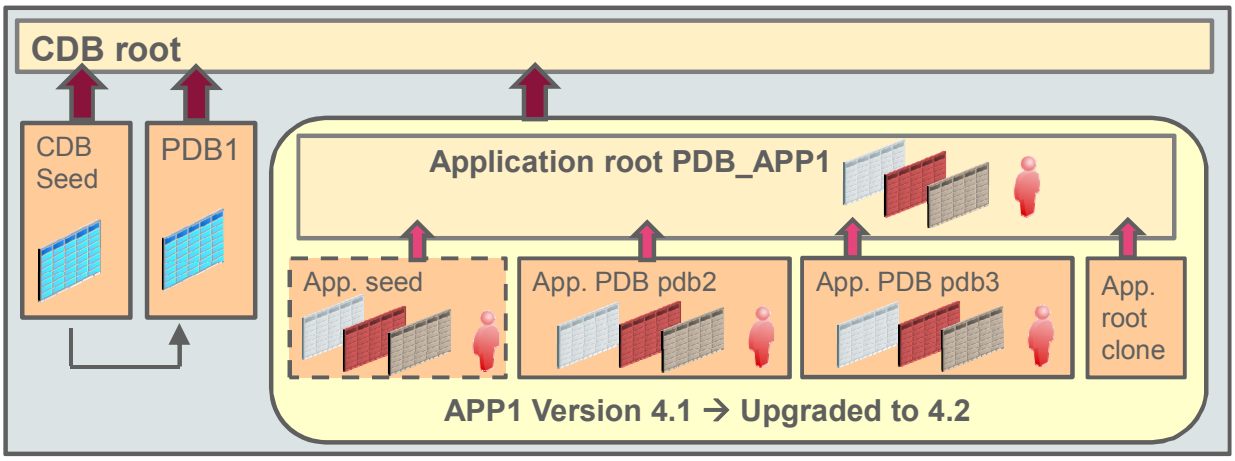
Use the `ALTER PLUGGABLE DATABASE APPLICATION <app_name> BEGIN UPGRADE '4.1' TO '4.2'` and `ALTER PLUGGABLE DATABASE APPLICATION <app_name> END UPGRADE TO '4.2'` statements to indicate the start and end boundaries of the operation, respectively.

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 Common Objects

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





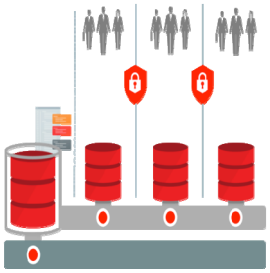

**Application**

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

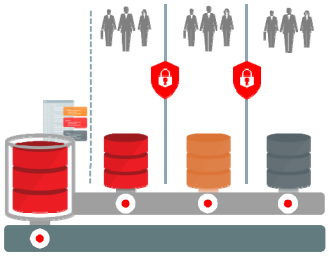
### Pure SaaS

- 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.



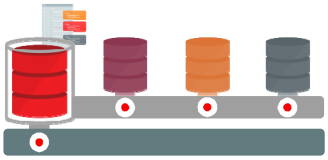
### Hybrid SaaS

- 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.



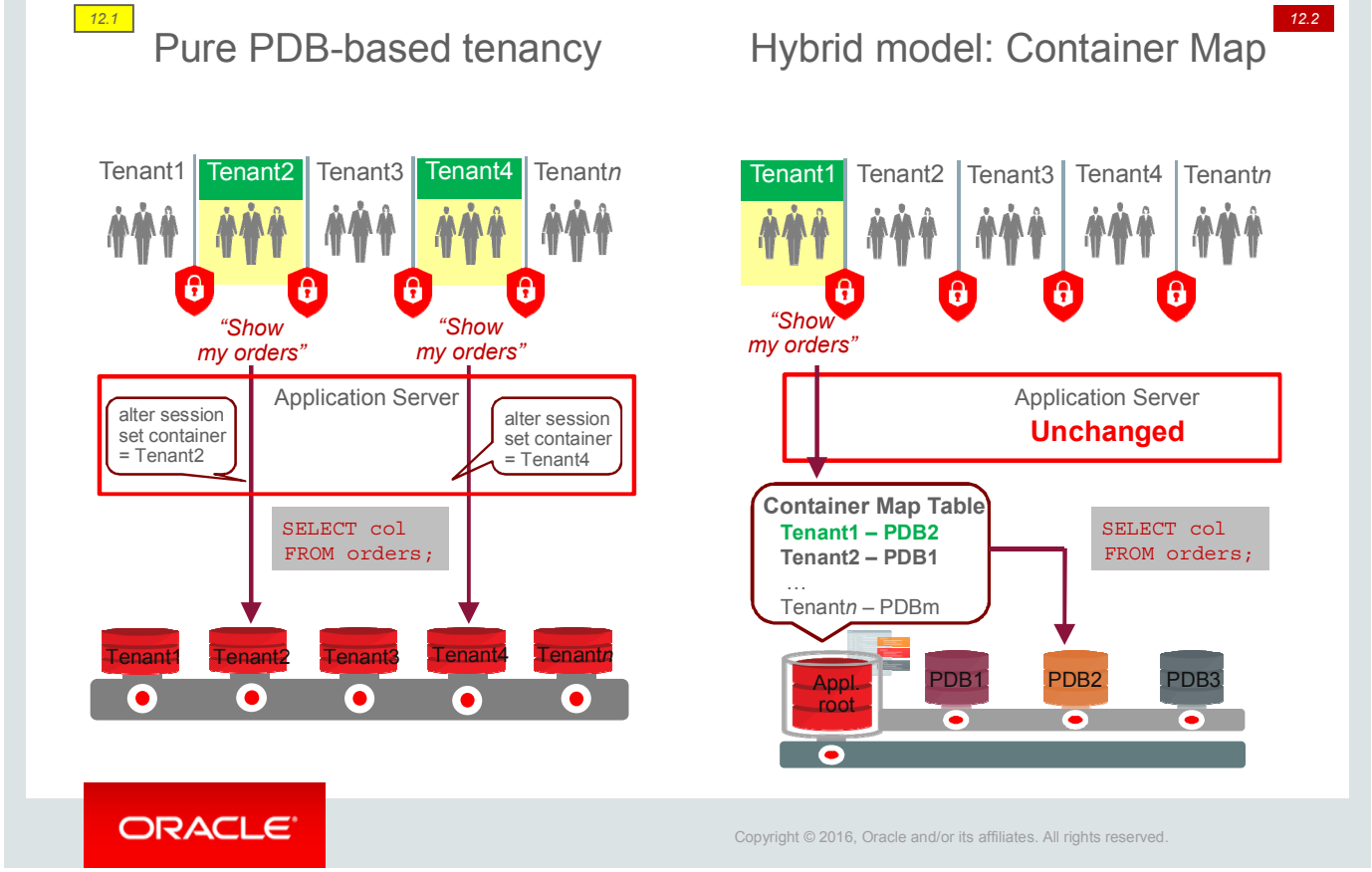


In a pure PDB-based tenancy, each customer's data resides in an individual PDB. All pluggable database–level operations, such as unplug, plug, and clone, are applicable on the individual customer data. Customer data is securely managed and thousands of tenants can be managed in this case.

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';
```

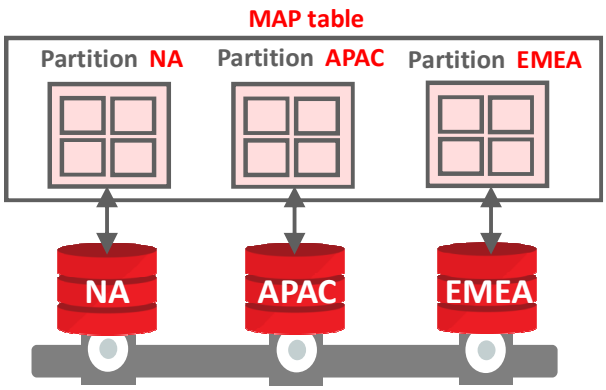
In this case, the query `select <columns> from orders` would be converted to:

```
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*



In Oracle Database 12.1, `CONTAINERS` (*table* or *view*) queried 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.

In Oracle Database 12.2, 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 in Oracle Database 12.2 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.

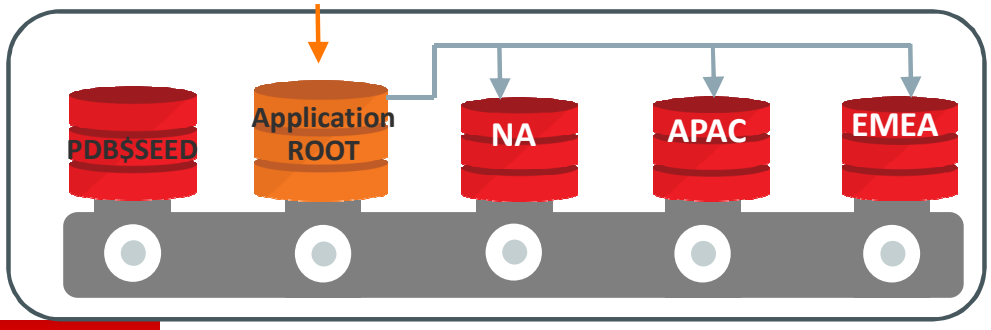
## Container Map: Example

```
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



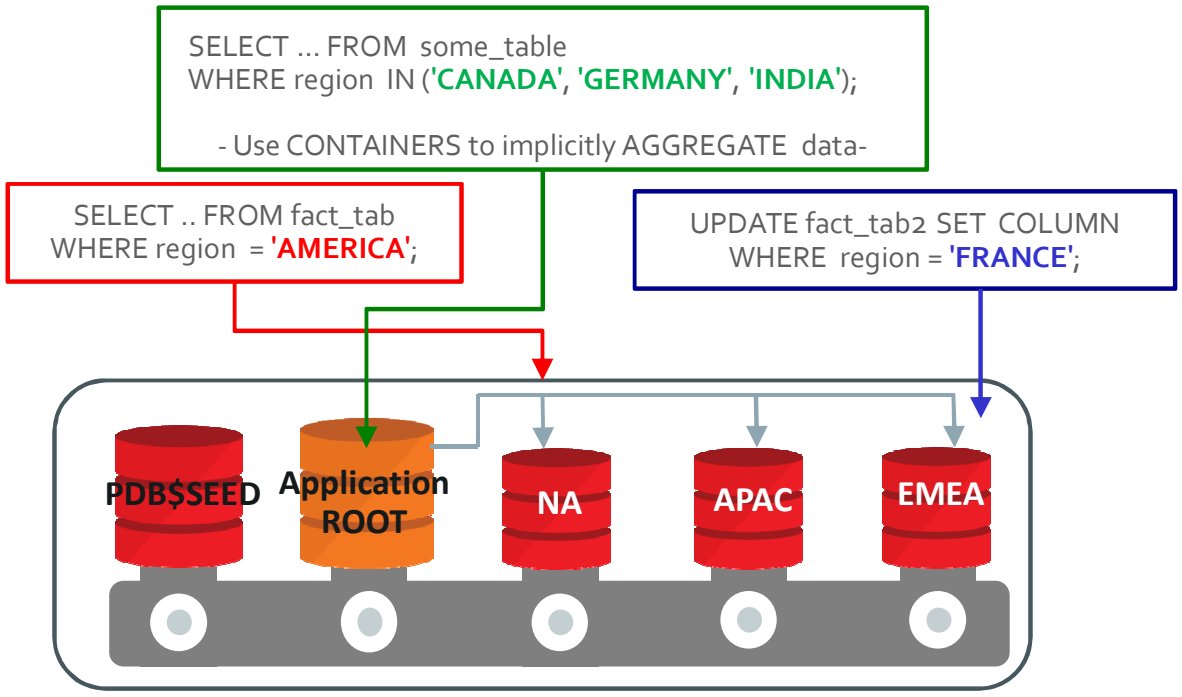


In a hybrid model, you can create common partitioned tables in the application root, mapping a partition of the table to an application PDB of the application container where, for example, the `TENANT_GRP1` partition would store data for customers of group1 in the `Tenant_GRP1` application PDB, and where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

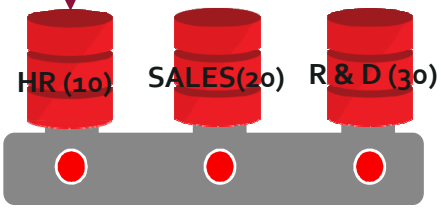
# Container Map and Containers Default

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.

```
SELECT EMPNO
FROM CONTAINERS(EMP)
WHERE CON_ID = 6 ;
```


```
SELECT EMPNO
FROM EMP
WHERE DEPT = 'HR' ;
```




*DBA_TABLES*
*CONTAINERS_DEFAULT= YES*
*CONTAINER_MAP = YES*



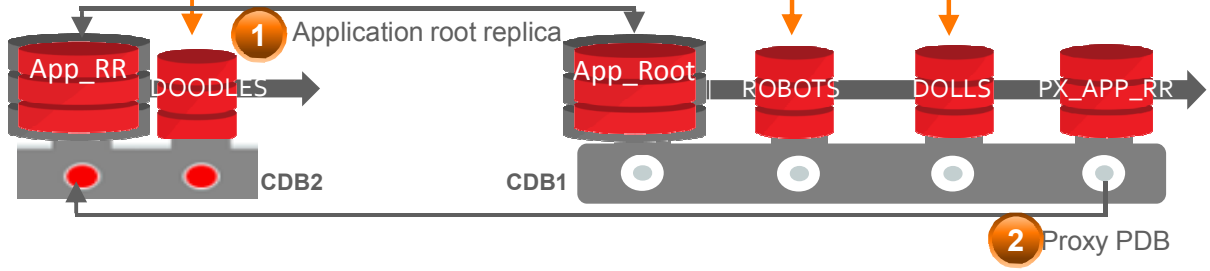
Metadata-linked objects in application containers are not automatically enabled for CONTAINERS(). This can be turned off or on by resetting and setting the CONTAINERS_DEFAULT attribute on the table.

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

```
SELECT sum(revenue), year, CDB$NAME, CON$NAME
FROM   CONTAINERS(sales_data)
WHERE  year =  2014  GROUP  BY year, CDB$NAME, CON$NAME;
```




➔Retrieves all rows from the shared table whose data is stored in all 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 |



In Oracle Database 12.1, `CONTAINERS()` implicitly adds a column named `CON_ID`.

In Oracle Database 12.2, `CONTAINERS()` adds two more implicit columns, `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

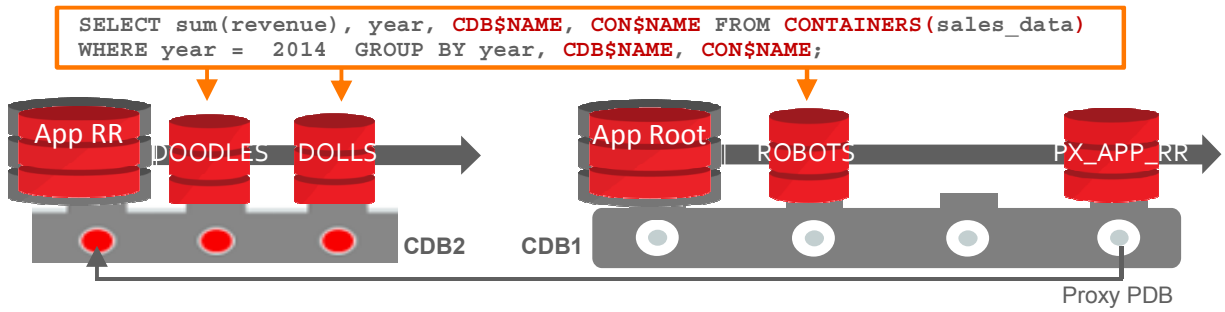
**Note:** Proxy PDB creation is detailed in the lesson titled “Creating Container Databases, Application Containers, and PDBs.”

## 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.




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

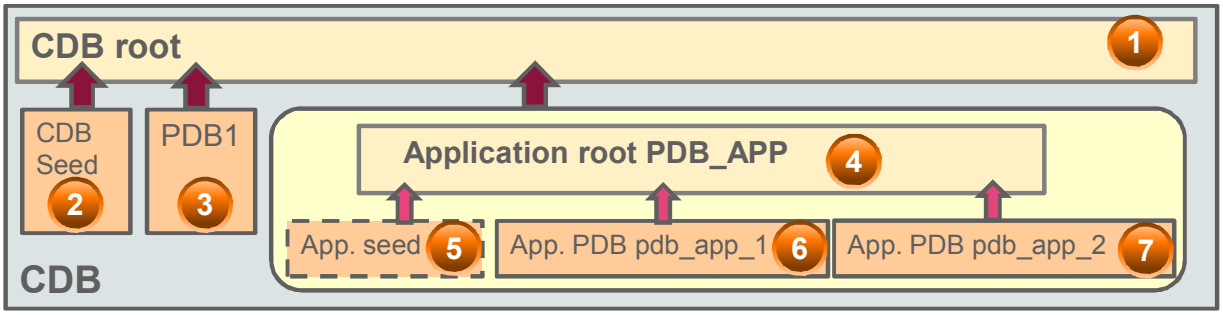


6. To load balance, relocate the `dolls` application PDB from `app_root` to `app_rr`. Relocation implies performing a transparent unplug or plug operation and/or a proxy PDB operation.

   **Note:** PDB relocation and proxy PDB are detailed in the lesson titled “Creating Container Databases, Application Containers, and PDBs.”

Note that 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
```



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

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



The terminology introduced in Oracle Database 12*c* Release 1 for common users, roles, granted privileges and roles, profiles, and tables is still valid in the context of 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.

- In Oracle Database 12*c* Release 1, common objects exist in Oracle-supplied schemas in the CDB root only. In Oracle Database 12*c* Release 2, users can create 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 also 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<br>• BEGIN UPGRADE / END UPGRADE<br>• BEGIN PATCH / END PATCH | Create, alter, or drop a common user. |
| | Create, alter, or drop a common role. |
| | Create, alter, or drop a common profile. |
| | Commonly grant privileges or roles to or revoke them from a common user or common role. |
| | Create, alter, and drop common objects. |



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

- Subset of PDBs in a standby database
- 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
- Heat maps and automatic data optimization (ADO) supported
- 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



- In Oracle Database 12.1.0.1, a standby database must have all the PDBs that the primary database has. With Active Data Guard (ADG) used for reporting purposes, it is not necessary to replicate all the PDBs. Oracle Database 12.1.0.2 introduced the `STANDBYS` clause with two possible values, `ALL` or `NONE`. Oracle Database 12.2 allows the implementation of a subset of PDBs in the standby database by using the `STANDBYS` clause with a list PDBs to be replicated. When a user checks the feasibility of switching over standbys with the subset of PDBs to primary by using the `ALTER DATABASE SWITCHOVER TO …VERIFY` command, the user gets a warning. Such standbys can become primary nevertheless.
  `SQL> CREATE PLUGGABLE DATABASE pdb1 … STANDBYS=(stdby1,stdby2);`
  In this case, `PDB1` will be created on *`stdby1`*, *`stdby2`*.
- In Oracle Database 12.1.0.2, the CDB has a single character set for all containers. In Oracle Database 12.2, 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.

