Oracle® Multitenant
Administrator's Guide

19c
E96136-08
January 2021

Oracle Multitenant Administrator's Guide, 19c

E96136-08

Primary Authors: Randy Urbano, Lance Ashdown

Contributing Authors: Patricia Huey, Donna Keesling, Roopesh Kumar, Bert Rich, Richard Strohm

Contributors: Penny Avril, Thomas Baby, Hermann Baer, Yasin Baskan, Dominique Djeunot, Andre Kruglikov, Kishy Kumar, Sue Lee, Siyu Liu, Bryn Llewellyn, Colin McGregor, John McHugh, Valarie Moore, Muthu Olagappan, Bhavesh Patel, Kumar Rajamani, Giridhar Ravipati, Can Tuzla, Patrick Wheeler

Contents

2 Overview of the Multitenant Architecture

4 Creating and Configuring a CDB

Part III Creating and Removing PDBs and Application Containers

5 Overview of PDB Creation

6 Creating a PDB from Scratch

7 Cloning a PDB or Non-CDB

10 Creating a PDB as a Proxy PDB

11 Removing a PDB

12 Creating and Removing Application Containers and Seeds

Part IV Administering a Multitenant Environment

13 Administering a CDB

18 Managing Security for a Multitenant Environment

19 Monitoring CDBs and PDBs

23 Using Oracle Scheduler with a CDB

24 Using Oracle Database Vault with a CDB

25 Using XStream with a CDB

Glossary

Index

Preface

This document describes how to create, configure, and administer an Oracle database.

Audience

This document is intended for database administrators who perform the following tasks:

- Create and configure one or more Oracle databases
- Monitor and tune Oracle databases
- Oversee routine maintenance operations for Oracle databases
- Create and maintain schema objects, such as tables, indexes, and views
- Schedule system and user jobs
- Diagnose, repair, and report problems

To use this document, you should be familiar with relational database concepts. You should also be familiar with the operating system environment under which you are running Oracle Database.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Documents

For more information, see these Oracle resources:

- *Oracle Database Concepts*
- *Oracle Database Administrator's Guide*
- *Oracle Database 2 Day DBA*
- *Oracle Database SQL Language Reference*

- *Oracle Database Reference*
- *Oracle Database PL/SQL Packages and Types Reference*
- *Oracle Automatic Storage Management Administrator's Guide*
- *Oracle Database VLDB and Partitioning Guide*
- *Oracle Database Error Messages Reference*
- *Oracle Database Net Services Administrator's Guide*
- *Oracle Database Backup and Recovery User's Guide*
- *Oracle Database Performance Tuning Guide*
- *Oracle Database SQL Tuning Guide*
- *Oracle Database Development Guide*
- *Oracle Database PL/SQL Packages and Types Reference*
- *SQL*Plus User's Guide and Reference*

Many of the examples in this book use the sample schemas. See *Oracle Database Sample Schemas* for information about these schemas.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Changes in This Release for Oracle Multitenant Administrator's Guide

There are changes in this document for recent releases of Oracle Database.

Changes in Oracle Database Release 19c, Version 19.1

Oracle Multitenant Administrator's Guide for Oracle Database release 19c, version 19.1 has the following changes.

New Features

The following major features are new in this release.

- Workload capture and replay in a PDB

  A local user can capture, replay, and report on a workload at the PDB level.

  See "About Using Manageability Features in a CDB" and *Oracle Database Testing Guide*.

- ADDM analysis for PDBs

  You can use ADDM to analyze AWR data stored inside the PDB through an AWR snapshot taken inside the PDB. You can also analyze AWR data of a non-CDB, CDB root, or PDB imported into the AWR storage of a PDB. Automatic ADDM of a PDB is disabled by default. You can enable it for a PDB by enabling automatic AWR snapshots.

  See "About Using Manageability Features in a CDB" and *Oracle Database Performance Tuning Guide*.

- Database Vault Operations Control for infrastructure database administrators

  You can use Oracle Database Vault to block common users (for example, infrastructure DBAs) from accessing local data in PDBs. Thus, common users are blocked from accessing local data. Oracle Database Vault enables you to store sensitive data for your business applications and allow operations to manage the database infrastructure without having to access sensitive customer data.

  See "Using Database Vault Operations Control to Restrict Multitenant Common User Access to Local PDB Data".

- Support for multiple PDB shards in the same CDB

  A CDB can contain multiple PDBs as shard catalog databases. Also, a CDB can contain shard PDBs from different sharded databases (SDBs), each managed by its own separate catalog database.

  See *Using Oracle Sharding*.

- Automated PDB relocation

In Oracle Grid Infrastructure, you can use Oracle Fleet Patching and Provisioning to automate relocation of a PDB from one CDB to another. Automated relocation enables you to patch individual PDBs more quickly without exposing other PDBs to the changes in the patch.

See *Oracle Clusterware Administration and Deployment Guide*.

- Cloning a remote PDB using DBCA

  You can clone a remote PDB using DBCA in silent mode.

  See "About Cloning a Remote PDB".

- Remote PDB relocation

  You can use Database Configuration Assistant (DBCA) to relocate a PDB from a remote CDB to a local CDB.

  See "Relocating a PDB Using DBCA: Example".

- Cloud object store support for Data Pump Import

  The `credential` parameter of `impdb` specifies the name of a credential object that contains the user name and password required to access an object store bucket. You can also specify a default credential using the database property `DEFAULT_CREDENTIAL`.

  See "Setting the Default Credential in a PDB" and "Using Data Pump to Move Data Into a CDB".

> **See Also:**
>
> *Oracle Database Licensing Information User Manual* for details on which features are supported for different editions and services

Changes in Oracle Database Release 18c, Version 18.1

The following are changes in *Oracle Multitenant Administrator's Guide* for Oracle Database release 18c, version 18.1.

New Features

The following features are new in this release:

- CDB fleet

  A **CDB fleet** is a collection of different CDBs that can be managed as one logical CDB.

  See "Administering a CDB Fleet".

- PDB snapshot carousel

  A **PDB snapshot** is a point-in-time copy of a PDB. The source PDB can be open read-only or read/write while the snapshot is created. You can create snapshots manually using the `SNAPSHOT` clause of `CREATE PLUGGABLE DATABASE` (or `ALTER PLUGGABLE DATABASE`), or automatically using the `EVERY` *`interval`* clause. When a PDB is enabled for snapshots, you can create multiple snapshots (point-in-time

copies) of the PDB. The library of snapshots is called a PDB snapshot carousel. You can quickly clone a new PDB based on any snapshot in the carousel. In this way, you can perform point-in-time recovery to any snapshot in the carousel, or rapidly create a PDB by cloning any snapshot.

See "User Interface for PDB Snapshot Carousel" and "Administering a PDB Snapshot Carousel".

- Logical partitioning

  A container map enables a session to issue SQL statements that are routed to the appropriate PDB, depending on the value of a predicate used in the SQL statement. The partitioning column in the map table does not need to match a column in the metadata-linked table. For example, if the table `sales` is enabled for the container map `pdb_map_tbl`, and if `sales` does not have the column used to partition `pdb_map_tbl`, then queries with the predicate `CONTAINERS(sales)` are still routed to the PDBs specified in the map table.

  See "Container Maps".

- Refreshable PDB switchover

  A **refreshable clone PDB** is a read-only clone that can periodically synchronize with its source PDB. You can reverse the roles, transforming the source PDB into the clone and the clone into the source. This technique can be useful for load balancing. Also, if the source PDB fails, then you can resume operations on the clone PDB, rendering a CDB-level Oracle Data Guard failover unnecessary.

  See "About Refreshable Clone PDBs" and "Switching Over a Refreshable Clone PDB".

- Lockdown profile enhancements

  You can create, alter, or drop lockdown profiles in application containers. Also, you can create lockdown profiles based on a static or a dynamic base profile.

  See "Overview of PDB Lockdown Profiles", "About Restricting PDB Users for Enhanced Security", and "Restricting Operations on PDBs Using PDB Lockdown Profiles".

- DBCA enhancements

  You can use DBCA to clone a local PDB or duplicate a CDB. Duplication is only supported in silent mode.

  See "About CDB Creation with DBCA" and "About Cloning a Local PDB".

- Usable backups of non-CDBs and relocated PDBs

  When you are cloning a non-CDB as a PDB or relocating a PDB, you can use the `DBMS_PDB.EXPORTRMANBACKUP` procedure to export RMAN backup metadata into the PDB dictionary. This metadata enables backups of the source non-CDB or PDB to be usable for restore and recovery of the target PDB.

  See "General Prerequisites for PDB Creation".

- RMAN duplication of a PDB to another CDB

  You can clone a PDB from a source CDB to an existing CDB that is open read/write.

  See "Techniques for Creating a PDB"

- Relocation of sessions during planned maintenance

Application Continuity can drain database sessions during planned maintenance when the application submits a connection test, at request boundaries, and at good places to fail over. The relocation is transparent to applications. This feature is on by default for all maintenance operations invoked at the database service and PDB levels: stop service, relocate service, relocate PDB, and stop PDB.

See "Managing Services for PDBs", "How PDB Relocation Works", and *Oracle Real Application Clusters Administration and Deployment Guide*.

- Copying a PDB in an Oracle Data Guard environment

  When performing a remote clone in a primary database, or plugging in a PDB in a primary database, you can set initialization parameters in a standby database that automates copying the data files for the newly created PDB.

  See "Cloning a Remote PDB: Basic Steps" and "Plugging In an Unplugged PDB".

- Parallel statement queuing at the PDB level

  You can configure parallel statement queuing for a PDB just as for a non-PDB using the `PARALLEL_SERVERS_TARGET` initialization parameter. At the PDB level, the default is based on the `CPU_COUNT` setting for the PDB. At the CDB level, the default value is the value of the `PARALLEL_MAX_SERVERS` initialization parameter.

  See "Utilization Limits for PDBs".

- Split mirror clone PDBs

  When a PDB resides in Oracle ASM, you can use a split mirroring technique to clone a PDB. The cloned PDB is independent of the original PDB. The principal use case is to rapidly provision test and development PDBs in an Oracle ASM environment.

  See "Creating a Split Mirror Clone PDB".

Part I
Multitenant Architecture

The **multitenant architecture** enables an Oracle database to function as a multitenant container database (CDB).

1

Introduction to the Multitenant Architecture

Become familiar with the **Oracle Multitenant** option.

About the Multitenant Architecture

The **multitenant architecture** enables an Oracle database to function as a multitenant container database (CDB).

A CDB includes zero, one, or many customer-created pluggable databases (PDBs). A PDB is a portable collection of schemas, schema objects, and nonschema objects that appears to an Oracle Net client as a non-CDB. All Oracle databases before Oracle Database 12*c* were non-CDBs.

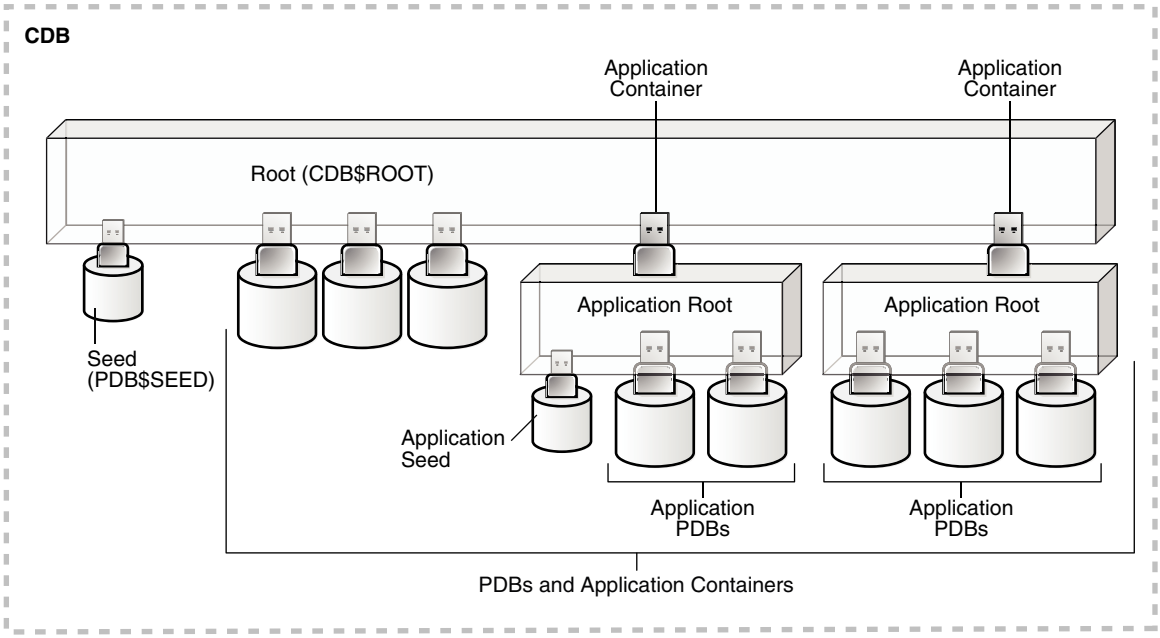
Video

About Containers in a CDB

A **container** is logical collection of data or metadata within the multitenant architecture.

The following figure represents possible containers in a CDB.

Figure 1-1 Containers in a CDB



Every CDB has the following containers:

- Exactly one CDB root container (also called simply *the root*)

  The CDB root is a collection of schemas, schema objects, and nonschema objects to which all PDBs belong (see "Overview of Containers in a CDB"). The root stores Oracle-supplied metadata and common users. An example of metadata is the source code for Oracle-supplied PL/SQL packages (see "Data Dictionary Architecture in a CDB"). A common user is a database user known in every container (see "Common Users in a CDB"). The root container is named `CDB$ROOT`.

- Exactly one system container

  The system container includes the root CDB and all PDBs in the CDB. Thus, the system container is the logical container for the CDB itself.

- Zero or more application containers

  An application container consists of exactly one application root, and the PDBs plugged in to this root. Whereas the system container contains the CDB root and *all* the PDBs within the CDB, an application container includes only the PDBs plugged into the application root. An application root belongs to the CDB root and no other container.

- Zero or more user-created PDBs

  A PDB contains the data and code required for a specific set of features (see "PDBs"). For example, a PDB can support a specific application, such as a human resources or sales application. No PDBs exist at creation of the CDB. You add PDBs based on your business requirements.

  A PDB belongs to exactly zero or one application container. If a PDB belongs to an application container, then it is an application PDB. For example, the `cust1_pdb` and `cust2_pdb` application PDBs might belong to the `saas_sales_ac` application container, in which case they belong to no other application containers. An application seed is an optional application PDB that acts as a user-created PDB template, enabling you to create new application PDBs rapidly.

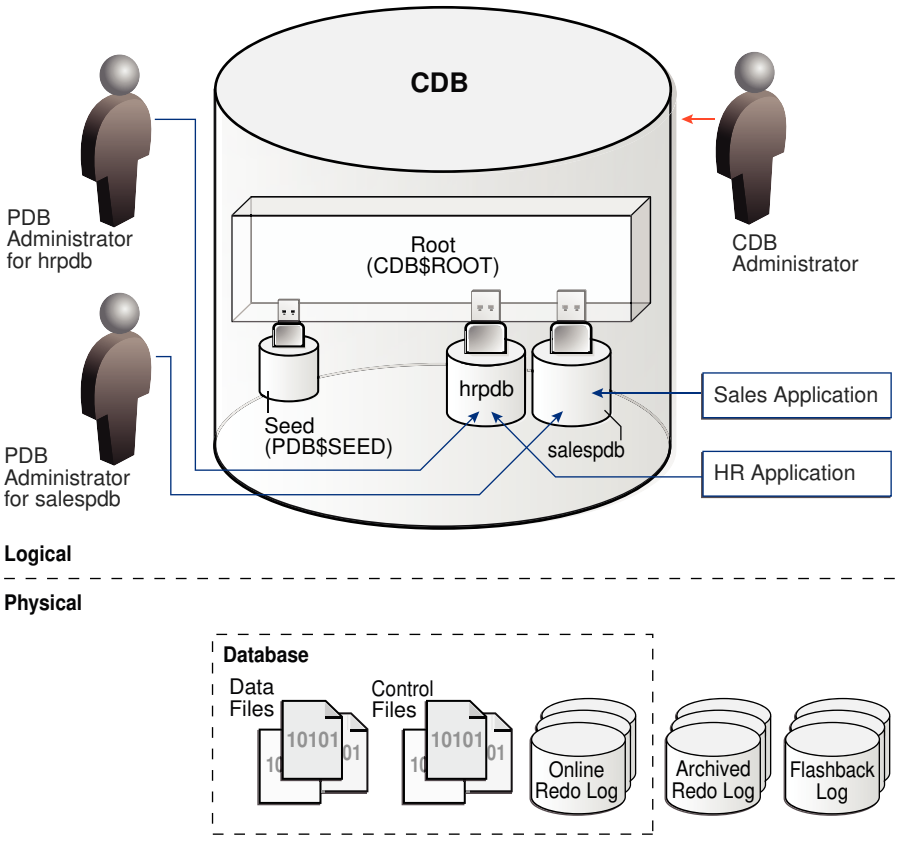
- Exactly one seed PDB

  The seed PDB is a system-supplied template that the CDB can use to create new PDBs. The seed PDB is named `PDB$SEED`. You cannot add or modify objects in `PDB$SEED`.

Example 1-1 CDB with No Application Containers

This example shows a simple CDB with five containers: the system container (the entire CDB), the CDB root, the PDB seed (`PDB$SEED`), and two PDBs. Each PDB has its own dedicated application. A different PDB administrator manages each PDB. A common user exists across a CDB with a single identity. In this example, common user `SYS` can manage the root and every PDB. At the physical level, this CDB has a database instance and database files, just as a non-CDB does.

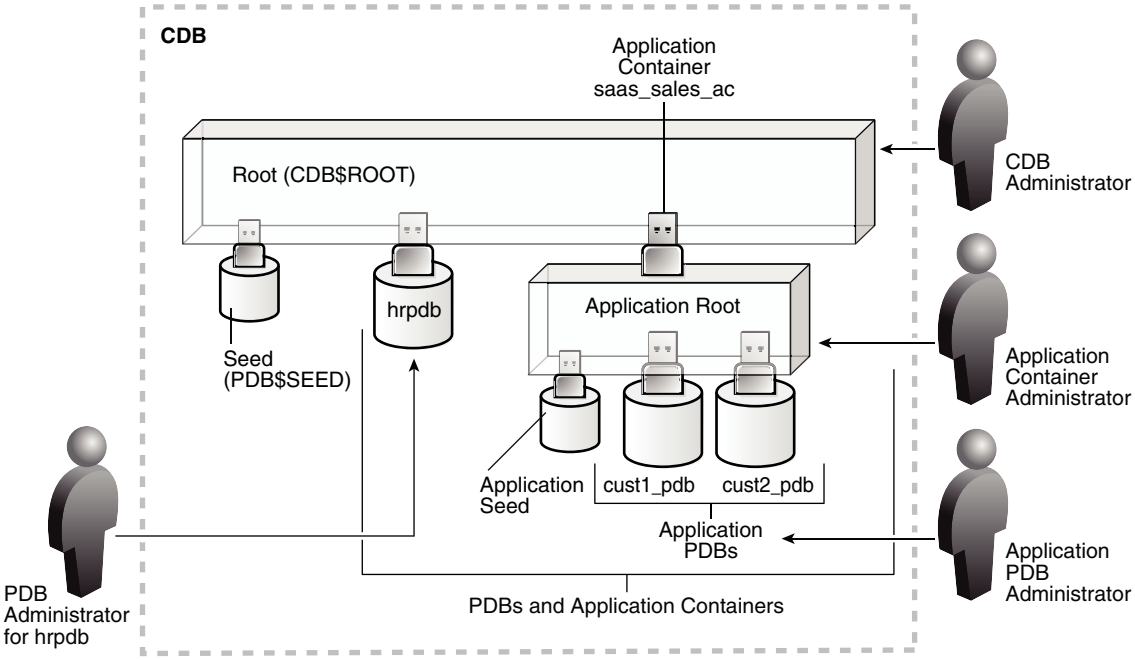
Figure 1-2 CDB with No Application Containers



Example 1-2 CDB with an Application Container

In this variation, the CDB contains an application container named `saas_sales_ac`. Within the application container, the application PDB `cust1_pdb` supports an application for one customer, and the application PDB `cust2_pdb` supports an application for a different customer. The CDB also contains a PDB named `hrpdb`, which supports an HR application, but does not belong to an application container.

Figure 1-3 CDB with an Application Container



In this example, multiple DBAs manage the CDB environment:

- A CDB administrator manages the CDB itself.
- An application container administrator manages the `saas_sales_ac` container, including application installation and upgrades.
- An application PDB administrator manages the two PDBs in the `saas_sales_ac` container: `cust1_pdb` and `cust2_pdb`.
- A PDB administrator manages `hrpdb`.

> **See Also:**
>
> " Overview of Configuring and Managing a Multitenant Environment"

About User Interfaces for the Multitenant Architecture

You can use the same administration tools for both CDBs and non-CDBs.

Table 1-1 Tools in a Multitenant Environment

Interface	Description	See Also
SQL*Plus and SQL Developer for command-line access	SQL*Plus is an interactive and batch query tool that is installed with Oracle Database.	*SQL*Plus User's Guide and Reference*

Table 1-1 (Cont.) Tools in a Multitenant Environment

Interface	Description	See Also
Oracle Enterprise Manager Cloud Control (Cloud Control)	Cloud Control is an Oracle Database administration tool that provides a graphical user interface (GUI). Cloud Control supports Oracle Database 12c targets, including PDBs, CDBs, and non-CDBs.	The Cloud Control online help
Oracle Enterprise Manager Database Express (EM Express)	EM Express is a web-based management product built into the Oracle database. EM Express enables you to provision and manage PDBs, including the following operations: • Creating and dropping PDBs • Plugging in and unplugging and PDBs • Cloning PDBs • Setting resource limits for PDBs	*Oracle Database Performance Tuning Guide* to learn more about using EM Express for managing CDBs and PDBs
Oracle Database Configuration Assistant (DBCA)	DBCA is a utility with a graphical user interface that enables you to create and duplicate CDBs. It also enables you to create, relocate, clone, plug in, and unplug PDBs.	*Oracle Database Performance Tuning Guide* and *Oracle Database Administrator's Guide* for more information about DBCA

See Also:

Oracle Database Concepts for more information about tools for database administrators

Benefits of the Multitenant Architecture

The multitenant architecture solves several problems posed by the traditional non-CDB architecture.

Challenges for a Non-CDB Architecture

Large enterprises may use hundreds or thousands of databases. Often these databases run on different platforms on multiple physical servers.

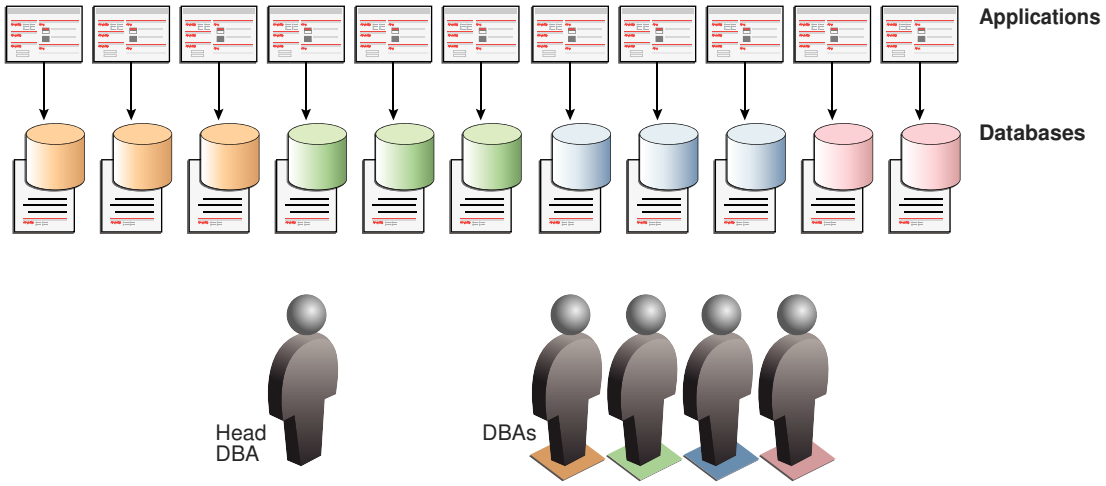
Because of improvements in hardware technology, especially the increase in the number of CPUs, servers can handle heavier workloads than before. A database

may use only a fraction of the server hardware capacity. This approach wastes both hardware and human resources.

For example, 100 servers may have one database each, with each database using 10% of hardware resources and 10% of an administrator's time. A team of DBAs must manage the SGA, database files, accounts, security, and so on of each database separately, while system administrators must maintain 100 different computers.

To show the problem in reduced scale, Figure 1-4 depicts 11 databases, each with its own application and server. A head DBA oversees a team of four DBAs, each of whom is responsible for two or three databases.

Figure 1-4 Database Environment Before Database Consolidation



Typical responses include:

- Use virtual machines (VMs).

  In this model, you replicate the operating infrastructure of the physical server —operating system and database—in a virtual machine. VMs are agile, but use technical resources inefficiently, and require individual management. Virtual sprawl, which is just as expensive to manage, replaces the existing physical sprawl.

- Place multiple databases on each server.

  Separate databases eliminate operating system replication, but do not share background processes, system and process memory, or Oracle metadata. The databases require individual management.

- Separate the data logically into schemas or virtual private databases (VPDs).

  This technique uses technical resources efficiently. You can manage multiple schemas or VPDs as one. However, this model is less agile than its alternatives, requiring more effort to manage, secure, and transport. Also, the logical model typically requires extensive application changes, which discourages adoption.

Benefits of the Multitenant Architecture for Database Consolidation

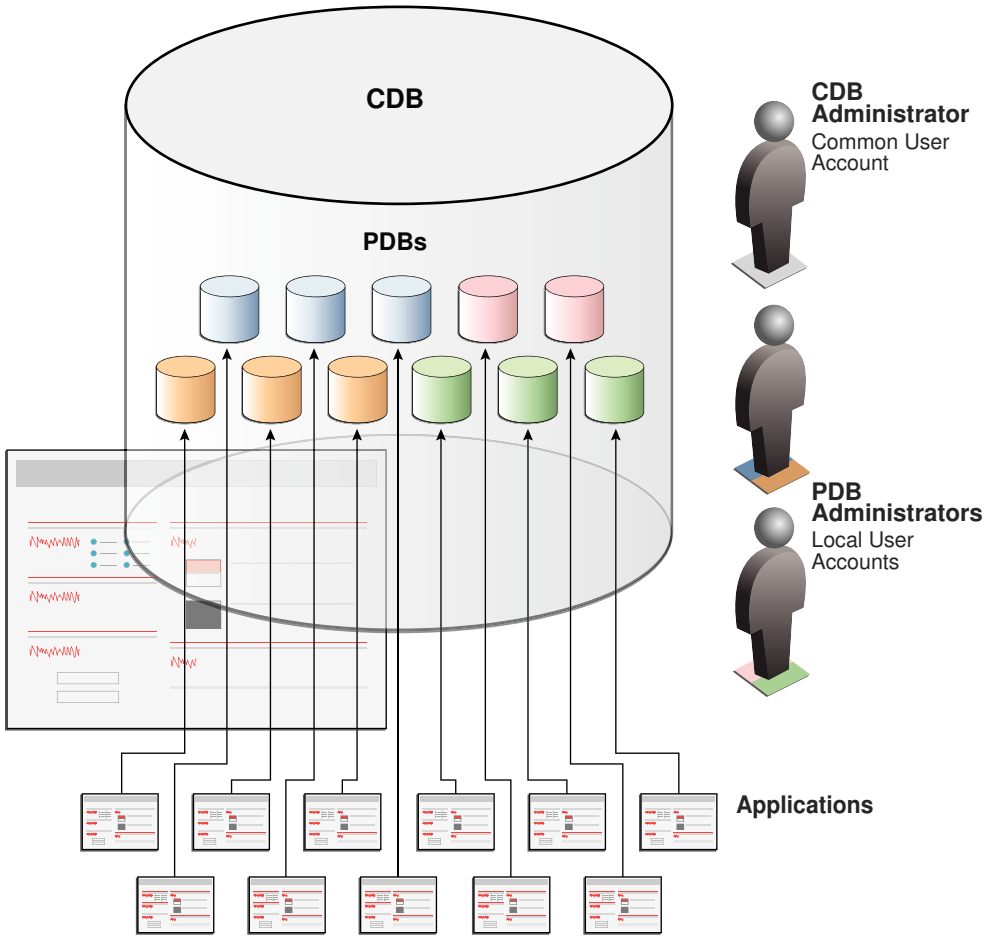
Database consolidation is the process of consolidating data from multiple databases into one database on one computer. The Oracle Multitenant option enables you to consolidate data and code *without altering existing schemas or applications*.

The PDB/non-CDB compatibility guarantee means that a PDB behaves the same as a non-CDB as seen from a client connecting with Oracle Net. The installation scheme for an application definition (for example, tables and PL/SQL packages) that runs against a non-CDB runs the same against a PDB and produces the same result. Also, the run-time behavior of client code that connects to the PDB containing the application definition is identical to the behavior of client code that connected to the non-CDB containing this application definition.

Operations that act on an entire non-CDB act in the same way on an entire CDB, for example, when using Oracle Data Guard and database backup and recovery. Thus, the users, administrators, and developers of a non-CDB have substantially the same experience after the database has been consolidated.

The following graphic depicts the databases in Figure 1-4 after consolidation onto one computer. The DBA team is reduced from five to three, with one CDB administrator managing the CDB while two PDB administrators split management of the PDBs.

Figure 1-5 Single CDB



Starting in Oracle Database 12c Release 2 (12.2), you can create an application container that contains application PDBs. This approach enables you to create and manage an application within this container. Most benefits that apply to consolidation into a CDB also apply to consolidation within an application container.

Using the multitenant architecture for database consolidation has the following benefits:

- Cost reduction

  By consolidating hardware and database infrastructure to a single set of background processes, and efficiently sharing computational and memory resources, you reduce costs for hardware and maintenance. For example, 100 PDBs on a single server share one database instance.

- Easier and more rapid movement of data and code

  By design, you can quickly plug a PDB into a CDB, unplug the PDB from the CDB, and then plug this PDB into a different CDB. You can also clone PDBs while they remain available. You can plug in a PDB with any character set and access it without character set conversion. If the character set of the CDB is AL32UTF8, then PDBs with different database character sets can exist in the same CDB.

- Easier management and monitoring of the physical database

  The CDB administrator can manage the environment as an aggregate by executing a single operation, such as patching or performing an RMAN backup, for all hosted tenants and the CDB root. Backup strategies and disaster recovery are simplified.

- Separation of data and code

  Although consolidated into a single physical database, PDBs mimic the behavior of non-CDBs. For example, if user error loses critical data, then a PDB administrator can use Oracle Flashback or point-in-time recovery to retrieve the lost data without affecting other PDBs.

- Secure separation of administrative duties

  A common user can connect to any container on which it has sufficient privileges, whereas a local user is restricted to a specific PDB. Administrators can divide duties as follows:

  – An administrator uses a common account to manage a CDB or application container. Because a privilege is contained within the container in which it is granted, a local user on one PDB does not have privileges on other PDBs within the same CDB.

  – An administrator uses a local account to manage an individual PDB.

- Ease of performance tuning

  It is easier to collect performance metrics for a single database than for multiple databases. It is easier to size one SGA than 100 SGAs.

- Fewer database patches and upgrades

  It is easier to apply a patch to one database than to 100 databases, and to upgrade one database than to upgrade 100 databases.

> **See Also:**
>
> - " Overview of Configuring and Managing a Multitenant Environment"
> - *Oracle Database Security Guide* to learn about common users

Benefits of the Multitenant Architecture for Manageability

The multitenant architecture has benefits beyond database consolidation. These benefits derive from storing the data *and* metadata specific to a PDB in the PDB itself rather than storing all dictionary metadata in one place.