- Automatic Data Optimization (ADO) in Oracle Database 12*c* Release 1 enabled automation of Information Lifecycle Management (ILM) actions, automating movement of data to the appropriate storage format through compression and storage tiering, in non-CDBs only. ADO relies on the statistics reported and collected by Heat Map, a tracking activity at both the segment level as well as the block level. Heat Map and ADO are now extended in Oracle Database 12*c* Release 2 to consolidated databases.
- 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. Oracle Database 12.2 introduces common audit policies 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.
- In Oracle Database Vault 12.1, each PDB has its own Database Vault metadata. Database Vault constructs, such as realms, are isolated within a PDB. Oracle Database 12.2 introduces protection on common objects 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 regular and 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
- Describe enhancements in various areas

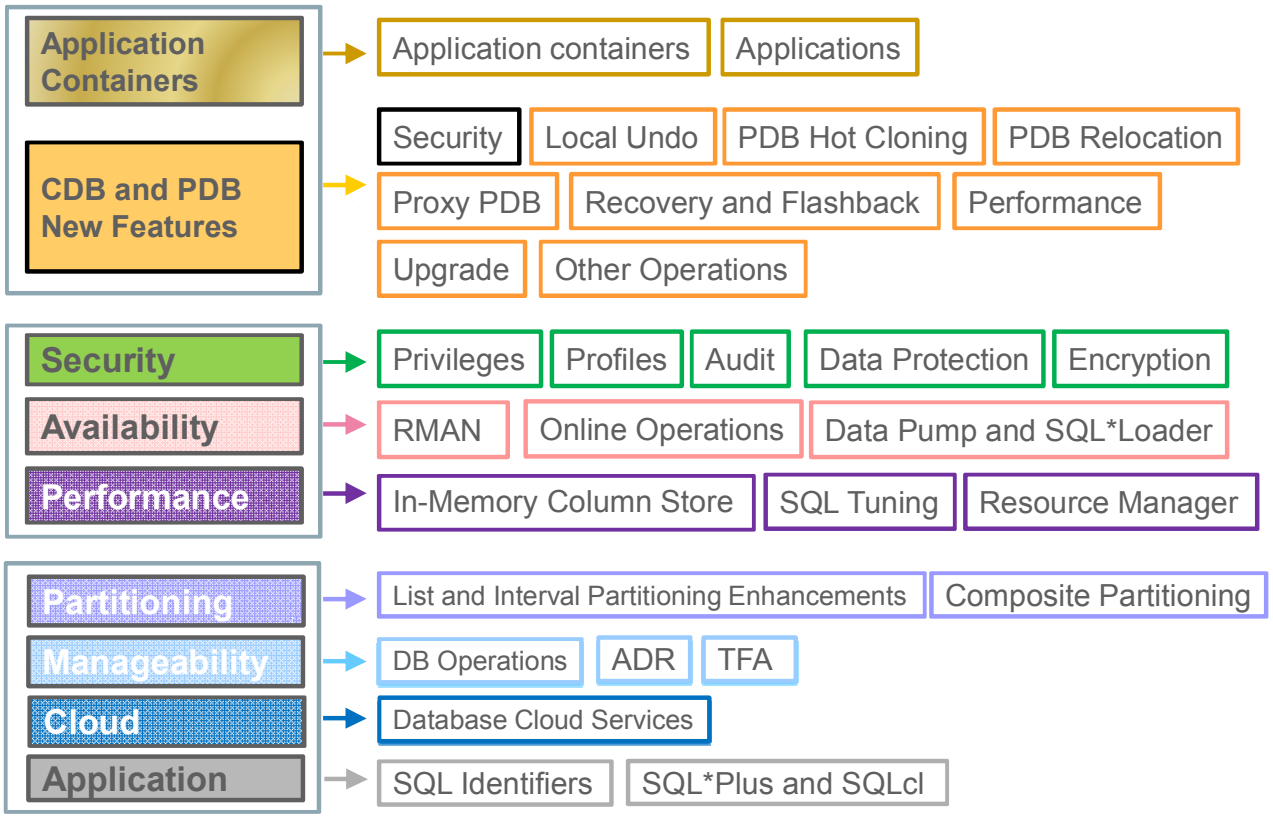
# Practice 2: Overview

- 2-1: Installing an application in an application container
- 2-2: Upgrading an application
- 2-3: Opening and closing application PDBs

3

# Security in CDB, Application Containers, and PDBs

# Security in Application Containers and PDBs



# Objectives

After completing this lesson, you should be able to:

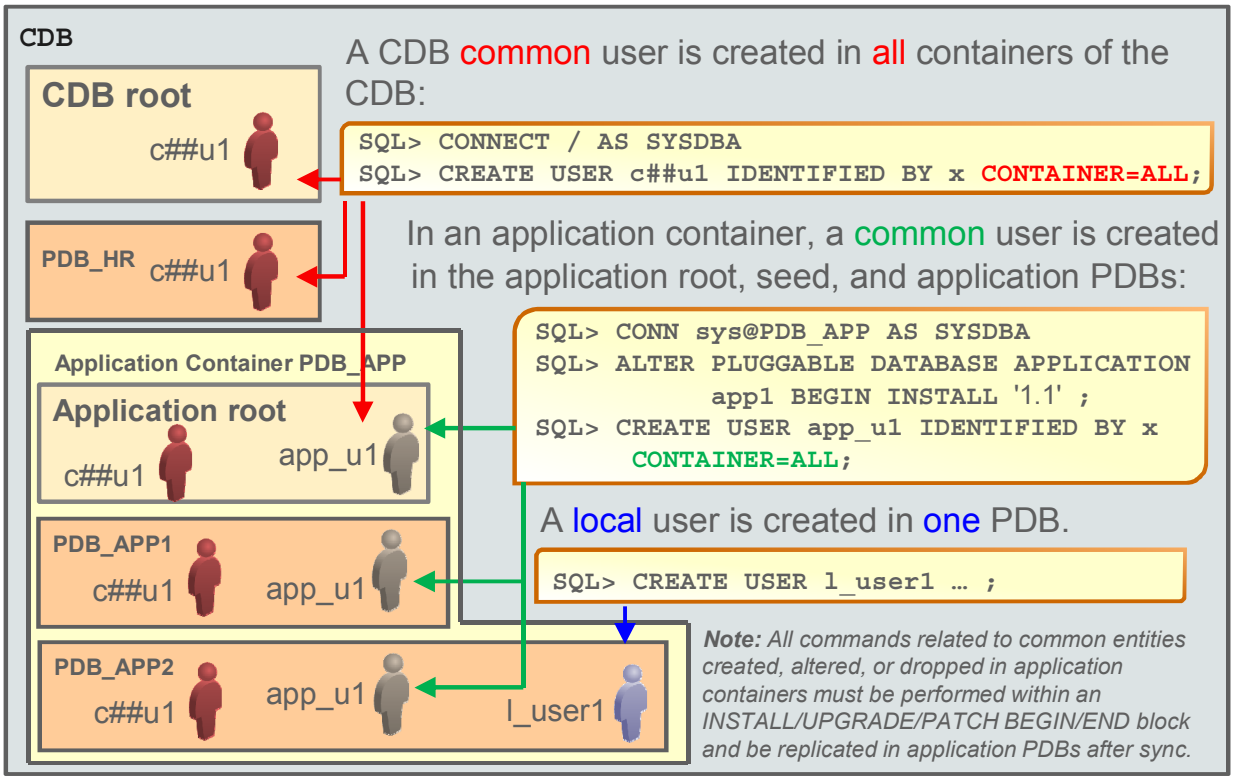
- Manage common and local users, roles, privileges, and profiles in application containers
- Manage common and local objects in application containers
- Manage PDB lockdown profiles
- Audit users in application containers
- Manage other types of policies in application containers
- Use Database Vault policies to protect common objects in application containers
- Allow per-PDB wallets for certificates
- Unplug and plug an encrypted PDB in a one-step operation



For a complete understanding of security in application containers in a Multitenant architecture, refer to the following guides in Oracle documentation:

- *Oracle Database Administrator's Guide 12c Release 2 (12.2)*
- *Oracle Database Security Guide 12c Release 2 (12.2)*
- *Oracle Database Advanced Security Guide 12c Release 2 (12.2)*
- *Oracle Database Vault Administrator's Guide 12c Release 2 (12.2)*

## Creating Common Users in Application Containers






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

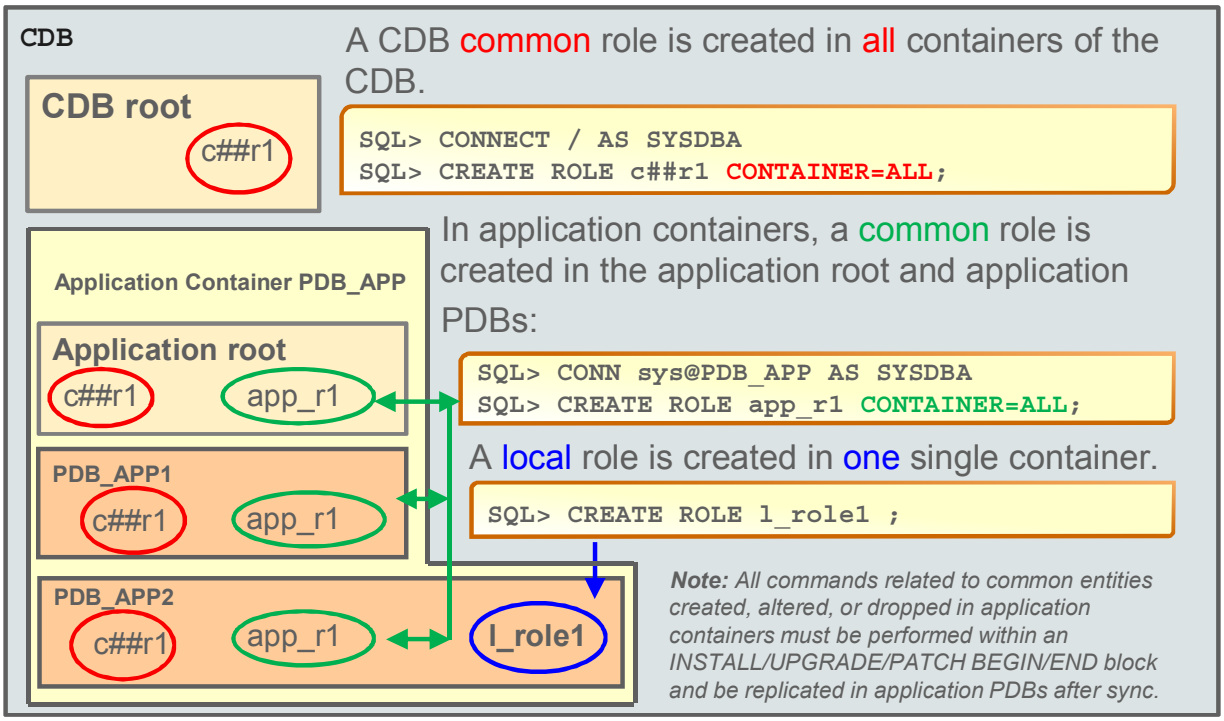
A common user cannot have the same name as any local user across all the PDBs. A common user can be created in the CDB root container or in an application root: a common user is a database user that has the same identity in the CDB root and in every existing and future PDBs in the CDB, or in an application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



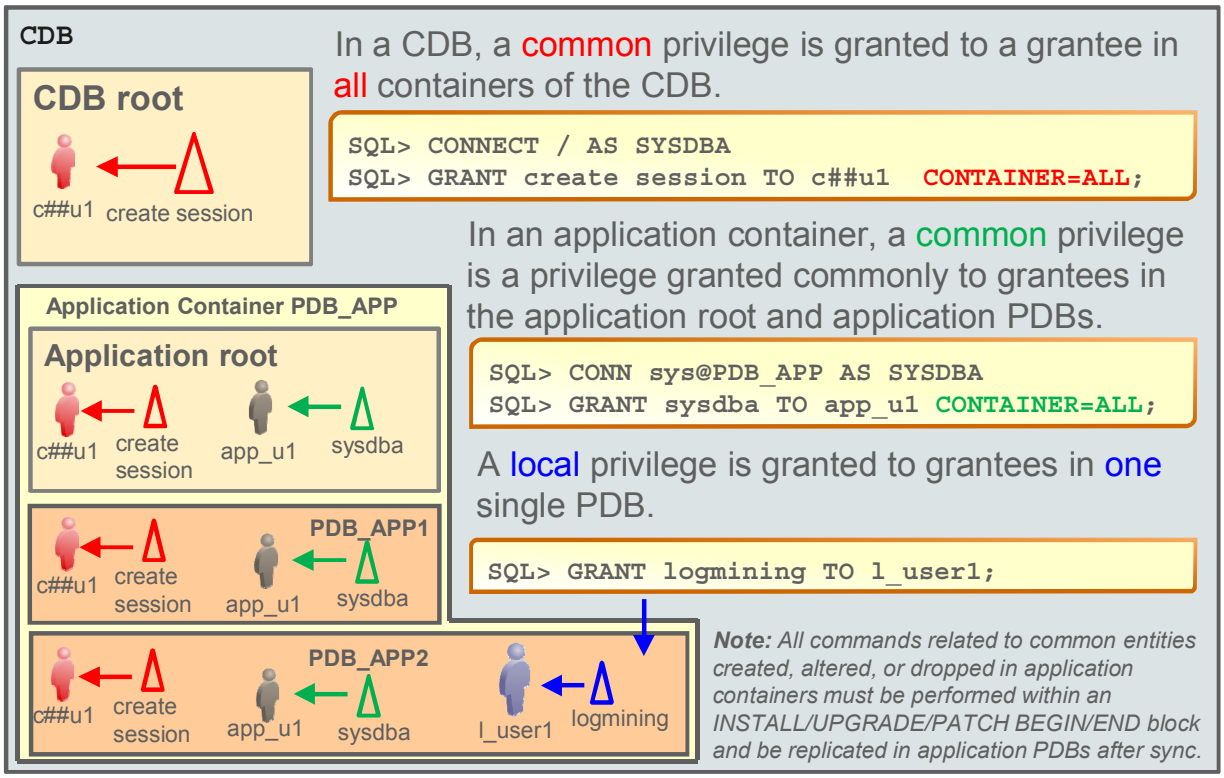



A CDB common role is created from the CDB root and replicated across all PDBs in the CDB. In the example in the slide, the `c##r1` role is created commonly by using the `CONTAINER=ALL` clause. The create operation is replicated in all containers of the CDB. Consequently, the same role `c##r1` is created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

A role, such as `app_r1`, created in an application root is common in the application container when you use the `CONTAINER=ALL` clause. But it 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 Application Containers





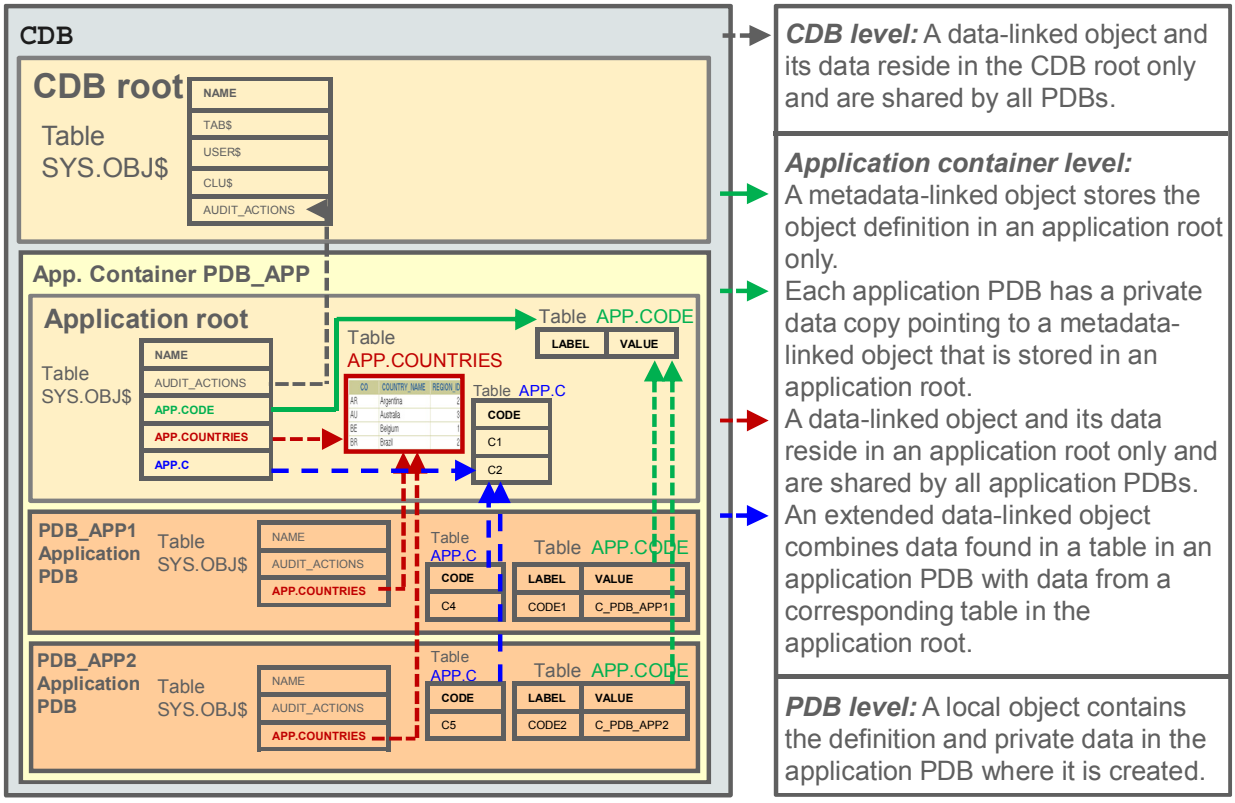

In a CDB, a privilege granted across all containers is a common privilege. In the example in the slide, the `CREATE SESSION` privilege is granted commonly to the `c##u1` user by using the `CONTAINER=ALL` clause. The grant operation is replicated in all containers, including the CDB root where it is initially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

# Common Objects in Application Containers






Common objects are being introduced in Oracle Database 12*c* to avoid having to store redundant representations of data and metadata across a CDB and to simplify the process of upgrading a CDB. In Oracle Database 12.1, common objects exist only in Oracle-supplied schemas and therefore, cannot be user-defined.

Oracle Database 12.2 allows the creation of common objects, metadata-linked or data-linked, 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.

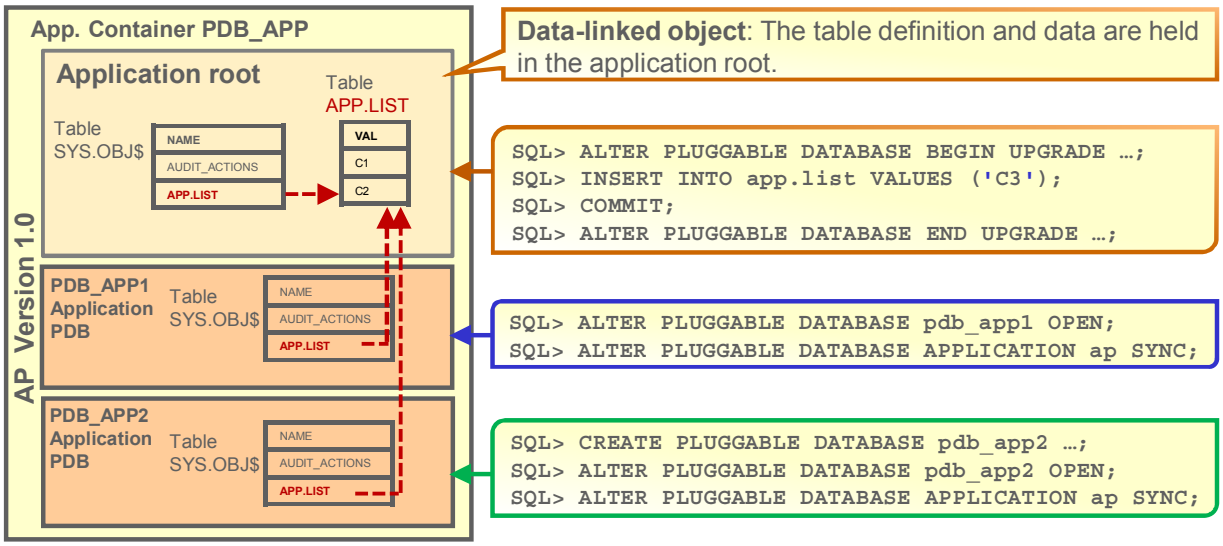
- Metadata-linked objects share the customer-created object definitions that are created in the application root but data is stored in application PDBs.
- Data-linked objects share the data that is stored in the application root that is accessible across all application PDBs. Both the definition and data reside in the application root.
- Extended-data-linked objects share data that is stored in the application root that is 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 new `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:

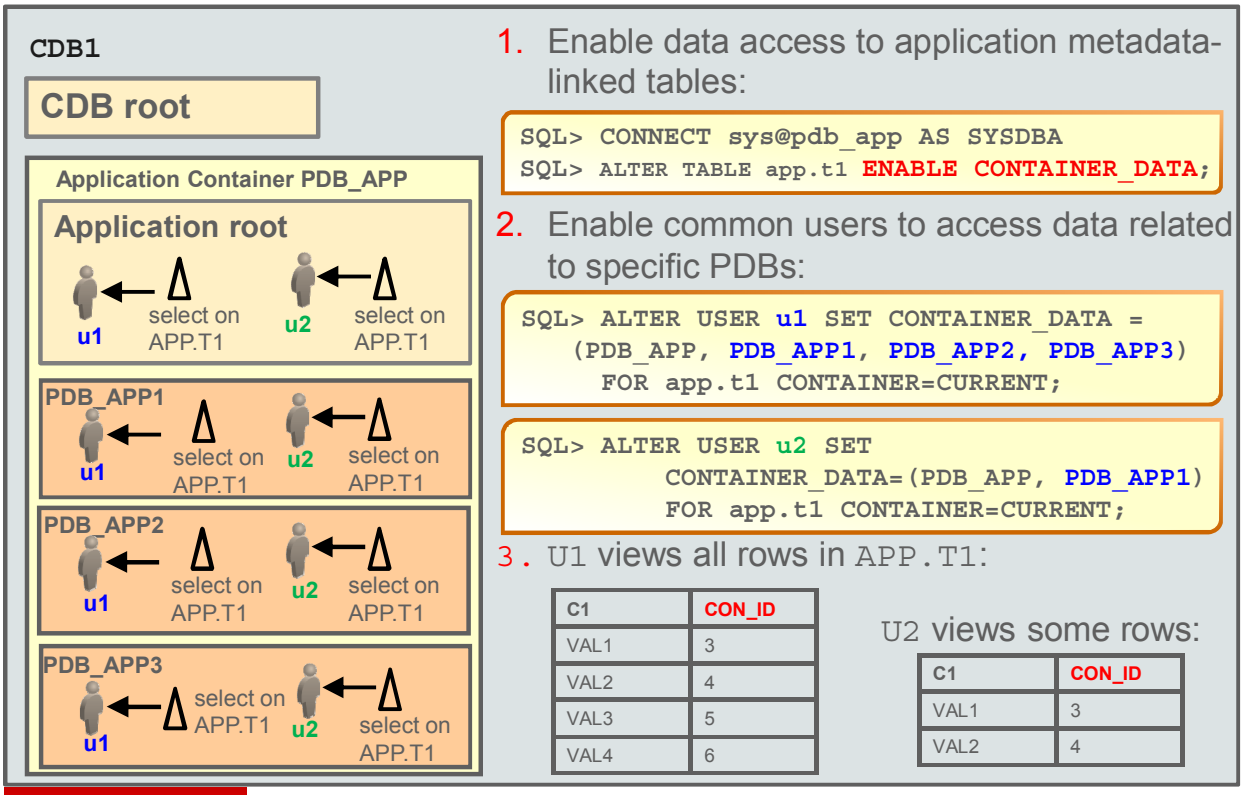
- To new application PDBs
- To PDBs that were closed when the DDL or DML statements were issued






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 Application PDBs





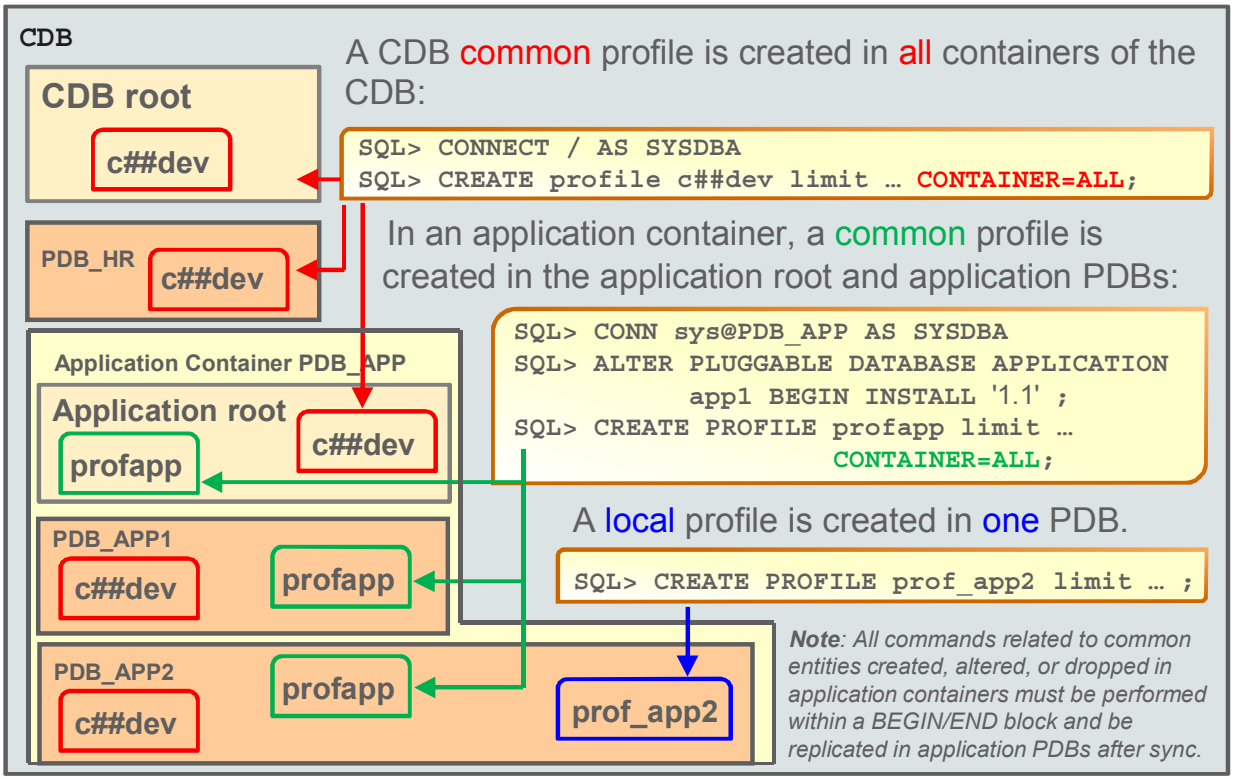

In Oracle Database 12.1, common users can view information about CDB root and about the entire CDB by querying a set of Oracle-supplied views that are created as container data objects. A common user that is connected to the CDB root can view data from the metadata-linked objects pertaining to the PDBs 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.

In Oracle Database 12.2, 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. On the other hand, the common application user, `U1`, can view all rows of the same table, rows pertaining to the application root, and all application PDBs.

## Creating Common Profiles in Application Containers






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.

## Restricting Operations with PDB Lockdown Profile

- 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 clauses.

| STATEMENT | FEATURE | OPTION |
|---|---|---|
| ALTER SYSTEM<br>FLUSH SHARED_POOL, CHECKPOINT, SWITCH LOGFILE, SET | NETWORK_ACCESS<br>UTL_TCP, UTL_SMTP, UTL_HTTP UTL_INADDR, XDB_PROTOCOLS DBMS_DEBUG_JDWP | Partitioning |
| | COMMON_SCHEMA_ACCESS | Advanced Queuing |
| | OS_ACCESS:<br>UTL_FILE, JAVA_OS_ACCESS EXTERNAL_PROCEDURES | Real Application Clusters |
| | XDB_PROTOCOLS | Oracle Data Guard |
| | JAVA, JAVA_RUNTIME | |

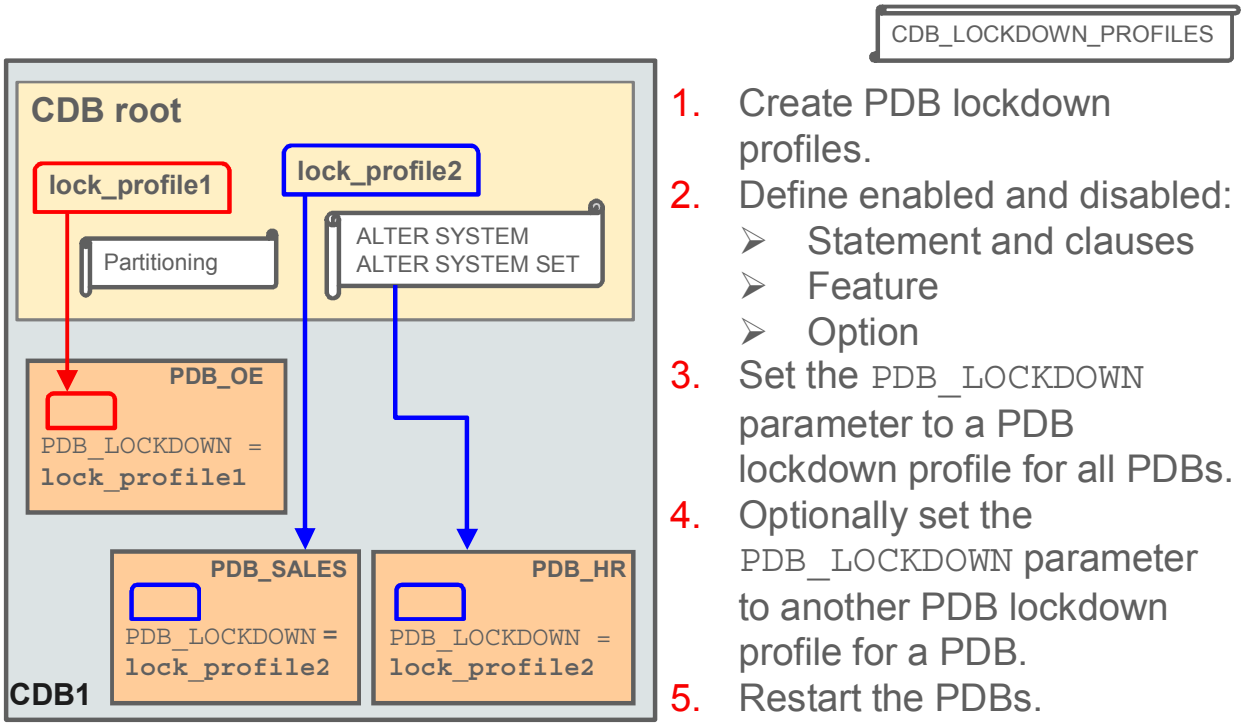


A PDB lockdown profile is a mechanism to restrict users connected to a given PDB or in all PDBs from completing operations such as setting instance parameters or using certain features related to network access or common schema access, or using options such as partitioning.

- **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
  - 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 with PDB Lockdown Profile




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.
5. Restart the PDBs.

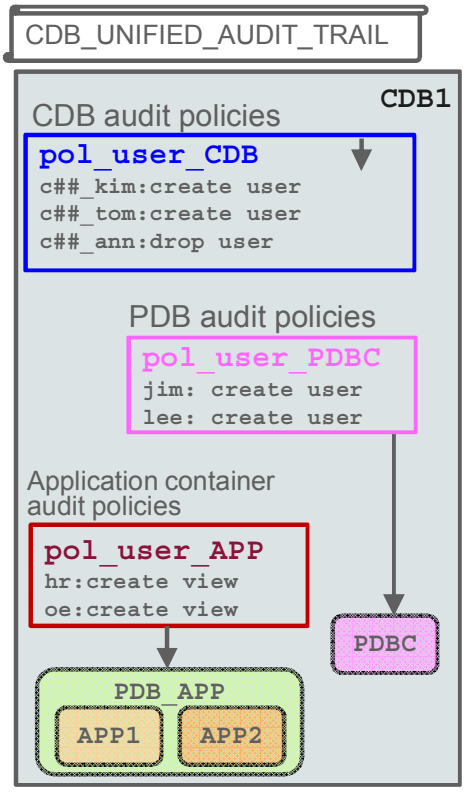


PDB lockdown profiles can be created by users who are granted the `CREATE LOCKDOWN PROFILE` system privilege.

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, the PDB lockdown profile defined at the CDB root level determines the operations that may be performed in all the PDBs.

And after the `PDB_LOCKDOWN` parameter is set, the PDB needs to be bounced before the lockdown profile can take effect.

## Auditing Actions in a CDB and PDBs




1. Connect to the CDB root or to an application root or to a regular PDB.
2. Create common or local audit policies:
   - For all PDBs (*connect to CDB root*)
   - For all application PDBs of an application container (*connect to the application root*)
   - For a regular PDB or a specific application PDB (*connect to the PDB*)
3. Enable/disable audit policies: `AUDIT` and `NOAUDIT`
   - Define users or users being granted roles to be audited (*DBA role*)

ORACLE





Audit policies can be created commonly at the CDB level for the whole CDB or at the application container level for all application PDBs belonging to the application container, or locally at a PDB level for a regular or an application PDB.

Enabled in the CDB root, in the application root, or in a PDB, the 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 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` audit policy that is created in the CDB root audits any `CREATE` or `DROP USER` performed by specific users (`C##_KIM`, `C##_TOM`, and `C##_ANN`).

- The pol_user_APP 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_PDBC audit policy that is created in the regular PDBC 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.

Oracle Database 12.2 introduces a new enablement option for unified audit policies by using database roles. A local 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 audit policy, on the other hand, can be enabled only on common roles, and becomes effective only for common users to whom the common role is granted.

The new clause is BY USERS WITH GRANTED ROLES <*role_list*>. A good example of using the new feature is the predefined role called DBA, which contains most of the system privileges, granted to special privileged users, which might be considered for auditing.

In Oracle Database 12.1, the audit-administrator must enable the 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 has to keep track of such changing auditing requirements and enable the audit policies appropriately for a new set of DBA users. Similarly the users, who no longer have the DBA role granted, should be excluded from auditing to avoid generating unnecessary audit records. This is a tedious and repetitive admin task. The new enhancement allows you to enable an audit policy on the DBA role and make it effective for all users to whom the DBA role is granted.

# 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 |



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 that are 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 Security 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.

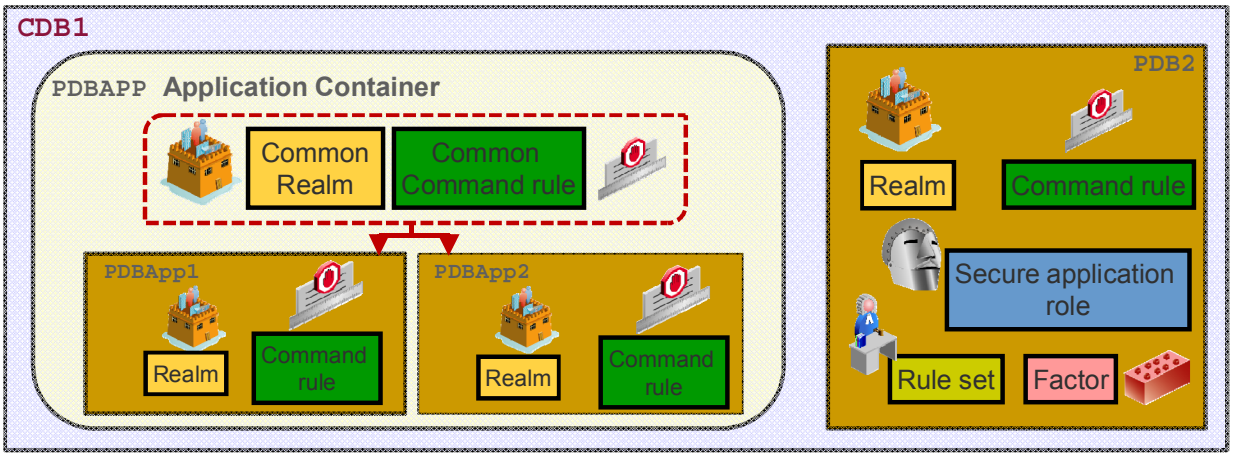
Application-common OLS policies cannot be created in an application root either within or outside of the application install/patch `BEGIN-END` block and therefore, cannot be applied on common objects within application PDBs.

# Securing Data with Oracle Database Vault

DVSYS.DBA_DV_POLICY
DVSYS.DBA_DV_POLICY_OBJECT

- 12.1 Each PDB has its own Database Vault metadata.
- 12.2 DV common protection can protect the common objects of an application container:
  - DV common realm
  - DV common command rule

DV policy (12.2)






In Oracle Database 12.1, each PDB has its own Database Vault metadata. Database Vault (DV) 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 DV in every PDB, you have to create realms and command rules in every PDB separately to protect the common objects. This is costly if there are many PDBs in the CDB that have the same requirement to protect the same common objects.

Oracle Database 12.2 introduces DV 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.