By storing its own dictionary metadata, a PDB becomes easier to manage as a distinct unit. This benefit occurs even when only one PDB resides in a CDB. Grouping PDBs into a separately managed application container increases manageability even further.

In a CDB, the data dictionary metadata is split between the root and the PDBs. Benefits of data dictionary separation include the following:

- Easier upgrade of data and code

  For example, instead of upgrading a CDB from one database release to another, you can rapidly unplug a PDB from the existing CDB, and then plug it into a newly created CDB from a higher release.

- Easier migration between servers

  To perform load balancing or to meet SLAs, you can migrate an application database from an on-premise data center to the cloud, or between two servers in the same environment.

- Protection against data corruption within a PDB

  You can flash back a PDB to an SCN or PDB-specific restore point, without affecting other PDBs. This feature is analogous to the Flashback Database feature for a non-CDB.

- Ability to install, administer, and upgrade application-specific data and metadata in a single place

  You can define a set of application-specific PDBs as a single component, called an application container. You can then define one or more applications within this container. Each application is a named, versioned set of common metadata and data shared within this application container.

  For example, each customer of a SaaS vendor could have its own application PDB. Each application PDB might have identically defined tables named `sales_mlt`, with different data in each PDB. The PDBs could share a data-linked common object named `countries_olt`, which has identical data in each PDB. As an application administrator, you could manage the master application definition so that every new customer gets a PDB with the same objects, and every change to existing schemas (for example, the addition of a new table, or a change in the definition of a table) applies to all PDBs that share the application definition.

- Integration with Oracle Database Resource Manager

  In a multitenant environment, one concern is contention for system resources among the PDBs running on the same server. Another concern is limiting resource

usage for more consistent, predictable performance. To address such resource contention, usage, and monitoring issues, use Oracle Database Resource Manager.

See Also:

- "Overview of Oracle Resource Manager in a CDB"
- "Data Dictionary Architecture in a CDB"
- "Administering Application Containers"

Path to Database Consolidation

For the duration of its existence, a database is either a CDB or a non-CDB.

You must define a database as a CDB at creation, and then create PDBs and application containers within this CDB. You cannot later transform a non-CDB into a CDB, or a CDB into a non-CDB.

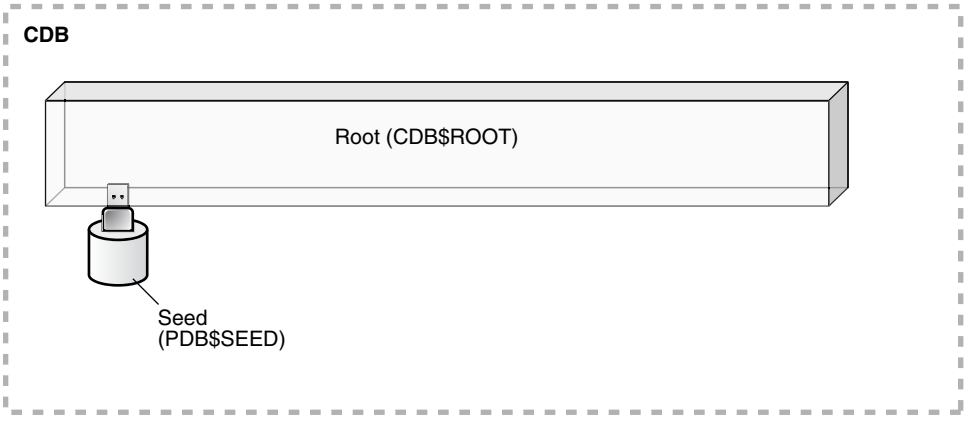
Creation of a CDB

The `CREATE DATABASE ... ENABLE PLUGGABLE DATABASE` SQL statement creates a new CDB.

If you do not specify the `ENABLE PLUGGABLE DATABASE` clause, then the newly created database is a non-CDB. In this case, the non-CDB can never contain PDBs.

When you create a CDB, Oracle Database automatically creates a root container (`CDB$ROOT`) and a seed PDB (`PDB$SEED`). The following graphic shows a newly created CDB:

Figure 1-6 CDB with Seed PDB



Example 1-3 Determining Whether a Database Is a CDB

The following simple query determines whether the database to which an administrative user is currently connected is a non-CDB, or a container in a CDB:

```
SQL> SELECT NAME, CDB, CON_ID FROM V$DATABASE;

NAME      CDB     CON_ID
--------- --- ----------
CDB1      YES          0
```

> **See Also:**
>
> - " Creating and Configuring a CDB"
> - *Oracle Database SQL Language Reference* for more information about specifying the clauses and parameter values for the `CREATE DATABASE` statement

Creation of a PDB

The `CREATE PLUGGABLE DATABASE` SQL statement creates a PDB.

The created PDB automatically includes a full data dictionary including metadata and internal links to system-supplied objects in the CDB root. You must define every PDB from a single root: either the CDB root or an application root.

Each PDB has a globally unique identifier (GUID). The PDB GUID is primarily used to generate names for directories that store the PDB's files, including both Oracle Managed Files directories and non-Oracle Managed Files directories.

> **See Also:**
>
> "Creating and Removing PDBs and Application Containers"

Creation of a PDB by Cloning

One technique for creating a PDB is called *cloning*.

You can clone a PDB from `PDB$SEED`, an application seed, a remote or local PDB, or a non-CDB.

Creation of a PDB from a Seed

You can use the `CREATE PLUGGABLE DATABASE` statement to create a PDB from a seed.

A seed is a PDB that serves as a template for creation of another PDB. Creating a PDB from a seed copies some or all of the contents of a PDB, and then assigns a new unique identifier.

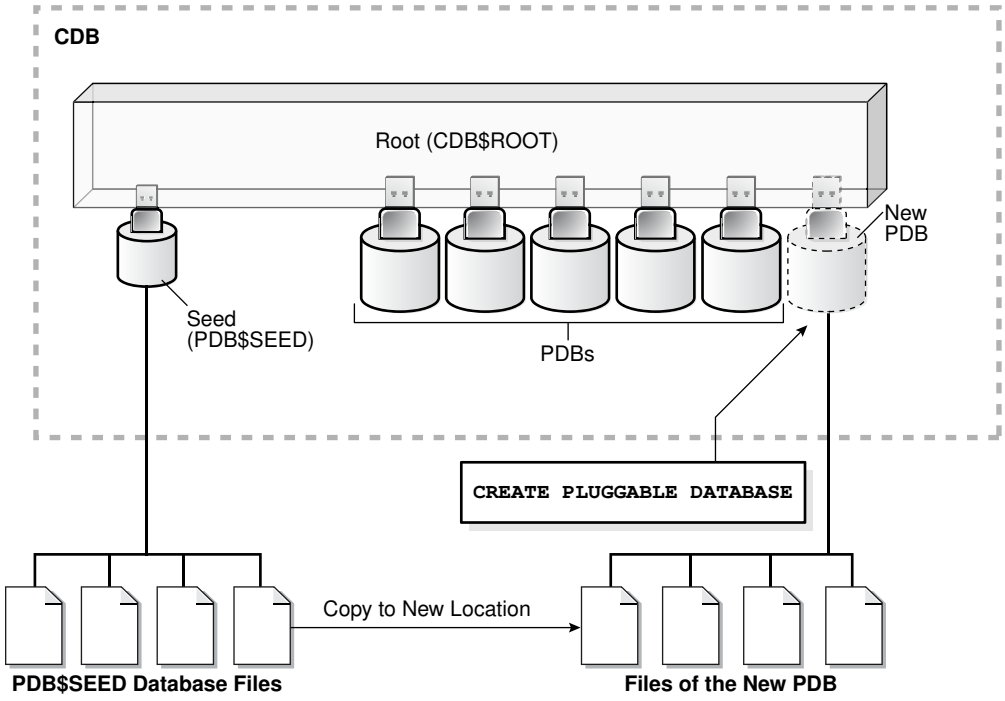
A seed PDB is either of the following:

- The PDB seed (`PDB$SEED`), which is a system-supplied template for creating PDBs

  Every CDB has exactly one `PDB$SEED`, which cannot be modified or dropped.
- An application seed, which is a user-created PDB for a specified application root

  Within an application container, you can create an application seed using the `CREATE PLUGGABLE DATABASE AS SEED` statement, which you can then use to accelerate creation of new application PDBs.

Figure 1-7 Creation from PDB$SEED



Example 1-4 Creation of a PDB from PDB$SEED

The following SQL statement creates a PDB named `hrpdb` from `PDB$SEED` using Oracle Managed Files:

```
CREATE PLUGGABLE DATABASE hrpdb
 ADMIN USER dba1 IDENTIFIED BY password;
```

> **See Also:**
>
> "Creating a PDB from Scratch"

Creation of a PDB by Cloning a PDB or a Non-CDB

To clone a PDB or non-CDB, use the `CREATE PLUGGABLE DATABASE` statement with the `FROM` clause.

In this technique, the source is either a non-CDB, or a PDB in a local or remote CDB. The target is the PDB copied from the source. The cloning operation copies the files associated with the source to a new location, and then assigns a new GUID to create the PDB.

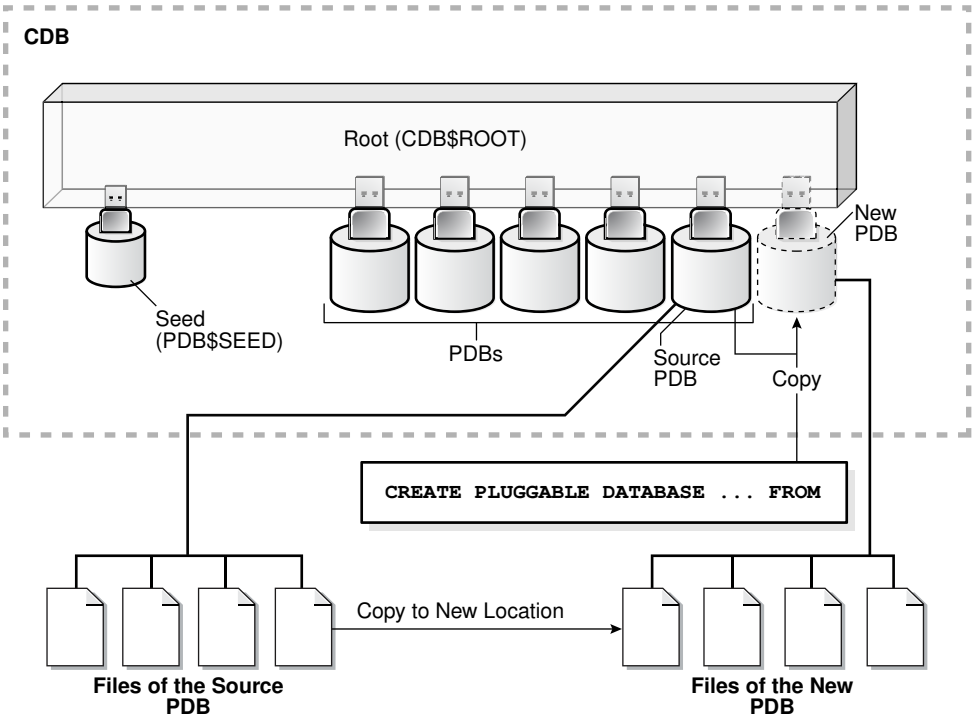
This technique is useful for quickly creating PDBs for testing and development. For example, you might test a new or modified application on a cloned PDB before deploying the application in a production PDB. If a PDB is in local undo mode, then the source PDB can be open in read/write mode during the operation, referred to as hot cloning.

> **Note:**
>
> If you clone a PDB from a remote CDB, then you must use a database link.

If you run `CREATE PLUGGABLE DATABASE` statement in an application root, then the cloned PDB is created in the application container. In this case, the application name and version of the source PDB must be compatible with the application name and version of the application container.

The following graphic illustrates cloning a PDB when both source and target are in the same CDB.

Figure 1-8 Cloning a PDB



Starting in Oracle Database 19c, you can clone a remote PDB using DBCA.

Example 1-5 Cloning a PDB

The following SQL statement clones a PDB named `salespdb` from the plugged-in PDB named `hrpdb`:

```
CREATE PLUGGABLE DATABASE salespdb FROM hrpdb;
```

See Also:

- "Cloning a PDB or Non-CDB"
- "Overview of Tablespaces and Database Files in a CDB"
- "Application Maintenance"

Clones from PDB Snapshots

Create a **clone from a PDB snapshot** by specifying `USING SNAPSHOT` clause of the `CREATE PLUGGABLE DATABASE` command.

Creation of PDB Snapshots with the SNAPSHOT Clause

A **PDB snapshot** is a point-in-time copy of a PDB. The source PDB can be open read-only or read/write while the snapshot is created. A PDB snapshot taken while the source PDB is open is called a **hot clone**. You can create clones from PDB snapshots. These clone PDBs are useful in development and testing.

You can create snapshots manually using the `SNAPSHOT` clause of `CREATE PLUGGABLE DATABASE` (or `ALTER PLUGGABLE DATABASE`), or automatically using the `EVERY` *`interval`* clause. The following statement creates a PDB snapshot with the name `pdb1_wed_4_1201`:

```
ALTER PLUGGABLE DATABASE SNAPSHOT pdb1_wed_4_1201;
```

If the storage system supports sparse clones, then the preceding command creates a sparse copy. Otherwise, the command creates a full copy.

Every PDB snapshot is associated with a snapshot name and the SCN and timestamp at snapshot creation.

Creation of a PDB Clone with the USING SNAPSHOT Clause

A clone from a PDB snapshot is a full, standalone PDB. Unlike a snapshot copy PDB, which is based on a storage-managed snapshot, you do not need to materialize a clone created from a PDB snapshot.

To create a clone from a PDB snapshot, specify the `USING SNAPSHOT` clause of the `CREATE PLUGGABLE DATABASE` statement. For example, the following statement clones a PDB named `pdb1_copy` from the PDB-level snapshot named `pdb1_wed_4_1201`:

```
CREATE PLUGGABLE DATABASE pdb1_copy FROM pdb1
  USING SNAPSHOT pdb1_wed_4_1201;
```

> **See Also:**
>
> - "About Cloning PDBs from PDB Snapshots"
> - *Oracle Database Licensing Information User Manual* for details on which features are supported for different editions and services

Snapshot Copy PDBs

A **snapshot copy PDB** is based on a copy of the underlying storage system. Snapshot copy PDBs reduce the amount of storage required for testing purposes and reduce creation time significantly.

If the file system supports storage snapshots, then `CREATE PLUGGABLE DATABASE ... FROM ... SNAPSHOT COPY` copies a PDB from a source PDB, which can be read/write during the operation. The snapshot copy PDB files use copy-on-write technology. Only modified blocks require extra storage on disk. If the file system does *not* support storage snapshots or use Oracle Exadata sparse files, then the `CLONEDB` initialization parameter must be `true`, and the source PDB must be read-only for as long as the snapshot copy PDB exists.

Because a snapshot copy PDB depends on storage-managed snapshots, you cannot unplug a snapshot copy PDB from the CDB root or application root. You cannot drop the storage snapshot on which a snapshot copy PDB is based.

You can transform a snapshot copy PDB, which uses sparse files, into a full PDB. This process is known as **materializing** the snapshot copy PDB. Because a materialized PDB does not depend on the source PDB, you can drop it. Materialize a PDB by running the `ALTER PLUGGABLE DATABASE MATERIALIZE` command.

> **Note:**
>
> A PDB created with the `USING SNAPSHOT` clause and a PDB created with the `SNAPSHOT COPY` clause have different properties. You cannot specify both clauses in a single `CREATE PLUGGABLE DATABASE` command. The `CREATE PLUGGABLE DATABASE … FROM … USING SNAPSHOT` clause creates a full, standalone PDB that does not need to be materialized. The `CREATE PLUGGABLE DATABASE … FROM … SNAPSHOT COPY` clause creates a sparse PDB that must be materialized if you want to drop the storage-level snapshot on which it is based.

> **Note:**
>
> "Creating and Materializing Snapshot Copy PDBs"

Refreshable Clone PDBs

A **refreshable clone PDB** is a read-only clone that can periodically synchronize with its source PDB.

Depending on the value specified in the `REFRESH MODE` clause, synchronization occurs automatically or manually. For example, if `hrpdb_re_clone` is a clone of `hrpdb`, then every month you could manually refresh `hrpdb_re_clone` with changes from `hrpdb`. Alternatively, you could configure `hrpdb` to propagate changes to `hrpdb_re_clone` automatically every 24 hours.

You can switch the roles of a source PDB and its refreshable clone. This switchover can be useful for load balancing between CDBs, and when the source PDB suffers a failure.

> **Note:**
>
> "About Cloning a PDB or Non-CDB" to learn how to clone a PDB using the `REFRESH MODE` clause

Creation of a PDB by Plugging In

You can create a PDB by plugging in an unplugged PDB, or plugging in a non-CDB as a PDB.

Creation of a PDB by Plugging In an Unplugged PDB

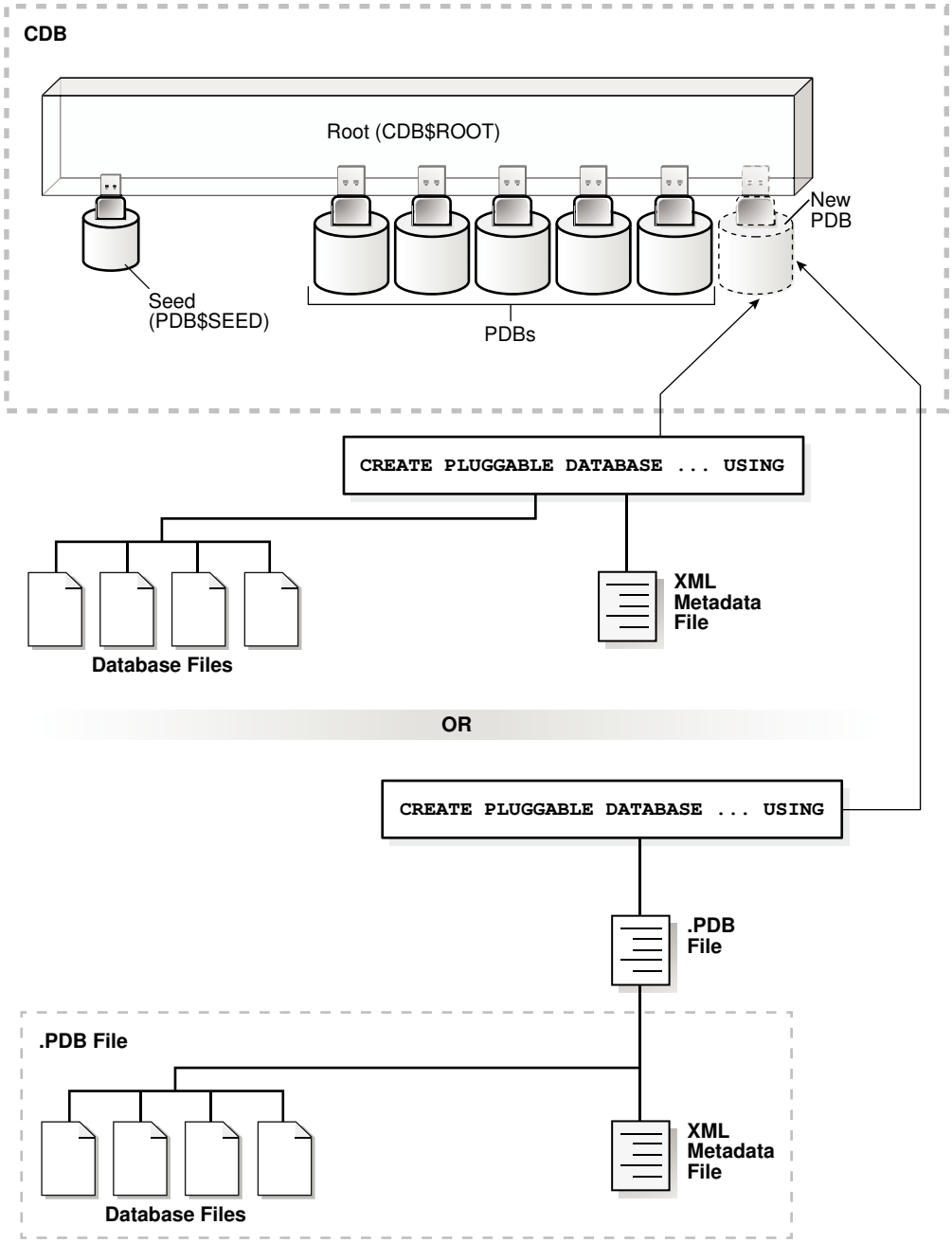
An **unplugged PDB** is a self-contained set of data files, and an XML metadata file that specifies the locations of the PDB files. To plug in an unplugged PDB, use the `CREATE PLUGGABLE DATABASE` statement with the `USING` clause.

When plugging in an unplugged PDB, you have the following options:

- Specify the XML metadata file that describes the PDB and the files associated with the PDB.
- Specify a PDB archive file, which is a compressed file that contains both the XML file and PDB data files. You can create a PDB by specifying the archive file, and thereby avoid copying the XML file and the data files separately.

The following graphic illustrates plugging in an unplugged PDB using the XML file.

Figure 1-9 Plugging In an Unplugged PDB



Example 1-6 Plugging In a PDB

The following SQL statement plugs in a PDB named `salespdb` based on the metadata stored in the named XML file, and specifies `NOCOPY` because the files of the unplugged PDB do not need to be moved to a new location:

```
CREATE PLUGGABLE DATABASE salespdb USING '/disk1/usr/salespdb.xml'
NOCOPY;
```

> **See Also:**
>
> "Plugging In an Unplugged PDB"

Creation of a PDB from a Non-CDB

You can move a non-CDB into a PDB.

You can accomplish this task in the following ways:

- Executing `DBMS_PDB.DESCRIBE` on a non-CDB in Oracle Database 12c

  You place a non-CDB in a transactionally consistent state, and then run the `DBMS_PDB.DESCRIBE` function to generate XML metadata about this database. While connected to the root in the CDB, you execute the `CREATE PLUGGABLE DATABASE` statement to create a PDB from the existing non-CDB. Finally, to convert the definitions in the PDB data dictionary to references to objects in `CDB$ROOT`, log in to the PDB and run the `noncdb_to_pdb.sql` script.

- Using Oracle Data Pump with or without transportable tablespaces

  You can define a data set on a non-CDB using Oracle Data Pump. This non-CDB can be in the current or a previous Oracle Database release, for example, Oracle Database 10g. You create an empty PDB in an existing CDB, and then use Oracle Data Pump to import the data set into the PDB.

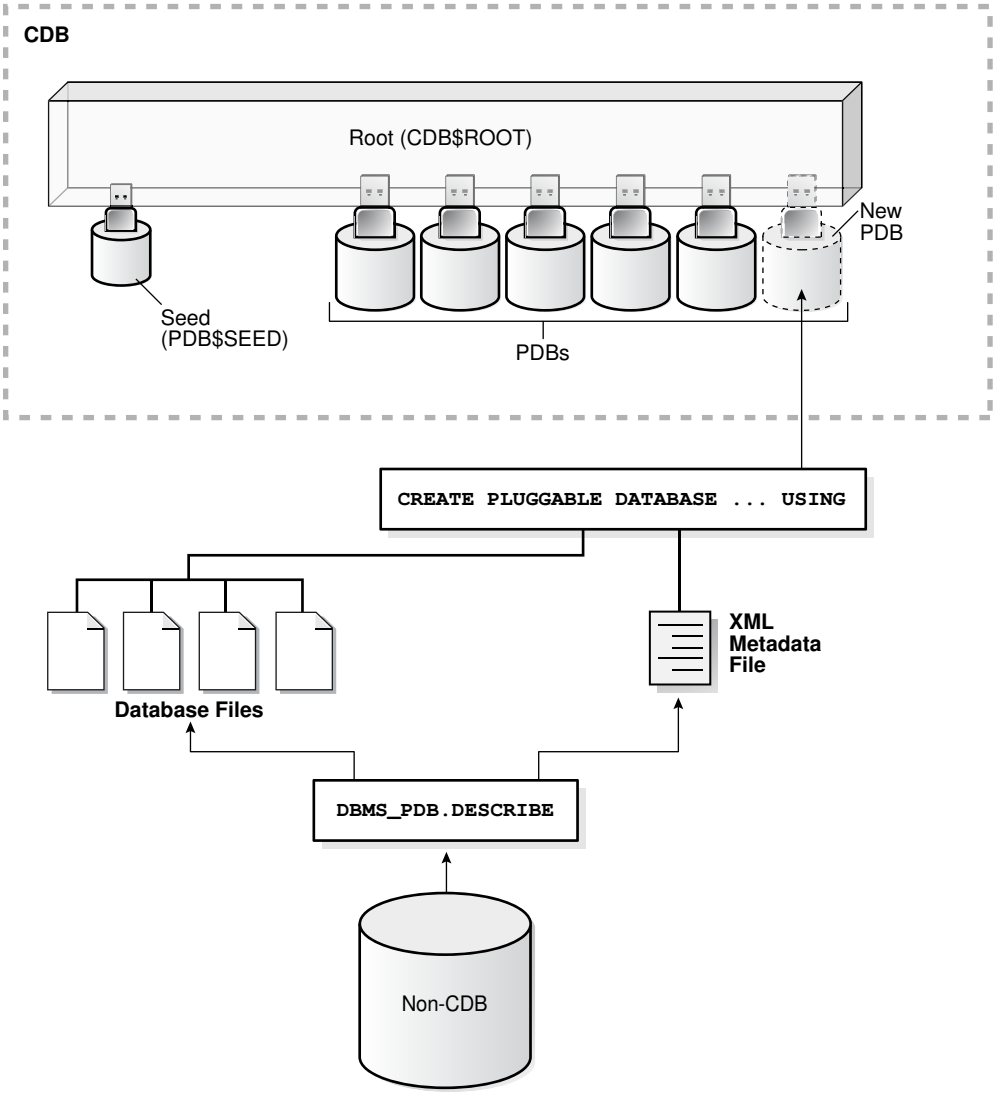
  A Full Transportable Export using Oracle Data Pump exports all objects and data necessary to create a complete copy of the database. Oracle Data Pump exports objects using direct path unload and external tables, and then imports objects using direct path `INSERT` and external tables. The Full Transportable dump file contains all objects in the database, not only table-related objects. Full Transportable Export is available starting in Oracle Database 11g Release 2 (11.2.0.3) for import into Oracle Database 12c and later.

- Using Oracle GoldenGate replication

  You replicate the data from the non-CDB to a PDB. When the PDB becomes current with the non-CDB, you switch over to the PDB.

The following figure illustrates running the `DBMS_PDB.DESCRIBE` function on a non-CDB, and then creating a PDB using the non-CDB files.

Figure 1-10 Creating a PDB from a Non-CDB



> **See Also:**
>
> - "Options for Creating a PDB from a Non-CDB"
> - *Oracle Database Concepts* for information about Oracle GoldenGate

Creation of a PDB by Relocating

To relocate a PDB from one CDB to another, use either the `CREATE PLUGGABLE DATABASE ... RELOCATE` statement or DBCA.

This technique has the following advantages:

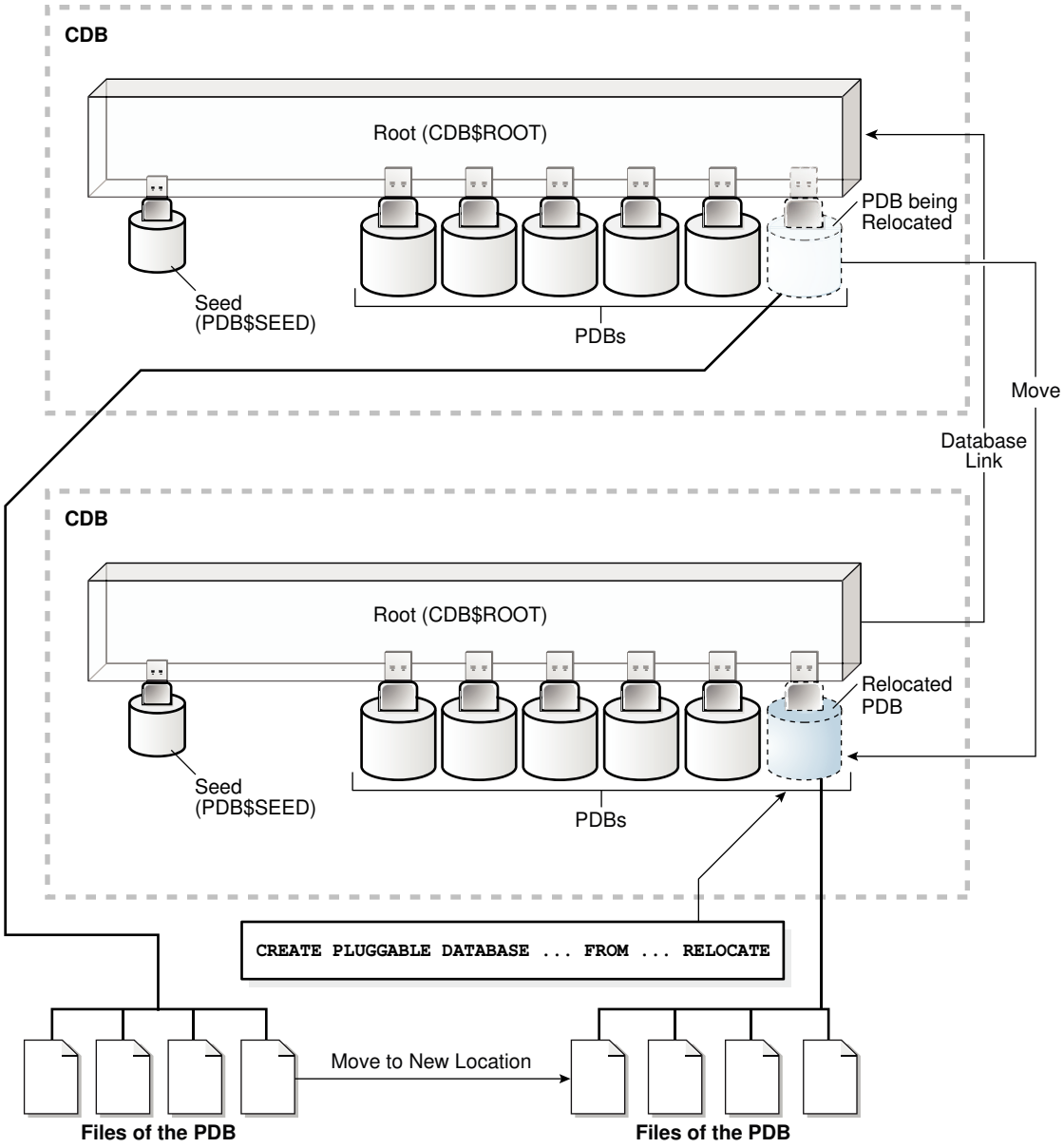
- The relocation occurs with minimal downtime.

- The technique keeps the PDB being relocated open in read/write mode during the relocation, and then brings the PDB online in its new location.

You must create a database link at the target CDB, which is the CDB that will contain the relocated PDB. Also, the source PDB must use local undo data.

The following graphic depicts a PDB relocation.

Figure 1-11 Relocating a PDB



Starting in Oracle Database 19c, you can relocate a remote PDB using DBCA in silent mode.

Example 1-7 PDB Relocation

The following statement, which is issued at a target CDB, relocates `hrpdb` from the source CDB to the target CDB:

```
CREATE PLUGGABLE DATABASE hrpdb FROM hrpdb@lnk_to_source RELOCATE;
```

> **See Also:**
>
> - "Overview of Tablespaces and Database Files in a CDB"
> - "Relocating a PDB"

Creation of a PDB as a Proxy PDB

A **proxy PDB** provides access to different PDB, called the **referenced PDB**, in a remote CDB.

Proxy PDBs enable you to aggregate data from multiple sources. A SQL statement submitted for execution in a proxy PDB executes within the referenced PDB.