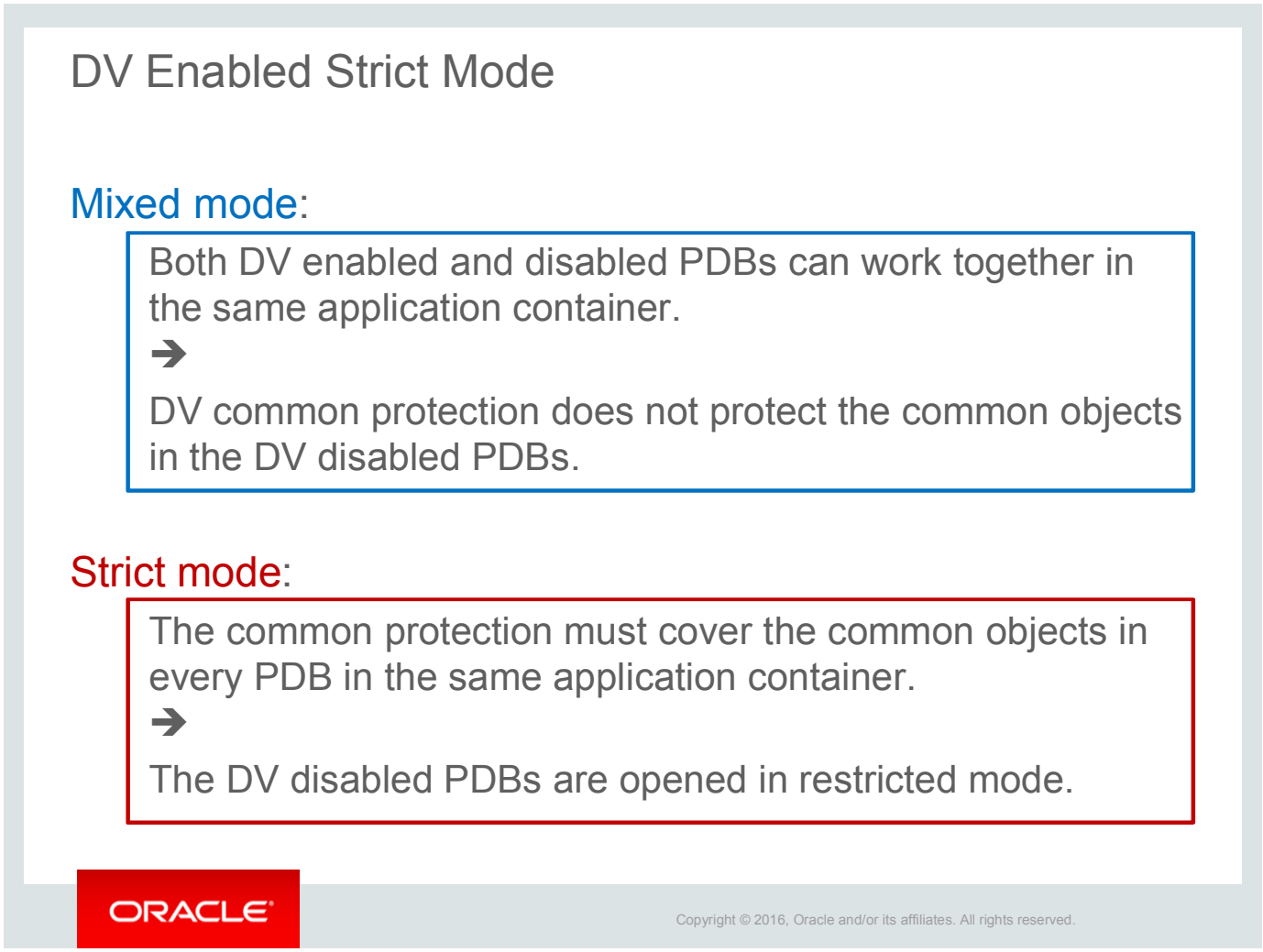
You can use DV policies to group realm and command rule definitions into one policy, which then can be collectively enabled or disabled, or partially enabled, setting the associated realms and command rules to take precedence over the policy. Or you can enable it in simulation mode, writing violations to a designated log table. Common realms can be added to DV policies in application roots only. In application PDBs, you can add local realms to DV policies.

The DV common realm does not protect any local object owned by local users in PDBs. A common DV administrator is a common user who is granted the `DV_OWNER` or `DV_ADMIN` role with `CONTAINER=ALL` in an application root. The DV common realm can be created, updated, and deleted only by the common DV administrator in an application root.

The DV 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 DV 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 12.2 introduces a new behavior for common protection on objects pertaining to application PDBs where some of them are DV enabled and others are DV disabled.

By default, when DV is enabled in the CDB root, the CDB supports mixed mode. Both the DV enabled PDBs and DV disabled PDBs plugged in the CDB function normally.

**Mixed Mode**

By default, when DV is enabled in an application root, the application container supports mixed mode. Both the DV enabled PDBs and DV disabled PDBs plugged in the application container function normally. The DV common protection cannot protect the common objects in DV disabled PDBs because the DV common protection requires that PDBs have DV enabled.

**Strict Mode**

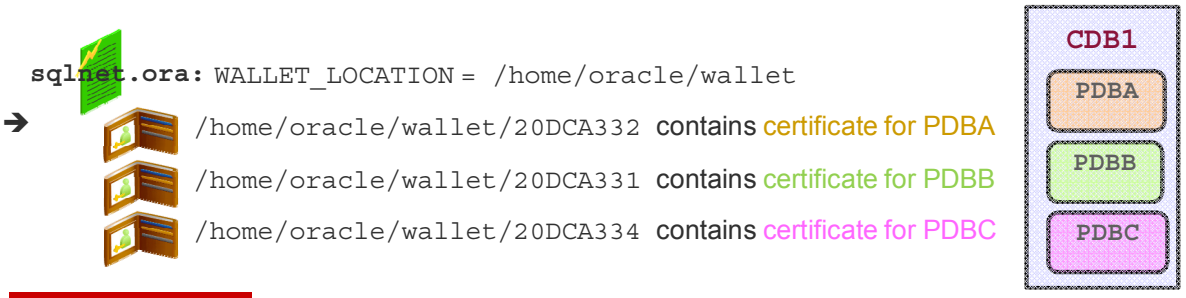
The DV strict mode is a new behavior. It allows you to make the common protection cover common objects in every application PDB. When DV is enabled in strict mode in an application root, the DV disabled PDBs that are plugged in the application container are opened in restricted mode. To make the PDB open normally, DV must be enabled in the PDB, and then the PDB must be restarted.

# Per-PDB Wallet for PDB Certificates

12.1 There is only one `sqlnet.ora` file and one `WALLET_LOCATION` parameter per CDB.

- All PDBs plugged into a CDB must share a common wallet location and, therefore, a common set of keys.
- All PDBs have to share the same identity and allow one PDB to potentially eavesdrop on the communication meant for another PDB.

12.2 Each PDB has its own keystore to store the TLS credentials and identity to communicate with other PDBs.






In Oracle Database 12.2, 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:

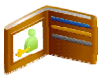
```
$ORACLE_HOME/admin/<db_unique_name>/wallet/<pdb GUID>/
```

# 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";
```

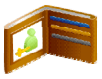
**PDB wallet opened**



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**







In Oracle Database 12.1, an unplugging or a plugging PDB operation requires two operations at each of the two steps: Exporting the Master Encryption Key from the PDB before unplugging the PDB, and then importing the Master Encryption Key into the new plugged PDB after plugging the PDB.

In Oracle Database 12.2, 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.

# Summary

In this lesson, you should have learned how to:

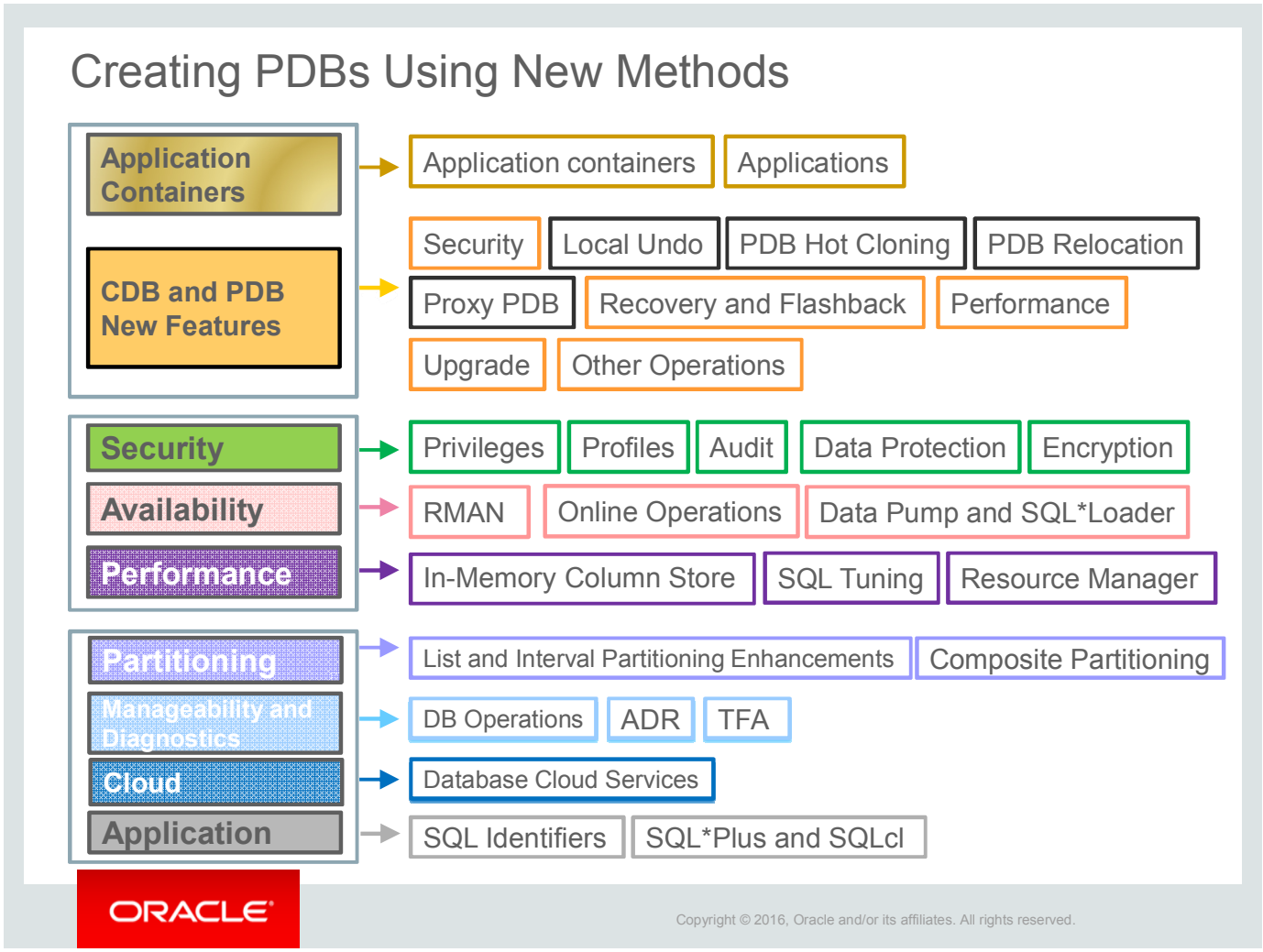
- Manage common and local users, roles, privileges, and profiles in application containers
- Manage common and local objects in application containers
- Manage PDB lockdown profiles
- Audit users in application containers
- Manage other types of policies in application containers
- Use Database Vault to protect common objects in application containers
- Allow per-PDB wallets for certificates
- Unplug and plug an encrypted PDB in a one-step operation

# Practice 3: Overview

- 3-1: Managing common entities and shared data in application containers
- 3-2: Querying data across application PDBs
- 3-3: Applying recorded statements in new application PDBs
- 3-4: Managing PDB lockdown profiles
- 3-5: Auditing operations in PDBs by using Unified Auditing
- 3-6: Protecting application common objects with DV common realms *(Optional)*
- 3-7: Unplugging and plugging encrypted PDBs *(Optional)*

4

# Creation of PDBs Using New Methods




This lesson explains new methods of PDB creation, some of them by using local UNDO tablespaces.

# Objectives

After completing this lesson, you should be able to:

- Explain enhancements in existing methods of PDB creation
- 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 the application PDBs, refreshable PDBs, relocated PDBs, and proxy PDBs



For a complete understanding of the new features and usage of Oracle Pluggable Database, refer to the following guide in Oracle documentation:

- *Oracle Database Administrator's Guide 12c Release 2 (12.2)*

# Enhancements in PDB Creation and Opening

- Cloning with USER_TABLESPACES and COMPATIBLE=12.2
- Cloning metadata only with NO DATA
- Allowing per-PDB character set
- 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');
```





- The USER_TABLESPACES clause introduced in Oracle Database 12.1 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 offlined, 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).
- 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.
- PDBs with different database character sets can exist in the same CDB. 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.
- 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.