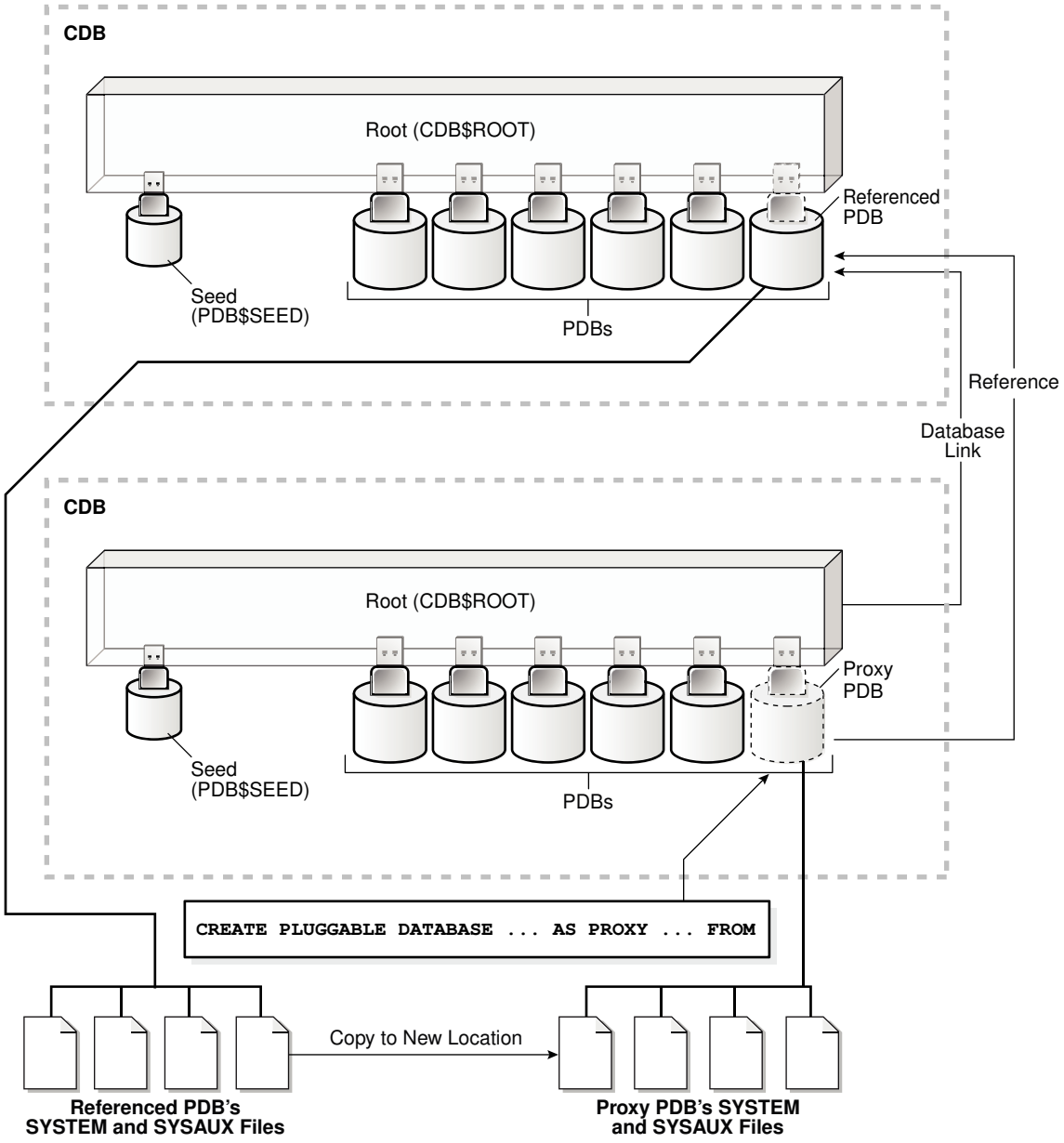
A typical use case is a proxy PDB that references an application root replica. If multiple CDBs have the same application definition (for example, same tables and PL/SQL packages), then you can create a proxy PDB in the application container of the master application root. The referenced PDB for the proxy PDB is the application root in a different CDB. By running installation scripts in the master root, the application roots in the other CDBs become replicas of the master application root.

To create a proxy PDB, use the `CREATE PLUGGABLE DATABASE` statement with the `FROM` clause, which must specify a database link to the referenced PDB in the remote CDB, and the `AS PROXY` clause.

> **Note:**
>
> If you plug a proxy PDB directly into `CDB$ROOT`, then you must have created the proxy in `CDB$ROOT`. A proxy of an application PDB must both be plugged in to an application root.

The following graphic shows the creation of a proxy PDB that references a PDB in a remote CDB.

Figure 1-12 Creating a Proxy PDB



Example 1-8 Creation of a Proxy PDB

This example creates a proxy PDB named `pdb1`. The referenced PDB is specified using a database link.

```
CREATE PLUGGABLE DATABASE pdb1 AS PROXY FROM pdb1@pdb1_link;
```

> **Note:**
>
> "Creating a PDB as a Proxy PDB"

Multitenant Environment Documentation Roadmap

This topic lists the most important topics for understanding and using CDBs, and includes cross-references to the appropriate documentation.

Table 1-2 Road map for the Multitenant Architecture Documentation

Category	Topic	Documentation
Concepts	Overview of CDBs and PDBs	"Overview of the Multitenant Architecture "
Administration	Creating and configuring a CDB	" Creating and Configuring a CDB"
Administration	Managing a CDB	" Administering a CDB"
Administration	Creating and configuring PDBs	"Creating and Removing PDBs and Application Containers"
Administration	Managing PDBs	" Administering PDBs"
Administration	Creating and removing application containers	"Creating and Removing Application Containers"
Administration	Administering application containers	"Administering Application Containers"
Performance	Troubleshooting PDBs	*Oracle Database Performance Tuning Guide*
Monitoring	Viewing information about CDBs and PDBs	"Monitoring CDBs and PDBs"
Backup and Recovery	Performing backup and recovery in a CDB	*Oracle Database Backup and Recovery User's Guide*
Security	Managing common users, roles, and privileges in a CDB	*Oracle Database Security Guide*
Miscellaneous	All other tasks relating to managing a CDB or PDB, including Oracle RAC, resource management, data transfer, and so on	This guide is the primary task-oriented intermediate and advanced documentation for managing CDBs. This guide also contains See Also links to books that cover different CDB topics. For example, *Oracle Database Utilities* explains concepts and tasks specific to PDBs when using Oracle Data Pump.

2
Overview of the Multitenant Architecture

This chapter describes the most important components of the multitenant architecture.

Overview of Containers in a CDB

A **container** is a collection of schemas, objects, and related structures in a **multitenant container database (CDB)**. Within a CDB, each container has a unique ID and name.

The CDB Root and System Container

The **CDB root**, also called simply *the root*, is a collection of schemas, schema objects, and nonschema objects to which all PDBs belong.

Every CDB has one and only one root container named `CDB$ROOT`. The root stores the system metadata required to manage PDBs. All PDBs belong to the root. The system container is the CDB root and all PDBs that belong to this root.

The CDB root does not store user data. Oracle recommends that you do *not* add common objects to the root or modify Oracle-supplied schemas in the root. However, you can create common users and roles for database administration. A common user with the necessary privileges can switch between containers.

Oracle recommends AL32UTF8 for the root character set. PDBs with different character sets can reside in the same CDB without requiring character set conversion.

Example 2-1 All Containers in a CDB

The following query, issued by an administrative user connected to the CDB root, lists all containers in the CDB (including the seed and CDB root), ordered by `CON_ID`.

```
COL NAME FORMAT A15
SELECT NAME, CON_ID, DBID, CON_UID, GUID
FROM   V$CONTAINERS ORDER BY CON_ID;

NAME          CON_ID       DBID    CON_UID GUID
------------- ------ ---------- ----------
--------------------------------
CDB$ROOT           1 1895287725          1
2003321EDD4F60D6E0534E40E40A41C5
PDB$SEED           2 2795386505 2795386505
200AC90679F07B55E05396C0E40A23FE
SAAS_SALES_AC      3 1239646423 1239646423
200B4CE0A8DC1D24E05396C0E40AF8EE
SALESPDB           4 3692549634 3692549634
200B4928319C1BCCE05396C0E40A2432
HRPDB              5 3784483090 3784483090
200B4928319D1BCCE05396C0E40A2432
```

> **See Also:**
>
> - "Common Users in a CDB"
> - " Overview of Configuring and Managing a Multitenant Environment"

PDBs

A **PDB** is a user-created set of schemas, objects, and related structures that appears logically to a client application as a separate database.

Every PDB is owned by `SYS`, regardless of which user created the PDB. `SYS` is a common user in the CDB.

Types of PDBs

All PDBs are user-created with the `CREATE PLUGGABLE DATABASE` statement except for `PDB$SEED`, which is Oracle-supplied.

You can create the following types of PDBs.

Standard PDB

This type of PDB results from running `CREATE PLUGGABLE DATABASE` *without* specifying the PDB as a seed, proxy PDB, or application root. Its capabilities depend on the container in which you create it:

- PDB plugged in to the CDB root

  This PDB belongs to the CDB root container and not an application container. This type of PDB cannot use application common objects. See "Application Common Objects".

- Application PDB

  An application PDB belongs to exactly one application container. Unlike PDBs plugged in to the CDB root, application PDBs can share a master application definition within an application container. For example, a `usa_zipcodes` table in an application root might be a data-linked common object, which means it contains data accessible by all application PDBs plugged in to this root. PDBs that do not reside within the application container cannot access its application common objects.

Application Root

Consider an application root as an application-specific root container. It serves as a repository for a master definition of an application back end, including common data and metadata. To create an application root, connect to the CDB root and specify the `AS APPLICATION CONTAINER` clause in a `CREATE PLUGGABLE DATABASE` statement. See "Application Root".

Seed PDBs

Unlike a standard PDB, a seed PDB is not intended to support an application. Rather, the seed is a *template* for the creation of PDBs that support applications. A seed can be either of the following:

- Seed PDB plugged in the CDB root (`PDB$SEED`)

  You can use this system-supplied template to create new PDBs either in an application container or the system container. The system container contains exactly one PDB seed. You cannot drop `PDB$SEED`, or add objects to or modify objects within it.

- Application seed PDB

  To accelerate creation of application PDBs within an application container, you can create an optional application seed. An application container contains either zero or one application seed.

  You create an application seed by connecting to the application container and executing the `CREATE PLUGGABLE DATABASE ... AS SEED` statement. See "Application Seed".

Proxy PDBs

A proxy PDB is a PDB that uses a database link to reference a PDB in a remote CDB. When you issue a statement in a proxy PDB while the PDB is open, the statement executes in the referenced PDB.

You must create a proxy PDB while connected to the CDB root or application root. You can alter or drop a proxy PDB just as you can a standard PDB.

> **See Also:**
>
> "Overview of PDB Creation"

Purpose of PDBs

For an application, a PDB is a self-contained, fully functional Oracle database. You can consolidate PDBs into a single CDB to achieve economies of scale, while maintaining isolation between PDBs.

You can use PDBs to achieve the following specific goals:

- Store data specific to an application

  For example, a sales application can have its own dedicated PDB, and a human resources application can have its own dedicated PDB. Alternatively, you can create an application container, which is a named collection of PDBs, to store an application back end containing common data and metadata (see "About Application Containers").

- Move data into a different CDB

  A database is "pluggable" because you can package it as a self-contained unit, called an unplugged PDB, and then move it into another CDB.

- Perform rapid upgrades

  You can unplug a PDB from CDB at a lower Oracle Database release, and then plug it in to a CDB at a higher release.

- Copy data quickly without loss of availability

  For testing and development, you can clone a PDB while it remains open, storing the clone in the same or a different CDB. Optionally, you can specify the PDB as a refreshable clone PDB. Alternatively, you use the Oracle-supplied seed PDB or a user-created application seed to copy new PDBs.

- Reference data in a different CDB

  You can create a proxy PDB that refers to a different PDB, either in the same CDB or in a separate CDB. When you issue statements in the proxy PDB, they execute in the referenced PDB.

- Isolate grants within PDBs

  A local or common user with appropriate privileges can grant `EXECUTE` privileges on a schema object to `PUBLIC` within an individual PDB.

> **See Also:**
>
> - "Benefits of the Multitenant Architecture"
> - *Oracle Database Security Guide* to learn how to grant roles and privileges in a CDB

Proxy PDBs

A **proxy PDB** refers to a remote PDB, called the **referenced PDB**.

Although you issue SQL statements in the proxy (referring) PDB, the statements execute in the referenced PDB. In this respect, a proxy PDB is loosely analogous to a symbolic link file in Linux.

Proxy PDBs provide the following benefits:

- Aggregate data from multiple application models

  Proxy PDBs enable you to build location-transparent applications that can aggregate data from multiple sources. These sources can be in the same data center or distributed across data centers.

- Enable an application root in one CDB to propagate application changes to a different application root

  Assume that CDBs `cdb_prod` and `cdb_test` have the same application model. You create a proxy PDB in an application container in `cdb_prod` that refers to an application root in `cdb_test`. When you run installation and upgrade scripts in the application root in `cdb_prod`, Oracle Database propagates these statements to the proxy PDB, which in turn sends them remotely to the application root in `cdb_test`. In this way, the application root in `cdb_test` becomes a replica of the application root in `cdb_prod`.

To create a proxy PDB, execute `CREATE PLUGGABLE DATABASE` with the `AS PROXY FROM` clause, where `FROM` specifies the referenced PDB name and a database link. The creation statement copies only the data files belonging to the `SYSTEM` and `SYSAUX` tablespaces.

Example 2-2 Creating a Proxy PDB

This example connects to the container `saas_sales_ac` in a local production CDB. The `sales_admin` common user creates a proxy PDB named `sales_sync_pdb`. This application PDB references an application root named `saas_sales_test_ac` in a remote development CDB, which it accesses using the `cdb_dev_rem` database link. When an application upgrade occurs in `saas_sales_ac` in the production CDB, the upgrade automatically propagates to the application root `saas_sales_test_ac` in the remote development CDB.

```
CONNECT sales_admin@saas_sales_ac
Password: ***********

CREATE PLUGGABLE DATABASE sales_sync_pdb AS PROXY FROM
saas_sales_test_ac@cdb_dev_rem;
```

> **Note:**
>
> "Creating a Proxy PDB"

Names for PDBs

Containers in a CDB share the same namespace, which means that they must have unique names within this namespace.

Names for the following containers must not conflict within the same CDB:

- The CDB root
- PDBs plugged in to the CDB root
- Application roots
- Application PDBs

For example, if the same CDB contains the application containers `saas_sales_ac` and `saas_sales_test_ac`, then two application PDBs that are both named `cust1` cannot simultaneously reside in both containers. The namespace rules also prevent creation of a PDB named `cust1pdb` in the CDB root and a PDB named `cust1pdb` in an application root.

Names for PDBs and application root containers must follow the same rules as net service names. Moreover, because a PDB or application root has a service with its own name, the container name must be unique across all CDBs whose services are exposed through a specific listener. The first character of a user-created container name must be alphanumeric, with remaining characters either alphanumeric or an underscore (_). Because service names are case-insensitive, container names are case-insensitive, and are in upper case even if specified using delimited identifiers.

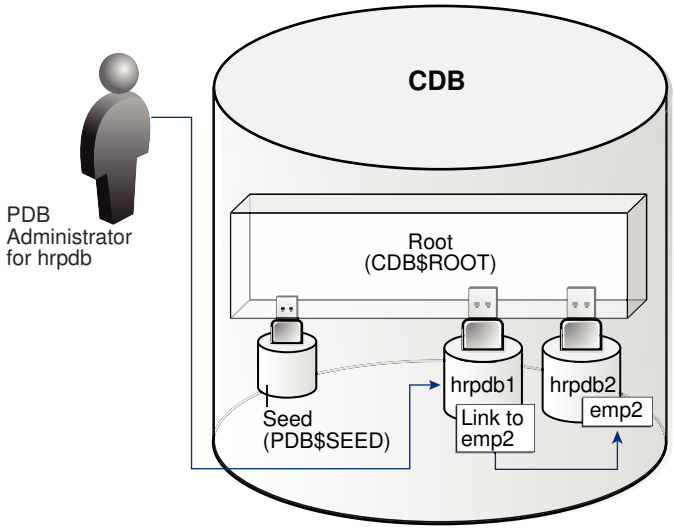
> **See Also:**
>
> *Oracle Database Net Services Reference* for the rules for service names

Database Links Between PDBs

By default, a user connected to one PDB must use database links to access objects in a different PDB. This behavior is directly analogous to a user in a non-CDB accessing objects in a different non-CDB.

Figure 2-1 Database Link Between PDBs

In this illustration, a PDB administrator is connected to the PDB named `hrpdb1`. By default, during this user session, `c##dba` cannot query the `emp2` table in `hrpdb2` without specifying a database link.



Exceptions to the rule include:

- A data-linked common object, which is accessible by all application PDBs that contain a data link that points to this object. For example, the application container `saas_sales_ac` might contain the data-linked table `usa_zipcodes` within its application. In this case, common CDB user `c##dba` can connect to an application PDB in this container, and then query `usa_zipcodes` even though the actual table resides in the application root. In this case, no database link is required.
- The `CONTAINERS()` clause in SQL issued from the CDB root or application root. Using this clause, you can query data across all PDBs plugged in to the root.

When creating a proxy PDB, you must specify a database link name in the `FROM` clause of the `CREATE PLUGGABLE DATABASE ... AS PROXY` statement. If the proxy PDB and referenced PDB reside in separate CDBs, then the database link must be defined in the root of the CDB that will contain the proxy PDB. The database link must connect either to the remote referenced PDB or to the CDB root of the remote CDB.

> **See Also:**
>
> - "Overview of Common and Local Objects in a CDB"
> - "About PDB Administration" to learn how to access objects in other PDBs using database links

Data Dictionary Architecture in a CDB

From the user and application perspective, the data dictionary in each container in a CDB is separate, as it would be in a non-CDB.

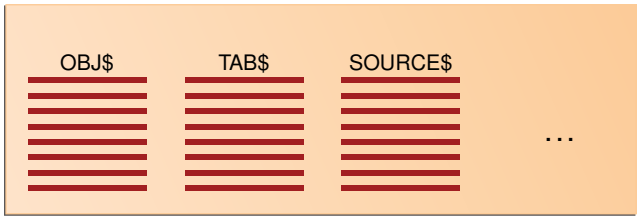
For example, the `DBA_OBJECTS` view in each PDB can show a different number of rows. This dictionary separation enables Oracle Database to manage the PDBs separately from each other and from the root.

Purpose of Data Dictionary Separation

In a newly created non-CDB that does not yet contain user data, the data dictionary contains only system metadata. For example, the `TAB$` table contains rows that describe only Oracle-supplied tables, for example, `TRIGGER$` and `SERVICE$`.

The following graphic depicts three underlying data dictionary tables, with the red bars indicating rows describing the system.

Figure 2-2 Unmixed Data Dictionary Metadata in a Non-CDB

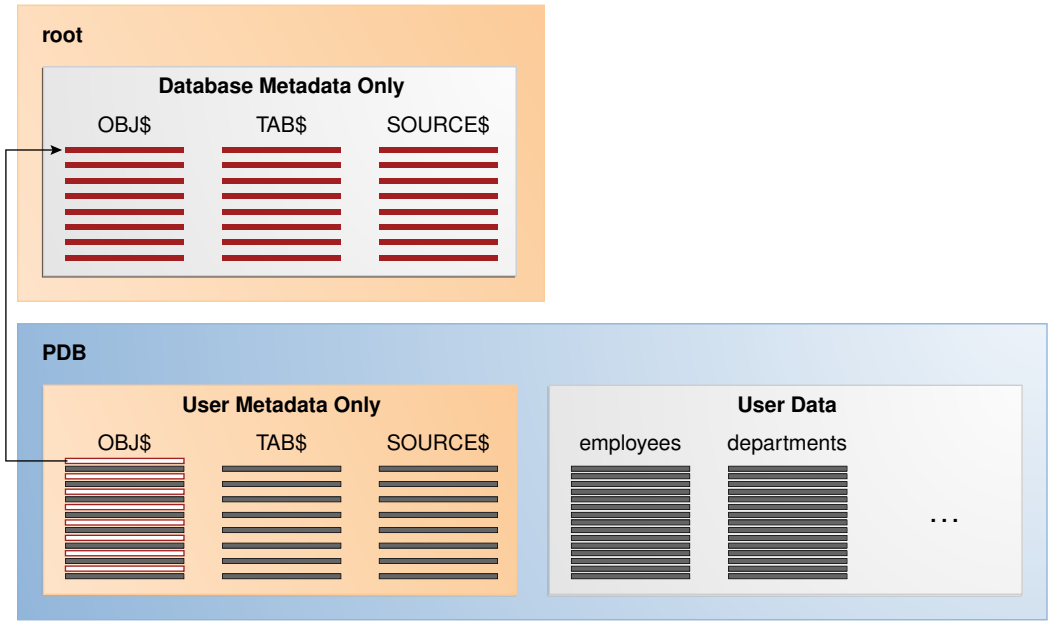


If users create their own schemas and tables in this non-CDB, then the data dictionary now contains some rows that describe Oracle-supplied entities, and other rows that describe user-created entities. For example, the `TAB$` dictionary table now has a row describing `employees` and a row describing `departments`.

Figure 2-3 Mixed Data Dictionary Metadata in a Non-CDB

In a CDB, the data dictionary metadata is split between the root and the PDBs. In the following figure, the `employees` and `departments` tables reside in a PDB. The data dictionary for this user data also resides in the PDB. Thus, the `TAB$` table in the PDB has a row for the `employees` table and a row for the `departments` table.

Figure 2-4 Data Dictionary Architecture in a CDB



The preceding graphic shows that the data dictionary in the PDB contains pointers to the data dictionary in the root. Internally, Oracle-supplied objects such as data dictionary table definitions and PL/SQL packages are represented *only* in the root. This architecture achieves two main goals within the CDB:

- Reduction of duplication

  For example, instead of storing the source code for the `DBMS_ADVISOR` PL/SQL package in every PDB, the CDB stores it only in `CDB$ROOT`, which saves disk space.

- Ease of database upgrade

  If the definition of a data dictionary table existed in every PDB, and if the definition were to change in a new release, then each PDB would need to be upgraded separately to capture the change. Storing the table definition only once in the root eliminates this problem.

Metadata and Data Links

The CDB uses an internal linking mechanism to separate data dictionary information.

Specifically, Oracle Database uses the following automatically managed pointers:

- Metadata links

  Oracle Database stores metadata about dictionary objects only in the CDB root. For example, the column definitions for the `OBJ$` dictionary table, which underlies the `DBA_OBJECTS` data dictionary view, exist only in the root. As depicted in

Figure 2-4, the OBJ$ table in each PDB uses an internal mechanism called a metadata link to point to the definition of OBJ$ stored in the root.

The *data* corresponding to a metadata link resides in its PDB, not in the root. For example, if you create table mytable in hrpdb and add rows to it, then the rows are stored in the PDB data files. The data dictionary views in the PDB and in the root contain different rows. For example, a new row describing mytable exists in the OBJ$ table in hrpdb, but not in the OBJ$ table in the CDB root. Thus, a query of DBA_OBJECTS in the CDB root and DBA_OBJECTS in hrdpb shows different results.

- Data links

  > **Note:**
  >
  > Data links were called *object links* in Oracle Database 12c Release 1 (12.1.0.2).

  In some cases, Oracle Database stores the data (not only metadata) for an object only once in the application root. An application PDB uses an internal mechanism called a data link to refer to the objects in the application root. The application PDB in which the data link was created also stores the data link description. A data link inherits the data type of the object to which it refers.

- Extended data link

  An extended data link is a hybrid of a data link and a metadata link. Like a data link, an extended data link refers to an object in an application root. However, the extended data link also refers to a corresponding object in the application PDB. Like a metadata link, the object in the application PDB inherits metadata from the corresponding object in the application root.

  When queried in the application root, an extended data-linked object fetches rows only from the application root. However, when queried in an application PDB, an extended data-linked object fetches rows from both the application root and application PDB.

Oracle Database automatically creates and manages metadata and data links to CDB$ROOT. Users cannot add, modify, or remove these links.

> **See Also:**
>
> - "Application Common Objects"
> - *Oracle Database Concepts* for an overview of the data dictionary

Container Data Objects in a CDB

A **container data object** is a table or view containing data pertaining to multiple containers or the whole CDB.

Container data privileges support a general requirement in which multiple PDBs reside in a single CDB, but with different local administration requirements. For example, if application DBAs do not want to administer locally, then they can grant container

data privileges on appropriate views to the common users. In this case, the CDB administrator can access the data for these PDBs. In contrast, PDB administrators who do not want the CDB administrator accessing their data do not grant container data privileges.

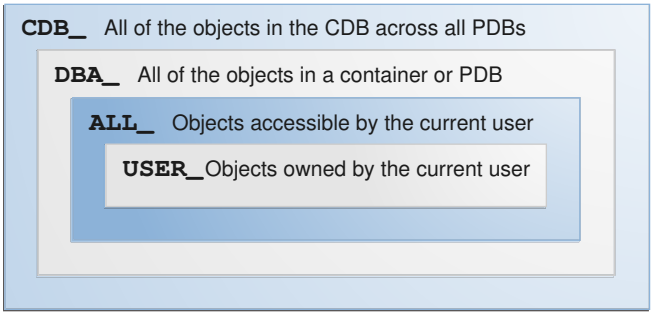
Examples of container data objects are Oracle-supplied views whose names begin with `V$` and `CDB_`. All container data objects have a `CON_ID` column. The following table shows the meaning of the values for this column.

Table 2-1 Container ID Values

Container ID	Rows pertain to
`0`	Whole CDB, or non-CDB
`1`	`CDB$ROOT`
`2`	`PDB$SEED`
All Other IDs	User-created PDBs, application roots, or application seeds

In a CDB, for every `DBA_` view, a corresponding `CDB_` view exists. The owner of a `CDB_` view is the owner of the corresponding `DBA_` view. The following graphic shows the relationship among the different categories of dictionary views:

Figure 2-5 Dictionary Views in a CDB



When the current container is a PDB, a user can view data dictionary information for the current PDB only. To an application connected to a PDB, the data dictionary appears as it would for a non-CDB. When the current container is the root, however, a common user can query `CDB_` views to see metadata for the root and for PDBs for which this user is privileged.

> **Note:**
>
> When queried from the root container, `CDB_` and `V$` views implicitly convert data to the AL32UTF8 character set. If a character set needs more bytes to represent a character when converted to AL32UTF8, and if the view column width cannot accommodate data from a specific PDB, then data truncation is possible.

The following table shows a scenario involving queries of CDB_ views. Each row describes an action that occurs after the action in the preceding row.

Table 2-2 Querying CDB_ Views

Operation	Description

`SQL> CONNECT SYSTEM`
`Enter password: ********`
`Connected.`
	The SYSTEM user, which is common to all containers in the CDB, connects to the root (see "Common Users in a CDB").

`SQL> SELECT COUNT(*) FROM CDB_USERS`
`WHERE CON_ID=1;`
 	
`COUNT(*)`
`--------`
`41`
	SYSTEM queries CDB_USERS to obtain the number of common users in the CDB. The output indicates that 41 common users exist.

`SQL> SELECT COUNT(DISTINCT(CON_ID))`
`FROM CDB_USERS;`
 	
`COUNT(DISTINCT(CON_ID))`
`----------------------`
`4`
	SYSTEM queries CDB_USERS to determine the number of distinct containers in the CDB.

`SQL> CONNECT SYSTEM@hrdb`
`Enter password: ********`
`Connected.`
	The SYSTEM user now connects to the PDB named hrpdb.

`SQL> SELECT COUNT(*) FROM CDB_USERS;`
 	
`COUNT(*)`
`----------`
`45`
	SYSTEM queries CDB_USERS. The output indicates that 45 users exist. Because SYSTEM is not connected to the root, the CDB_USERS view shows the same output as DBA_USERS. Because DBA_USERS only shows the users in the *current* container, it shows 45.

See Also:

"About CDB and Container Information in Views" to learn more about container data objects

Data Dictionary Storage in a CDB

The data dictionary that stores the metadata for the CDB as a whole is stored only in the system tablespaces.

The data dictionary that stores the metadata for a specific PDB is stored in the self-contained tablespaces dedicated to this PDB. The PDB tablespaces contain both the data and metadata for an application back end. Thus, each set of data dictionary tables is stored in its own dedicated set of tablespaces.

> **See Also:**
>
> - *Oracle Database Concepts* for an overview of the data dictionary
> - "Overview of Tablespaces and Database Files in a CDB"

Current Container

For a given session, the current container is the one in which the session is running. The current container can be the CDB root, an application root, or a PDB.

Each session has exactly one current container at any point in time. Because the data dictionary in each container is separate, Oracle Database uses the data dictionary in the current container for name resolution and privilege authorization.

> **See Also:**
>
> "About the Current Container"

Cross-Container Operations

A **cross-container operation** is a DDL or DML statement that affects multiple containers at once.

Only a common user connected to either the CDB root or an application root can perform cross-container operations. A cross-container operation can affect:

- The CDB itself
- Multiple containers within a CDB
- Multiple phenomena such as common users or common roles that are represented in multiple containers
- A container to which the user issuing the DDL or DML statement is currently not connected

Examples of cross-container DDL operations include user `SYSTEM` granting a privilege commonly to another common user (see "Roles and Privileges Granted Commonly in a CDB"), and an `ALTER DATABASE . . . RECOVER` statement that applies to the entire CDB.

When you are connected to either the CDB root or an application root, you can execute a single DML statement to modify tables or views in multiple PDBs within the container. The database infers the target PDBs from the value of the `CON_ID` column specified in the DML statement. If no `CON_ID` is specified, then the database uses the `CONTAINERS_DEFAULT_TARGET` property specified by the `ALTER PLUGGABLE DATABASE CONTAINERS DEFAULT TARGET` statement.

Example 2-3 Updating Multiple PDBs in a Single DML Statement

In this example, your goal is to set the `country_name` column to the value `USA` in the `sh.sales` table. This table exists in two separate PDBs, with container IDs of `7` and `8`. Both PDBs are in the application container named `saas_sales_ac`. You can connect to the application root as an administrator, and make the update as follows:

```
CONNECT sales_admin@saas_sales_ac
Password: *******

UPDATE CONTAINERS(sh.sales) sal
  SET sal.country_name = 'USA'
  WHERE sal.CON_ID IN (7,8);
```

In the preceding `UPDATE` statement, `sal` is an alias for `CONTAINERS(sh.sales)`.

> **See Also:**
>
> - "Common Users in a CDB"
> - " Administering a CDB"

Overview of Commonality in the CDB

In a CDB, every user, role, or object is either common or local. Similarly, a privilege is granted either commonly or locally.

About Commonality in a CDB

A common phenomenon defined in a CDB or application root is the same in all containers plugged in to this root.

Principles of Commonality

In a CDB, a phenomenon can be common within either the system container (the CDB itself), or within a specific application container.

For example, if you create a common user account while connected to `CDB$ROOT`, then this user account is common to all PDBs and application roots in the CDB. If you create an application common user account while connected to an application root, however, then this user account is common only to the PDBs in this application container.

Within the context of CDB$ROOT or an application root, the principles of commonality are as follows:

- A common phenomenon is the same in every existing and future container.

  Therefore, a common user defined in the CDB root has the same identity in every PDB plugged in to the CDB root; a common user defined in an application root has the same identity in every application PDB plugged in to this application root. In contrast, a local phenomenon is scoped to exactly one existing container.
- Only a common user can alter the existence of common phenomena.

  More precisely, only a common user logged in to either the CDB root or an application root can create, destroy, or modify attributes of a user, role, or object that is common to the current container.

Namespaces in a CDB

In a CDB, the namespace for every object is scoped to its container.