# Enhancements in PDB Creation from CDB Seed


- 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;
```



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 recreated in order for it to pick up the new port number for its target PDB.

# Enhancements in PDB Cloning

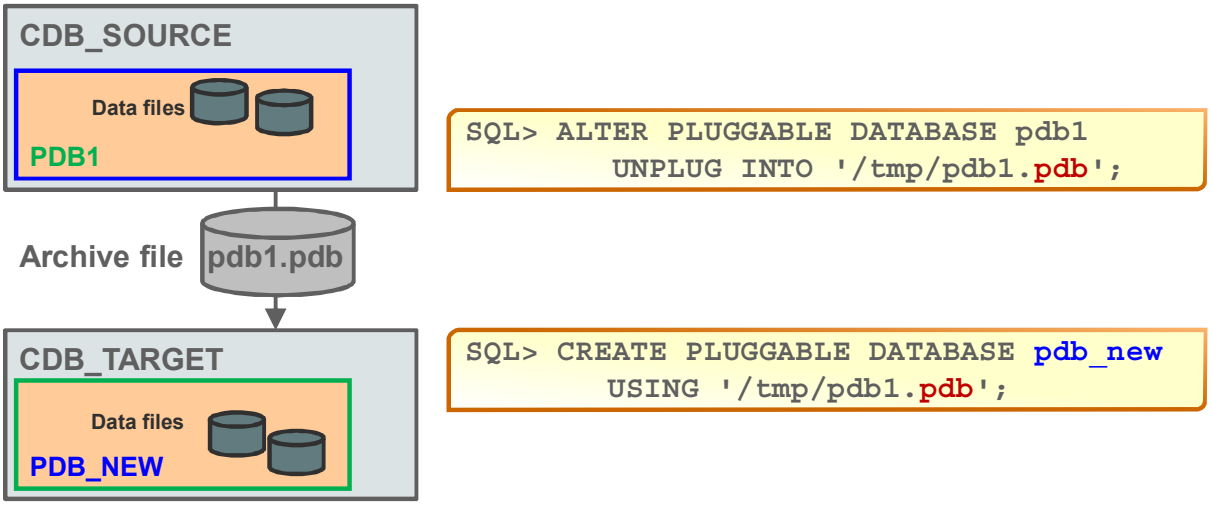
- 12.1 Define the default tablespace of a new PDB created from the CDB seed.
- 12.2 Define the default tablespace of a new PDB during PDB cloning.



Oracle Database 12.2 allows the definition of a default tablespace at PDB creation from another PDB, provided that the tablespace in the source PDB already exists. In Oracle Database 12.1, this clause is allowed only if creating a PDB by using the CDB seed.

# Enhancements in PDB Unplugging and Plugging

- Unplugging a PDB into a single archive file includes:
  - XML file
  - Data files
- Plugging the PDB requires only the archive file.






In Oracle Database 12.1, when a PDB is unplugged, all the data files 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.

With Oracle Database 12.2, 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 data files 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.

## Provisioning Pluggable Databases

Different methods:

- Create a new PDB from the CDB seed.
- Unplug a non-CDB and plug it into a CDB.
- Clone a non-CDB into a CDB. 12.1.0.2
- Unplug a local or remote PDB and plug it into a CDB.
- Clone a PDB from a local or remote PDB 12.1.0.2 in hot mode. 12.2
- Relocate a PDB. 12.2
- Create a proxy PDB. 12.2
- Convert a regular PDB to an application PDB. 12.2

Using the following tools:

- SQL*Plus, SQLcl, SQL Developer
- EM Cloud Control or EM Database Express
- DBCA

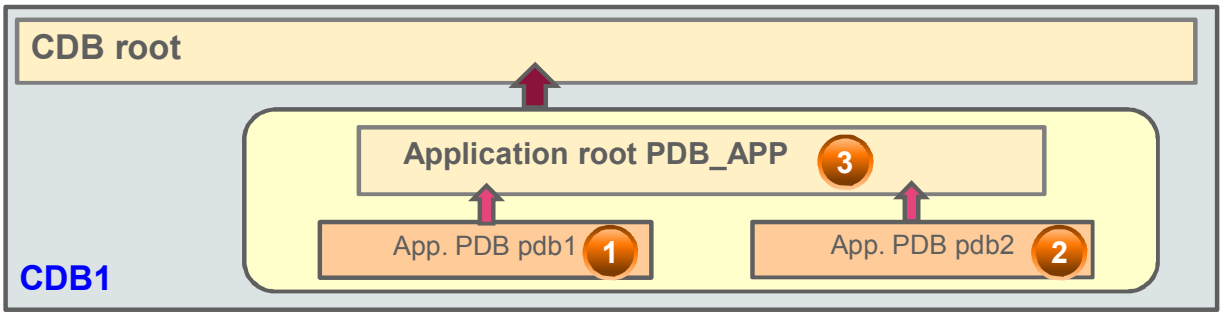


There are different methods to provisioning new PDBs in a CDB. Oracle Database 12.2 introduces new methods for creating new PDBs:

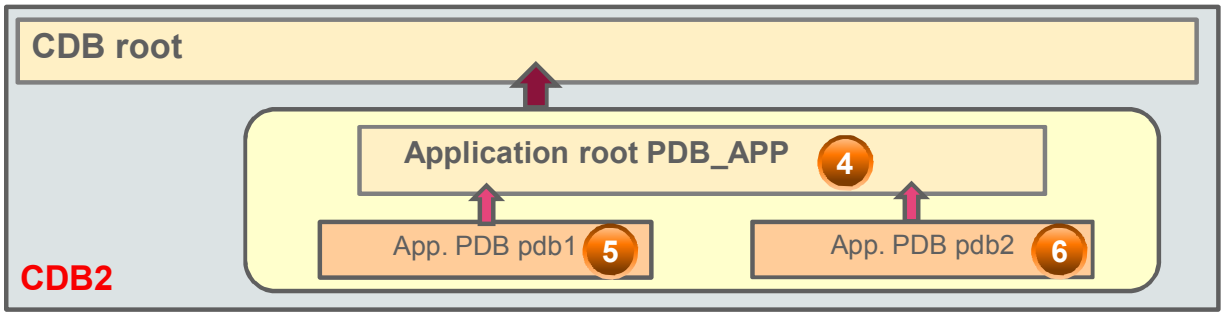
- Hot cloning: This technique copies a remote source PDB into a CDB while the remote source PDB is still up and fully functional.
- Near-zero downtime PDB relocation: A single DDL statement can relocate a PDB from one CDB to another CDB without interrupting ongoing connections.
- Proxy PDB: A proxy PDB allows SQL statements execution in a remote PDB as if it were in the local PDB in the CDB.

Note a new utility, SQLcl, in the tools list. SQLcl is covered in detail in the lesson titled “SQL and SQLcl.”

## Unplugging and Plugging Application PDBs




1. Unplug all application PDBs, and then the application root.
2. Plug the application root, and then all the application PDBs.



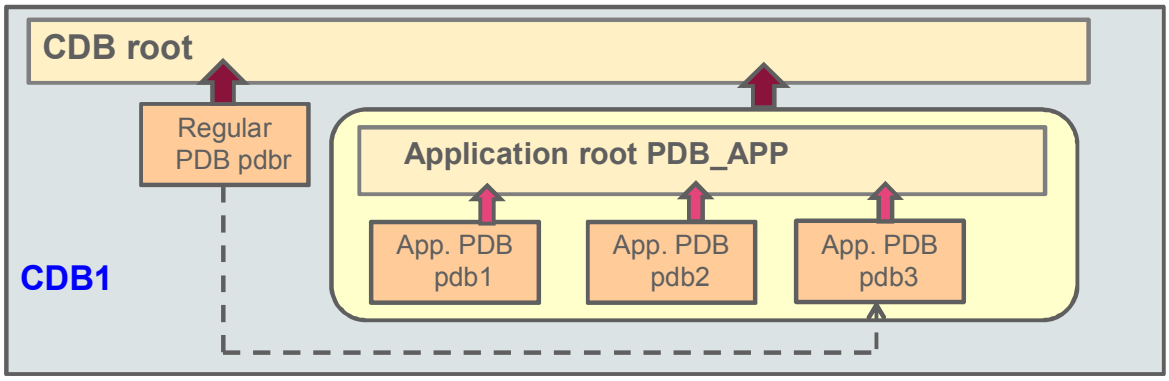



To plug an application container from one CDB to another, proceed with the following sequence:

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.



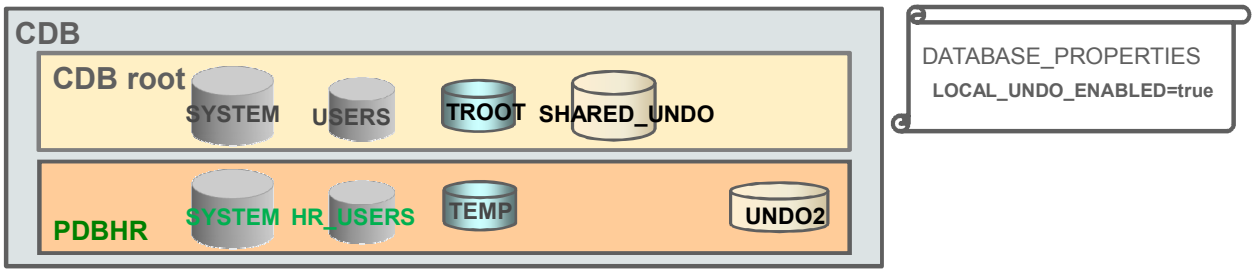
Applications that are already installed in a regular PDB can also take advantage of application containers.

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 the application root.

# 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

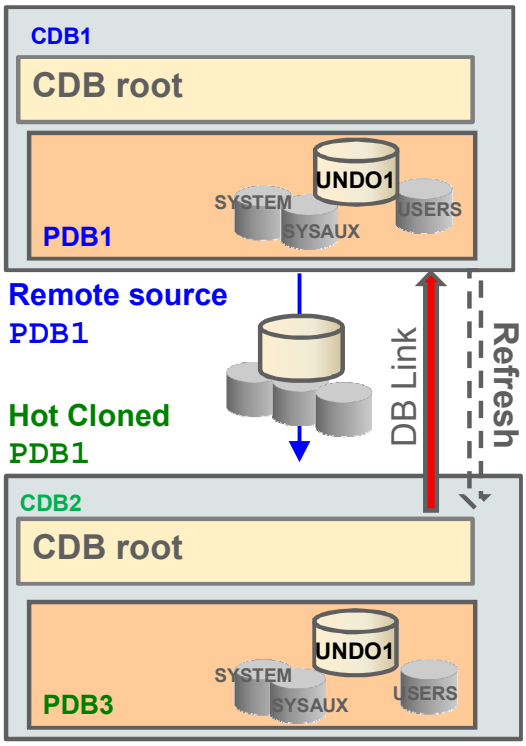


Using the local UNDO mode is required when cloning a PDB in hot mode or performing a near-zero downtime PDB relocation or refreshing PDBs or using proxy PDBs.

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)



Cloning Oracle Database 12.1 PDBs to get a test PDB required the source PDB to be in read-only mode, which was nearly unacceptable on a production PDB. The new Oracle Database 12.2 technique copies a remote source PDB into a CDB while the remote source 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

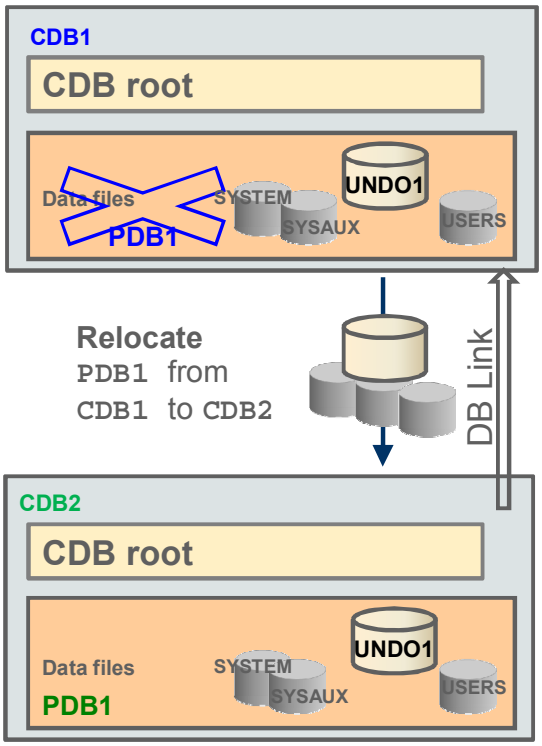
In addition, hot cloning allows incremental refreshing in that the cloned copy of the production database can be refreshed at regular intervals. Incremental refreshing means refreshing an existing clone from a source PDB at a point in time that is more recent than the original clone creation to provide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

## 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 data files to a new location
- Plug the PDB in the target CDB
- Drop the source PDB from the source CDB

ORACLE



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 Oracle Database 12.2 feature such as near-zero downtime PDB relocating.

In Oracle Database 12.1, unplugging and plugging a PDB requires several steps such as unplugging the PDB from the source CDB, copying the database files to a new location, creating the new PDB by plugging the source PDB at the target CDB, and finally dropping the PDB from the source CDB.

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.

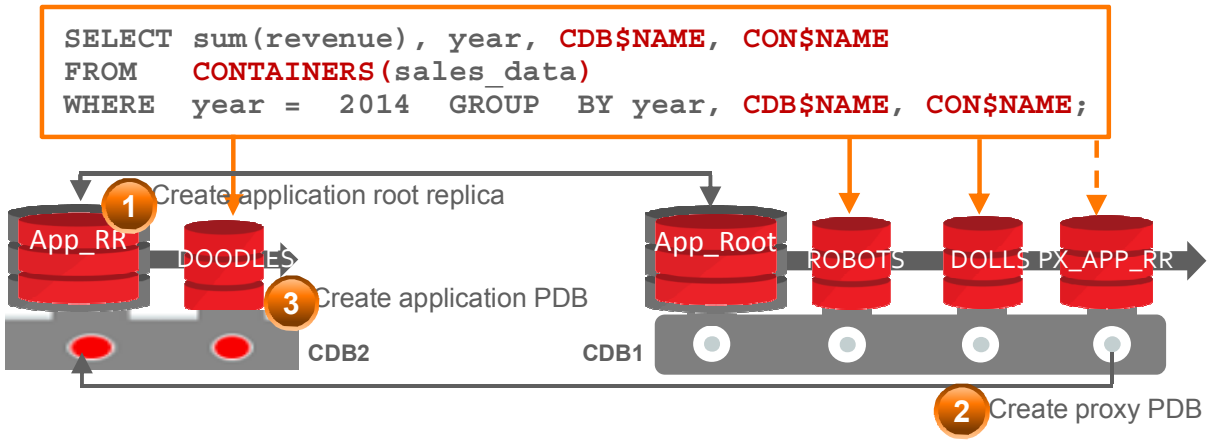
There are two relocation methods:

- 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 |



Oracle Database 12.2 introduces a new type of PDB, the proxy PDB, which allows you to execute SQL statements in a remote PDB as if it were a local PDB in the CDB. This new type of PDB is very helpful to query data spread over PDBs in different CDBs.

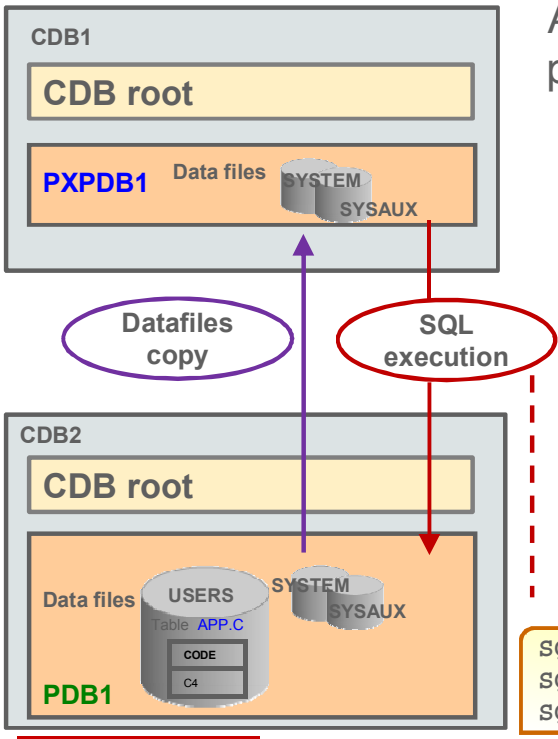
In the example in the slide, when connected to the `toys_root` application root, a query on a table shared in the application PDBs, `robots`, `dolls`, and `doodles` cannot retrieve rows from the remote `doodles` if two conditions are not satisfied:

- 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 the **CDB1** CDB root.
4. Create the **PXPDB1** proxy PDB in **CDB1** as a view referencing the entire proxied **PDB1** in **CDB2**.
5. Execute all the statements in the **PXPDB1** proxy PDB context to have them executed in the proxied **PDB1** PDB in **CDB2**.

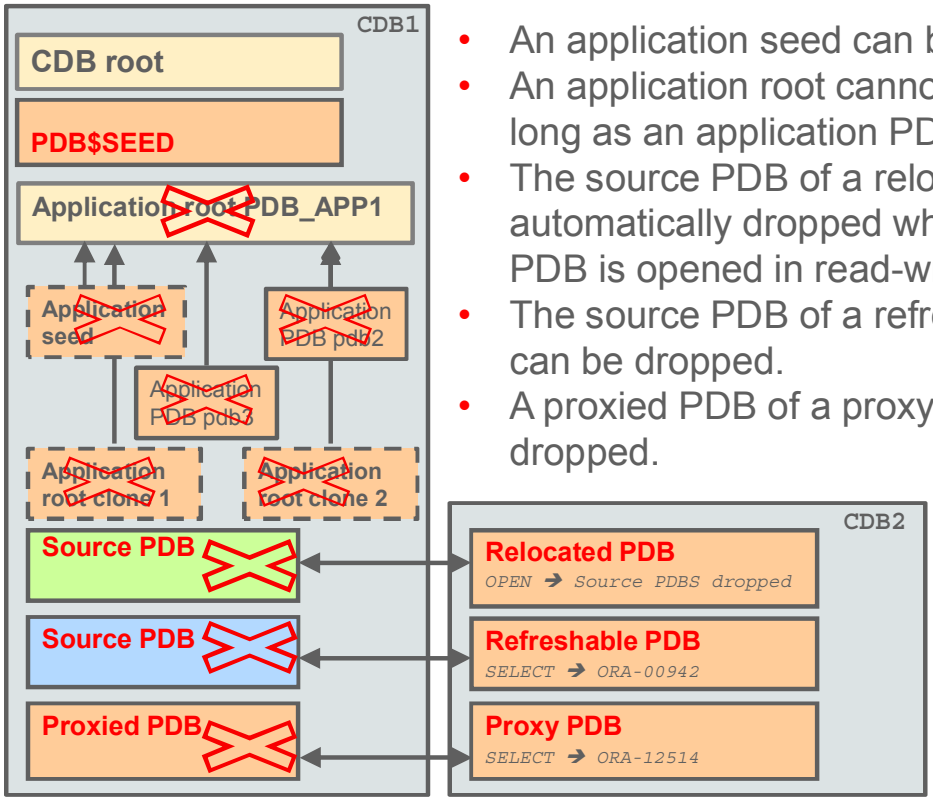
```
SQL> CONNECT sys@pxpdb1 AS SYSDBA
SQL> ALTER PLUGGABLE DATABASE pxpdb1 OPEN;
SQL> SELECT * FROM app.c;
```



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




- 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 read-write mode.
- The source PDB of a refreshable PDB can be dropped.
- A proxied PDB of a proxy PDB can be dropped.

ORACLE



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:

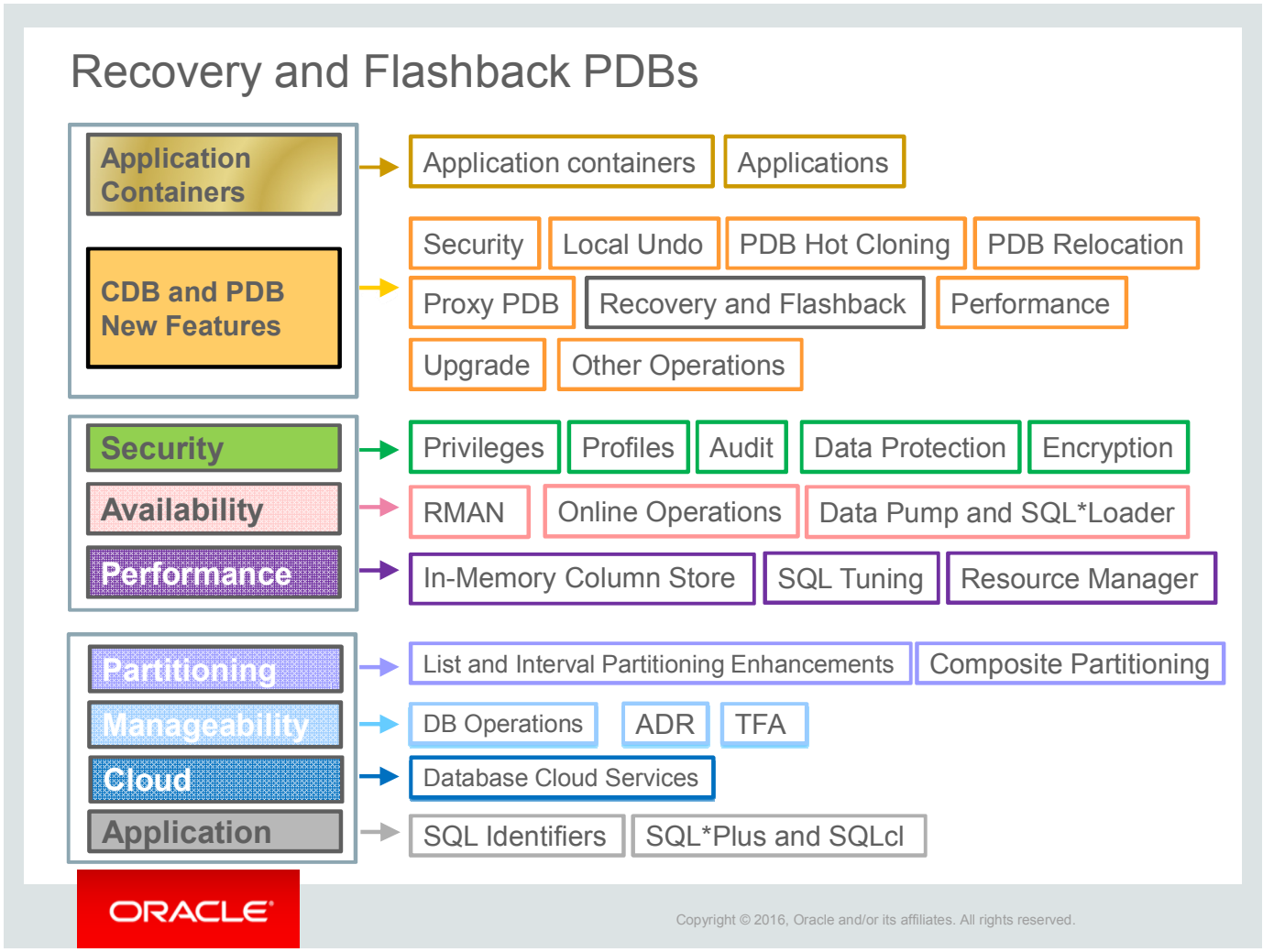
- Explain enhancements in existing methods of PDB creation
- Unplug and plug an application container
- Convert regular PDBs to application PDBs
- Configure and use the local UNDO mode
- Perform hot cloning
- Perform near-zero downtime PDB relocation
- Create and use a proxy PDB
- Drop the application PDBs, refreshable PDBs, relocated PDBs, and proxy PDBs

# Practice 4: Overview

- 4-1: Unplugging and plugging application containers by using archive files (*Optional*)
- 4-2: Renaming services (*Optional*)
- 4-3: Converting regular PDBs to application PDBs (*Optional*)
- 4-4: Cloning remote PDBs in hot mode and automatic refreshing
- 4-5: Relocating PDBs
- 4-6: Querying data across CDBs using proxy PDBs
- 4-7: Dropping unnecessary PDBs

5

# Recovery and Flashback of PDBs




This lesson explains the Oracle Database 12.2 enhancements in recovering and flashing back PDBs.

# Objectives

After completing this lesson, you should be able to:

- Recover a PDB from essential file damage by using a PDB close abort
- Perform PDB flashback
- Use clean restore points to complete PDB flashback



For a complete understanding of new recovery and flashback procedures in Oracle Pluggable Database, refer to the following guides in Oracle documentation:

- *Oracle Database Backup and Recovery User's Guide 12c Release 2 (12.2)*
- *Oracle Database Administrator's Guide 12c Release 2 (12.2)*

## SYSTEM Tablespace PDB 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 data file:

```
$ rman target sys@sales_pdb
RMAN> RESTORE DATABASE;
RMAN> RECOVER DATABASE;
RMAN> ALTER PLUGGABLE DATABASE sales_pdb OPEN;
```



Oracle Database 12.2 introduces the capability to close abort a PDB. 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 data files.

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 data file that is missing or corrupted belongs to a PDB and more specifically to the SYSTEM tablespace, the PDB must be shut down in ABORT mode.

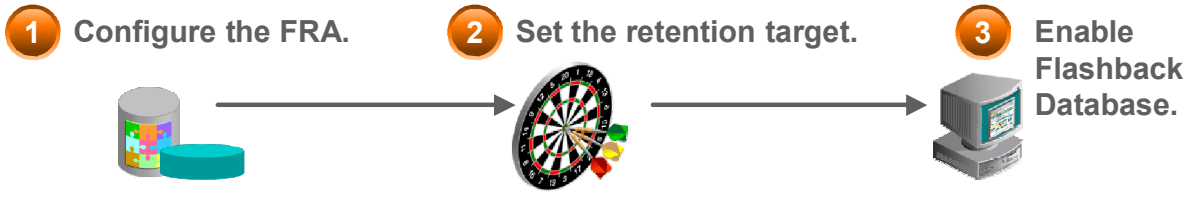
The health checker can automatically detect that there is a severe failure and aborts the PDB.

```
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 data file 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.

# CDB Flashback




```
SQL> SHUTDOWN IMMEDIATE
SQL> STARTUP MOUNT
SQL> ALTER DATABASE ARCHIVELOG;
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 12.1
- PDB flashback similar to CDB flashback 12.2

```
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



## CDB Flashback

In Oracle Database 12.1, you can configure Flashback Database for the CDB as you would for any non-CDB. You cannot flash back the CDB root alone without flashing back the entire CDB.

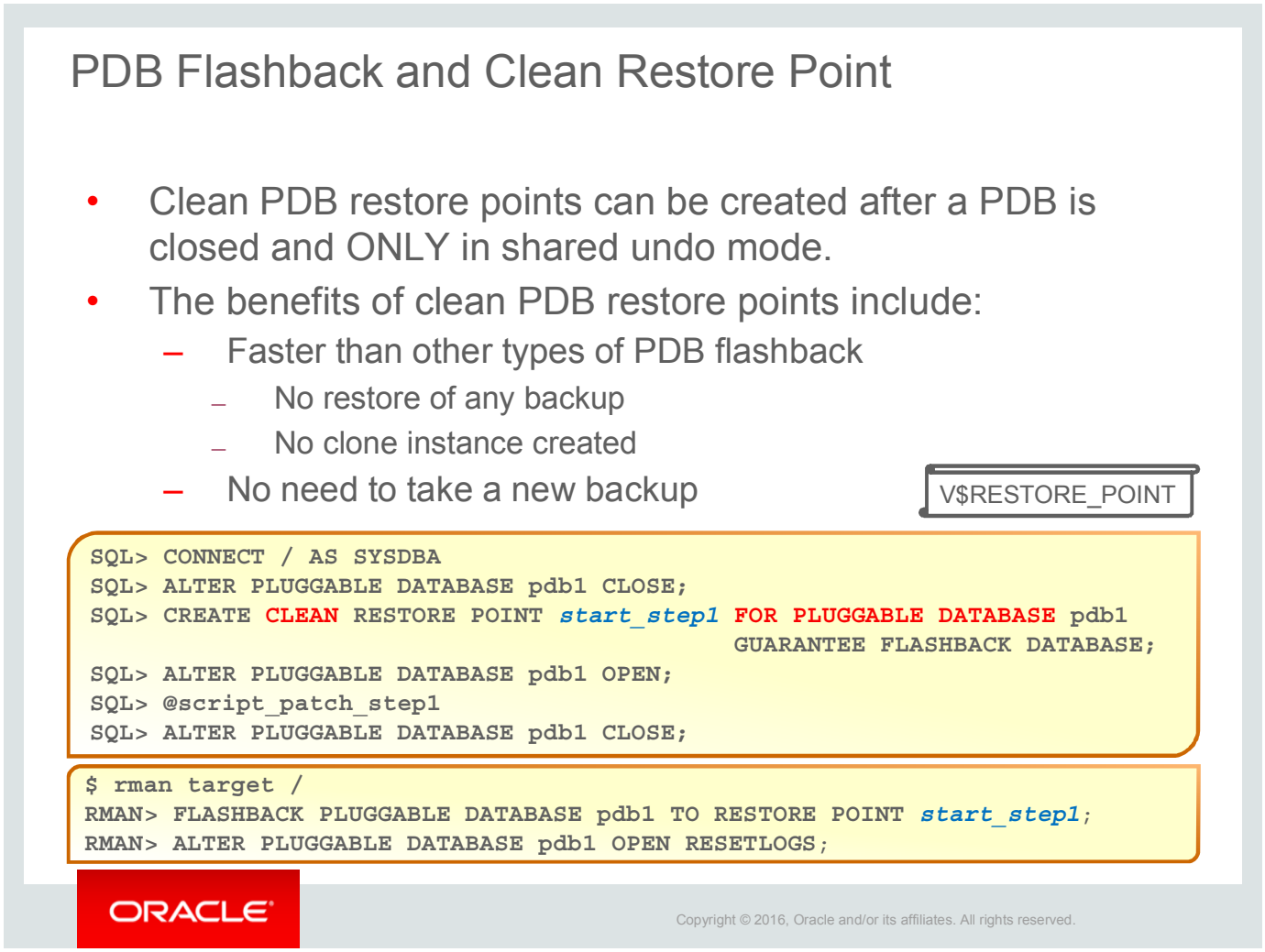
## PDB Flashback

Oracle Database 12.2 introduces Flashback Pluggable Database. All data files 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.




## 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.

# Summary

In this lesson, you should have learned how to:

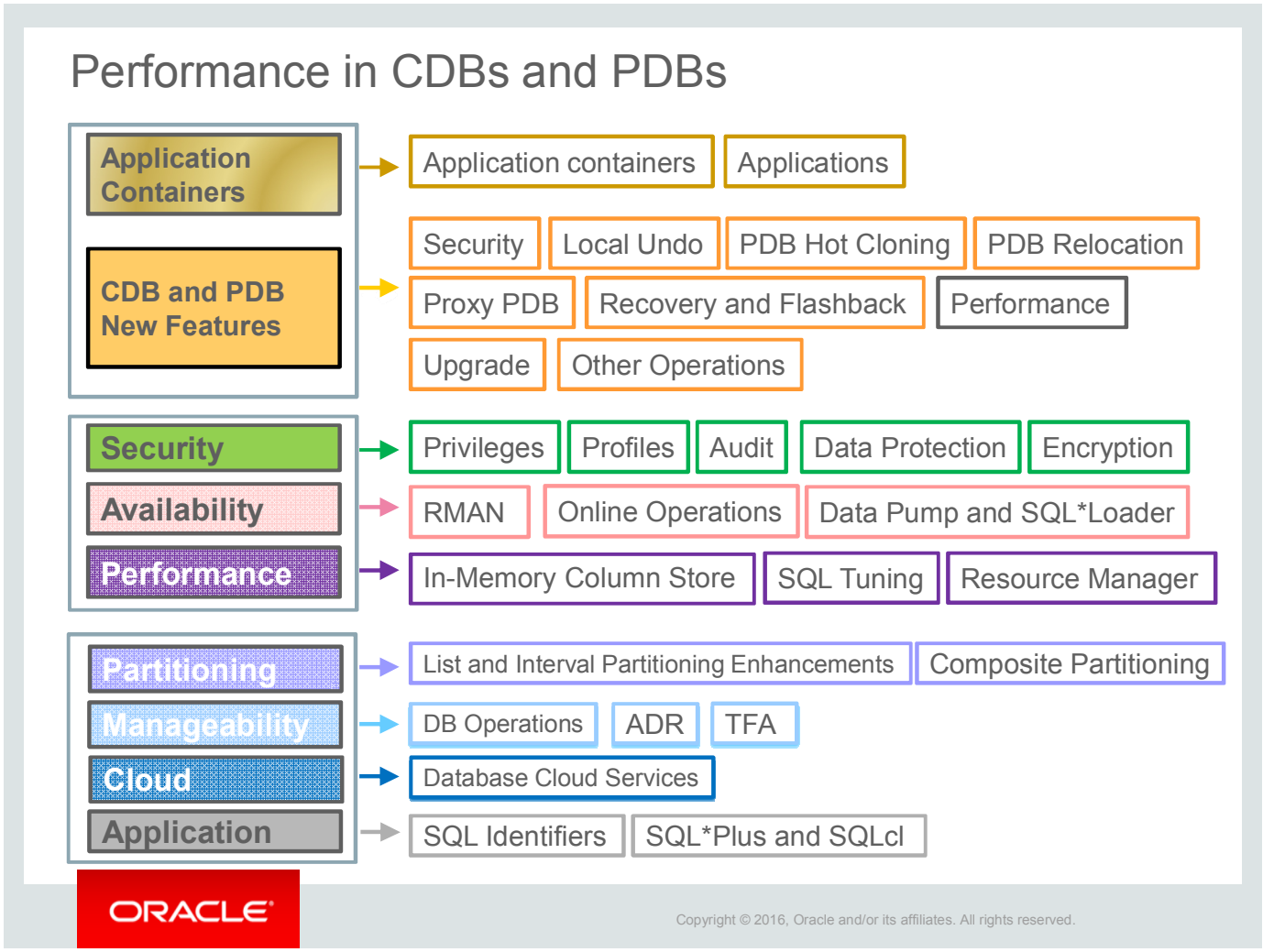
- Recover a PDB from essential file damage by using a PDB close abort
- Perform PDB flashback
- Use clean restore points to complete PDB flashback

# Practice 5: Overview

- 5-1: Recovering from essential PDB datafiles damage
- 5-2: Flashing back an application container from the loss of application common users (*Optional*)
- 5-3: Flashing back an application upgrade using restore points (*Optional*)

6

# Performance in CDBs and PDBs




This lesson explains how to manage AWR snapshots at the CDB and PDB levels, get PDB recommendations from ADDM tasks, allocate memory resources for PDBs, allocate resources to PDBs by using performance profiles, and finally benefit from Heat Map statistics to use Automatic Data Optimization (ADO) policies.

## Objectives

After completing this lesson, you should be able to:

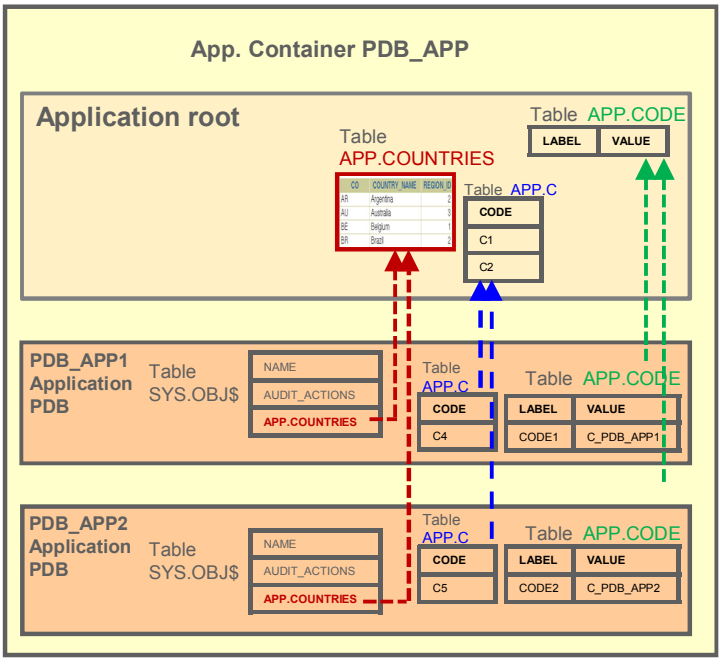
- Manage application shared object statistics
- Control query DOP involving the `containers()` construct
- Manage AWR snapshots at the CDB and PDB levels
- Run ADDM tasks for CDB and PDB recommendations
- Manage SGA and PGA at the PDB level
- Manage resource allocation between PDBs and within a PDB
- Manage Heat Map and ADO policy declaration in a PDB



For a complete understanding of new performance enhancements and Resource Manager features in Oracle Pluggable Database, refer to the following guides in Oracle documentation:

- *Oracle Database Administrator's Guide 12c Release 2 (12.2)*
- *Oracle Database Performance Tuning Guide 12c Release 2 (12.2)*

# 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.



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.

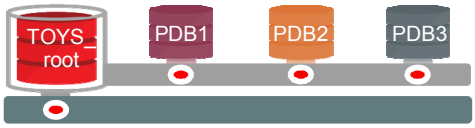
SQL statements are tuned at the PDB level. The SQL Tuning Advisor runs in a specific PDB. SQL Profiles are applied at the PDB level. Object statistics are collected with the `DBMS_STATS` package in the PDB where the object resides.

Oracle Database 12.2 introduces user-defined common objects in application roots. Statistics for user-defined 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



Oracle Database 12.2 introduces a new instance parameter to control 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()`:

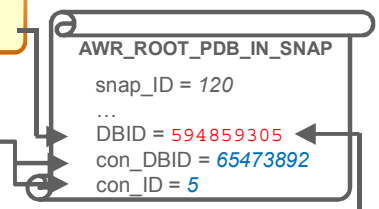
- In the session if the value was set at the session level
- In the instance if the value was set at the system level

# AWR and ADDM Enhancements

- 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;
/
```



Oracle Database 12.2 introduces PDB-level snapshots—collection of statistics that matter at the PDB level—that are created when the PDB is opened. Even if the snapshot is created while being connected to the 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.

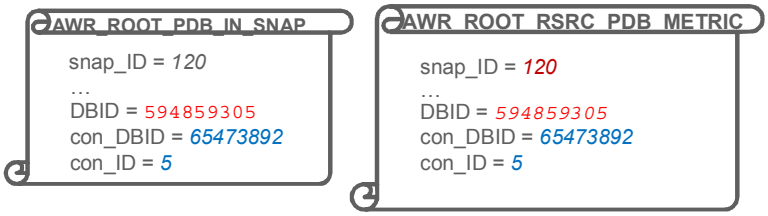
A new view, `AWR_ROOT_PDB_IN_SNAP`, captures the list of opened PDBs at the time of the Automatic Workload Repository (AWR) snapshot creation. The `CON_ID` column displays the ID of the PDB to which the data pertains. The same information is kept in the `DBA_HIST_PDB_IN_SNAP` view.