The following principles summarize the scoping rules:

- From an application perspective, a PDB is indistinguishable from a non-CDB.
- Local phenomena are created within and restricted to a single container.

> **Note:**
>
> In this topic, the word “phenomenon” means “user account, role, or database object.”

- Common phenomena are defined in a CDB root or application root, and exist in all PDBs that are or will be plugged into this root.

The preceding principles have implications for local and common phenomena.

Local Phenomena

A local phenomenon must be uniquely named *within* a container, but not across all containers in the CDB. Identically named local phenomena in different containers are distinct. For example, local user sh in one PDB does not conflict with local user sh in another PDB.

CDB$ROOT Common Phenomena

Common phenomena defined in CDB$ROOT exist in multiple containers and must be unique within each of these namespaces. For example, the CDB root includes predefined common users such as SYSTEM and SYS. To ensure namespace separation, Oracle Database prevents creation of a SYSTEM user within another container.

To ensure namespace separation, the name of user-created common phenomena in the CDB root must begin with the value specified by the COMMON_USER_PREFIX initialization parameter. The default prefix is c## or C##. The names of all *other* user-created phenomena must *not* begin with c## or C##. For example, you cannot create a local user in hrpdb named c##hr, nor can you create a common user in the CDB root named hr.

Application Common Phenomena

Within an application container, names for local and application common phenomena must not conflict.

- Application common users and roles

  The same principles apply to application common users as to CDB common users. The difference is that for CDB common users, the default value for the common user prefix is `c##` or `C##`, whereas in application root the default value for the common user prefix is the empty string.

  The multitenant architecture assumes that you create application PDBs from an application root, or convert a single-tenant application to a multitenant application.

- Application common objects

  The multitenant architecture assumes that you create application common objects in the application root. Later, you add data locally within the application PDBs. However, Oracle Database supports creation of *local* tables within an application PDB. In this case, the local tables reside in the same namespace as application common objects within the application PDB.

> **See Also:**
>
> *Oracle Database Security Guide* to learn more about common users and roles

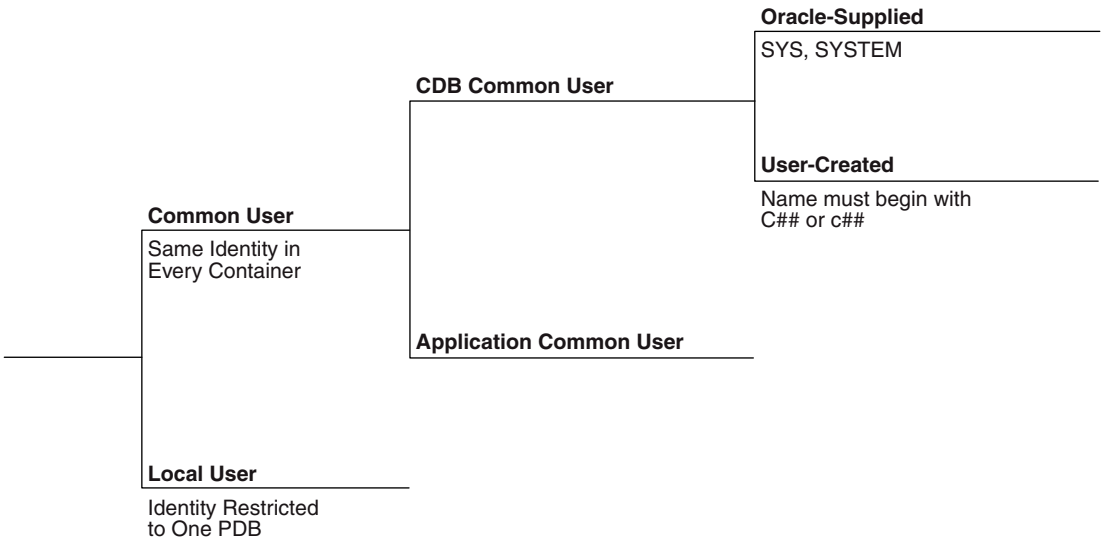
Overview of Common and Local Users in a CDB

If a user account owns objects that define the database, then this user account is common. User accounts that are *not* Oracle-supplied are either local or common.

A CDB common user is a common user that is created in the CDB root. An application common user is a user that is created in an application root, and is common only within this application container.

The following graphic shows the possible user account types in a CDB.

Figure 2-6 User Accounts in a CDB



A CDB common user can connect to *any* container in the CDB to which it has sufficient privileges. In contrast, an application common user can only connect to the application root in which it was created, or a PDB that is plugged in to this application root, depending on its privileges.

> **See Also:**
>
> *Oracle Database Security Guide* for an overview of common and local users

Common Users in a CDB

Within the context of either the system container (CDB) or an application container, a **common user** is a database user that has the same identity in the root and in every existing and future PDB within this container.

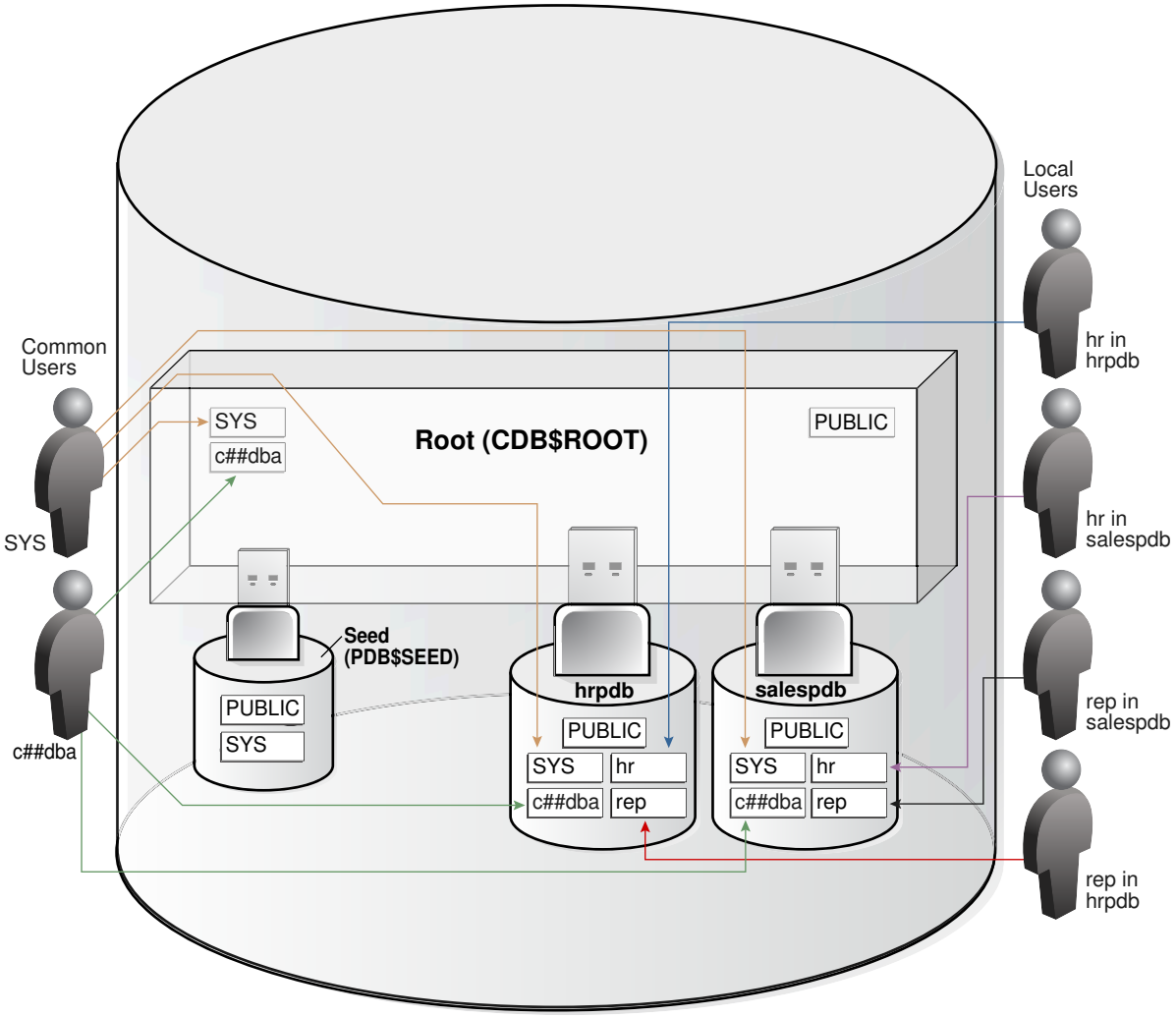
Every common user can connect to and perform operations within the root of its container, and within any PDB in which it has sufficient privileges. Some administrative tasks must be performed by a common user. Examples include creating a PDB and unplugging a PDB.

For example, `SYSTEM` is a CDB common user with DBA privileges. Thus, `SYSTEM` can connect to the CDB root and any PDB in the database. You might create a common user `saas_sales_admin` in the `saas_sales` application container. In this case, the `saas_sales_admin` user could *only* connect to the `saas_sales` application root or to an application PDB within the `saas_sales` application container.

Every common user is either Oracle-supplied or user-created. Examples of Oracle-supplied common users are `SYS` and `SYSTEM`. Every user-created common user is either a CDB common user, or an application common user.

Figure 2-7 shows sample users and schemas in two PDBs: hrpdb and salespdb. SYS and c##dba are CDB common users who have schemas in CDB$ROOT, hrpdb, and salespdb. Local users hr and rep exist in hrpdb. Local users hr and rep also exist in salespdb.

Figure 2-7 Users and Schemas in a CDB



Common users have the following characteristics:

- A common user can log in to any container (including CDB$ROOT) in which it has the CREATE SESSION privilege.

  A common user need not have the same privileges in every container. For example, the c##dba user may have the privilege to create a session in hrpdb and in the root, but *not* to create a session in salespdb. Because a common user with the appropriate privileges can switch between containers, a common user in the root can administer PDBs.

- An application common user does not have the CREATE SESSION privilege in any container outside its own application container.

Thus, an application common user is restricted to its own application container. For example, the application common user created in the `saas_sales` application can connect only to the application root and the PDBs in the `saas_sales` application container.

- The names of user-created CDB common users must follow the naming rules for other database users. Additionally, the names must begin with the characters specified by the `COMMON_USER_PREFIX` initialization parameter, which are `c##` or `C##` by default. Oracle-supplied common user names and user-created application common user names do not have this restriction.

  No local user name may begin with the characters `c##` or `C##`.

- Every common user is uniquely named across all PDBs within the container (either the system container or a specific application container) in which it was created.

  A CDB common user is defined in the CDB root, but must be able to connect to every PDB with the same identity. An application common user resides in the application root, and may connect to every application PDB *in its container* with the same identity.

> **See Also:**
>
> - *Oracle Database Security Guide* to learn about common user accounts
> - *Oracle Database Reference* to learn about `COMMON_USER_PREFIX`

Local Users in a CDB

A **local user** is a database user that is not common and can operate only within a single PDB.

Local users have the following characteristics:

- A local user is specific to a PDB and may own a schema in this PDB.

  In Figure 2-7, local user `hr` on `hrpdb` owns the `hr` schema. On `salespdb`, local user `rep` owns the `rep` schema, and local user `hr` owns the `hr` schema.

- A local user can administer a PDB, including opening and closing it.

  A common user with `SYSDBA` privileges can grant `SYSDBA` privileges to a local user. In this case, the privileged user remains local.

- A local user in one PDB cannot log in to another PDB or to the CDB root.

  For example, when local user `hr` connects to `hrpdb`, `hr` cannot access objects in the `sh` schema that reside in the `salespdb` database without using a database link. In the same way, when local user `sh` connects to the `salespdb` PDB, `sh` cannot access objects in the `hr` schema that resides in `hrpdb` without using a database link.

- The name of a local user must not begin with the characters `c##` or `C##`.

- The name of a local user must only be unique within its PDB.

  The user name and the PDB in which that user schema is contained determine a unique local user. Figure 2-7 shows that a local user and schema named `rep` exist

on hrpdb. A completely independent local user and schema named rep exist on the salespdb PDB.

The following table describes a scenario involving the CDB in Figure 2-7. Each row describes an action that occurs after the action in the preceding row. Common user SYSTEM creates local users in two PDBs.

Table 2-3 Local Users in a CDB

Operation	Description

`SQL> CONNECT SYSTEM@hrpdb`
`Enter password: ********`
`Connected.`
	SYSTEM connects to the hrpdb container using the service name hrpdb.

`SQL> CREATE USER rep IDENTIFIED BY password;`
 	
`User created.`
 	
`SQL> GRANT CREATE SESSION TO rep;`
 	
`Grant succeeded.`
	SYSTEM now creates a local user rep and grants the CREATE SESSION privilege in this PDB to this user. The user is local because common users can only be created by a common user connected to the root.

`SQL> CONNECT rep@salespdb`
`Enter password: *******`
`ERROR:`
`ORA-01017: invalid username/password; logon denied`
	The rep user, which is local to hrpdb, attempts to connect to salespdb. The attempt fails because rep does not exist in PDB salespdb. This behavior mimics the behavior of non-CDBs. A user account on one non-CDB is independent of user accounts on a different non-CDB.

`SQL> CONNECT SYSTEM@salespdb`
`Enter password: ********`
`Connected.`
	SYSTEM connects to the salespdb container using the service name salespdb.

`SQL> CREATE USER rep IDENTIFIED BY password;`
 	
`User created.`
 	
`SQL> GRANT CREATE SESSION TO rep;`
 	
`Grant succeeded.`
	SYSTEM creates a local user rep in salespdb and grants the CREATE SESSION privilege in this PDB to this user. Because the name of a local user must only be unique within its PDB, a user named rep can exist in both salespdb and hrpdb.

`SQL> CONNECT rep@salespdb`
`Enter password: *******`
`Connected.`
	The rep user successfully logs in to salespdb.

> **See Also:**
>
> *Oracle Database Security Guide* to learn about local user accounts

Overview of Common and Local Roles in a CDB

User-created roles are either local or common. Common roles are either common to the CDB itself or to a specific application container.

Every Oracle-supplied role is common, for example, the predefined `DBA` role. In Oracle-supplied scripts, every privilege or role granted to Oracle-supplied users and roles is granted commonly, with one exception: system privileges are granted locally to the common role `PUBLIC`.

> **See Also:**
>
> "Grants to PUBLIC in a CDB"

Common Roles in a CDB

A **common role** exists either in the CDB root or an application root, and applies to every PDB within the root container (either the CDB or the application container).

Common roles are useful for cross-container operations, ensuring that a common user has a role in every PDB. Every common role is one of the following types:

- Oracle-supplied

  All Oracle-supplied roles, such as `DBA` and `PUBLIC`, are common to the CDB.

- User-created

  Create a common role by executing `CREATE ROLE ... CONTAINER=ALL` in either the CDB root or application root, which determines the container to which the role is common. The standard naming conventions apply. Additionally, the names of CDB common roles must begin with the characters specified by the `COMMON_USER_PREFIX` initialization parameter, which are `c##` or `C##` by default.

The scope of the role is the scope of the root within which it is defined. If you define the role in `CDB$ROOT`, then its scope is the entire CDB. If you define the role within application root, then its scope is the application container.

> **See Also:**
>
> - "Cross-Container Operations"
> - *Oracle Database Security Guide* to learn how to manage common roles
> - *Oracle Database SQL Language Reference* to learn about the `CREATE ROLE` statement

Local Roles in a CDB

A **local role** exists only in a single PDB, just as a role in a non-CDB exists only in the non-CDB.

A local role can only contain roles and privileges that apply within the container in which the role exists. For example, if you create the local role `pdbadmin` in `hrpdb`, then the scope of this role is restricted to this PDB.

PDBs in the same CDB, or in the same application container, may contain local roles with the same name. For example, the user-created role `pdbadmin` may exist in both `hrpdb` and `salespdb`. However, these roles are completely independent of each other, just as they would be in separate non-CDBs.

> **See Also:**
>
> *Oracle Database Security Guide* to learn how to manage local roles

Overview of Privilege and Role Grants in a CDB

Just as in a non-CDB, users in a CDB can grant and be granted roles and privileges. Roles and privileges in a CDB, however, are either locally or commonly granted.

A privilege or role granted locally is exercisable only in the PDB in which it was granted. A privilege or role granted commonly is exercisable in every existing and future PDB in the container—either the CDB or an application container—in which it was granted.

Users and roles may be common or local. However, a privilege is *in itself* neither common nor local. If a user grants a privilege locally using the `CONTAINER=CURRENT` clause, then the grantee has a privilege exercisable only in the current container. If a user connects to either the CDB root or an application root, and if this user grants a privilege commonly using the `CONTAINER=ALL` clause, then the grantee has this privilege in any existing or future PDB within the current container.

> **See Also:**
>
> *Oracle Database Security Guide* to learn how to manage common privileges

Principles of Privilege and Role Grants in a CDB

In a CDB, every act of granting, whether local or common, occurs within a container. The container may be the CDB root, an application root, or a PDB.

If the current container is the CDB root, then granting commonly means granting to all containers in the CDB. If the current container is an application root, however, then granting commonly means granting to all PDBs in the current application container.

The basic principles of granting are as follows:

- Both common and local phenomena may grant and be granted locally.
- Only common phenomena may grant or be granted commonly.

Local users, roles, and privileges are restricted to a particular PDB. Thus, local users may not grant roles and privileges commonly, and local roles and privileges may not be granted commonly.

The following sections describe the implications of the preceding principles.

Privileges and Roles Granted Locally in a CDB

Roles and privileges may be granted locally to users and roles *regardless* of whether the grantees, grantors, or roles being granted are local or common.

The following table explains the valid possibilities for locally granted roles and privileges.

Table 2-4 Local Grants

Phenomenon	May Grant Locally	May Be Granted Locally	May Receive a Role or Privilege Granted Locally
Common User	Yes	N/A	Yes
Local User	Yes	N/A	Yes
Common Role	N/A	Yes[1]	Yes
Local Role	N/A	Yes[2]	Yes
Privilege	N/A	Yes	N/A

1 Privileges in this role are available to the grantee only in the container in which the role was granted, regardless of whether the privileges were granted to the role locally or commonly.
2 Privileges in this role are available to the grantee only in the container in which the role was granted and created.

What Makes a Privilege or Role Grant Local

To grant a role or privilege locally, use the `GRANT` statement with the `CONTAINER=CURRENT` clause, which is the default.

Specifically, a role or privilege is granted locally only when the following criteria are met:

- The grantor has the necessary privileges to grant the specified role or privileges.

  For system roles and privileges, the grantor must have the `ADMIN OPTION` for the role or privilege being granted. For object privileges, the grantor must have the `GRANT OPTION` for the privilege being granted.

- The grant applies to only one container.

  By default, the `GRANT` statement includes the `CONTAINER=CURRENT` clause, which indicates that the privilege or role is granted locally.

Example 2-4 Granting a Privilege Locally

In this example, both `SYSTEM` and `c##hr_admin` are common users. The example connects to `hrpdb` as `SYSTEM` (which has administrator privileges), and then locally

grants read privileges on the `employees` table to `c##hr_admin`. This grant applies *only* to `c##hr_admin` within `hrpdb`, not within any other PDBs.

```
CONNECT SYSTEM@hrpdb
Enter password: password
Connected.

GRANT READ ON employees TO c##hr_admin CONTAINER=CURRENT;
```

> **See Also:**
>
> *Oracle Database Security Guide* to learn more about granting local roles and privileges

Roles and Privileges Granted Locally

A user or role may be locally granted a *privilege* (`CONTAINER=CURRENT`).

For example, a `READ ANY TABLE` privilege granted locally to a local or common user in `hrpdb` applies only to *this* user in *this* PDB. Analogously, the `READ ANY TABLE` privilege granted to user `hr` in a non-CDB has no bearing on the privileges of an `hr` user that exists in a separate non-CDB.

A user or role may be locally granted a *role* (`CONTAINER=CURRENT`). As shown in Table 2-4, a *common* role may receive a privilege granted *locally*. For example, the common role `c##dba` may be granted the `READ ANY TABLE` privilege locally in `hrpdb`. If the `c##cdb` common role is granted locally, then privileges in the role apply *only* in the container in which the role is granted. In this example, a common user who has the `c##cdba` role does not, because of a privilege granted locally to this role in `hrpdb`, have the right to exercise this privilege in any PDB other than `hrpdb`.

> **See Also:**
>
> *Oracle Database Security Guide* to learn how to grant roles and privileges in a CDB

Roles and Privileges Granted Commonly in a CDB

Privileges and common roles may be granted commonly.

User accounts or roles may be granted roles and privileges commonly only if the grantees and grantors are both *common*. If a role is being granted commonly, then the role itself must be common. The following table explains the possibilities for common grants.

Table 2-5 Common Grants

Phenomenon	May Grant Commonly	May Be Granted Commonly	May Receive Roles and Privileges Granted Commonly
Common User Account	Yes	N/A	Yes
Local User Account	No	N/A	No
Common Role	N/A	Yes[1]	Yes
Local Role	N/A	No	No
Privilege	N/A	Yes	N/A

1 Privileges that were granted commonly to a common role are available to the grantee across all containers. In addition, any privilege granted locally to a common role is available to the grantee only in the container in which that privilege was granted to the common role.

See Also:

Oracle Database Security Guide to learn more about common grants

What Makes a Grant Common

The `CONTAINER=ALL` clause specifies that the privilege or role is being granted commonly.

A role or privilege is granted commonly when the following criteria are met:

- The grantor is a common user.

  The user that performs the grant is either common to the CDB itself, or common to a specific application container.

- The grantee is a common user or common role.

  The recipient of the grant is either common to the CDB itself, or common to a specific application container.

- The grantor has the necessary privileges to grant the specified role or privileges.

  For system roles and privileges, the grantor must have the `ADMIN OPTION` for the role or privilege being granted. For object privileges, the grantor must have the `GRANT OPTION` for the privilege being granted.

- The grant applies to all PDBs within the container (either CDB or application container) in which the grant occurred.

  The `GRANT` statement includes a `CONTAINER=ALL` clause specifying that the privilege or role is granted commonly.

- If a role is being granted, then it must be common, and if an object privilege is being granted, then the object on which the privilege is granted must be common.

Example 2-5 Granting a Privilege Commonly

In this example, both SYSTEM and c##hr_admin are common users. SYSTEM connects to the CDB root, and then grants the CREATE ANY TABLE privilege commonly to c##hr_admin. In this case, c##hr_admin can now create a table in any PDB in the CDB.

```
CONNECT SYSTEM@root
Enter password: password
Connected.

GRANT CREATE ANY TABLE TO c##hr_admin CONTAINER=ALL;
```

> **See Also:**
>
> *Oracle Database Security Guide* to learn how to grant common privileges

Roles and Privileges Granted Commonly

A common user account or role may be granted a *privilege* commonly (CONTAINER=ALL).

Within the context of either the CDB root or an application root, the privilege is granted to this common user account or role in all existing and future PDBs within the current container. For example, if SYSTEM connects to the CDB root and grants a SELECT ANY TABLE privilege commonly to CDB common user account c##dba, then the c##dba user has this privilege in all PDBs in the CDB. A role or privilege granted *commonly* cannot be revoked *locally*.

A user or role may receive a common role granted commonly. As mentioned in a footnote on Table 2-5, a common role may receive a privilege granted locally. Thus, a common user can be granted a common role, and this role may contain locally granted privileges.

For example, the common role c##admin may be granted the SELECT ANY TABLE privilege that is local to hrpdb. Locally granted privileges in a common role apply *only* in the container in which the privilege was granted. Thus, the common user with the c##admin role does not have the right to exercise an hrpdb-contained privilege in salespdb or any PDB other than hrpdb.

> **See Also:**
>
> *Oracle Database Security Guide* to learn how to grant roles and privileges in a CDB

Grants to PUBLIC in a CDB

In a CDB, PUBLIC is a common role. In a PDB, privileges granted locally to PUBLIC enable all local and common user account to exercise these privileges in this PDB only.

Every privilege and role granted to Oracle-supplied users and roles is granted commonly except for system privileges granted to PUBLIC, which are granted locally. This exception exists because you may want to revoke some grants included by default in Oracle Database, such as EXECUTE on the SYS.UTL_FILE package.

Assume that local user account hr exists in hrpdb. This user locally grants the SELECT privilege on hr.employees to PUBLIC. Common and local users in hrpdb may exercise the privilege granted to PUBLIC. User accounts in salespdb or any other PDB do not have the privilege to query hr.employees in hrpdb.

Privileges granted commonly to PUBLIC enable all local users to exercise the granted privilege in their respective PDBs and enable all common users to exercise this privilege in the PDBs to which they have access. Oracle recommends that users do not commonly grant privileges and roles to PUBLIC.

> **See Also:**
>
> *Oracle Database Security Guide* to learn how the PUBLIC role works in a multitenant environment

Grants of Privileges and Roles: Scenario

In this scenario, SYSTEM creates common user c##dba and tries to give this user privileges to query a table in the hr schema in hrpdb.

The scenario shows how the CONTAINER clause affects grants of roles and privileges. The first column shows operations in CDB$ROOT. The second column shows operations in hrpdb.

Table 2-6 Granting Roles and Privileges in a CDB

t	Operations in CDB$ROOT	Operations in hrpdb	Explanation

t1	`SQL> CONNECT SYSTEM@root`
	`Enter password: *******`
	`Connected.`
		n/a	Common user SYSTEM connects to the root container.

t2	`SQL> CREATE USER c##dba`
	`IDENTIFIED BY password`
	`CONTAINER=ALL;`
		n/a	SYSTEM creates common user c##dba. The clause CONTAINER=ALL makes the user a common user.

Table 2-6 (Cont.) Granting Roles and Privileges in a CDB

t	Operations in CDB$ROOT	Operations in hrpdb	Explanation
t3	`SQL> GRANT CREATE SESSION TO c##dba;`	n/a	`SYSTEM` grants the `CREATE SESSION` system privilege to `c##dba`. Because the clause `CONTAINER=ALL` is absent, this privilege is granted locally and thus applies *only* to the root, which is the current container.
t4	`SQL> CREATE ROLE c##admin CONTAINER=ALL;`	n/a	`SYSTEM` creates a common role named `c##admin`. The clause `CONTAINER=ALL` makes the role a common role.
t5	`SQL> GRANT SELECT ANY TABLE TO c##admin;` `Grant succeeded.`	n/a	`SYSTEM` grants the `SELECT ANY TABLE` privilege to the `c##admin` role. The absence of the `CONTAINER=ALL` clause makes the privilege local to the root. Thus, this common role contains a privilege that is exercisable only in the root.
t6	`SQL> GRANT c##admin TO c##dba;` `SQL> EXIT;`	n/a	`SYSTEM` grants the `c##admin` role to `c##dba`. Because the `CONTAINER=ALL` clause is absent, the role applies *only* to the current container, even though it is a common role. If `c##dba` connects to a PDB, then `c##dba` does not have this role.

t7	n/a	`SQL> CONNECT c##dba@hrpdb`
		`Enter password: *******`
		`ERROR: ORA-01045: user c##dba lacks CREATE SESSION privilege;`
		`logon denied`
			`c##dba` fails to connect to `hrpdb` because the grant at t3 was local to the root.

Table 2-6 (Cont.) Granting Roles and Privileges in a CDB

t	Operations in CDB$ROOT	Operations in hrpdb	Explanation

t8	n/a	`SQL> CONNECT SYSTEM@hrpdb`
		`Enter password: *******`
		`Connected.`
			SYSTEM connects to hrpdb.

t9	n/a	`SQL> GRANT CONNECT, RESOURCE`
		`TO c##dba;`
		`Grant succeeded.`
		`SQL> EXIT`
			SYSTEM grants the CONNECT and RESOURCE roles to common user c##dba. Because the clause CONTAINER=ALL is absent, the grant is local to hrpdb.

t10	n/a	`SQL> CONNECT c##dba@hrpdb`
		`Enter password: *******`
		`Connected.`
			Common user c##dba connects to hrpdb.

t11	n/a	`SQL> SELECT COUNT(*)`
		`FROM hr.employees;`
		`select * from hr.employees`
		`*`
		`ERROR at line 1:`
		`ORA-00942: table or view does not exist`
			The query of hr.employees still returns an error because c##dba does not have select privileges on tables in hrpdb. The SELECT ANY TABLE privilege granted locally at t5 is restricted to the root and thus does not apply to hrpdb.

t12	`SQL> CONNECT SYSTEM@root`
	`Enter password: *******`
	`Connected.`
		n/a	Common user SYSTEM connects to the root container.

t13	`SQL> GRANT SELECT ANY TABLE`
	`TO c##admin CONTAINER=ALL;`
	`Grant succeeded.`
		n/a	SYSTEM grants the SELECT ANY TABLE privilege to the c##admin role. The presence of CONTAINER=ALL means the privilege is being granted commonly.

Table 2-6 (Cont.) Granting Roles and Privileges in a CDB

t	Operations in CDB$ROOT	Operations in hrpdb	Explanation

t14	n/a	`SQL> SELECT COUNT(*) FROM hr.employees;`
		`select * from hr.employees`
		`*`
		`ERROR at line 1:`
		`ORA-00942: table or view does not exist`
			A query of `hr.employees` still returns an error. The reason is that at t6 the `c##admin` common role was granted to `c##dba` in the root only.

t15	`SQL> GRANT c##admin TO c##dba`
	`CONTAINER=ALL;`
	`Grant succeeded.`
		n/a	`SYSTEM` grants the common role named `c##admin` to `c##dba`, specifying `CONTAINER=ALL`. Now user `c##dba` has the role in *all* containers, not just the root.

t17	n/a	`SQL> SELECT COUNT(*)`
		`FROM hr.employees;`
		
		`COUNT(*)`
		`----------`
		`107`
			The query succeeds.

> **See Also:**
>
> *Oracle Database Security Guide* to learn how to manage common and local roles

Overview of Common and Local Objects in a CDB

A **common object** is defined in either the CDB root or an application root, and can be referenced using metadata links or object links. A local object is every object that is not a common object.

Database-supplied common objects are defined in `CDB$ROOT` and cannot be changed. Oracle Database does not support creation of common objects in `CDB$ROOT`.

You can create most schema objects—such as tables, views, PL/SQL and Java program units, sequences, and so on—as common objects in an application root. If the object exists in an application root, then it is called an application common object.

A local user can own a common object. Also, a common user can own a local object, but only when the object is not data-linked or metadata-linked, and is also neither a metadata link nor a data link.

> **See Also:**
>
> - "Application Common Objects"
> - "About Application Common Objects"
> - *Oracle Database Security Guide* to learn more about privilege management for common objects

Overview of Common Audit Configurations

For both mixed mode and unified auditing, a **common audit configuration** is visible and enforced across all PDBs.

Audit configurations are either local or common. The scoping rules that apply to other local or common phenomena, such as users and roles, all apply to audit configurations.

> **Note:**
>
> Audit initialization parameters exist at the CDB level and not in each PDB.

PDBs support the following auditing options:

- Object auditing

  Object auditing refers to audit configurations for specific objects. Only common objects can be part of the common audit configuration. A local audit configuration cannot contain common objects.

- Audit policies

  Audit policies can be local or common:

  - Local audit policies

    A local audit policy applies to a single PDB. You can enforce local audit policies for local and common users in this PDB only. Attempts to enforce local audit policies across all containers result in an error.

    In all cases, enforcing of a local audit policy is part of the local auditing framework.

  - Common audit policies

    A common audit policy applies to all containers. This policy can only contain actions, system privileges, common roles, and common objects. You can apply a common audit policy only to common users. Attempts to enforce a common audit policy for a local user across all containers result in an error.

A common audit configuration is stored in the SYS schema of the root. A local audit configuration is stored in the SYS schema of the PDB to which it applies.

Audit trails are stored in the SYS or AUDSYS schemas of the relevant PDBs. Operating system and XML audit trails for PDBs are stored in subdirectories of the directory specified by the AUDIT_FILE_DEST initialization parameter.