```
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
```

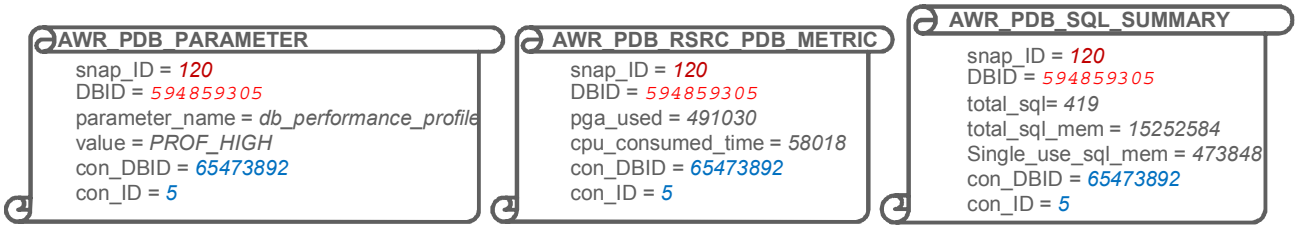
Thus, ADDM tasks can provide recommendations for the CDB and for PDBs as well.

## PDB-Level Snapshot Views

- CDB-level AWR views




- PDB-level AWR views




- List of all PDB-level AWR views

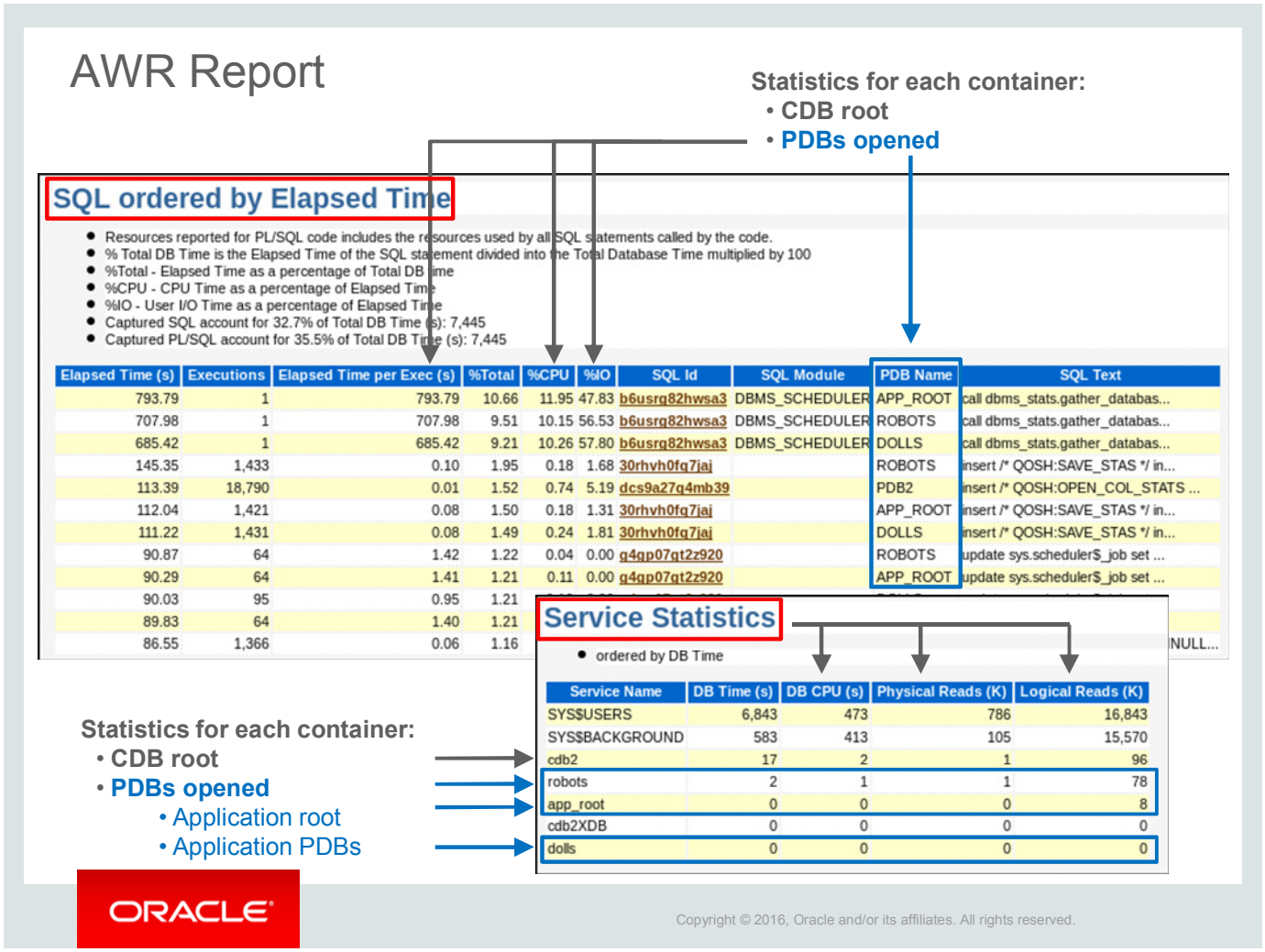
```
SQL> SELECT view_name FROM dba_views WHERE  view_name LIKE '%AWR%_PDB%';
```



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.

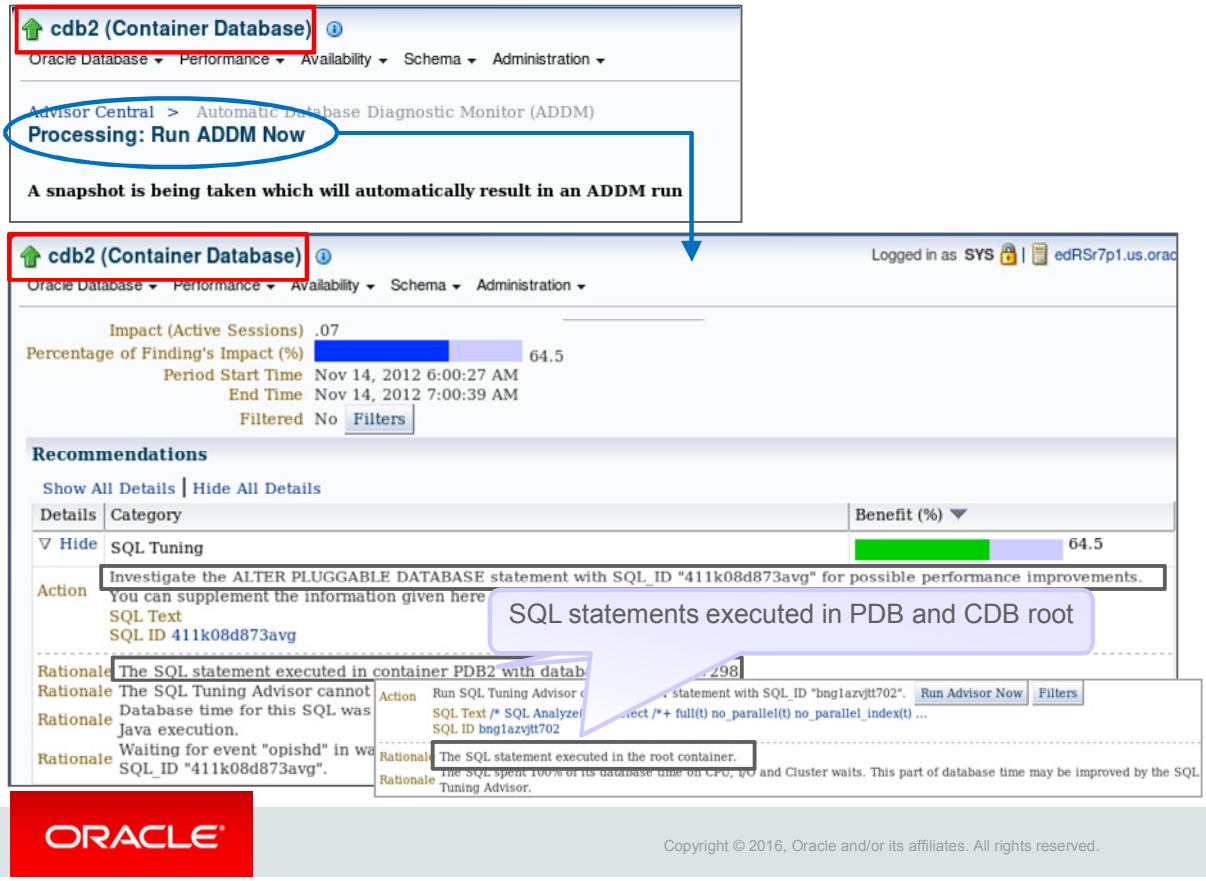



AWR reporting is performed at the CDB level.

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






All ADDM runs must be performed in the CDB root and all ADDM results are stored in the CDB root.

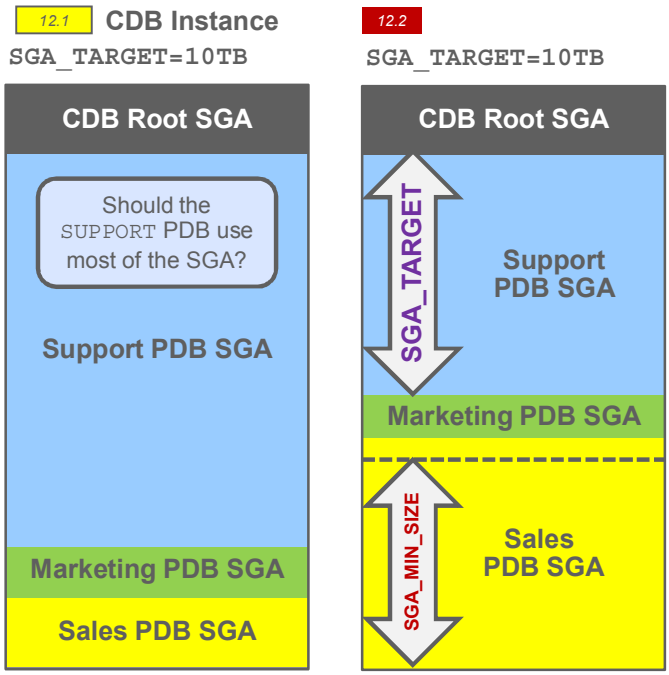
However, ADDM tasks provide recommendations on statements executed in any container—in the CDB root, in application roots, in regular PDBs, and in application PDBs. However, ADDM results cannot be viewed when the current container is a PDB.

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.

**Note:** ASH, ASH Analytics, SQL Tuning Advisor, and Segment Advisor still work at the CDB level as they do in Oracle Database 12.1 Release.

# Managing SGA for PDBs




- 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
- Σ(PDB minimums) cannot be > 50% of memory.



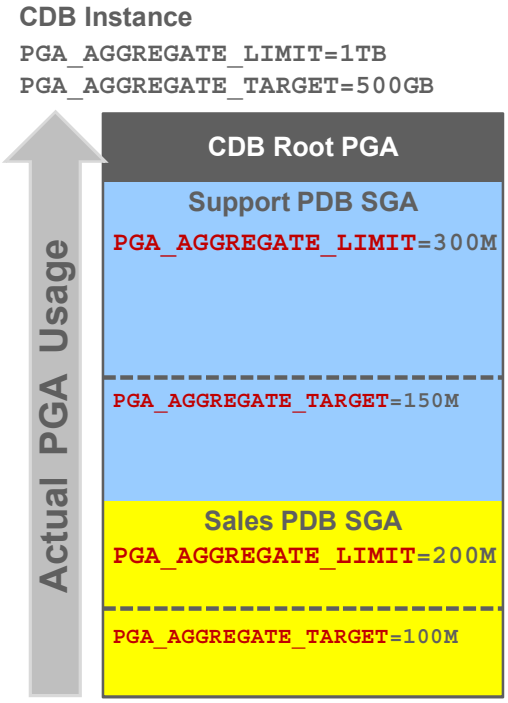


In a CDB, there is one SGA allocation for the instance, which is shared by all containers, the CDB root, and all PDBs. Most of the SGA is a cache that favors frequently accessed objects in the buffer cache, the shared pool, and the In-Memory column store.
In Oracle Database 12.1, active PDBs dominate the space in the SGA cache. In the example in the slide, 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.
Oracle Database 12.2 introduces SGA and PGA memory management 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 nor 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% 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




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 12.2

PDB PGA_AGGREGATE_TARGET 12.2

- These parameters set the same behavior at the PDB level.

ORACLE

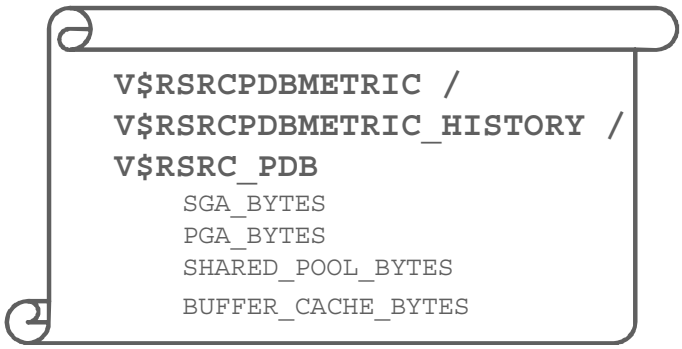


In Oracle Database 12.1, PGA_AGGREGATE_TARGET specifies the target aggregate PGA memory that is available to all server processes attached to the instance. To set a hard limit for aggregate PGA memory, use the PGA_AGGREGATE_LIMIT parameter. PGA_AGGREGATE_LIMIT is set by default to the greater of 2 GB, 200% of PGA_AGGREGATE_TARGET, and 3 MB times the PROCESSES parameter. It is set below 200% of PGA_AGGREGATE_TARGET if it is larger than 90% of the physical memory size minus the total SGA size, but not below 100% of PGA_AGGREGATE_TARGET.

Oracle Database 12.2 allows the use of these parameters at the PDB level.

# Monitoring PDB Memory Usage

- Monitor memory usage before and after configuring PDB memory parameters.




- Monitor per PDB history statistics.






Several views have been added to allow you to monitor the memory and Resource Manager operations at the PDB level.

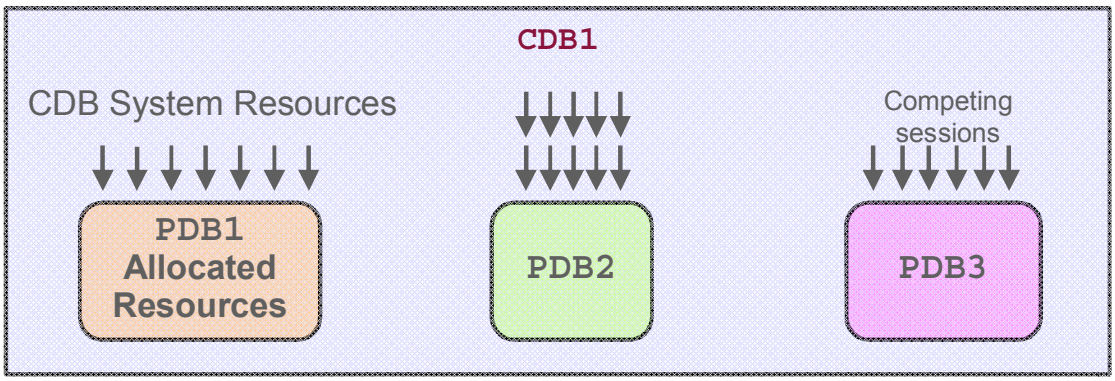
Statistics are collected every minute into `V$RSRCPDBMETRIC` and `V$RSRCPDBMETRIC_HISTORY`, showing PDB memory and Resource Manager statistics rolled up to the PDB level. This is different from the `V$RSRCMGRMETRIC` view that shows statistics at the consumer group level.

All these views include information that allows you to monitor different aspects of SGA and PGA memory allocations by PDB.

## Resource Manager and Pluggable Databases

In a CDB, the Resource Manager manages resources:

- Between PDBs
- Within each PDB






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.

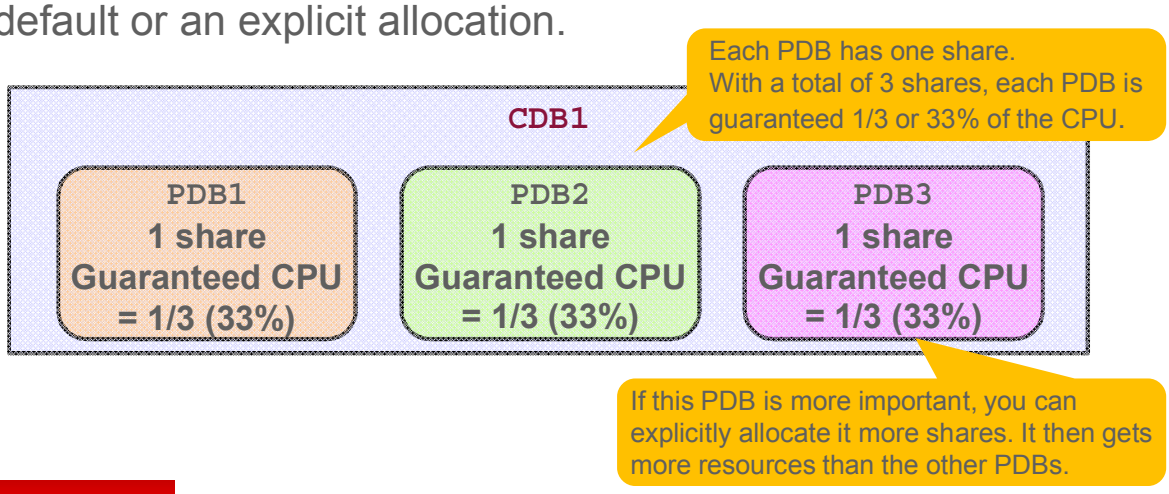
The Resource Manager allocates resources in two steps:

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

12.1

- 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



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 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% 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% of the system resources.

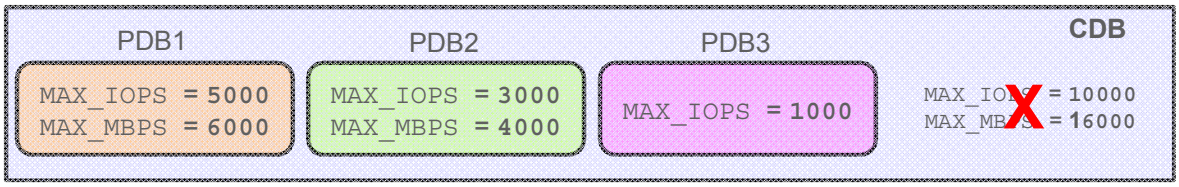
Depending on the workload, each PDB can get up to 100% of the system resources but may actually use much less than the guaranteed level of these resources.

**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.

# PDB IO Rate Limit

12.2

- 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



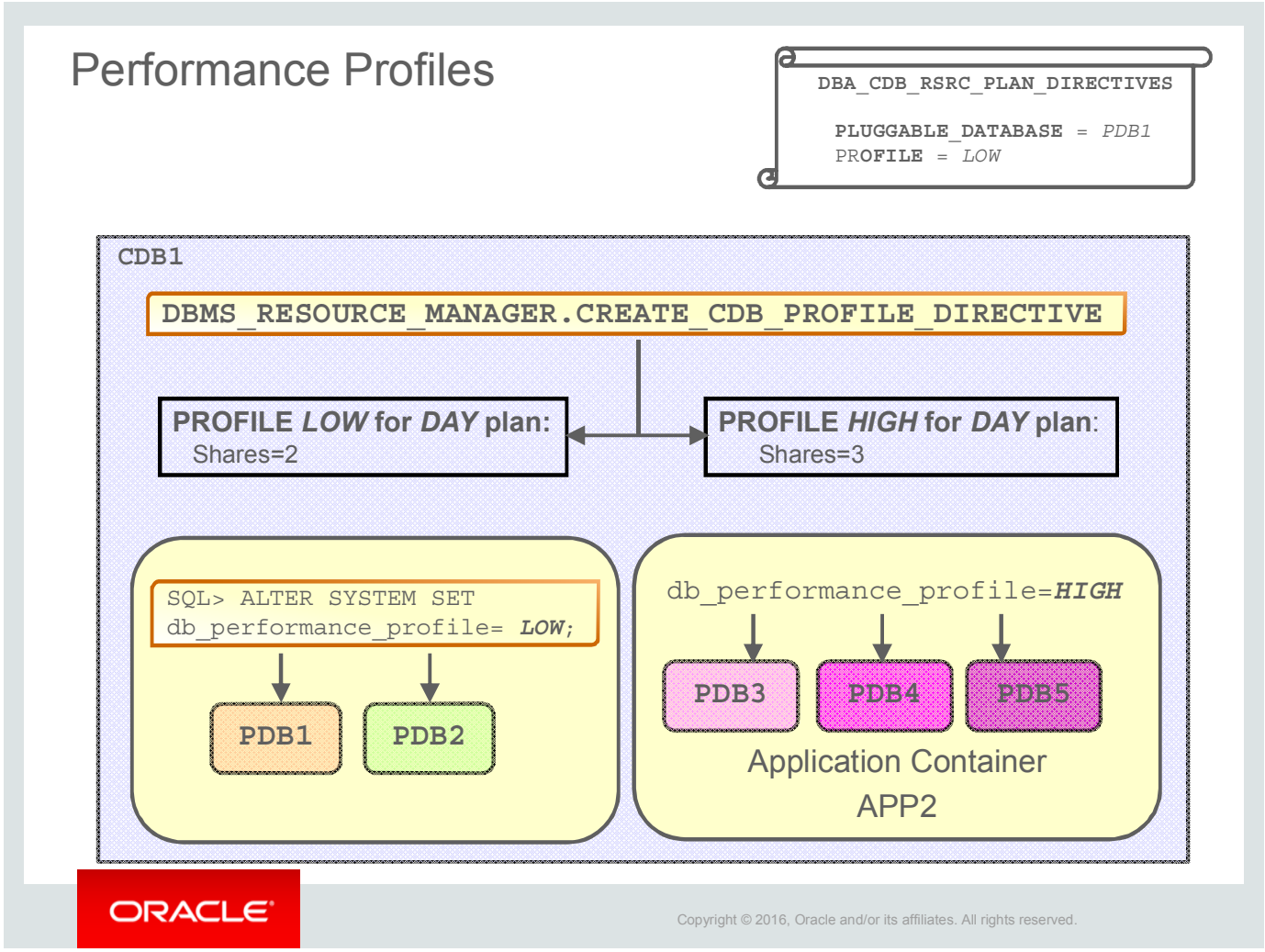
Managing the IO issued per PDB can be important in consolidation environments. The PDB IO rate limits set limits on the IOs that are issued in a CDB on a per PDB basis. Two dynamic parameters are introduced in Oracle Database 12.2 that can be set from within a PDB, and can be changed dynamically as required:

- 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.




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.

## Heat Map and ADO Enhancements

Oracle Database 12.2 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
```



In Oracle Database 12.1.0.1, Automatic Data Optimization (ADO) policies automatically execute actions under predefined conditions:

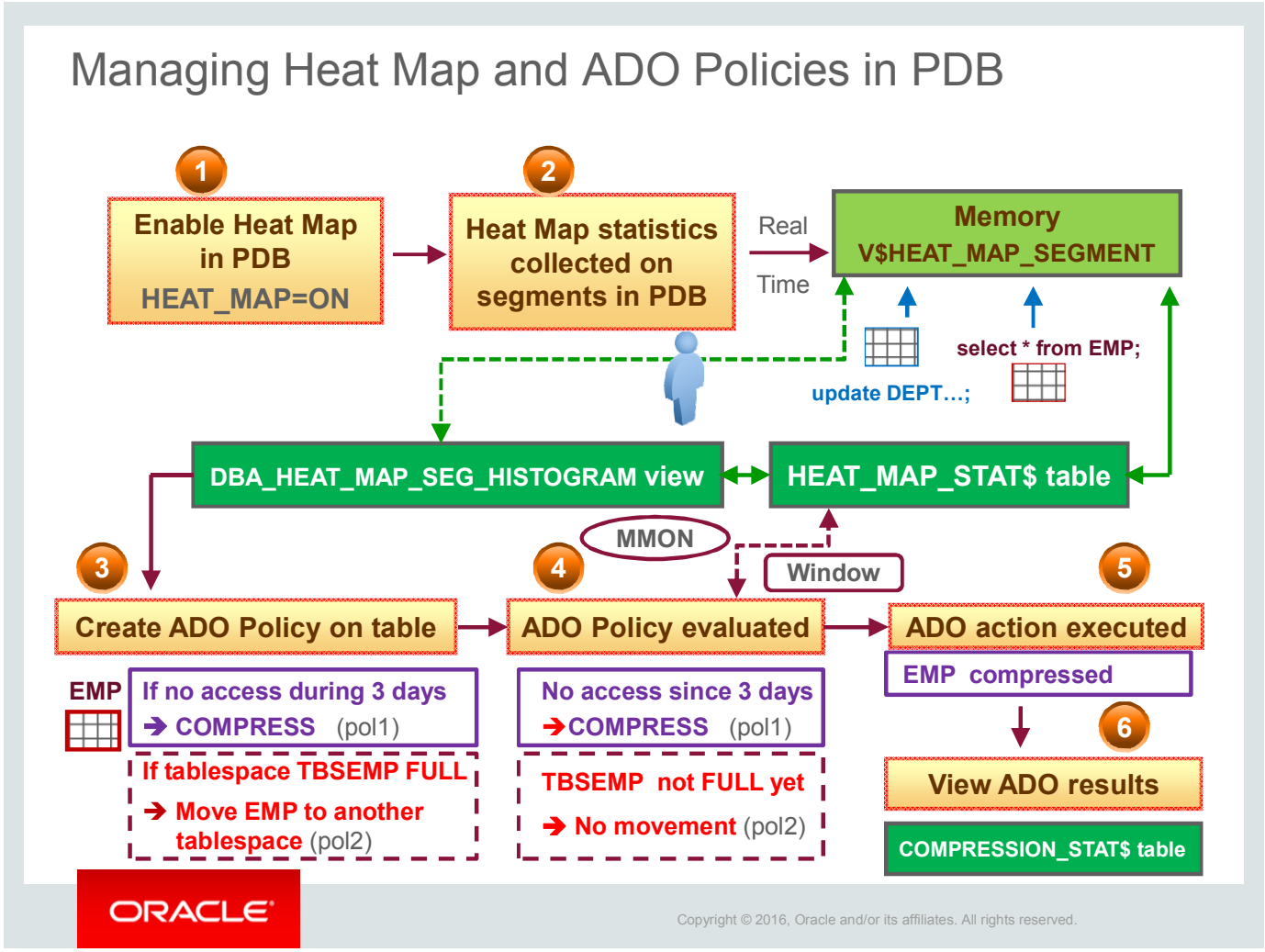
- Compress data only
- Move segments to other storage tiers under space pressure

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.