> **See Also:**
>
> - *Oracle Database Concepts* for information about database auditing
> - *Oracle Database Security Guide* to learn about common audit configurations

Overview of PDB Lockdown Profiles

A **PDB lockdown profile** is a named set of features that control operations available to users connected to a PDB. For example, a PDB lockdown profile can disable privileges that come with the ALTER SYSTEM statement.

A potential for elevation of privileges exists when PDBs share an identity. For example, identity can be shared at a network level, or when PDBs access common objects or connect through database links. To increase security, a CDB administrator may want to compartmentalize access, thereby restricting the operations that a user can perform in a PDB.

A use case might be the creation of lockdown profiles at high, medium, and low levels. The high level might greatly restrict access, whereas the low level might enable access.

You can restrict the following types of access:

- Network access

  For example, restrict access to UTL_HTTP or UTL_MAIL.
- Common user and common object access

  For example, restrict operations in which a local user in a PDB can proxy through a common user or access objects in a common schema.
- Operating system access

  For example, restrict access to the UTL_FILE or DBMS_FILE_TRANSFER PL/SQL packages.
- Connections

  For example, you can restrict common users from connecting to the PDB or you can restrict a local user who has the SYSOPER administrative privilege from connecting to a PDB that is open in restricted mode.
- Administrative features

  For example, you can restrict the use of ALTER SYSTEM, ALTER SESSION, and ALTER DATABASE.
- Database options

  For example, you can use lockdown profiles to disable access to database options such as Oracle Partitioning or Oracle Database Advanced Queuing.

When logged in to the CDB root or application root, create a lockdown profile by issuing the `CREATE LOCKDOWN PROFILE` statement, which supports the following optional clauses:

- `FROM` *`static_base_profile`* creates a new lockdown profile by using the values from an existing profile. Any subsequent changes to the existing profile will not affect the new profile.
- `INCLUDING` *`dynamic_base_profile`* creates a new lockdown profile by using the values from an existing profile, except that this new lockdown profile inherits the `DISABLE STATEMENT` rules that comprise the base profile, and any subsequent changes to the base profile.

The user issuing the statement must have the `CREATE LOCKDOWN PROFILE` system privilege in the current container. You can add and remove restrictions with the `ALTER LOCKDOWN PROFILE` statement. The user must issue the `ALTER` statement in the CDB root or application root and must have the have `ALTER LOCKDOWN PROFILE` system privilege in the current container.

Specify a lockdown profile by using the `PDB_LOCKDOWN` initialization parameter. This parameter determines whether the PDB lockdown profile applies to a given PDB. You can set this parameter at the following levels:

- PDB

  The profile applies only to the PDB in which it is set.

- Application container

  The profile applies to all application PDBs in the application container. The value can be modified only by an application common user who has application common `SYSDBA` or common `ALTER SYSTEM` privileges or a CDB common user who has common `SYSDBA` or common `ALTER SYSTEM` privileges.

- CDB

  The profile applies to all PDBs. A common user who has common `SYSDBA` or common `ALTER SYSTEM` privileges can override a CDB-wide setting for a specific PDB.

If the `PDB_LOCKDOWN` parameter in a PDB is set to the name of a lockdown profile different from the container for this PDB (CDB or application container), then a set of rules govern the interaction between restrictions.

Example 2-6 Creating a PDB Lockdown Profile

In this example, you connect to the CDB root as a common user with the `CREATE LOCKDOWN PROFILE` privilege. You create a profile called medium that disables all `ALTER SYSTEM` statements except for `ALTER SYSTEM FLUSH SHARED POOL`:

```
CREATE LOCKDOWN PROFILE medium;
ALTER LOCKDOWN PROFILE medium DISABLE STATEMENT=('ALTER SYSTEM');
ALTER LOCKDOWN PROFILE medium ENABLE STATEMENT=('ALTER SYSTEM')
CLAUSE=('FLUSH SHARED POOL');
```

You can connect as the same common user to each PDB that requires this profile, and then use `ALTER SYSTEM` to set the `PDB_LOCKDOWN` initialization parameter to `medium`. For example, you could set `PDB_LOCKDOWN` to `medium` for `hrpdb`, but not `salespdb`.

The following example creates a `medium2` profile from `medium`:

```
CREATE LOCKDOWN PROFILE medium2 FROM medium;
```

Note:

- "About Restricting PDB Users for Enhanced Security" to learn more about PDB lockdown profiles
- *Oracle Database Security Guide* to learn how to create, enable, and drop PDB lockdown profiles

Overview of Applications in an Application Container

Within an application container, an **application** is the named, versioned set of common data and metadata stored in the application root.

In this context of an application container, the term “application” means “master application definition.” For example, the application might include definitions of tables, views, and packages.

See Also:

- "Overview of Common and Local Objects in a CDB" to learn about application common objects
- "Creating and Removing Application Containers and Seeds"
- "Administering Application Containers"

About Application Containers

An **application container** is an optional, user-created CDB component that stores data and metadata for one or more application back ends. A CDB includes zero or more application containers.

For example, you might create multiple sales-related PDBs within one application container, with these PDBs sharing an application back end that consists of a set of common tables and table definitions. You might store multiple HR-related PDBs within a separate application container, with their own common tables and table definitions.

The `CREATE PLUGGABLE DATABASE` statement with the `AS APPLICATION CONTAINER` clause creates the application root of the application container, and thus implicitly creates the application container itself. When you first create the application container, it contains no PDBs. To create application PDBs, you must connect to the application root, and then execute the `CREATE PLUGGABLE DATABASE` statement.

In the `CREATE PLUGGABLE DATABASE` statement, you must specify a container name (which is the same as the application root name), for example, `saas_sales_ac`. The application container name must be unique within the CDB, and within the scope of all

the CDBs whose instances are reached through a specific listener. Every application container has a default service with the same name as the application container.

> **See Also:**
>
> - "Overview of Applications in an Application Container"
> - "Creating and Removing Application Containers and Seeds"
> - "Administering Application Containers"

Purpose of Application Containers

In some ways, an application container functions as an application-specific CDB *within* a CDB. An application container, like the CDB itself, can include multiple PDBs, and enables these PDBs to share metadata and data.

The application root enables application PDBs to share an application, which in this context means a named, versioned set of common metadata and data. A typical application installs application common users, metadata-linked common objects, and data-linked common objects.

Key Benefits of Application Containers

Application containers provide several benefits over storing each application in a separate PDB.

- The application root stores metadata and data that all application PDBs can share.

  For example, all application PDBs can share data in a central table, such as a table listed default application roles. Also, all PDBs can share a table definition to which they add PDB-specific rows.
- You maintain your master application definition in the application root, instead of maintaining a separate copy in each PDB.

  If you upgrade the application in the application root, then the changes are automatically propagated to all application PDBs. The application back end might contain the data-linked common object `app_roles`, which is a table that list default roles: `admin`, `manager`, `sales_rep`, and so on. A user connected to any application PDB can query this table.
- An application container can include an application seed, application PDBs, and proxy PDBs (which refer to PDBs in other CDBs).
- You can rapidly create new application PDBs from the application seed.
- You can query views that report on all PDBs in the application container.
- While connected to the application root, you can use the `CONTAINERS` function to perform DML on objects in multiple PDBs.

  For example, if the `products` table exists in every application PDB, then you can connect to the application root and query the products in all application PDBs using a single `SELECT` statement.

- You can unplug a PDB from an application root, and then plug it in to an application root in a higher Oracle database release. Thus, PDBs are useful in an Oracle database upgrade.

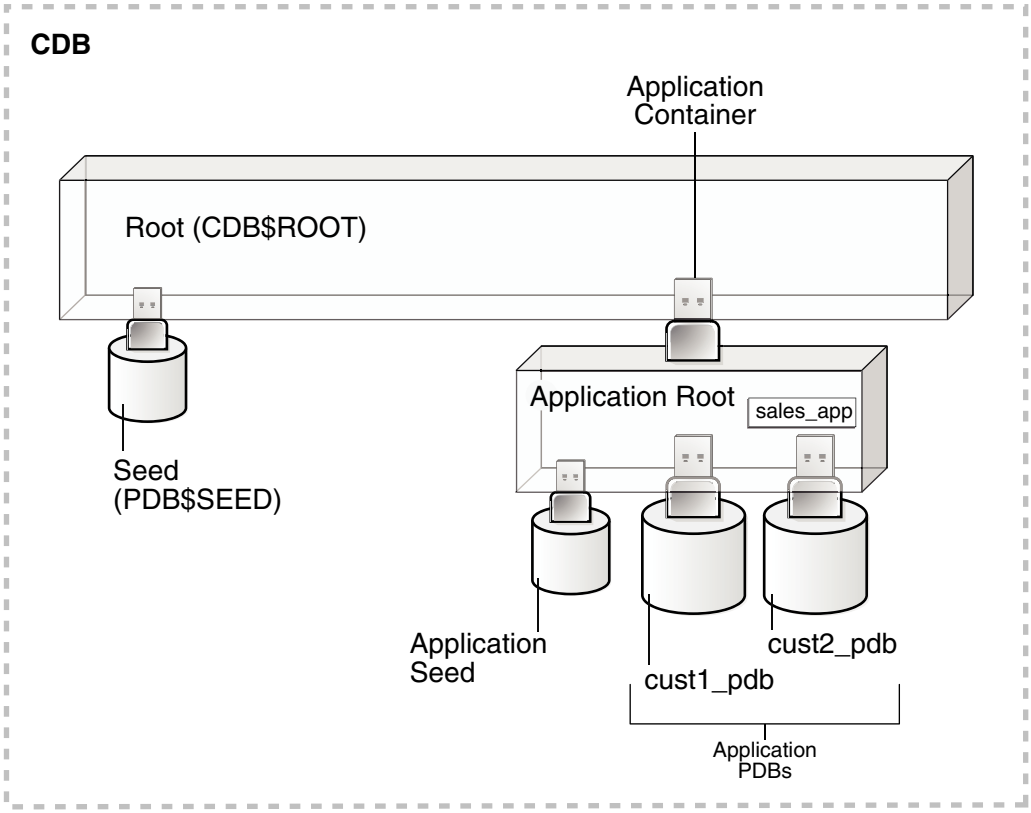
> **See Also:**
>
> "Overview of Applications in an Application Container"

Application Container Use Case: SaaS

A SaaS deployment can use multiple application PDBs, each for a separate customer, that share metadata and data.

In a pure SaaS environment, the master application definition resides in the application root, but the customer-specific data resides in its own application PDB. For example, `sales_app` is the application model in the application root. The application PDB named `cust1_pdb` contains sales data only for customer 1, whereas the application PDB named `cust2_pdb` contains sales data only for customer 2. Plugging, unplugging, cloning, and other PDB-level operations are available for individual customer PDBs.

Figure 2-8 SaaS Use Case



A pure SaaS configuration provides the following benefits:

- Performance
- Security
- Support for multiple customers

  The data for each customer resides in its own container, but is consolidated so that you can manage many customers collectively. This model extends the economies of scale of managing many as one to the application administrator, not only the DBA.

Application Containers Use Case: Logical Data Warehouse

A customer can use multiple application PDBs to address data sovereignty issues.

In a sample use case, a company puts data specific to each financial quarter in a separate PDB. For example, the application container named `sales_ac` includes `q1_2016_pdb`, `q2_2016_pdb`, `q3_2016_pdb`, and `q4_2016_pdb`. You define each transaction in the PDB corresponding to the associated quarter. To generate a report that aggregates performance across a year, you aggregate across the four PDBs using the `CONTAINERS()` clause.

Benefits of this logical warehouse design include:

- ETL for data specific to a single PDB does not affect the other PDBs.
- Execution plans are more efficient because they are based on actual data distribution.

Application Root

An application container has exactly one **application root**, which is the parent of the application PDBs in the container.

The property of being an application root is established at creation time, and cannot be changed. The only container to which an application root belongs is the CDB root. An application root is like the CDB root in some ways, and like a PDB in other ways:

- Like the CDB root, an application root serves as parent container to the PDBs plugged into it. When connected to the application root, you can manage common users and privileges, create application PDBs, switch containers, and issue DDL that applies to all PDBs in the application container.
- Like a PDB, you create an application root with the `CREATE PLUGGABLE DATABASE` statement, alter it with `ALTER PLUGGABLE DATABASE`, and change its availability with `STARTUP` and `SHUTDOWN`. You can use DDL to plug, unplug, and drop application roots. The application root has its own service name, and users can connect to the application root in the same way that they connect to a PDB.

An application root differs from both the CDB root and standard PDB because it can store *user-created* common objects, which are called **application common objects**. Application common objects are accessible to the application PDBs plugged in to the application root. Application common objects are not visible to the CDB root, other application roots, or PDBs that do not belong to the application root.

> **See Also:**
>
> - "Creating and Removing Application Containers and Seeds"
> - "Administering Application Containers"

Example 2-7 Creating an Application Root

In this example, you log in to the CDB root as administrative common user `c##system`. You create an application container named `saas_sales_ac`, and then open the application root, which has the same name as the container.

```
-- Create the application container called saas_sales_ac
CREATE PLUGGABLE DATABASE saas_sales_ac AS APPLICATION CONTAINER
  ADMIN USER saas_sales_ac_adm IDENTIFIED BY manager;

-- Open the application root
ALTER PLUGGABLE DATABASE saas_sales_ac OPEN;
```

You set the current container to `saas_sales_ac`, and then verify that this container is the application root:

```
-- Set the current container to saas_sales_ac
ALTER SESSION SET CONTAINER = saas_sales_ac;

COL NAME FORMAT a15
COL ROOT FORMAT a4
SELECT CON_ID, NAME, APPLICATION_ROOT AS ROOT,
       APPLICATION_PDB AS PDB,
FROM   V$CONTAINERS;

    CON_ID NAME            ROOT PDB
---------- --------------- ---- ---
         3 SAAS_SALES_AC   YES    NO
```

Application PDBs

An **application PDB** is a PDB that resides in an application container. Every PDB in a CDB resides in either zero or one application containers.

For example, the `saas_sales_ac` application container might support multiple customers, with each customer application storing its data in a separate PDB. The application PDBs `cust1_sales_pdb` and `cust2_sales_pdb` might reside in `saas_sales_ac`, in which case they belong to no other application container (although as PDBs they necessarily belong also to the CDB root).

Create an application PDB by executing `CREATE PLUGGABLE DATABASE` while connected to the application root. You can either create the application PDB from a seed, or clone a PDB or plug in an unplugged PDB. Like a PDB that is plugged in to CDB root, you can clone, unplug, or drop an application PDB. However, an application PDB must always belong to an application root.

> **See Also:**
>
> "Creating and Removing Application Containers and Seeds"

Application Seed

An **application seed** is an optional, user-created PDB within an application container. An application container has either zero or one application seed.

An application seed enables you to create application PDBs quickly. It serves the same role within the application container as `PDB$SEED` serves within the CDB itself.

The application seed name is always *`application_container_name`*`$SEED`, where *`application_container_name`* is the name of the application container. For example, use the `CREATE PDB ... AS SEED` statement to create `saas_sales_ac$SEED` in the `saas_sales_ac` application container.

> **See Also:**
>
> "Creating and Removing Application Seeds"

Application Common Objects

An **application common object** is a common object created within an application in an application root. Common objects are either data-linked or metadata-linked.

For a data-linked common object, application PDBs share a single set of data. For example, an application for the `saas_sales_ac` application container is named `saas_sales_app`, has version `1.0`, and includes a data-linked `usa_zipcodes` table. In this case, the rows are stored once in the table in the application root, but are visible in all application PDBs.

For a metadata-linked common object, application PDBs share only the metadata, but contain different sets of data. For example, a metadata-linked `products` table has the same definition in every application PDB, but the rows themselves are specific to the PDB. The application PDB named `cust1pdb` might have a `products` table that contains books, whereas the application PDB named `cust2pdb` might have a `products` table that contains auto parts.

> **See Also:**
>
> - "Overview of Common and Local Objects in a CDB" to learn about common objects
> - "About Application Common Objects"

Creation of Application Common Objects

To create common objects, connect to an application root, and then execute a `CREATE` statement that specifies a sharing attribute.

You can only create or change application common objects as part of an application installation, upgrade, or patch. You can specify sharing in the following ways:

- `DEFAULT_SHARING` initialization parameter

  The setting is the default sharing attribute for all database objects of a supported type created in the root.

- `SHARING` clause

  You specify this clause in the `CREATE` statement itself. When a `SHARING` clause is included in a SQL statement, it takes precedence over the value specified in the `DEFAULT_SHARING` initialization parameter. Possible values are `METADATA`, `DATA`, `EXTENDED DATA`, and `NONE`.

The following table shows the types of application common objects, and where the data and metadata is stored.

Table 2-7 Application Common Objects

Object Type	SHARING Value	Metadata Storage	Data Storage
Data-Linked	`DATA`	Application root	Application root
Extended Data-Linked	`EXTENDED DATA`	Application root	Application root and application PDB
Metadata-Linked	`METADATA`	Application root	Application PDB

See Also:

- "Creating Application Common Objects"
- *Oracle Database Security Guide* to learn how to manage privileges for common objects

Metadata-Linked Application Common Objects

A **metadata link** is a dictionary object that supports referring to, and granting privileges on, common metadata shared by all PDBs in the application container.

Specifying the `METADATA` value in either the `SHARING` clause or the `DEFAULT_SHARING` initialization parameter specifies a link to an object's metadata, called a metadata-linked common object. The metadata for the object is stored once in the application root.

Tables, views, and code objects (such as PL/SQL procedures) can share metadata. In this context, "metadata" includes column definitions, constraints, triggers, and code. For example, if `sales_mlt` is a metadata-linked common table, then all application PDBs access the *same* definition of this table, which is stored in the application root,

by means of a metadata link. The rows in sales_mlt are different in every application PDB, but the column definitions are the same.

Typically, most objects in an application will be metadata-linked. Thus, you need only maintain one master application definition. This approach centralizes management of the application in multiple application PDBs.

Example 2-8 Creating a Metadata-Linked Common Object

In this example, the SYSTEM user logs in to the saas_sales_ac application container. SYSTEM installs an application named saas_sales_app at version 1.0 (see "Application Maintenance"). This application creates a common user account named saas_sales_adm. The schema contains a metadata-linked common table named sales_mlt.

```
-- Begin the install of saas_sales_app
ALTER PLUGGABLE DATABASE APPLICATION saas_sales_app BEGIN INSTALL '1.0';

-- Create the tablespace for the app
CREATE TABLESPACE saas_sales_tbs DATAFILE SIZE 100M AUTOEXTEND ON NEXT
10M MAXSIZE 200M;

-- Create the user account saas_sales_adm, which will own the app
CREATE USER saas_sales_adm IDENTIFIED BY ****** CONTAINER=ALL;

-- Grant necessary privileges to this user account
GRANT CREATE SESSION, DBA TO saas_sales_adm;

-- Makes the tablespace that you just created the default for
saas_sales_adm
ALTER USER saas_sales_adm DEFAULT TABLESPACE saas_sales_tbs;

-- Now connect as the application owner
CONNECT saas_sales_adm/******@saas_sales_ac

-- Create a metadata-linked table
CREATE TABLE saas_sales_adm.sales_mlt SHARING=METADATA
(YEAR       NUMBER(4),
 REGION     VARCHAR2(10),
 QUARTER    VARCHAR2(4),
 REVENUE    NUMBER);

-- End the application installation
ALTER PLUGGABLE DATABASE APPLICATION saas_sales_app END INSTALL '1.0';
```

You can use the ALTER PLUGGABLE DATABASE APPLICATION ... SYNC statement to synchronize the application PDBs to use the same master application definition. In this way, every application PDB has a metadata link to the saas_sales_adm.sales_mlt common table. The middle-tier code that updates sales_mlt within the PDB named cust1_pdb adds rows to this table in cust1_pdb, whereas the middle-tier code that updates sales_mlt in cust2_pdb adds rows to the copy of this table in cust2_pdb. Only the table metadata, which is stored in the application root, is shared.

> **Note:**
>
> - "About Application Common Objects"
> - *Oracle Database Security Guide* to learn more about how commonly granted object privileges work

Metadata Links

For metadata-linked application common objects, the metadata for the object is stored once in the application root. A metadata link is a dictionary object whose object type is the same as the metadata it is sharing.

The description of a metadata link is stored in the data dictionary of the PDB in which it is created. A metadata link must be owned by an application common user. You can only use metadata links to share metadata of common objects owned by their creator in the CDB root or an application root.

Unlike a data link, a metadata link depends *only* on common data. For example, if an application contains the local tables `dow_close_lt` and `nasdaq_close_lt` in the application root, then a common user cannot create metadata links to these objects. However, an application common table named `sales_mlt` may be metadata-linked.

If a privileged common user changes the metadata for `sales_mlt`, for example, adds a column to the table, then this change propagates to the metadata links. Application PDB users may not change the metadata in the metadata link. For example, a DBA who manages the application PDB named `cust1_pdb` cannot add a column to `sales_mlt` in this PDB only: such metadata changes can be made only in the application root.

> **See Also:**
>
> "About Application Common Objects"

Data-Linked Application Common Objects

A **data-linked object** is an object whose metadata and data reside in an application root, and are accessible from all application PDBs in this application container.

Specifying the `DATA` value in either the `SHARING` clause or the `DEFAULT_SHARING` initialization parameter specifies a link to a common object, called a data-linked common object. Dimension tables in a data warehouse are often good candidates for data-linked common tables.

A data link is a dictionary object that functions much like a synonym. For example, if `countries` is an application common table, then all application PDBs access the *same* copy of this table by means of a data link. If a row is added to this table, then this row is visible in all application PDBs.

A data link must be owned by an application common user. The link inherits the object type from the object to which it is pointing. The description of a data link is stored in the dictionary of the PDB in which it is created. For example, if an application container

contains 10 application PDBs, and if every PDB contains a link to the `countries` application common table, then all 10 PDBs contain dictionary definitions for this link.

Example 2-9 Creating a Data-Linked Object

In this example, `SYSTEM` connects to the `saas_sales_ac` application container. `SYSTEM` upgrades the application named `saas_sales_app` from version `1.0` to `2.0`. This application upgrade logs in to the container as common user `saas_sales_adm`, creates a data-linked table named `countries_dlt`, and then inserts rows into it.

```
-- Begin an upgrade of the application
ALTER PLUGGABLE DATABASE APPLICATION saas_sales_app BEGIN UPGRADE '1.0'
to '2.0';

-- Connect as application owner to application root
CONNECT saas_sales_adm/manager@saas_sales_ac

-- Create data-linked table named countries_dlt
CREATE TABLE countries_dlt SHARING=DATA
(country_id   NUMBER,
 country_name VARCHAR2(20));

-- Insert records into countries_dlt
INSERT INTO countries_dlt VALUES(1, 'USA');
INSERT INTO countries_dlt VALUES(44, 'UK');
INSERT INTO countries_dlt VALUES(86, 'China');
INSERT INTO countries_dlt VALUES(91, 'India');

-- End application upgrade
ALTER PLUGGABLE DATABASE APPLICATION saas_sales_app END UPGRADE TO
'2.0';
```

Use the `ALTER PLUGGABLE DATABASE APPLICATION ... SYNC` statement to synchronize application PDBs with the application root (see "Application Synchronization"). In this way, every synchronized application PDB has a data link to the `saas_sales_adm.countries_dlt` data-linked table.

> **Note:**
>
> "About Application Common Objects"

Extended Data-Linked Application Objects

An **extended data-linked object** is a hybrid of a data-linked object and metadata-linked object.

In an extended data-linked object, the data stored in the application root is common to all application PDBs, and all PDBs can access this data. However, each application PDB can create its own, PDB-specific data while sharing the common data in application root. Thus, the PDBs supplement the common data with their own data.

For example, a sales application might support several application PDBs. All application PDBs need the postal codes for the United States. In this case, you

might create a zipcodes_edt extended data-linked table in the application root. The application root stores the United States postal codes, so all application PDBs can access them. However, one application PDB requires the postal codes for the United States and Canada. This application PDB can store the postal codes for Canada in the extended data-linked object in the application PDB instead of in the application root.

Create an extended data-linked object by connecting to the application root and specifying the SHARING=EXTENDED DATA keyword in the CREATE statement.

Example 2-10 Creating an Extended-Data Object

In this example, SYSTEM connects to the saas_sales_ac application container, and then upgrades the application named saas_sales_app (created in "Example 2-8") from version 2.0 to 3.0. This application logs in to the container as common user saas_sales_adm, creates an extended data-linked table named zipcodes_edt, and then inserts rows into it.

```
-- Begin an upgrade of the app
ALTER PLUGGABLE DATABASE APPLICATION saas_sales_app BEGIN UPGRADE '2.0'
to '3.0';

-- Connect as app owner to app root
CONNECT saas_sales_adm/manager@saas_sales_ac

-- Create a common-data table named zipcodes_edt
CREATE TABLE zipcodes_edt SHARING=EXTENDED DATA
(code       VARCHAR2(5),
 country_id NUMBER,
 region     VARCHAR2(10));

-- Load rows into zipcodes_edt
INSERT INTO zipcodes_edt VALUES ('08820','1','East');
INSERT INTO zipcodes_edt VALUES ('10005','1','East');
INSERT INTO zipcodes_edt VALUES ('44332','1','North');
INSERT INTO zipcodes_edt VALUES ('94065','1','West');
INSERT INTO zipcodes_edt VALUES ('73301','1','South');
COMMIT;

-- End app upgrade
ALTER PLUGGABLE DATABASE APPLICATION saas_sales_app END UPGRADE TO
'3.0';
```

Use the ALTER PLUGGABLE DATABASE APPLICATION ... SYNC statement to synchronize application PDBs with the application (see "Application Synchronization"). In this way, every synchronized application PDB has a data link to the saas_sales_adm.zipcodes_edt data-linked table. Applications that connect to these PDBs can see the postal codes that were inserted into zipcodes_edt during the application upgrade, but can also insert their own postal codes into this table.

> **Note:**
>
> "About Application Common Objects"

Application Maintenance

In this context, **application maintenance** refers to installing, uninstalling, upgrading, or patching an application.

An application must have a name and version number. This combination of properties determines which maintenance operations you can perform. In all maintenance operations, you perform the following steps:

1. Begin by executing the `ALTER PLUGGABLE DATABASE ... APPLICATION` statement with the `BEGIN INSTALL`, `BEGIN UPGRADE`, or `BEGIN PATCH` clauses.
2. Execute statements to alter the application.
3. End by executing the `ALTER PLUGGABLE DATABASE ... APPLICATION` statement with the `END INSTALL`, `END UPGRADE`, or `END PATCH` clauses.

As the application evolves, the application container maintains all versions and patch changes.

> **Note:**
>
> "About Application Management"

About Application Maintenance

Perform application installation, upgrade, and patching operations using an `ALTER PLUGGABLE DATABASE APPLICATION` statement.

The basic steps for application maintenance are as follows:

1. Log in to the application root.
2. Begin the operation with an `ALTER PLUGGABLE DATABASE APPLICATION ... BEGIN` statement in the application root.
3. Execute the application maintenance statements.
4. End the operation with an `ALTER PLUGGABLE DATABASE APPLICATION ... END` statement.

Perform the maintenance using scripts, SQL statements, or GUI tools.

> **See Also:**
>
> "About Application Management"

Application Installation

An **application installation** is the initial creation of a master application definition. A typical installation creates user accounts, tables, and PL/SQL packages.

To install the application, specify the following in the `ALTER PLUGGABLE DATABASE APPLICATION` statement:

- Name of the application
- Application version number

Example 2-11 Installing an Application

This example assumes that you are logged in to the application container named `saas_sales_ac` as. The example installs an application named `saas_sales_app` at version 1.0. Note that you specify the version with a string rather than a number. The application creates an application common user named `saas_sales_adm`, grants necessary privileges, and then connects to the application root as this user. This user creates a metadata-linked table named `sales_mlt`.

```
-- Begin the install of saas_sales_app
ALTER PLUGGABLE DATABASE APPLICATION saas_sales_app BEGIN INSTALL '1.0';

-- Create the tablespace for the app
CREATE TABLESPACE saas_sales_tbs DATAFILE SIZE 100M AUTOEXTEND ON NEXT
10M MAXSIZE 200M;

-- Create the user account saas_sales_adm, which will own the
application
CREATE USER saas_sales_adm IDENTIFIED BY manager CONTAINER=ALL;

-- Grant necessary privileges to this user account
GRANT CREATE SESSION, DBA TO saas_sales_adm;

-- Make the tablespace that you just created the default for
saas_sales_adm
ALTER USER saas_sales_adm DEFAULT TABLESPACE saas_sales_tbs;

-- Now connect as the application owner
CONNECT saas_sales_adm/manager@saas_sales_ac

-- Create a metadata-linked table
CREATE TABLE saas_sales_adm.sales_mlt SHARING=METADATA
(YEAR       NUMBER(4),
 REGION     VARCHAR2(10),
 QUARTER    VARCHAR2(4),
 REVENUE    NUMBER);

-- End the application installation
ALTER PLUGGABLE DATABASE APPLICATION saas_sales_app END INSTALL '1.0';
```

PDB synchronization is the user-initiated update of an application PDB with the application in the application root. After you synchronize the application PDBs with the `saas_sales_app` application, each application PDB will contain an empty table named `products_mlt`. An application can connect to an application PDB, and then insert PDB-specific rows into this table.

> **See Also:**
>
> - "Application Synchronization"
> - "Installing an Application in an Application Container with Automated Propagation"

Application Upgrade

An **application upgrade** is a major change to an installed application.

Typically, an upgrade changes the physical architecture of the application. For example, an upgrade might add new user accounts, tables, and packages, or alter the definitions of existing objects.

To upgrade the application, you must specify the following in the `ALTER PLUGGABLE DATABASE APPLICATION` statement:

- Name of the application
- Old application version number
- New application version number

Example 2-12 Upgrading an Application Using the Automated Technique

In this example, you connect to the application root as an administrator, and then upgrade the application `saas_sales_app` from version 1.0 to version 2.0. The upgrade creates a data-linked table named `countries_dlt`, and then adds rows to it. It also creates an extended data-linked table named `zipcodes_edt`, and then adds rows to it.

```
-- Begin an upgrade of the app
ALTER PLUGGABLE DATABASE APPLICATION saas_sales_app
  BEGIN UPGRADE '1.0' to '2.0';

-- Connect as app owner to app root
CONNECT saas_sales_adm/manager@saas_sales_ac

-- Create data-linked table named countries_dlt
CREATE TABLE countries_dlt SHARING=DATA
(country_id   NUMBER,
 country_name VARCHAR2(20));

-- Insert records into countries_dlt
INSERT INTO countries_dlt VALUES(1, 'USA');
INSERT INTO countries_dlt VALUES(44, 'UK');
INSERT INTO countries_dlt VALUES(86, 'China');
INSERT INTO countries_dlt VALUES(91, 'India');

-- Create an extended data-linked table named zipcodes_edt
CREATE TABLE zipcodes_edt SHARING=EXTENDED DATA
(code       VARCHAR2(5),
 country_id NUMBER,
 region     VARCHAR2(10));
```

```
-- Load rows into zipcodes_edt
INSERT INTO zipcodes_edt VALUES ('08820','1','East');
INSERT INTO zipcodes_edt VALUES ('10005','1','East');
INSERT INTO zipcodes_edt VALUES ('44332','1','North');
INSERT INTO zipcodes_edt VALUES ('94065','1','West');
INSERT INTO zipcodes_edt VALUES ('73301','1','South');
COMMIT;

-- End app upgrade
ALTER PLUGGABLE DATABASE APPLICATION saas_sales_app END UPGRADE TO
'2.0';
```

See Also:

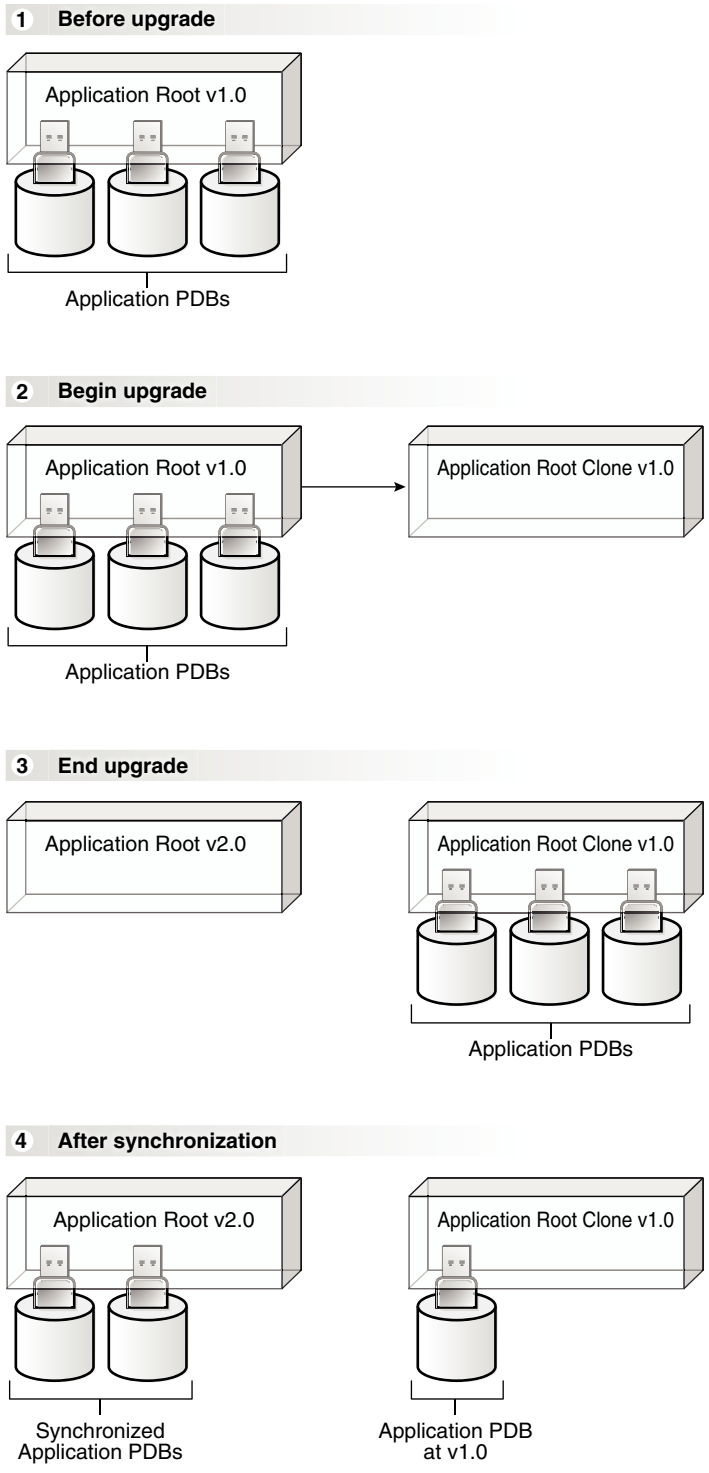
"Upgrading Applications in an Application Container"

How an Application Upgrade Works

During an application upgrade, the application remains available. To make this availability possible, Oracle Database clones the application root.

The following figure gives an overview of the application upgrade process.

Figure 2-9 Application Upgrade



An upgrade occurs as follows:

1. In the initial state, the application root has an application in a specific version.

2. The user executes the `ALTER PLUGGABLE DATABASE APPLICATION BEGIN UPGRADE` statement, and then issues the application upgrade statements.

   During the upgrade, the database automatically does the following:

   - Clones the application root

     For example, if the `saas_sales_app` application is at version 1.0 in the application root, then the clone is also at version 1.0

   - Points the application PDBs to the application root clone

     The clone is in read-only mode. The application remains available to the application PDBs.

3. The user executes the `ALTER PLUGGABLE DATABASE APPLICATION END UPGRADE` statement.

   At this stage, the application PDBs are still pointing to the application root clone, and the original application root is at a new version. For example, if the `saas_sales_app` application is at version 1.0 in the application root, then the upgrade might bring it to version 2.0. The application root clone, however, remains at version 1.0.

4. Optionally, the user synchronizes the application PDBs with the upgraded application root by issuing `ALTER PLUGGABLE DATABASE APPLICATION` statement with the `SYNC` clause.

   For example, after the synchronization, some application PDBs are plugged in to the application root at version 2.0. However, the application root clone continues to support application PDBs that must stay on version 1.0, or any new application PDBs that are plugged in to the application root at version 1.0.

See Also:

- "Application Synchronization"
- "Upgrading Applications in an Application Container"

Applications at Different Versions

Different application PDBs might use different versions of the application.

For example, one application PDB might have version 1.0 of the `saas_sales_app`. In the same application container, another application PDB has version 2.0 of this application.

A use case is a SaaS application provided to different customers. If each customer has its own application PDB, then some customers might wait longer to upgrade the application. In this case, some application PDBs may use the latest version of the application, whereas other application PDBs use an older version.

> **See Also:**
>
> "Upgrading Applications in an Application Container" to learn more about applications at different versions

Application Patch

An **application patch** is a minor change to an application.

Typical examples of application patching include bug fixes and security patches. New functions and packages are permitted within a patch.

In general, destructive operations are not permitted. For example, a patch cannot include `DROP` statements, or `ALTER TABLE` statements that drop a column or change a data type.

Just as the Oracle Database patching process restricts the kinds of operations permitted in an Oracle Database patch, the application patching process restricts the operations permitted in an application patch. If a fix includes an operation that raises an “operation not supported in an application patch” error, then perform an application upgrade instead.

> **Note:**
>
> You cannot patch an application when another application patch or upgrade is in progress.

To patch the application, specify the application name and patch number in the `ALTER PLUGGABLE DATABASE APPLICATION` statement. Optionally, you can specify an application minimum version.

Example 2-13 Patching an Application Using the Automated Technique

In this example, `SYSTEM` logs in to the application root, and then patches the application `saas_sales_app` at version 1.0 or greater. Patch 101 logs in to the application container as `saas_sales_adm`, and then creates a metadata-linked PL/SQL function named `get_total_revenue`.

```
ALTER PLUGGABLE DATABASE APPLICATION saas_sales_app BEGIN PATCH 101
MINIMUM VERSION '1.0';

-- Connect to the saas_sales_ac container as saas_sales_adm, who owns
the application
CONNECT saas_sales_adm/*******@saas_sales_ac

-- Now install the get_total_revenue() function
CREATE FUNCTION get_total_revenue SHARING=METADATA (p_year IN NUMBER)
RETURN SYS_REFCURSOR
AS
c1_cursor SYS_REFCURSOR;
BEGIN
```

```
OPEN c1_cursor FOR
   SELECT a.year,sum(a.revenue)
   FROM containers(sales_data) a
   WHERE a.year = p_year
   GROUP BY a.year;
RETURN c1_cursor;
END;
/

-- End the patch
ALTER PLUGGABLE DATABASE APPLICATION saas_sales_app END PATCH 101;
```

> **See Also:**
>
> "Patching Applications in an Application Container"

Migration of an Existing Application

You can migrate an application that is installed in a PDB to either an application root or to an application PDB.

Typical reasons for migrating a preexisting application include the following:

- Applications that use an installation program

  Some applications use an installation program rather than a script. In this case, you can run the installation program in a new application root, and then use the `DBMS_PDB_ALTER_SHARING` package to set the objects to the appropriate sharing mode: `METADATA`, `DATA`, or `EXTENDED DATA`. The root automatically propagates the changes to the application PDBs. Oracle Database creates a statement log of the installation, so PDBs with previous application versions can be plugged into the application root.

- Applications that are defined separately in each PDB

  Some applications are defined in each PDB, but no application container exists. In this case, you can update the installation script to set the appropriate sharing mode. You create an application root, and then create the master application definition in this root. You can adopt the existing PDBs as application PDBs by plugging them into the application root, and then running a SQL script to replace the full definitions with references to the common definitions.

For example, you can migrate an application installed in a PDB plugged into an Oracle Database 12c CDB to an application container in an Oracle Database 18c CDB.

> **See Also:**
>
> - "About Application Management" to learn how to migrate an existing application
> - *Oracle Database PL/SQL Packages and Types Reference* to learn more about the `DBMS_PDB_ALTER_SHARING` package

Implicitly Created Applications

In addition to user-created applications, application containers can also contain implicitly created applications.

An application is created implicitly in an application root when an application common user operation is issued with a `CONTAINER=ALL` clause without being preceded by an `ALTER PLUGGABLE DATABASE BEGIN` statement.

Application common user operations include operations such as creating a common user with a `CREATE USER` statement or altering a common user with an `ALTER USER` statement. The database automatically names an implicit application `APP$`*`guid`*, where *`guid`* is the global unique ID of the application root. An implicit application is created when the application root is opened for the first time.

> **See Also:**
>
> "Synchronizing Applications in an Application PDB" to learn more about implicitly created applications

Application Synchronization

Within an application PDB, synchronization is the user-initiated update of the application to the latest version and patch in the application root.

When an application is installed, upgraded, patched, or uninstalled in an application root, the changes do not automatically propagate to the application PDBs. You must synchronize the PDBs manually. When connected to an application PDB, you can synchronize one or more applications by issuing `ALTER PLUGGABLE DATABASE APPLICATION ... SYNC`.

Synchronization of a Single Application

If you specify one application name before `SYNC`, then the database synchronizes only the specified application.

The following statement, executed in an application PDB, synchronizes `apexapp` with the application PDB:

```
ALTER PLUGGABLE DATABASE APPLICATION apexapp SYNC;
```

You can use the `SYNC TO PATCH` *`patchnum`* clause to synchronize the application to a specific patch number. This following statement synchronizes an application named `saas_sales_app` to patch 100 in the application PDB:

```
ALTER PLUGGABLE DATABASE APPLICATION saas_sales_app SYNC TO PATCH 100;
```

To synchronize the application to a specific application version, use `SYNC TO version`. This following statement synchronizes an application named `saas_sales_app` to version 2.0 in the application PDB:

```
ALTER PLUGGABLE DATABASE APPLICATION saas_sales_app SYNC TO '2.0';
```

Synchronization of Multiple Applications

You can synchronize multiple applications by specifying the `ALL` keyword.

If you specify `ALL SYNC`, then the database synchronizes all applications, including those implicitly created. Note that `ALL` does not support the `SYNC TO PATCH patchno` and `SYNC TO version` clauses. The following statement synchronizes all applications:

```
ALTER PLUGGABLE DATABASE APPLICATION ALL SYNC;
```

During the synchronization, the replay order for application `BEGIN` and `END` blocks is the same as the capture order.

> **See Also:**
>
> "Synchronizing Applications in an Application PDB"

Container Maps

A **container map** enables a session connected to application root to issue SQL statements that are routed to the appropriate PDB, depending on the value of a predicate used in the SQL statement.

A map table specifies a column in a metadata-linked common table, and uses partitions to associate different application PDBs with different column values. In this way, container maps enable the partitioning of data at the PDB level when the data is not physically partitioned at the table level.

The key components for using container maps are:

- Metadata-linked table

  This table is intended to be queried using the container map. For example, you might create a metadata-linked table named `countries_mlt` that stores different data in each application PDB. In `amer_pdb`, the `countries_mlt.cname` column stores North American country names; in `euro_pdb`, the `countries_mlt.cname` column stores European country names; and in `asia_pdb`, the `countries_mlt.cname` column stores Asian country names.

- Map table

  In the application root, you create a single-column map table partitioned by list, hash, or range. The map table enables the metadata-linked table to be queried using the partitioning strategy that is enabled by the container map. The names of the partitions in the map object table must match the names of the application PDBs in the application container.

For example, the map table named `pdb_map_tbl` may partition by list on the `cname` column. The partitions named `amer_pdb`, `euro_pdb`, and `asia_pdb` correspond to the names of the application PDBs. The values in each partition are the names of the countries, for example, `PARTITION amer_pdb VALUES ('US','MEXICO','CANADA')`.

Starting in Oracle Database 18c, for a `CONTAINERS()` query to use a map, the partitioning column in the map table does not need to match a column in the metadata-linked table. Assume that the table `sh.sales` is enabled for the container map `pdb_map_tbl`, and `cname` is the partitioning column for the map table. Even though `sh.sales` does *not* include a `cname` column, the map table routes the following query to the appropriate PDB: `SELECT * FROM CONTAINERS(sh.sales) WHERE cname = 'US' ORDER BY time_id`.

- Container map

  A container map is a database property that specifies a map table. To set the property, you connect to the application root and execute the `ALTER PLUGGABLE DATABASE SET CONTAINER_MAP=`*`map_table`* statement, where *`map_table`* is the name of the map table.

Example 2-14 Creating a Metadata-Linked Table, Map Table, and Container Map: Part 1

In this example, you log in as an application administrator to the application root. Assume that an application container has three application PDBs: `amer_pdb`, `euro_pdb`, and `asia_pdb`. Each application PDB stores country names for a different region. A metadata-linked table named `oe.countries_mlt` has a `cname` column that stores the country name. For this partitioning strategy, you use partition by list to create a map object named `salesadm.pdb_map_tbl` that creates a partition for each region. The country name determines the region.

```
ALTER PLUGGABLE DATABASE APPLICATION saas_sales_app BEGIN INSTALL '1.0';

-- Create the metadata-linked table.
CREATE TABLE oe.countries_mlt SHARING=METADATA (
  region    VARCHAR2(30),
  cname     VARCHAR2(30));

-- Create the partitioned map table, which is list partitioned on the
-- cname column. The names of the partitions are the names of the
-- application PDBs.
CREATE TABLE salesadm.pdb_map_tbl (cname VARCHAR2(30) NOT NULL)
  PARTITION BY LIST (cname) (
    PARTITION amer_pdb VALUES ('US','MEXICO','CANADA'),
    PARTITION euro_pdb VALUES ('UK','FRANCE','GERMANY'),
    PARTITION asia_pdb VALUES ('INDIA','CHINA','JAPAN'));

-- Set the CONTAINER_MAP database property to the map object.
ALTER PLUGGABLE DATABASE SET CONTAINER_MAP='salesadm.pdb_map_tbl';

-- Enable the container map for the metadata-linked table to be queried.
ALTER TABLE oe.countries_mlt ENABLE CONTAINER_MAP;

-- Ensure that the table to be queried is enabled for the
-- CONTAINERS clause.
ALTER TABLE oe.countries_mlt ENABLE CONTAINERS_DEFAULT;
```

```
-- End the application installation.
ALTER PLUGGABLE DATABASE APPLICATION saas_sales_app END INSTALL '1.0';
```

> **Note:**
>
> Although you create container maps using partitioning syntax, the database does not use partitioning functionality. Defining a container map does not require Oracle Partitioning.

In the preceding script, the `ALTER TABLE oe.countries_mlt ENABLE CONTAINERS_DEFAULT` statement specifies that queries and DML statements issued in the application root must use the `CONTAINERS()` clause by default for the database object.

Example 2-15 Synchronizing the Application, and Adding Data: Part 2

This example continues from the previous example. While connected to the application root, you switch the current container to each PDB in turn, synchronize the `saas_sales_app` application, and then add PDB-specific data to the `oe.countries_mlt` table.

```
ALTER SESSION SET CONTAINER=amer_pdb;
ALTER PLUGGABLE DATABASE APPLICATION saas_sales_app SYNC;
INSERT INTO oe.countries_mlt VALUES ('AMER','US');
INSERT INTO oe.countries_mlt VALUES ('AMER','MEXICO');
INSERT INTO oe.countries_mlt VALUES ('AMER','CANADA');
COMMIT;

ALTER SESSION SET CONTAINER=euro_pdb;
ALTER PLUGGABLE DATABASE APPLICATION saas_sales_app SYNC;
INSERT INTO oe.countries_mlt VALUES ('EURO','UK');
INSERT INTO oe.countries_mlt VALUES ('EURO','FRANCE');
INSERT INTO oe.countries_mlt VALUES ('EURO','GERMANY');
COMMIT;

ALTER SESSION SET CONTAINER=asia_pdb;
ALTER PLUGGABLE DATABASE APPLICATION saas_sales_app SYNC;
INSERT INTO oe.countries_mlt VALUES ('ASIA','INDIA');
INSERT INTO oe.countries_mlt VALUES ('ASIA','CHINA');
INSERT INTO oe.countries_mlt VALUES ('ASIA','JAPAN');
COMMIT;
```

Example 2-16 Querying the Metadata-Linked Table: Part 3

This example continues from the previous example. You connect to the application root, and then query `oe.countries_mlt` multiple times, specifying different countries in the `WHERE` clause. The query returns the correct value from the `oe.countries_mlt.region` column.

```
ALTER SESSION SET CONTAINER=saas_sales_ac;
```

```
SELECT region FROM oe.countries_mlt WHERE cname='MEXICO';

REGION
------
AMER

SELECT region FROM oe.countries_mlt WHERE cname='GERMANY';

REGION
------
EURO

SELECT region FROM oe.countries_mlt WHERE cname='JAPAN';

REGION
------
ASIA
```

> **See Also:**
>
> "Partitioning by PDB with Container Maps"

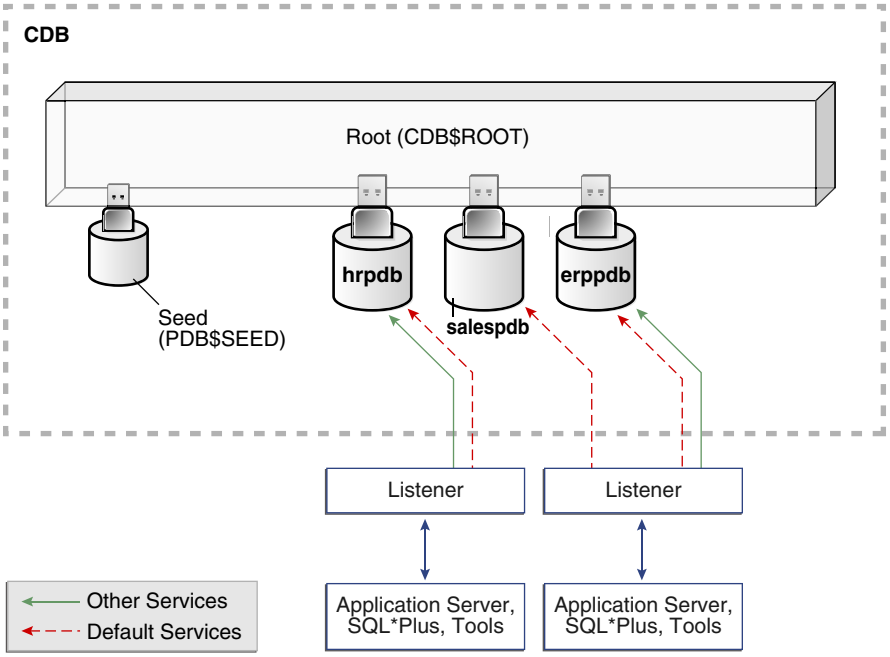
Overview of Services in a CDB

Clients must connect to PDBs or application roots using services.

A connection using a service name starts a new session in a PDB or application root. A foreground process, and therefore a session, at every moment of its lifetime, has a uniquely defined current container.

The following graphic shows two clients connecting to PDBs using two different listeners.

Figure 2-10 Services in a CDB



> **See Also:**
>
> "Managing Services for PDBs"

Service Creation in a CDB

When you execute the `CREATE PLUGGABLE DATABASE` statement to create a PDB, the database automatically creates and starts a service inside the CDB.

The default service has a property that identifies the PDB as the initial current container for the service. The property is shown in the `DBA_SERVICES.PDB` column.

> **See Also:**
>
> "Managing Services for PDBs"

Default Services in a CDB

The default service has the same name as the PDB. The PDB name must be a valid service name, which must be unique within the CDB.

When you create an application container, which requires specifying the `AS APPLICATION CONTAINER` clause, Oracle Database automatically creates a new default service for the application root. The service has the same name as the application container. Oracle Net Services must be configured properly for clients to access this

service. Similarly, every application PDB has its own default service name, and an application seed PDB has its own default service name.

Example 2-17 Switching to a PDB Using a Default Service

This example switches to the PDB names `salespdb` using the default service, which has the same name as the PDB:

```
ALTER SESSION SET CONTAINER = salespdb;
```

> **See Also:**
>
> - *Oracle Database Concepts* for information about service names
> - "Managing Services for PDBs"

Nondefault Services in a CDB

You can create additional services for each PDB, up to a per-CDB maximum of 10,000. Each additional service denotes its PDB as the initial current container.

In Figure 2-10, nondefault services exist for `erppdb` and `hrpdb`. Create, maintain, and drop additional services using the same techniques that you use in a non-CDB.

For example, in Figure 2-10 the PDB named `hrpdb` has a default service named `hrpdb`. The default service cannot be dropped.

When you switch to a container using `ALTER SESSION SET CONTAINER`, the session uses the default service for the container. Optionally, you can use a different service for the container by specifying `SERVICE = service_name`, where *`service_name`* is the name of the service. You might want to use a particular service so that the session can take advantage of its service attributes and features, such as service metrics, load balancing, Resource Manager settings, and so on.

Example 2-18 Switching to a PDB Using a Nondefault Service

In this example, the default service for `hrpdb` does not support all the service attributes and features such as service metrics, FAN, load balancing, Oracle Database Resource Manager, Transaction Guard, Application Continuity, and so on. You switch to a nondefault service as follows:

```
ALTER SESSION SET CONTAINER = hrpdb SERVICE = hrpdb_full;
```

Connections to Containers in a CDB

Typically, a CDB administrator must have appropriate privileges to provision PDBs and connect to various containers. CDB administrators are common users.

The CDB administrator can use either of the following techniques:

- Connect directly to a PDB or application root.

  The user requires the `CREATE SESSION` privilege in the container.

- Use the `ALTER SESSION SET CONTAINER` statement, which is useful for both connection pooling and advanced CDB administration, to switch between containers. The syntax is `ALTER SESSION SET CONTAINER = container_name [SERVICE = service_name]`.

  For example, a CDB administrator can connect to the root in one session, and then in the same session switch to a PDB. In this case, the user requires the `SET CONTAINER` system privilege in the container.

The following table describes a scenario involving the CDB in Figure 2-10. Each row describes an action that occurs after the action in the preceding row. Common user `SYSTEM` queries the name of the current container and the names of PDBs in the CDB.

Table 2-8 Services in a CDB

Operation	Description

`SQL> CONNECT SYSTEM@prod`
`Enter password: ********`
`Connected.`
	The `SYSTEM` user, which is common to all containers in the CDB, connects to the root using service named `prod`.

`SQL> SHOW CON_NAME`
 	
`CON_NAME`
`--------`
`CDB$ROOT`
	`SYSTEM` uses the SQL*Plus command `SHOW CON_NAME` to list the name of the container to which the user is currently connected. `CDB$ROOT` is the name of the root container.

`SQL> SELECT NAME, PDB FROM V$SERVICES`
`  2  ORDER BY PDB, NAME;`
 	
`NAME  PDB`
`----------  --------`
`SYS$BACKGROUND  CDB$ROOT`
`SYS$USERS  CDB$ROOT`
`prod.example.com  CDB$ROOT`
`erppdb.example.com  ERPPDB`
`erp.example.com  ERPPDB`
`hr.example.com  HRPDB`
`hrpdb.example.com  HRPDB`
`salespdb.example.com  SALESPDB`
 	
`8 rows selected.`
	A query of `V$SERVICES` shows that three PDBs exist with service names that match the PDB name. Both `hrpdb` and `erppdb` have an additional service.

`SQL> ALTER SESSION SET CONTAINER = hrpdb;`
 	
`Session altered.`
	`SYSTEM` uses `ALTER SESSION` to connect to `hrpdb`.

Table 2-8 (Cont.) Services in a CDB

Operation	Description

`SQL> SELECT SYS_CONTEXT`
`2 ('USERENV', 'CON_NAME')`
`3 AS CUR_CONTAINER FROM DUAL;`
 	
`CUR_CONTAINER`
`-------------`
`HRPDB`
	A query confirms that the current container is now `hrpdb`.

See Also:

- "Connecting to a PDB"
- *Oracle Database SQL Language Reference* for the syntax and semantics of `ALTER SESSION SET CONTAINER`

Overview of Tablespaces and Database Files in a CDB

A CDB has the same structure as a non-CDB, except that each PDB and application root has its own set of tablespaces, including its own `SYSTEM`, `SYSAUX`, and undo tablespaces.

A CDB contains the following files:

- One control file
- One online redo log
- One or more undo tablespaces

  Only a common user who has the appropriate privileges and whose current container is the root can create an undo tablespace. At any given time, a CDB is either in either of the following undo modes:

  – Local undo mode

    In this case, each PDB has its own undo tablespace. If a CDB is using local undo mode, then the database automatically creates an undo tablespace in every PDB. Local undo provides advantages such as the ability to perform a hot clone of a PDB, and speed the relocation of a PDB. Also, local undo provides level of isolation and enables faster unplug and point-in-time recovery operations.

    A local undo tablespace is required for each node in an Oracle Real Application Clusters (RAC) cluster in which the PDB is open. For example, if you move a PDB from a two-node cluster to a four-node cluster, and if the PDB is open in all nodes, then the database automatically creates the additional required undo tablespaces. If you move the PDB back again, then you can drop the redundant undo tablespaces.

> **Note:**
>
> By default, Database Configuration Assistant (DBCA) creates new CDBs with local undo enabled.

  - Shared undo mode

    In a single-instance CDB, only one active undo tablespace exists. For an Oracle RAC CDB, one active undo tablespace exists for every instance. All undo tablespaces are visible in the data dictionaries and related views of all containers.

  The undo mode applies to the entire CDB, which means that every container uses shared undo, or every container uses local undo. You can switch between undo modes in a CDB, which necessitates re-starting the database.

- `SYSTEM` and `SYSAUX` tablespaces for every container

  The primary physical difference between CDBs and non-CDBs is the data files in `SYSTEM` and `SYSAUX`. A non-CDB has only one `SYSTEM` tablespace and one `SYSAUX` tablespace. In contrast, the CDB root, each application root, and each PDB in a CDB has its own `SYSTEM` and `SYSAUX` tablespaces. Each container also has its own set of dictionary tables describing the objects that reside in the container.

- Zero or more user-created tablespaces

  In a typical use case, each PDB has its own set of non-system tablespaces. These tablespaces contain the data for user-defined schemas and objects in the PDB.

  Within a PDB, you manage permanent and temporary tablespaces in the same way that you manage them in a non-CDB. You can also limit the amount of storage used by the data files for a PDB by using the `STORAGE` clause in a `CREATE PLUGGABLE DATABASE` or `ALTER PLUGGABLE DATABASE` statement.

  The storage of the data dictionary within the PDB enables it to be portable. You can unplug a PDB from a CDB, and plug it in to a different CDB.

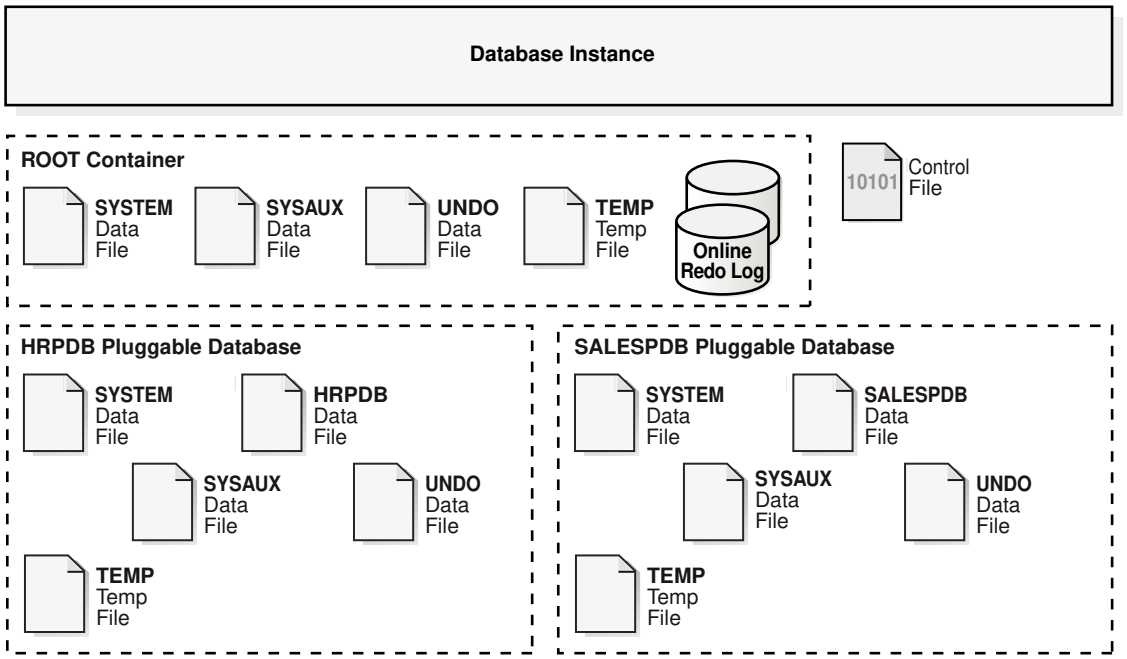
- A set of temp files for every container

  One default temporary tablespace exists for the CDB root, and one for each application root, application PDB, and PDB.

Example 2-19 CDB in Local Undo Mode

This example shows aspects of the physical storage architecture of a CDB with two PDBs: `hrpdb` and `salespdb`. In this example, the database uses local undo mode, and so has undo data files in the CDB root, `hrpdb`, and `salespdb`.

Figure 2-11 Physical Architecture of a CDB in Local Undo Mode



See Also:

- "Data Dictionary Architecture in a CDB"
- "After Creating a CDB" to learn about the state of a CDB after creation

Overview of Availability in a CDB

Many availability features that exist for a non-CDB also exist for individual PDBs within a CDB.

Overview of Backup and Recovery in a CDB

RMAN and Oracle Enterprise Manager Cloud Control provide full support for backup and recovery in a multitenant environment.

You can back up and recover a whole CDB, the root only, or one or more PDBs. You can also back up and recover individual tablespaces and data files within a PDB.

From the perspective of recovery, separately backing up the root and all PDBs is equivalent to backing up the whole CDB. The main difference is in the number of RMAN commands that you must enter and the time to recover. Recovering a whole CDB requires less time than recovering the CDB root plus all PDBs.

You can perform complete recovery of one or more PDBs without affecting operations of other open PDBs. RMAN also provides support for point-in-time recovery at the

PDB level. The procedure is similar to the procedure for point-in-time recovery of a non-CDB.

> **See Also:**
>
> *Oracle Database Backup and Recovery User's Guide* to learn about the state of a CDB after creation

Overview of Flashback PDB in a CDB

You can rewind a PDB using the `FLASHBACK PLUGGABLE DATABASE` command in SQL or Recovery Manager. This command is analogous to `FLASHBACK DATABASE` in a non-CDB.

Flashback PDB protects an individual PDB against data corruption, widespread user errors, and redo corruption. The operation does not rewind data in other PDBs in the CDB.

In releases prior to Oracle Database 12c Release 2 (12.2), you could create a restore point—an alias for an SCN—only when connected to the root. Now you can use `CREATE RESTORE POINT ... FOR PLUGGABLE DATABASE` to create a PDB restore point, which is only usable within a specified PDB. As with CDB restore points, PDB restore points can be normal or guaranteed. A guaranteed restore point never ages out of the control file and must be explicitly dropped. If you connect to the root, and if you do not specify the `FOR PLUGGABLE DATABASE` clause, then you create a CDB restore point, which is usable by all PDBs.

A special type of PDB restore point is a clean restore point, which you can only create when a PDB is closed. For PDBs with shared undo, rewinding the PDB to a clean restore point preserves database consistency and improves performance. The database avoids using the automatic infrastructure, which can reduce performance.