In Oracle Database 12.1, ADO is not supported in multitenant container databases.

Oracle Database 12.2 supports ADO and Heat Map statistics in CDBs and more precisely on objects in PDBs.




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_ILMEVALUATIONDETAILS` 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.

# Summary

In this lesson, you should have learned how to:

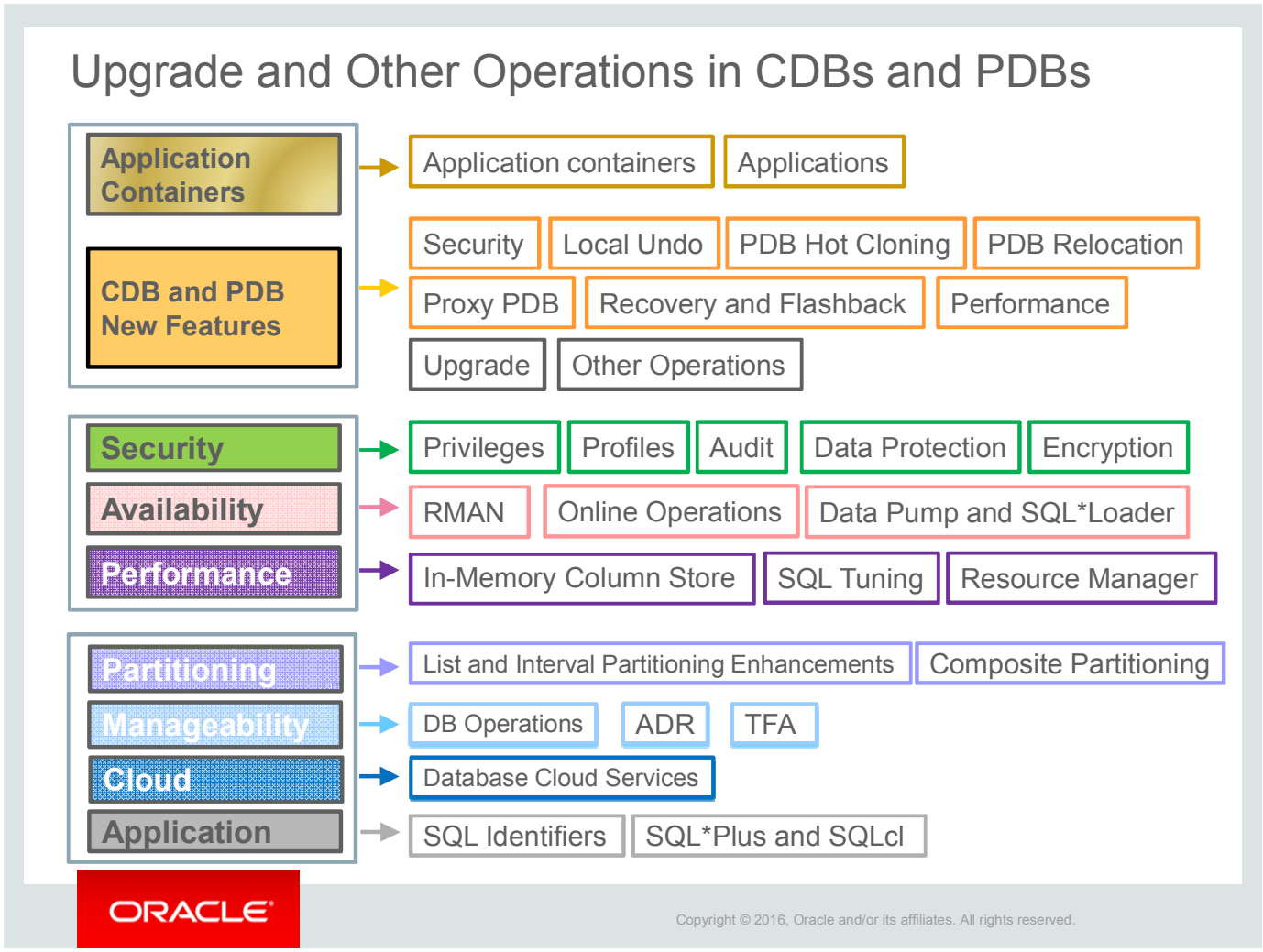
- Manage application shared object statistics
- Control query DOP involving the `containers()` construct
- Manage AWR snapshots at the CDB and PDB levels
- Run ADDM tasks for CDB and PDB recommendations
- Manage SGA and PGA at the PDB level
- Manage resource allocation between PDBs and within a PDB
- Manage Heat Map and ADO policy declaration in a PDB

# Practice 6: Overview

- 6-1: Monitoring performance at the CDB and PDB levels
- 6-2: Getting performance recommendations at CDB and PDB levels
- 6-3: Monitoring and tuning SQL executions at the PDB level
- 6-4: Using performance profiles to limit CPU between application PDBs

7

# Upgrade and Other Operations in CDBs and PDBs




This lesson essentially explains how to upgrade CDBs and individual PDBs from 12.1 to 12.2. The lesson also shows the export and import of application containers, and how to perform cross-platform PDB transport.

# Objectives

After completing this lesson, you should be able to:

- Upgrade CDBs or regular PDBs from 12.1 to 12.2
- Plug in a remote PDB through XTTS into a target CDB



To get detailed information about how to perform any of the operations explained in this lesson, refer to the following guides in Oracle documentation:

- *Oracle Database Upgrade Guide 12c Release 2 (12.2)*
- *Oracle Database Administrator's Guide 12c Release 2 (12.2)*
- *Oracle Database Utilities 12c Release 2 (12.2)*

## Upgrading CDB and PDBs to 12.2: Methods

- Data Pump Export / Import
  - Can provide better performance depending on data volume, metadata volume
  - Ensures support for new data types
- 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



There are three methods to upgrade a CDB from Oracle Database 12.1 to Oracle Database 12.2.

- Oracle Data Pump Export and Import
- Database Upgrade Assistant
- Manual upgrade by using scripts

# Upgrading a CDB Including PDBs from 12.1 to 12.2

1. Install the 12.2 Oracle Database software.
2. Execute the Pre-Upgrade Information Tool in the 12.1 CDB.

```
$ cd /u01/app/oracle/product/12.2.0/dbhome_1/rdbms/admin
$ $ORACLE_HOME/jdk/bin/java -jar preupgrade.jar
```

3. Back up the CDB.
4. Execute the `preupgrade_fixups*.sql` scripts on the 12.1 CDB.

```
$ cd /u01/app/oracle/product/12.1.0/dbhome_1/rdbms/admin
$ $ORACLE_HOME/perl/bin/perl catcon.pl -b preupgrade
    $ORACLE_BASE/cfgtoollogs/cdb1/preupgrade/preupgrade_fixups_CDB_ROOT.sql
$ $ORACLE_HOME/perl/bin/perl catcon.pl -b preupgrade
    $ORACLE_BASE/cfgtoollogs/cdb1/preupgrade/preupgrade_fixups_PDB_SEED.sql
…
```

5. Adjust the parameter file with the Oracle Database 12.2 parameters.



To upgrade the CDB and PDBs from Oracle Database 12.1 to Oracle Database 12.2, 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`.

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`.

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. 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 5, remove desupported 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 12.1 to 12.2

6. After shutting down the 12.1 CDB and all PDBs, start the CDB and all PDBs in `UPGRADE` mode in the 12.2 environment.

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

7. Execute the upgrade script on the CDB root and all PDBs.

```
$ cd $ORACLE_HOME/rdbms/admin
$ ./catctl.pl [ -C 'PDB1 PDB2' ] [ -l /tmp]
```

8. Open the CDB and upgraded PDBs in normal mode.
9. Execute the `postupgrade_fixups_<PDB>.sql` and `utlrp` scripts.

```
$ cd /u01/app/oracle/product/12.2.0/dbhome_1/rdbms/admin
$ $ORACLE_HOME/perl/bin/perl catcon.pl -c PDB1 -b postupgrade
    $ORACLE_BASE/cfgtoollogs/cdb1/preupgrade/postupgrade_fixups_CDB_ROOT.sql
$ $ORACLE_HOME/perl/bin/perl catcon.pl -c PDB1 -b postupgrade
    $ORACLE_BASE/cfgtoollogs/cdb1/preupgrade/postupgrade_fixups_PDB_SEED.sql
…
```



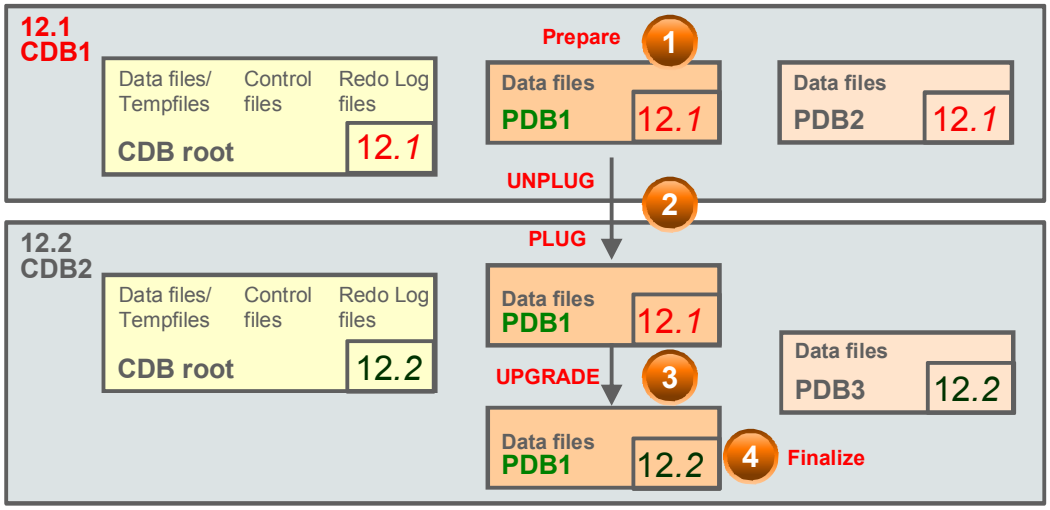
In step 7, DBUA or `catctl.pl` automatically runs the Parallel Upgrade Utility.

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 9, use the `catcon.pl` script to execute the `postupgrade_fixups.sql` and `utlrp.sql` scripts.

**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 to 12.2. 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.

## Upgrading a Single Regular PDB from 12.1 to 12.2




1. Execute `preupgrade.jar`, and then `preupgrade_fixups_<PDB>.sql` in the 12.1 PDB.
2. Unplug the PDB from the 12.1 CDB and plug the PDB into the 12.2 CDB.
3. Open the PDB in `UPGRADE` mode and upgrade the PDB.
4. Finalize by executing the `postupgrade_fixups _<PDB>.sql` and `utlrp` scripts.

ORACLE



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_BASE/cfgtoollogs/SID/preupgrade` directory. Execute the `preupgrade_fixups_<PDB>.sql` script in the PDB.
2. Close the PDB from the 12.1 CDB to unplug it. Then plug it into the target 12.2 CDB.
3. Open the PDB in the target 12.2 CDB in upgrade mode and upgrade it to 12.2.

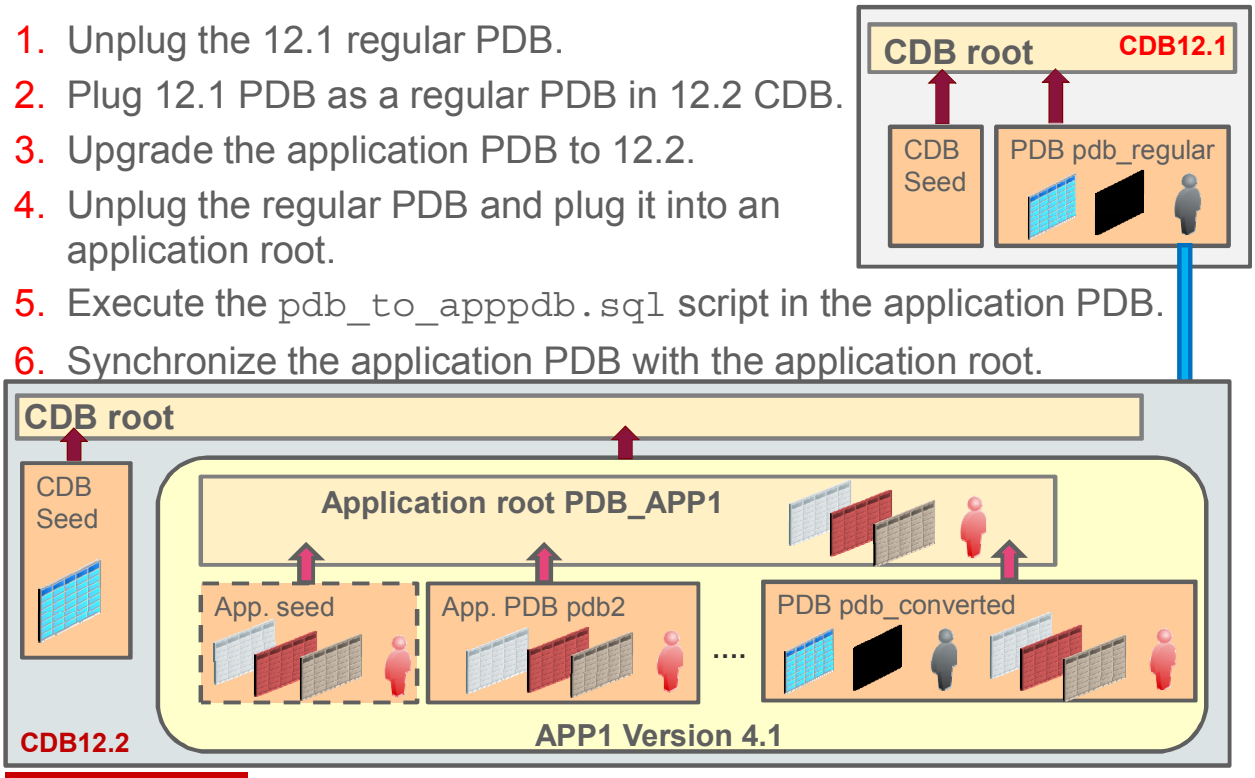
```
$ cd $ORACLE_HOME/rdbms/admin
$ ./catctl.pl -c 'PDB1' -l /tmp/upgrade catupgrd.sql
```

4. Finally close and open the PDB in normal mode. Use the `catcon.pl` script to execute the `postupgrade_fixups.sql` and `utlrp.sql` scripts. You can verify that all issues have been fixed by running the `utlu122s.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 12.1 regular PDB.
2. Plug 12.1 PDB as a regular PDB in 12.2 CDB.
3. Upgrade the application PDB to 12.2.
4. Unplug the regular PDB and plug it into an application root.
5. Execute the `pdb_to_apppdb.sql` script in the application PDB.
6. Synchronize the application PDB with the application root.



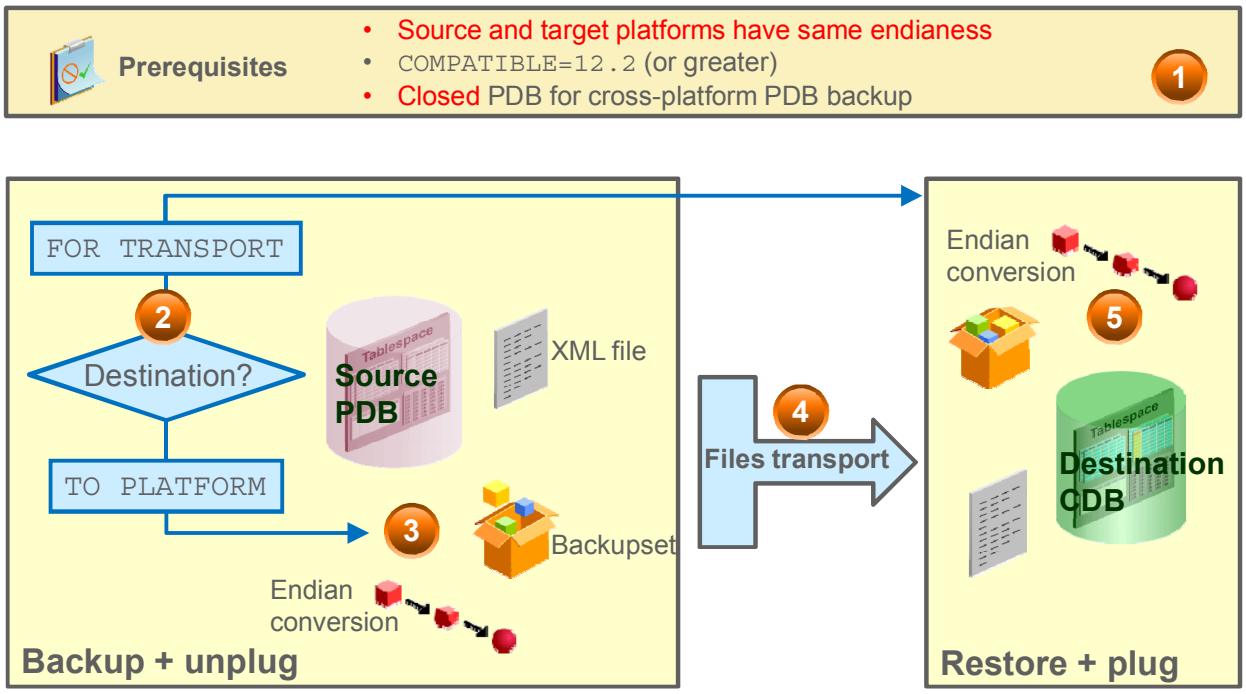



Applications that are already installed in an Oracle Database 12.1.0.2 PDB can also take advantage of applications containers. Perform the steps listed in the slide.

# Practice 7: Overview

- 7-1: Upgrading and converting a 12.1.0.2 regular PDB to a 12.2 application PDB

# 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, either using 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 Using XTTS: Phase 1

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



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 Using XTTS: Phase 2


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';
```

ORACLE



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 or regular PDBs from 12.1 to 12.2
- Plug in a remote PDB through XTTS into a target CDB

# Practice 7: Overview

- 7-2: Plugging remote PDBs through XTTS
- 7-3: Upgrading a 12.1.0.2 CDB to a 12.2 CDB (Optional)