> **See Also:**
>
> *Oracle Database Backup and Recovery User's Guide* to learn about using `FLASHBACK PLUGGABLE DATABASE`

Overview of Oracle Resource Manager in a CDB

Using Oracle Resource Manager (Resource Manager), you can create CDB resource plans and set initialization parameters to allocate resources to PDBs.

In a non-CDB, you can use Resource Manager to manage multiple workloads that are contending for system and database resources. Therefore, in a CDB, multiple workloads within multiple PDBs can also complete for system and CDB resources.

In a CDB, Resource Manager can manage resources on two levels: CDB and PDB.

CDB Resource Plans

A CDB resource plan allocates resources to its PDBs according to its set of resource plan directives (directives). A parent-child relationship exists between a CDB resource plan and its directives. Each resource plan directive references either a set of PDBs or an individual PDB.

A performance profile specifies shares of system resources for a set of PDBs. PDB performance profiles enable you to manage resources for large numbers of PDBs by specifying Resource Manager directives for profiles instead of individual PDBs.

The directives control allocation of CPU and parallel execution servers. A directive can control the allocation of resources to PDBs based on the share value that you specify for each PDB or PDB performance profile. A higher share value results in more guaranteed resources. For PDBs and PDB performance profiles, you can also set utilization limits for CPU and parallel servers.

You can create a CDB resource plan by using the `CREATE_CDB_PLAN` procedure in the `DBMS_RESOURCE_MANAGER` PL/SQL package, and set a CDB resource plan using the `RESOURCE_MANAGER_PLAN` parameter. You create directives for a CDB resource plan by using the `CREATE_CDB_PLAN_DIRECTIVE` procedure.

PDB Resource Plans

A CDB resource plan allocates a portion of the system resources to a PDB. A PDB resource plan determines how this portion is allocated within the PDB.

Create a PDB resource plan in the same way that you create a resource plan for a non-CDB: by using procedures in the `DBMS_RESOURCE_MANAGER` package to create the plan.

You can create a PDB resource plan by using the `CREATE_PLAN` procedure in the `DBMS_RESOURCE_MANAGER` PL/SQL package, and set a PDB resource plan using the `RESOURCE_MANAGER_PLAN` parameter. You create directives for a PDB resource plan by using the `CREATE_PLAN_DIRECTIVE` procedure.

PDB-Level Memory Controls

In a CDB, PDBs may contend for SGA or PGA memory. Several initialization parameters can control the memory usage of a PDB, either guaranteeing memory or limiting memory. When you set the following initialization parameters with the PDB as the current container, the parameters control the memory usage of the current PDB.

Examples of important parameters include:

- `SGA_MIN_SIZE` sets the minimum guaranteed SGA size of the PDB.
- `SGA_TARGET` specifies the maximum SGA that the PDB can use at any time.
- `PGA_AGGREGATE_LIMIT` sets the maximum PGA that the PDB can use at any time.

PDB-Level I/O Controls

Intensive disk I/O can cause poor performance. Several factors can result in excess disk I/O, such as poorly designed SQL or index and table scans in high-volume transactions. If one PDB generates excessive disk I/O, then it can degrade the performance of other PDBs in the same CDB.

On non-Engineered Systems, use one or both of the following initialization parameters to limit the I/O generated by a particular PDB:

- `MAX_IOPS` limits the number of I/O operations for each second.
- `MAX_MBPS` limits the MB/s for I/O operations.

For Engineered Systems, manage PDB I/Os with I/O Resource Management.

See Also:

- *Oracle Database Concepts* for information about Database Resource Manager
- "Using Oracle Resource Manager for PDBs"
- *Oracle Database Reference* to learn more about `DB_CACHE_SIZE` and other initialization parameters
- *Oracle Database PL/SQL Packages and Types Reference* to learn more about the `DBMS_RESOURCE_MANAGER` package
- *Oracle Exadata Storage Server Software User's Guide* to learn more about I/O Resource Management

Part II

Creating and Configuring a Multitenant Environment

You can create and configure a multitenant environment.

3

Overview of Configuring and Managing a Multitenant Environment

Become familiar with basic concepts related to configuring and managing a multitenant environment.

About Configuring and Managing a Multitenant Environment

You can use the Oracle Multitenant option to configure and manage a multitenant environment.

You must meet certain prerequisites before configuring and managing a multitenant environment. To do so, you complete some common tasks and use a set of tools to complete those tasks.

The multitenant architecture enables an Oracle database to function as a multitenant container database (CDB) that includes zero, one, or many customer-created pluggable databases (PDBs). A PDB is a portable collection of schemas, schema objects, and nonschema objects that appears to an Oracle Net client as a non-CDB. All Oracle databases before Oracle Database 12c were non-CDBs.

> **See Also:**
>
> - "Introduction to the Multitenant Architecture "
> - "Overview of the Multitenant Architecture "

Common Users and Local Users

A common user is a user that has the same identity in the root and in every existing and future PDB.

A common user can log in to the root and any container in which it has been granted `CREATE SESSION` privilege. The operations that a common user can perform depend on the privileges granted to the common user. Some administrative tasks, such as creating a PDB or unplugging a PDB, must be performed by a common user.

A CDB also supports local users. A local user is a user that exists in exactly one PDB.

> **See Also:**
>
> - "Overview of Commonality in the CDB"
> - *Oracle Database Security Guide* for more information about common users and local users

Separation of Duties in CDB and PDB Administration

Some database administrators manage an entire CDB, while others manage individual PDBs.

DBAs who manage an entire CDB connect to the CDB as common users, and manage attributes of the entire CDB and the root, as well as some attributes of PDBs. For example, these DBAs can create, unplug, plug in, and drop PDBs. They can also specify the temporary tablespace and the default tablespace for the root, and they can change the open mode of PDBs.

DBAs can also connect to a specific PDB as a local PDB administrator and then perform a subset of management tasks on the PDB that a DBA performs on a non-CDB. The subset includes tasks required for the PDB to support an application. For example, tasks can include management of tablespaces and schemas in a PDB, specification of storage parameters for that PDB, changing the open mode of the current PDB, and setting PDB-level initialization parameters.

Prerequisites for a Multitenant Environment

Prerequisites must be met for a multitenant environment.

The following minimum prerequisites must be met before you can create and use a multitenant environment:

- You must install or upgrade to Oracle Database 12c or later releases. Oracle Multitenant is not supported in Oracle Database 11g and earlier releases.

  The installation includes setting various environment variables unique to your operating system and establishing the directory structure for software and database files.
- The database compatibility level must be set to `12.0.0` or later.
- Sufficient memory must be available to start the Oracle Database instance.

  Size the memory required by a CDB to accommodate the workload of each of its containers and the number of containers.
- Sufficient disk storage space must be available for the planned PDBs on the computer that runs Oracle Database. In an Oracle RAC environment, sufficient shared storage must be available.

  The disk storage space required by a CDB is the sum of the space requirements for all PDBs that will reside in the CDB.

These prerequisites are discussed in the *Oracle Database Installation Guide* or *Oracle Grid Infrastructure Installation and Upgrade Guide* specific to your operating system. If you use the Oracle Universal Installer, then it will guide you through your installation

and provide help in setting environment variables and establishing directory structure and authorizations.

> **See Also:**
>
> - *Oracle Database Installation Guide* specific to your operating system
> - *Oracle Database Upgrade Guide* for information about the database compatibility level

Tasks and Tools for a Multitenant Environment

There are common tasks you perform for a multitenant environment, and you use tools to complete the tasks.

Tasks for a Multitenant Environment

A multitenant environment enables you to achieve several goals. You can complete general tasks to configure and use a multitenant environment.

These goals are described in "Benefits of the Multitenant Architecture". To do so, you must complete the following general tasks:

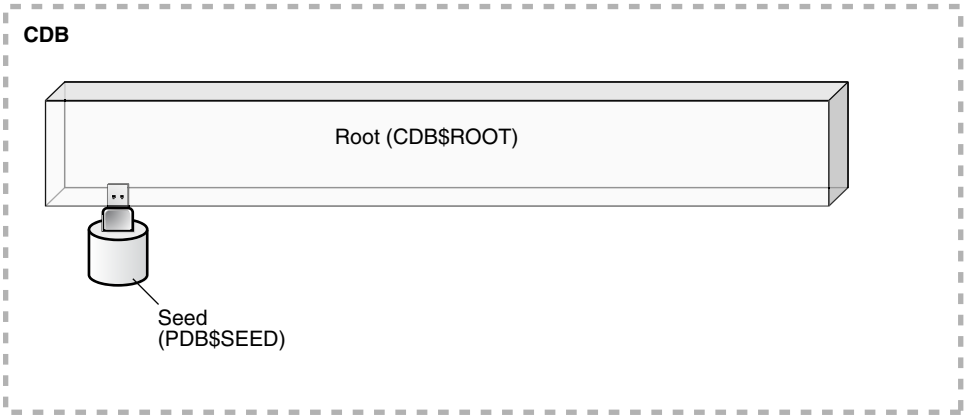
Task 1 Plan for the Multitenant Environment
Creating and configuring any database requires careful planning. A CDB requires special considerations. For example, consider the following factors when you plan for a CDB:

- The number of PDBs that will be plugged into each CDB
- The resources required to support the planned CDB
- Container management policies executed as an aggregate on the entire CDB or executed locally on individual PDBs
- Container database topology, which could consist of application containers with application PDBs or a CDB with PDBs, or a combination of both

See "Planning for CDB Creation" for detailed information about planning for a CDB.

Task 2 Create One or More CDBs
When you have completed the necessary planning, you can create one or more CDBs using either the Database Configuration Assistant (DBCA) or the `CREATE DATABASE` statement. In either case, you must specify the configuration details for each CDB. See "About CDB Creation with DBCA" and "Creating a CDB" for detailed information about creating a CDB.
After a CDB is created, it consists of the root and `PDB$SEED`, as shown in Figure 3-1. The CDB root contains only Oracle maintained objects and data structures, and `PDB$SEED` is a generic seed database for cloning purposes.

Figure 3-1 A Newly Created CDB

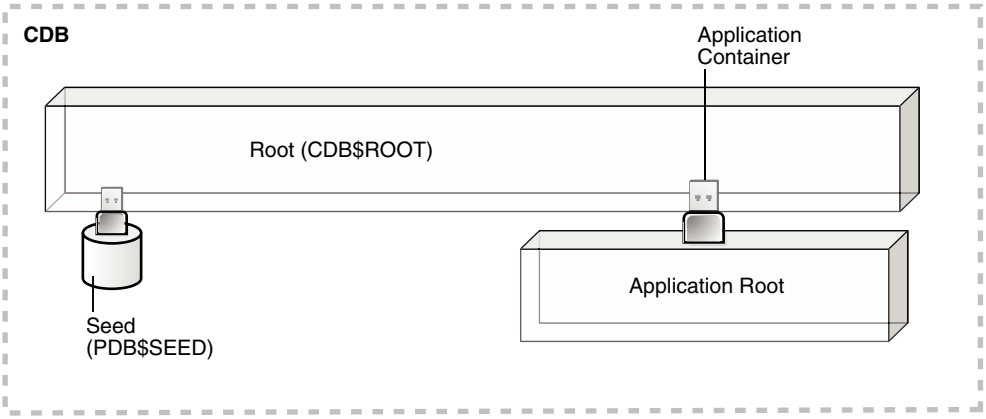


Task 3 Optionally, Create Application Containers

An application container is an optional component of a CDB that consists of an application root and the application PDBs associated with it. An application container stores data for one or more applications.

The following graphic shows a CDB with one empty application container.

Figure 3-2 An Application Container



See "Overview of Applications in an Application Container".

Task 4 Create, Plug In, and Unplug PDBs

PDBs contain user data. After creating a CDB, you can create PDBs, plug unplugged PDBs into it, and unplug PDBs from it whenever necessary. You can unplug a PDB from a CDB and plug this PDB into a different CDB. You might move a PDB from one CDB to another if, for example, you want to move the workload for the PDB from one server to another.

See "Creating and Removing PDBs and Application Containers" for information about creating PDBs, plugging in PDBs, and unplugging PDBs.

Figure 3-3 shows a CDB with several PDBs.

Figure 3-3 A CDB with PDBs

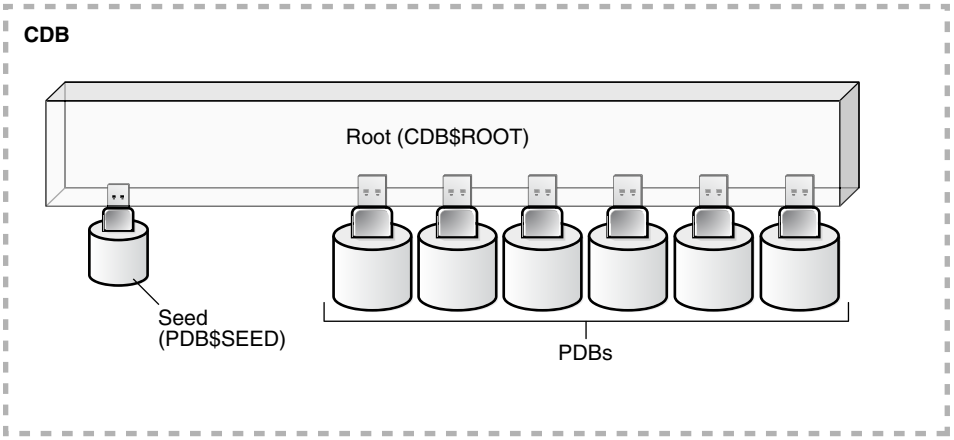
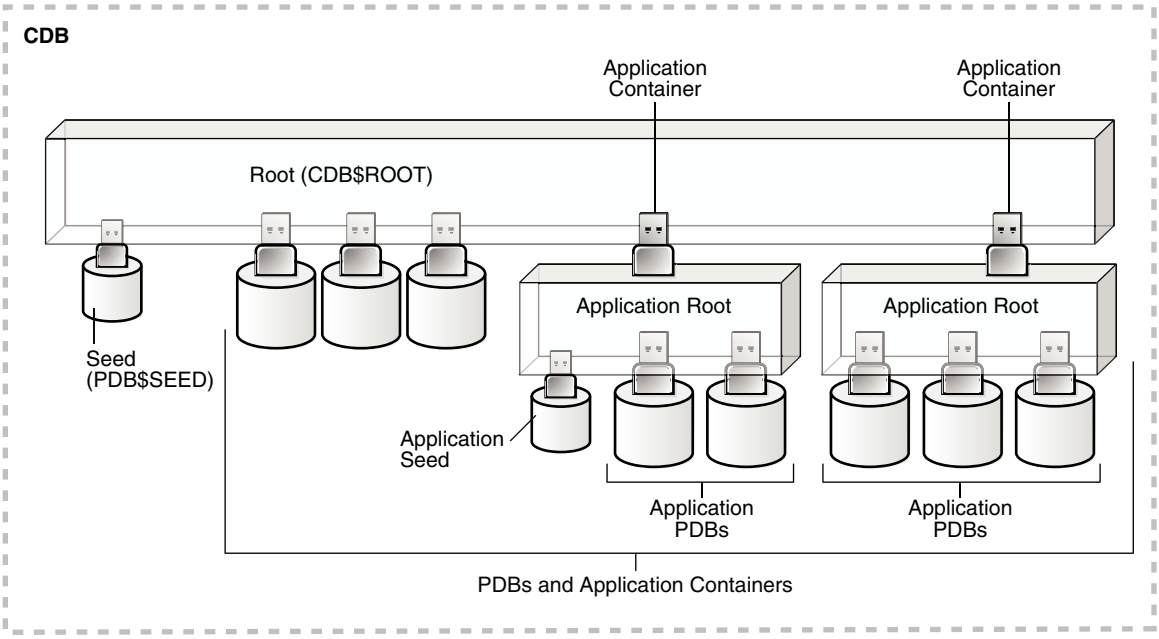


Figure 3-4 shows a CDB with PDBs, application containers, and application PDBs.

Figure 3-4 A CDB with PDBs, Application Containers, and Application PDBs



Task 5 Administer and Monitor the CDB and Application Containers
Administering and monitoring a CDB involves managing the entire CDB, the root, and some attributes of PDBs. Some management tasks are the same for CDBs and non-CDBs, and some are different.
Administering and monitoring an application container is similar to administering and monitoring a CDB, but your actions only affect the application root and the application PDBs that are part of the application container.
See "After Creating a CDB" for descriptions of tasks that are similar and tasks that are different. Also, see " Administering a CDB" and "Monitoring CDBs and PDBs".

You can use Oracle Resource Manager to allocate and manage resources among PDBs hosted in a CDB, and you can use it to allocate and manage resource use among user processes within a PDB. See "Using Oracle Resource Manager for PDBs".
You can also use Oracle Scheduler to schedule jobs in a CDB and in individual PDBs. See " Using Oracle Scheduler with a CDB".

Task 6 Administer and Monitor PDBs and Application PDBs
Administering and monitoring a PDB or an application PDB is similar to administering and monitoring a non-CDB, but there are some differences. See " Administering PDBs" and "Monitoring CDBs and PDBs".

Tools for a Multitenant Environment

You can use various tools to configure and administer a multitenant environment.

Table 3-1 Tools for a Multitenant Environment

Tool	Description	See Also
SQL*Plus	SQL*Plus is a command-line tool that enables you to create, manage, and monitor CDBs and PDBs. You use SQL statements and Oracle-supplied PL/SQL packages to complete these tasks in SQL*Plus.	*SQL*Plus User's Guide and Reference*
Oracle Database Configuration Assistant (DBCA)	DBCA is a utility with a graphical user interface that enables you to create and duplicate CDBs. It also enables you to create, relocate, clone, plug in, and unplug PDBs.	*Oracle Database 2 Day DBA*, *Oracle Database Installation Guide*, and the DBCA online help
Oracle Enterprise Manager Cloud Control	Cloud Control is a system management tool with a graphical user interface that enables you to manage and monitor a CDB and its PDBs.	Cloud Control online help
Oracle SQL Developer	Oracle SQL Developer is a client application with a graphical user interface that enables you to configure a CDB, create PDBs, plug and unplug PDBs, modify the state of a PDB, clone a PDB to the Oracle Cloud, hot clone/refresh a PDB, relocate a PDB between application roots, and more. Additionally, Oracle SQL Developer has graphical interfaces for resource management, storage, security, configuration, and reporting of performance metrics on containers and pluggable databases in a CDB.	*Oracle SQL Developer User's Guide*
The Server Control (SRVCTL) utility	The SRVCTL utility can create and manage services for PDBs.	"Managing Services for PDBs"

Table 3-1 (Cont.) Tools for a Multitenant Environment

Tool	Description	See Also
EM Express	EM Express is a management and monitoring tool with a graphical user interface which ships with Oracle Database. It can be configured for the CDB, individual hosted PDBs, or both. This tool is intended for PDB administrative use, in the context of the PDB, to manage and monitor application development as an application DBA.	*Oracle Database 2 Day DBA*
Oracle Multitenant Self-Service Provisioning application	This application enables the self-service provisioning of PDBs. CDB administrators control access to this self-service application and manage quotas on PDBs.	`http://www.oracle.com/goto/multitenant` To access the application, click the **Downloads** tab, and select **Oracle Pluggable Database Self-Service Provisioning application** in the Downloads for Oracle Multitenant section.

4

Creating and Configuring a CDB

Creating and configuring a multitenant container database (CDB) includes tasks such as planning, creating the CDB, and optionally configuring EM Express.

About Creating a CDB

The procedure for creating a multitenant container database (CDB) is similar to the procedure for creating a non-CDB.

The procedure for creating a non-CDB is described in *Oracle Database Administrator's Guide*. Before creating a CDB, you must understand the concepts and tasks described in this documentation.

This chapter describes special considerations for creating a CDB. This chapter also describes differences between the procedure for creating a non-CDB in *Oracle Database Administrator's Guide* and the procedure for creating a CDB.

After you plan your CDB using some of the guidelines presented in "Planning for CDB Creation", you can create the CDB either during or after Oracle Database software installation. The following are typical reasons to create a CDB after installation:

- You used Oracle Universal Installer (OUI) to install software only, and did not create a CDB.
- You want to create another CDB on the same host as an existing CDB or an existing non-CDB. In this case, this chapter assumes that the new CDB uses the same Oracle home as the existing database. You can also create the CDB in a new Oracle home by running OUI again.

The specific methods for creating a CDB are:

- With the Database Configuration Assistant (DBCA), a graphical tool.

  See "About CDB Creation with DBCA".
- With the `CREATE DATABASE` SQL statement.

  See "Creating a CDB".

Planning for CDB Creation

CDB creation prepares several operating system files to work together as a CDB.

> **Note:**
>
> Before planning for CDBs, review the conceptual information about CDBs and PDBs in "Introduction to the Multitenant Architecture ".

Decide How to Configure the CDB

Prepare to create the CDB by research and careful planning.

As a first step, consult the *Oracle Database Release Notes*, which includes a list of Oracle Database features that are currently not supported in a CDB. If you must use one or more of these features, then create a non-CDB.

The following sections describe recommended actions and considerations that apply to CDBs:

Plan the PDBs

Plan the tables and indexes for the pluggable databases (PDBs) and estimate the amount of space they require.

In a CDB, most user data resides in the PDBs. The root contains no user data or minimal user data. Plan for the PDBs that will be part of the CDB. The disk storage space requirement for a CDB is the space required for the Oracle Database installation plus the sum of the space requirements for all PDBs that will be part of the CDB.

The `MAX_PDBS` initialization parameter specifies a limit on the total number of PDBs that you can create in a CDB root or application root. The default value and maximum value for `MAX_PDBS` depend on your Oracle Database offering. See *Oracle Database Licensing Information User Manual* for details on which features are supported for different editions and services.

You can also create application containers in a CDB. An application container is a collection of application PDBs that store the data for one or more applications. In addition, application containers support user-created application common objects that can be shared by the application PDBs in the application container.

See Also:

- "Creating and Removing PDBs and Application Containers"
- "Overview of Applications in an Application Container"
- *Oracle Database Administrator's Guide* to learn more about database structure and storage and schema objects
- *Oracle Database Reference* to learn more about `MAX_PDBS`

Plan the Physical Layout

Plan the layout of the underlying operating system files your CDB will comprise.

There are separate data files for the CDB root, `PDB$SEED`, each PDB, each application root, and each application PDB.

There is one redo log for a single-instance CDB, or one redo log for each instance of an Oracle Real Application Clusters (Oracle RAC) CDB. Also, for Oracle RAC, all data files and redo log files must be on shared storage.

See Also:

- *Oracle Database Administrator's Guide* for information about using Oracle Managed Files
- *Oracle Automatic Storage Management Administrator's Guide*
- *Oracle Database Performance Tuning Guide*
- *Oracle Database Backup and Recovery User's Guide*
- *Oracle Grid Infrastructure Installation and Upgrade Guide* for information about configuring storage for Oracle RAC
- Your Oracle operating system–specific documentation, including the appropriate Oracle Database installation guide.

Learn How to Manage Initialization Parameters

Familiarize yourself with the initialization parameters that can be included in an initialization parameter file.

Before creating a CDB, ensure that you are familiar with the concept and operation of a server parameter file (SPFILE). An SPFILE file lets you store and manage your initialization parameters persistently in a server-side binary file.

A CDB uses a single SPFILE or a single text initialization parameter file (PFILE). Values of initialization parameters set for the root can be inherited by PDBs. You can set some initialization parameters for a PDB by using the `ALTER SYSTEM` statement.

The CDB root must be the current container when you operate on an SPFILE. The user who creates or modifies the SPFILE must be a common user with `SYSDBA`, `SYSOPER`, or `SYSBACKUP` administrative privilege, and the user must exercise the privilege by connecting `AS SYSDBA`, `AS SYSOPER`, or `AS SYSBACKUP` respectively.

The following initialization parameters are important:

- To create a CDB, the `ENABLE_PLUGGABLE_DATABASE` initialization parameter must be set to `TRUE`.
- Create the global database name for the CDB root by setting both the `DB_NAME` and `DB_DOMAIN` initialization parameters. The global database name of the root is the global database name of the CDB. The global database name of a PDB is defined by the PDB name and the `DB_DOMAIN` initialization parameter.

See Also:

- "About the Current Container"
- "Modifying a CDB with ALTER SYSTEM"
- "Listing the Modifiable Initialization Parameters in PDBs"
- *Oracle Database Administrator's Guide* for information about schema objects
- *Oracle Database Administrator's Guide* for information about determining the global database name
- *Oracle Database Reference*

Select the Character Set

You must choose a character set for the CDB.

When selecting the database character set for the CDB, you must consider the current character sets of the databases that you want to consolidate (plug) into this CDB. Oracle recommends AL32UTF8 for the CDB database character set and AL16UTF6 for the CDB national character set because they provide the most flexibility.

When the character set of the CDB root is AL32UTF8, PDBs that are plugged into the CDB can have a different character set from the CDB root. PDBs that are created from `PDB$SEED` inherit the AL32UTF8 character set from it, but you can migrate the PDB to a different character set. When the character set of the root is not AL32UTF8, all PDBs in the CDB use the character set of the CDB root.

Note:

Oracle Multitenant does not support a LOB in one container from being accessed by a container with a different character set using data links, extended data links, or the `CONTAINERS()` clause. For example, if the CDB root and `salespdb` have different character sets, then a `CONTAINERS()` query run in the CDB root should not access LOBs stored in `salespdb`.

When moving a non-CDB to a CDB, it is best to migrate the non-CDB to AL32UTF8 first. You can use Oracle Database Migration Assistant for Unicode (DMU) to migrate a non-CDB to AL32UTF8. After a CDB is created, you cannot migrate the character set of the CDB using DMU.

See Also:

Oracle Database Globalization Support Guide

Decide Which Time Zones to Support

Consider which time zones your CDB must support.

You can set the time zones for the entire CDB (including all PDBs). You can also set the time zones individually for each PDB.

> **See Also:**
>
> *Oracle Database Administrator's Guide* for information about specifying the database time zone and time zone file

Select the Database and Redo Log Block Sizes

Select the standard database block size for the CDB.

This is specified at CDB creation by the `DB_BLOCK_SIZE` initialization parameter and cannot be changed after the CDB is created. The standard block size applies to the entire CDB.

If you plan to store online redo log files on disks with a 4K byte sector size, then determine whether you must manually specify the online redo log block size.

- *Oracle Database Administrator's Guide* for information about specifying database block sizes
- *Oracle Database Administrator's Guide* for information about planning the block size of redo log files

Plan the SYSTEM and SYSAUX Tablespaces

There is a separate `SYSAUX` and `SYSTEM` tablespace for the CDB root and for each PDB.

You must determine the appropriate initial sizing for the `SYSAUX` tablespace. Also, plan to use a default tablespace for non-`SYSTEM` users to prevent inadvertently saving database objects in the `SYSTEM` tablespace. You can specify a separate default tablespace for the CDB root and for each PDB.

> **See Also:**
>
> - *Oracle Database Administrator's Guide* for information about the `SYSAUX` tablespace
> - *Oracle Database Administrator's Guide* for information about creating a default permanent tablespace
> - "About Container Modification When Connected to CDB Root"

Plan the Temporary Tablespaces

Plan to use default temporary tablespaces.

A default temporary tablespace exists for every container in the CDB. Therefore, the CDB root and every PDB, application root, and application PDB has its own default temporary tablespace.

Oracle Database uses the shared temporary tablespace for recursive SQL only. Hosted PDB tenants do not use this tablespace directly.

See Also:

- "About Container Modification When Connected to CDB Root"
- *Oracle Database Administrator's Guide* for information about creating a default temporary tablespace

Choose the Undo Mode

Plan to use an undo tablespace to manage your undo data.

A CDB can run in different undo modes. You can configure a CDB to have one active undo tablespace for the entire CDB or a separate undo tablespace for each container in the CDB. You can specify the undo mode during CDB creation, and you can change the undo mode after the CDB is created.

When you choose to have one active undo tablespace for the entire CDB, shared undo is used, and local undo is disabled. In this configuration, there is one active undo tablespace for a single-instance CDB. When local undo is enabled, there is one undo tablespace for each container in a single instance configuration. For an Oracle RAC CDB, each PDB has one undo tablespace in each node in which it is open. With shared undo, only a common user who has the appropriate privileges and whose current container is the root can create an undo tablespace.

The best practice is to use local undo for a CDB. Shared undo is supported primarily for upgrade and transitional purposes only. Although there is minor overhead associated with local undo when compared with shared undo, the benefits of local undo make it preferable in most environments. Local undo makes unplug operations and point in time recovery faster, and it is required for some features, such as relocating a PDB. By default, DBCA creates new CDBs with local undo enabled.

In a CDB, the `UNDO_MANAGEMENT` initialization parameter must be set to `AUTO`, and an undo tablespace is required to manage the undo data.

When local undo is not enabled, undo tablespaces are visible in static data dictionary views and dynamic performance (V$) views when the current container is the root. Undo tablespaces are visible only in dynamic performance views when the current container is a PDB.

Also, when local undo is disabled, Oracle Database silently ignores undo tablespace and rollback segment operations when the current container is a PDB.

> **See Also:**
>
> - "Setting the Undo Mode in a CDB Using ALTER DATABASE"
> - "About the Current Container"
> - *Oracle Database Administrator's Guide* for information about managing undo

Plan the Services for Your Application

Plan for the database services required to meet the needs of your applications.

The root and each PDB might require several services. You can create services for the root or for individual PDBs.

Database services have an optional `PDB` property. You can create services and associate them with a particular PDB by specifying the `PDB` property. Services with a null `PDB` property are associated with the CDB root.

You can also use the `DBMS_SERVICE` supplied PL/SQL package to create services and associate them with PDBs. When you run `CREATE_SERVICE` procedure, the service is associated with the current container.

You can manage services with the SRVCTL utility, Oracle Enterprise Manager Cloud Control, and the `DBMS_SERVICE` supplied PL/SQL package.

When you create a PDB, a new default service for the PDB is created automatically. The service has the same name as the PDB. You cannot manage this service with the SRVCTL utility. However, you can create user-defined services and customize them for your applications.

> **See Also:**
>
> - "Managing Services for PDBs"
> - *Oracle Database Administrator's Guide* to learn about database services and using SRVCTL with a single-instance database
> - *Oracle Real Application Clusters Administration and Deployment Guide* for information about using the SRVCTL utility with an Oracle RAC database

Learn How to Start Up and Shut Down a CDB

Familiarize yourself with the principles and options of starting up and shutting down a database instance and mounting and opening a CDB.

In a CDB, the CDB root and all containers share a single database instance, or, when using Oracle RAC, multiple concurrent instances. You can start up and shut down an entire CDB, which in turn determines the state of hosted PDBs. When the CDB is open, you can control the open mode of PDBs by using either an `ALTER PLUGGABLE`

DATABASE statement in the context of the CDB or PDB to open or close hosted PDBs. To maintain backward compatibility, the ALTER DATABASE OPEN statement is supported when it is executed and a PDB is the current container. You can also use the SQL*Plus STARTUP command and the SQL*Plus SHUTDOWN command when a PDB is the current container. However, the SQL*Plus STARTUP MOUNT command is a CDB-only operation and cannot be used when a PDB is the current container.

See Also:

- "Modifying the Open Mode of PDBs"
- "Modifying a PDB with the ALTER PLUGGABLE DATABASE Statement"
- *Oracle Database Administrator's Guide* for information about starting up a database

Plan for Oracle RAC

If you plan to use Oracle RAC, then plan for an Oracle RAC environment.

The Oracle RAC documentation describes special considerations for a CDB in an Oracle RAC environment. See your platform-specific Oracle RAC installation guide for information about creating a CDB in an Oracle RAC environment.

See Also:

Oracle Real Application Clusters Administration and Deployment Guide

Prerequisites for CDB Creation

You must complete prerequisites before creating a new CDB.

Before you can create a new CDB, the following prerequisites must be met:

- Ensure that the prerequisites described in "Prerequisites for a Multitenant Environment" are met.
- Sufficient memory must be available to start the Oracle Database instance.

  Size the memory required by a CDB to accommodate the workload of each of its containers and the number of containers.
- Sufficient disk storage space must be available for the planned PDBs on the computer that runs Oracle Database. In an Oracle RAC environment, sufficient shared storage must be available.

  The disk storage space required by a CDB is the sum of the space requirements for all PDBs that will reside in the CDB.

These prerequisites are discussed in the *Oracle Database Installation Guide* or *Oracle Grid Infrastructure Installation and Upgrade Guide* specific to your operating system. If you use the Oracle Universal Installer, then it will guide you through your installation

and provide help in setting environment variables and establishing directory structure and authorizations.

Creating a CDB

You can create a CDB using DBCA or by manually issuing the `CREATE DATABASE` SQL statement.

> **Note:**
>
> Oracle strongly recommends using the Database Configuration Assistant (DBCA) template deployment instead of the `CREATE DATABASE` SQL statement to create a CDB, because using DBCA is a more automated approach, and your CDB is ready to use when DBCA completes.

About CDB Creation with DBCA

Oracle strongly recommends using the Database Configuration Assistant (DBCA) to create a CDB.

DBCA offers the following advantages over alternative techniques:

- Creation is largely automated.
- DBCA enables you to specify the number of PDBs in the CDB when it is created.
- When DBCA completes, the CDB is ready to use.
- After a CDB is created, you can use DBCA to do the following:
  - Clone local PDBs
  - Plug in and unplug PDBs
  - Duplicate a CDB (silent mode only)

Depending on the type of install that you select, Oracle Universal Installer (OUI) can launch DBCA. You can also launch DBCA as a standalone tool at any time after Oracle Database installation.

You can use DBCA to create a CDB in either of the following modes:

- Interactive mode

  This mode provides a graphical interface and guided workflow for creating and configuring a CDB.

- Noninteractive mode (also called *silent mode*)

  This mode enables you to script a preconfigured CDB template deployment with customized PDB seed databases that are suitable for cloning. Run DBCA in silent mode by specifying command-line arguments, a response file, or both.

> **See Also:**
>
> - *Oracle Database Administrator's Guide* to learn how to create a database with DBCA
> - *Oracle Database 2 Day DBA*
> - The DBCA online help

About CDB Creation with SQL Statements

Creating a CDB using the `CREATE DATABASE` SQL statement is similar to creating a non-CDB.

This section describes additional requirements for creating a CDB. When you create a CDB using `CREATE DATABASE`, you must do the following:

- Enable PDBs
- Specify the names and locations of the CDB root files
- Specify the names and locations of the `PDB$SEED` files

> **Note:**
>
> Using the `CREATE DATABASE` SQL statement is a more manual approach to creating a CDB than using DBCA.

> **See Also:**
>
> *Oracle Database Concepts* for information about the files in a CDB

About Enabling PDBs

To create a CDB, the `CREATE DATABASE` statement must include the `ENABLE PLUGGABLE DATABASE` clause.

This clause affects the `CREATE DATABASE` statement as follows:

- `ENABLE PLUGGABLE DATABASE` is included

  The statement creates a CDB with the root and `PDB$SEED`. You can never change the CDB into a non-CDB.

- `ENABLE PLUGGABLE DATABASE` is not included

  The newly created database is a non-CDB, which means that it does not include the CDB root or `PDB$SEED`. The non-CDB can never contain PDBs.

About the Names and Locations of Files for the CDB Root and PDB$SEED

To create the CDB, Oracle Database must know the names and locations of the files for the CDB root and PDB$SEED.

After the CREATE DATABASE statement completes successfully, you can use PDB$SEED and its files to create new PDBs. You cannot modify the PDB seed after it is created.

You must specify the names and locations of the files for PDB$SEED in one of the following ways:

1. The ENABLE PLUGGABLE DATABASE SEED FILE_NAME_CONVERT clause of CREATE DATABASE
2. Oracle Managed Files
3. The PDB_FILE_NAME_CONVERT initialization parameter

If you use more than one technique, then the CREATE DATABASE statement uses one technique in the order of precedence of the list. For example, if you use all techniques, then the CREATE DATABASE statement only uses the specifications in the ENABLE PLUGGABLE DATABASE SEED FILE_NAME_CONVERT clause because it is first in the list.

> **See Also:**
>
> "Creating a PDB from Scratch"

The SEED FILE_NAME_CONVERT Clause

The SEED FILE_NAME_CONVERT clause of the CREATE DATABASE statement specifies how to generate the names of the PDB$SEED files using the names of the CDB root files.

You can use this clause to specify one of the following options:

- One or more file name patterns and replacement file name patterns, in the following form:

  ```
  'string1' , 'string2' , 'string3' , 'string4' , ...
  ```

  The *string2* file name pattern replaces the *string1* file name pattern, and the *string4* file name pattern replaces the *string3* file name pattern. You can use as many pairs of file name pattern and replacement file name pattern strings as required.

  If you specify an odd number of strings (the last string has no corresponding replacement string), then an error is returned. Do not specify more than one pattern/replace string that matches a single file name or directory.

  File name patterns cannot match files or directories managed by Oracle Managed Files.

- NONE when no file names should be converted. Omitting the SEED FILE_NAME_CONVERT clause is the same as specifying NONE.

Example 4-1 SEED FILE_NAME_CONVERT Clause

This SEED FILE_NAME_CONVERT clause generates file names for the PDB$SEED files in the /oracle/pdbseed/ directory using file names in the /oracle/dbs directory.

```
SEED FILE_NAME_CONVERT = ('/oracle/dbs/', '/oracle/pdbseed/')
```

> **See Also:**
>
> *Oracle Database SQL Language Reference* for the syntax of the SEED FILE_NAME_CONVERT clause

Oracle Managed Files

When Oracle Managed Files is enabled, it can determine the names and locations of the PDB$SEED files.

> **See Also:**
>
> *Oracle Database Administrator's Guide*

The PDB_FILE_NAME_CONVERT Initialization Parameter

The PDB_FILE_NAME_CONVERT initialization parameter can specify the names and locations of the seed's files.

To use this technique, ensure that the PDB_FILE_NAME_CONVERT initialization parameter is included in the initialization parameter file when you create the CDB.

File name patterns specified in this initialization parameter cannot match files or directories managed by Oracle Managed Files.

> **See Also:**
>
> *Oracle Database Reference*

About the Attributes of the Data Files for PDB$SEED

You can use the PDB seed (PDB$SEED) as a template to create new containers.

The attributes of the data files for the CDB root SYSTEM and SYSAUX tablespaces might not be suitable for the PDB seed. In this case, you can specify different attributes for the PDB seed data files by using the *tablespace_datafile* clauses. Use these clauses to specify attributes for all data files comprising the SYSTEM and SYSAUX tablespaces in the PDB seed. The values inherited from the root are used for any attributes whose values have not been provided.

The syntax of the *tablespace_datafile* clauses is the same as the syntax for a data file specification, excluding the name and location of the data file and the REUSE attribute. You can use the *tablespace_datafile* clauses with any of the methods for specifying the names and locations of the PDB seed's data files described in "About the Names and Locations of Files for the CDB Root and PDB$SEED".

The *tablespace_datafile* clauses do not specify the names and locations of the PDB seed's data files. Instead, they specify the attributes of SYSTEM and SYSAUX data files in the PDB seed that differ from those in the root. If SIZE is not specified in the *tablespace_datafile* clause for a tablespace, then data file size for the tablespace is set to a predetermined fraction of the size of a corresponding root data file.

Example 4-2 Using the tablespace_datafile Clauses

Assume the following CREATE DATABASE clauses specify the names, locations, and attributes of the data files that comprise the SYSTEM and SYSAUX tablespaces in the root.

```
DATAFILE '/u01/app/oracle/oradata/newcdb/system01.dbf'
  SIZE 325M REUSE
SYSAUX DATAFILE '/u01/app/oracle/oradata/newcdb/sysaux01.dbf'
  SIZE 325M REUSE
```

You can use the following *tablespace_datafile* clauses to specify different attributes for these data files:

```
SEED
  SYSTEM DATAFILES
    SIZE 125M AUTOEXTEND ON NEXT 10M MAXSIZE UNLIMITED
  SYSAUX DATAFILES
    SIZE 100M
```

In this example, the data files for the PDB seed's SYSTEM and SYSAUX tablespaces inherit the REUSE attribute from the root's data files. However, the following attributes of the PDB seed's data files differ from the root's:

- The data file for the SYSTEM tablespace is 125 MB for the PDB seed and 325 MB for the root.
- AUTOEXTEND is enabled for the PDB seed's SYSTEM data file, and it is disabled by default for the root's SYSTEM data file.
- The data file for the SYSAUX tablespace is 100 MB for the PDB seed and 325 MB for the root.

See Also:

Oracle Database SQL Language Reference for information about data file specifications

About the CDB Undo Mode

Shared undo is the default. You can use the *undo_mode_clause* to an `ENABLE PLUGGABLE DATABASE` clause to specify the undo mode of the CDB.

The *undo_mode_clause* specifies whether the CDB undo mode is local or shared. Local undo mode means that every container in the CDB uses local undo. To configure local undo mode for the CDB, specify `LOCAL UNDO ON`.

Shared undo mode means that there is one active undo tablespace for a single-instance CDB, or for an Oracle RAC CDB, there is one active undo tablespace for each instance. To configure shared undo mode for the CDB, either do not specify *undo_mode_clause*, or specify `LOCAL UNDO OFF`.

Creating a CDB with the CREATE DATABASE Statement

When you use the `CREATE DATABASE` statement to create a CDB, you must complete additional actions before you have an operational CDB.

These actions include building views on the data dictionary tables and installing standard PL/SQL packages in the root. Perform these actions by running the supplied `catcdb.sql` script, which installs all components required by a CDB.

Prerequisites

Note the following prerequisites:

- The instructions in this section apply to *single-instance database installations only*. See the Oracle Real Application Clusters (Oracle RAC) installation guide for your platform for instructions for creating an Oracle RAC CDB.
- If you are using Oracle ASM to manage your disk storage, then you must start the Oracle ASM instance and configure your disk groups before performing these steps.
- The `ENABLE_PLUGGABLE_DATABASE` initialization parameter must be set to `true`.

To create a CDB with the CREATE DATABASE statement:

1. Complete steps 1 - 8 in the “Creating a Database with the `CREATE DATABASE` Statement” topic in *Oracle Database Administrator’s Guide*.

   In a CDB, the `DB_NAME` initialization parameter specifies the name of the root. Also, it is common practice to set the SID to the name of the root. The maximum number of characters for this name is 30. For more information, see the discussion of the `DB_NAME` initialization parameter in *Oracle Database Reference*.

2. Use the `CREATE DATABASE` statement to create a new CDB.

   See the examples in "Creating a CDB Using Oracle Managed Files: Example" and "Creating a CDB Without Using Oracle Managed Files".

> **Tip:**
>
> If your `CREATE DATABASE` statement fails, and if you did not complete Step 7 in the "Creating a Database with the `CREATE DATABASE` Statement" topic in *Oracle Database Administrator's Guide*, then ensure that there is not a preexisting server parameter file (SPFILE) for this instance that is setting initialization parameters in an unexpected way. For example, an SPFILE contains a setting for the complete path to all control files, and the `CREATE DATABASE` statement fails if those control files do not exist. Ensure that you shut down and restart the instance (with `STARTUP NOMOUNT`) after removing an unwanted SPFILE.

3. Run the `catcdb.sql` SQL script.

   Enter the following in SQL*Plus to run the script:

   ```
   @?/rdbms/admin/catcdb.sql
   ```

4. When prompted by the script, enter the log file directory for parameter `1` and the log file name for parameter `2`.

   For following example enters `/tmp` for the first prompt and `create_cdb.log` for the second prompt:

   ```
   SQL> host perl -I &&rdbms_admin &&rdbms_admin_catcdb --logDirectory
   &&1 --logFilename &&2
   Enter value for 1: /tmp
   Enter value for 2: create_cdb.log
   ```

5. When prompted by the script, enter any other required information.

   For example, the scripts prompts for administrator passwords and the temporary tablespace name:

   ```
   Enter new password for SYS: ********
   Enter new password for SYSTEM: ********
   Enter temporary tablespace name: TEMP
   ```

6. After `catcdb.sql` completes, perform steps 12 - 14 in the "Creating a Database with the `CREATE DATABASE` Statement" topic in *Oracle Database Administrator's Guide*.

> **See Also:**
>
> - *Oracle Database Administrator's Guide* to learn more about `CREATE DATABASE` clauses and Oracle Managed Files
> - *Oracle Real Application Clusters Administration and Deployment Guide* for more information on Oracle RAC
> - *Oracle Automatic Storage Management Administrator's Guide*
> - *Oracle Database SQL Language Reference* to learn more about the clauses and parameter values for the `CREATE DATABASE` statement

Creating a CDB with the CREATE DATABASE Statement: Examples

These examples create a CDB named `newcdb`.

The examples assume that you completed steps 1 - 8 in the "Creating a Database with the `CREATE DATABASE` Statement" topic in *Oracle Database Administrator's Guide*.

Creating a CDB Without Using Oracle Managed Files

The following statement creates a CDB named `newcdb`. This name must agree with the `DB_NAME` parameter in the initialization parameter file.

Assumptions

This example assumes the following:

- The initialization parameter file specifies the number and location of control files with the `CONTROL_FILES` parameter.
- The directory `/u01/app/oracle/oradata/newcdb` exists.
- The directory `/u01/app/oracle/oradata/pdbseed` exists.
- The directories `/u01/logs/my` and `/u02/logs/my` exist.

This example includes the `ENABLE PLUGGABLE DATABASE` clause to create a CDB with the root and the PDB seed. This example also includes the `SEED FILE_NAME_CONVERT` clause to specify the names and locations of the PDB seed's files. This example also includes *`tablespace_datafile`* clauses that specify attributes of the PDB seed data files for the `SYSTEM` and `SYSAUX` tablespaces that differ from the root data files. This example includes the *`undo_mode_clause`* to specify that the CDB undo mode is local.

```
CREATE DATABASE newcdb
  USER SYS IDENTIFIED BY sys_password
  USER SYSTEM IDENTIFIED BY system_password
  LOGFILE GROUP 1 ('/u01/logs/my/redo01a.log','/u02/logs/my/redo01b.log')
             SIZE 100M BLOCKSIZE 512,
          GROUP 2 ('/u01/logs/my/redo02a.log','/u02/logs/my/redo02b.log')
             SIZE 100M BLOCKSIZE 512,
          GROUP 3 ('/u01/logs/my/redo03a.log','/u02/logs/my/redo03b.log')
             SIZE 100M BLOCKSIZE 512
  MAXLOGHISTORY 1
```

```
MAXLOGFILES 16
MAXLOGMEMBERS 3
MAXDATAFILES 1024
CHARACTER SET AL32UTF8
NATIONAL CHARACTER SET AL16UTF16
EXTENT MANAGEMENT LOCAL
DATAFILE '/u01/app/oracle/oradata/newcdb/system01.dbf'
  SIZE 700M REUSE AUTOEXTEND ON NEXT 10240K MAXSIZE UNLIMITED
SYSAUX DATAFILE '/u01/app/oracle/oradata/newcdb/sysaux01.dbf'
  SIZE 550M REUSE AUTOEXTEND ON NEXT 10240K MAXSIZE UNLIMITED
DEFAULT TABLESPACE deftbs
   DATAFILE '/u01/app/oracle/oradata/newcdb/deftbs01.dbf'
   SIZE 500M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED
DEFAULT TEMPORARY TABLESPACE tempts1
   TEMPFILE '/u01/app/oracle/oradata/newcdb/temp01.dbf'
   SIZE 20M REUSE AUTOEXTEND ON NEXT 640K MAXSIZE UNLIMITED
UNDO TABLESPACE undotbs1
   DATAFILE '/u01/app/oracle/oradata/newcdb/undotbs01.dbf'
   SIZE 200M REUSE AUTOEXTEND ON NEXT 5120K MAXSIZE UNLIMITED
ENABLE PLUGGABLE DATABASE
  SEED
  FILE_NAME_CONVERT = ('/u01/app/oracle/oradata/newcdb/',
                       '/u01/app/oracle/oradata/pdbseed/')
  SYSTEM DATAFILES SIZE 125M AUTOEXTEND ON NEXT 10M MAXSIZE UNLIMITED
  SYSAUX DATAFILES SIZE 100M
USER_DATA TABLESPACE usertbs
  DATAFILE '/u01/app/oracle/oradata/pdbseed/usertbs01.dbf'
  SIZE 200M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED
LOCAL UNDO ON;
```

A CDB is created with the following characteristics:

- The CDB is named `newcdb`. Its global database name is `newcdb.us.example.com`, where the domain portion (`us.example.com`) is taken from the initialization parameter file. See *Oracle Database Administrator's Guide* for information about determining the global database name.
- Three control files are created as specified by the `CONTROL_FILES` initialization parameter, which was set before CDB creation in the initialization parameter file. See *Oracle Database Administrator's Guide* for a sample initialization parameter file and *Oracle Database Administrator's Guide* for information about specifying control files.
- The passwords for user accounts `SYS` and `SYSTEM` are set to the values that you specified. The passwords are case-sensitive. The two clauses that specify the passwords for `SYS` and `SYSTEM` are not mandatory in this release of Oracle Database. However, if you specify either clause, then you must specify both clauses. For further information about the use of these clauses, see *Oracle Database Administrator's Guide* for information about specifying passwords for users `SYS` and `SYSTEM`.
- The new CDB has three online redo log file groups, each with two members, as specified in the `LOGFILE` clause. `MAXLOGFILES`, `MAXLOGMEMBERS`, and `MAXLOGHISTORY` define limits for the redo log. See *Oracle Database Administrator's Guide* for information about choosing the number of redo log files. The block size for the redo logs is set to 512 bytes, the same size as physical sectors on disk. The

BLOCKSIZE clause is optional if block size is to be the same as physical sector size (the default). Typical sector size and thus typical block size is 512. Permissible values for BLOCKSIZE are 512, 1024, and 4096. For newer disks with a 4K sector size, optionally specify BLOCKSIZE as 4096. See *Oracle Database Administrator's Guide* for more information about planning the block size of redo log files.

- MAXDATAFILES specifies the maximum number of data files that can be open in the CDB. This number affects the initial sizing of the control file. For a CDB, set MAXDATAFILES to a high number that anticipates the aggregate number of data files for all containers, in addition to the CDB root files.

> **Note:**
>
> You can set several limits during CDB creation. Some of these limits are limited by and affected by operating system limits. For example, if you set MAXDATAFILES, then Oracle Database allocates enough space in the control file to store MAXDATAFILES file names, even if the CDB has only one data file initially. However, because the maximum control file size is limited and operating system dependent, you might not be able to set all CREATE DATABASE parameters at their theoretical maximums.
>
> For more information about setting limits during CDB creation, see the *Oracle Database SQL Language Reference* and your operating system–specific Oracle documentation.

- The AL32UTF8 character set is used to store data in this CDB.
- The AL16UTF16 character set is specified as the NATIONAL CHARACTER SET used to store data in columns specifically defined as NCHAR, NCLOB, or NVARCHAR2.
- The SYSTEM tablespace, consisting of the operating system file /u01/app/oracle/oradata/newcdb/system01.dbf, is created as specified by the DATAFILE clause. If a file with that name already exists, then it is overwritten.
- The SYSTEM tablespace is created as a locally managed tablespace. See *Oracle Database Administrator's Guide* for information about creating a locally managed SYSTEM tablespace.
- A SYSAUX tablespace is created, consisting of the operating system file /u01/app/oracle/oradata/newcdb/sysaux01.dbf as specified in the SYSAUX DATAFILE clause. See *Oracle Database Administrator's Guide* for information about the SYSAUX tablespace.
- The DEFAULT TABLESPACE clause creates and names a default permanent tablespace for this CDB.
- The DEFAULT TEMPORARY TABLESPACE clause creates and names a default temporary tablespace for the root of this CDB. See *Oracle Database Administrator's Guide* for information about creating a default temporary tablespace.
- The UNDO TABLESPACE clause creates and names an undo tablespace that is used to store undo data for this CDB. In a CDB, an undo tablespace is required to manage the undo data, and the UNDO_MANAGEMENT initialization parameter must be set to AUTO. If you omit this parameter, then it defaults to AUTO. See *Oracle Database Administrator's Guide* for information about creating an undo tablespace.

- Redo log files will not initially be archived, because the `ARCHIVELOG` clause is not specified in this `CREATE DATABASE` statement. This is customary during CDB creation. You can later use an `ALTER DATABASE` statement to switch to `ARCHIVELOG` mode. The initialization parameters in the initialization parameter file for `newcdb` relating to archiving are `LOG_ARCHIVE_DEST_1` and `LOG_ARCHIVE_FORMAT`. See *Oracle Database Administrator's Guide* for information about managing archived redo log files.
- The `ENABLE PLUGGABLE DATABASE` clause creates a CDB with the root and the PDB seed.
- `SEED` is required for the `FILE_NAME_CONVERT` clause and the *`tablespace_datafile`* clauses.
- The `FILE_NAME_CONVERT` clause generates file names for the PDB seed's files in the `/u01/app/oracle/oradata/pdbseed` directory using file names in the `/u01/app/oracle/oradata/newcdb` directory.
- The `SYSTEM DATAFILES` clause specifies attributes of the PDB seed `SYSTEM` tablespace data file(s) that differ from the root's.
- The `SYSAUX DATAFILES` clause specifies attributes of the PDB seed `SYSAUX` tablespace data file(s) that differ from the root's.
- The `USER_DATA TABLESPACE` clause creates and names the PDB seed's tablespace for storing user data and database options such as Oracle XML DB. PDBs created using the PDB seed include this tablespace and its data file. The tablespace and data file specified in this clause are not used by the root.
- The `LOCAL UNDO ON` clause sets the CDB undo mode to local, which means that each container in the CDB uses local undo.

  When the CDB is created in local undo mode, the PDB seed includes an undo tablespace so that any new PDB created from the PDB seed has an undo tablespace. When a PDB is created by plugging it in or cloning a remote PDB, and the source PDB was in shared undo mode, an undo tablespace is created for the PDB automatically the first time the PDB is opened.

Note:

- Ensure that all directories used in the `CREATE DATABASE` statement exist. The `CREATE DATABASE` statement does not create directories.
- If you are not using Oracle Managed Files, then every tablespace clause must include a `DATAFILE` or `TEMPFILE` clause.
- If CDB creation fails, then you can look at the alert log to determine the reason for the failure and to determine corrective actions. See *Oracle Database Administrator's Guide* for information about viewing the alert log. If you receive an error message that contains a process number, then examine the trace file for that process. Look for the trace file that contains the process number in the trace file name. See *Oracle Database Administrator's Guide* for more information.
- To resubmit the `CREATE DATABASE` statement after a failure, you must first shut down the instance and delete any files created by the previous `CREATE DATABASE` statement.

Creating a CDB Using Oracle Managed Files: Example

This example illustrates creating a CDB with Oracle Managed Files, which enables you to use a much simpler `CREATE DATABASE` statement.

To use Oracle Managed Files, the initialization parameter `DB_CREATE_FILE_DEST` must be set. This parameter defines the base directory for the various CDB files that the CDB creates and automatically names.

The following statement is an example of setting this parameter in the initialization parameter file:

```
DB_CREATE_FILE_DEST='/u01/app/oracle/oradata'
```

This example sets the parameter Oracle ASM storage:

```
DB_CREATE_FILE_DEST = +data
```

This example includes the `ENABLE PLUGGABLE DATABASE` clause to create a CDB with the root and the PDB seed. This example does not include the `SEED FILE_NAME_CONVERT` clause because Oracle Managed Files determines the names and locations of the PDB seed's files. However, this example does include *`tablespace_datafile`* clauses that specify attributes of the PDB seed data files for the `SYSTEM` and `SYSAUX` tablespaces that differ from the root data files.

With Oracle Managed Files and the following `CREATE DATABASE` statement, the CDB creates the `SYSTEM` and `SYSAUX` tablespaces, creates the additional tablespaces specified in the statement, and chooses default sizes and properties for all data files, control files, and redo log files. Note that these properties and the other default CDB properties set by this method might not be suitable for your production environment, so it is recommended that you examine the resulting configuration and modify it if necessary.

```
CREATE DATABASE newcdb
USER SYS IDENTIFIED BY sys_password
USER SYSTEM IDENTIFIED BY system_password
EXTENT MANAGEMENT LOCAL
DEFAULT TABLESPACE users
DEFAULT TEMPORARY TABLESPACE temp
UNDO TABLESPACE undotbs1
ENABLE PLUGGABLE DATABASE
   SEED
   SYSTEM DATAFILES SIZE 125M AUTOEXTEND ON NEXT 10M MAXSIZE UNLIMITED
   SYSAUX DATAFILES SIZE 100M;
```

A CDB is created with the following characteristics:

- The CDB is named `newcdb`. Its global database name is `newcdb.us.example.com`, where the domain portion (`us.example.com`) is taken from the initialization parameter file. See *Oracle Database Administrator's Guide* for information about determining the global database name.

- The passwords for user accounts SYS and SYSTEM are set to the values that you specified. The passwords are case-sensitive. The two clauses that specify the passwords for SYS and SYSTEM are not mandatory in this release of Oracle Database. However, if you specify either clause, then you must specify both clauses. For further information about the use of these clauses, see *Oracle Database Administrator's Guide* for information about specifying passwords for users SYS and SYSTEM.
- The DEFAULT TABLESPACE clause creates and names a default permanent tablespace for this CDB.
- The DEFAULT TEMPORARY TABLESPACE clause creates and names a default temporary tablespace for the root of this CDB. See *Oracle Database Administrator's Guide* for information about creating a default temporary tablespace.
- The UNDO TABLESPACE clause creates and names an undo tablespace that is used to store undo data for this CDB. In a CDB, an undo tablespace is required to manage the undo data, and the UNDO_MANAGEMENT initialization parameter must be set to AUTO. If you omit this parameter, then it defaults to AUTO. See *Oracle Database Administrator's Guide* for information about creating an undo tablespace.
- Redo log files will not initially be archived, because the ARCHIVELOG clause is not specified in this CREATE DATABASE statement. This is customary during CDB creation. You can later use an ALTER DATABASE statement to switch to ARCHIVELOG mode. The initialization parameters in the initialization parameter file for newcdb relating to archiving are LOG_ARCHIVE_DEST_1 and LOG_ARCHIVE_FORMAT. See *Oracle Database Administrator's Guide* for information about managing archived redo log files.
- The ENABLE PLUGGABLE DATABASE clause creates a CDB with the root and the PDB seed.
- SEED is required for the *tablespace_datafile* clauses.
- The SYSTEM DATAFILES clause specifies attributes of the PDB seed's SYSTEM tablespace data file(s) that differ from the root's.
- The SYSAUX DATAFILES clause specifies attributes of the PDB seed's SYSAUX tablespace data file(s) that differ from the root's.

Configuring EM Express for a CDB

For a CDB, you can configure Oracle Enterprise Manager Database Express (EM Express) for the root and for each PDB by setting a global HTTPS port, or you can set a different port for every container in the CDB.

You can set a global port, which enables you to use EM Express to connect to all PDBs in the CDB using the HTTPS port for the CDB. Alternatively, you can set a different HTTPS port for every container in a CDB.

To configure EM Express for a CDB:

1. In SQL*Plus, access a container in a CDB.

   The user must have common SYSDBA administrative privilege, and you must exercise this privilege using AS SYSDBA at connect time. The container can be the root or a PDB.

See "About Container Access in a CDB".

2. Set the port in one of the following ways:
   - To set the global port, connect to the `CDB$ROOT`, and issue the following SQL statement to configure the global port for the CDB:

     ```
     EXEC DBMS_XDB_CONFIG.SETGLOBALPORTENABLED(TRUE);
     ```

   - To set the HTTPS port for the current container, run the following procedure:

     ```
     exec DBMS_XDB_CONFIG.SETHTTPSPORT(https_port_number);
     ```

     Replace *`https_port_number`* with the appropriate HTTPS port number.
3. Access EM Express in one of the following ways:
   - To use the global port, enter the EM Express URL provided by Database Configuration Assistant (DBCA) when it configured the CDB that includes the PDB. When the EM Express login screen appears, specify your administrator credentials, and enter the name of the PDB that you want to connect to in the **Container Name** field.
   - The URL for the HTTPS port for a container:

     ```
     https://database_hostname:https_port_number/em/
     ```

     Replace *`database_hostname`* with the host name of the computer on which the database instance is running, and replace *`https_port_number`* with the appropriate HTTPS port number.

When connected to the root, EM Express displays data and enables actions that apply to the entire CDB. When connected to a PDB, EM Express displays data and enables actions that apply to the PDB only.

> **Note:**
>
> If the listener is not configured on port 1521, then you must manually configure the port for EM Express. See *Oracle Database 2 Day DBA* for instructions.

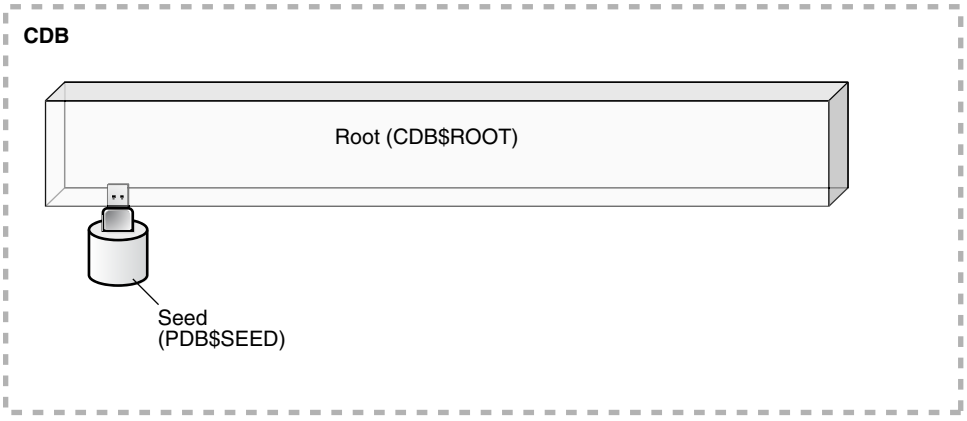
> **See Also:**
>
> *Oracle Database 2 Day DBA* for more information about EM Express

After Creating a CDB

After creation, a CDB consists of the root and the PDB seed.

The root contains system-supplied metadata and common users that can administer the PDBs. The PDB seed is a template that you can use to create new PDBs. The following graphic shows a newly created CDB.

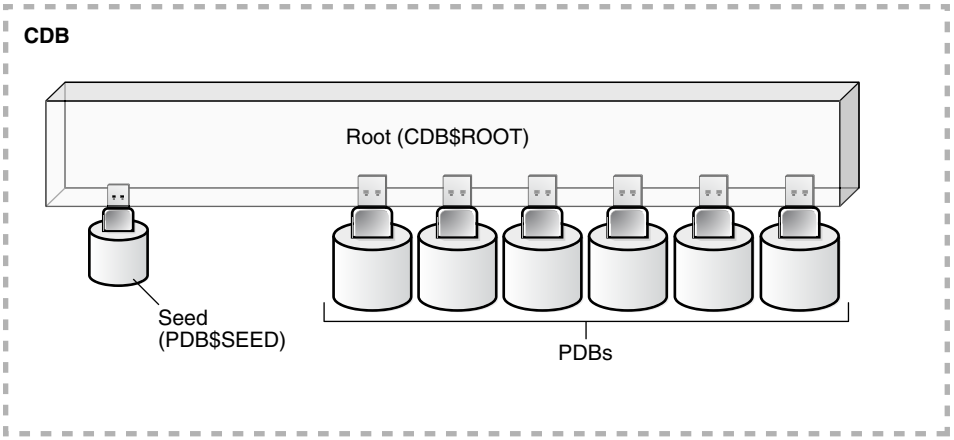
Figure 4-1 A Newly Created CDB



In a CDB, the root contains minimal user data or no user data. User data resides in the PDBs. Therefore, after creating a CDB, one of the first tasks is to add the PDBs that will contain the user data.

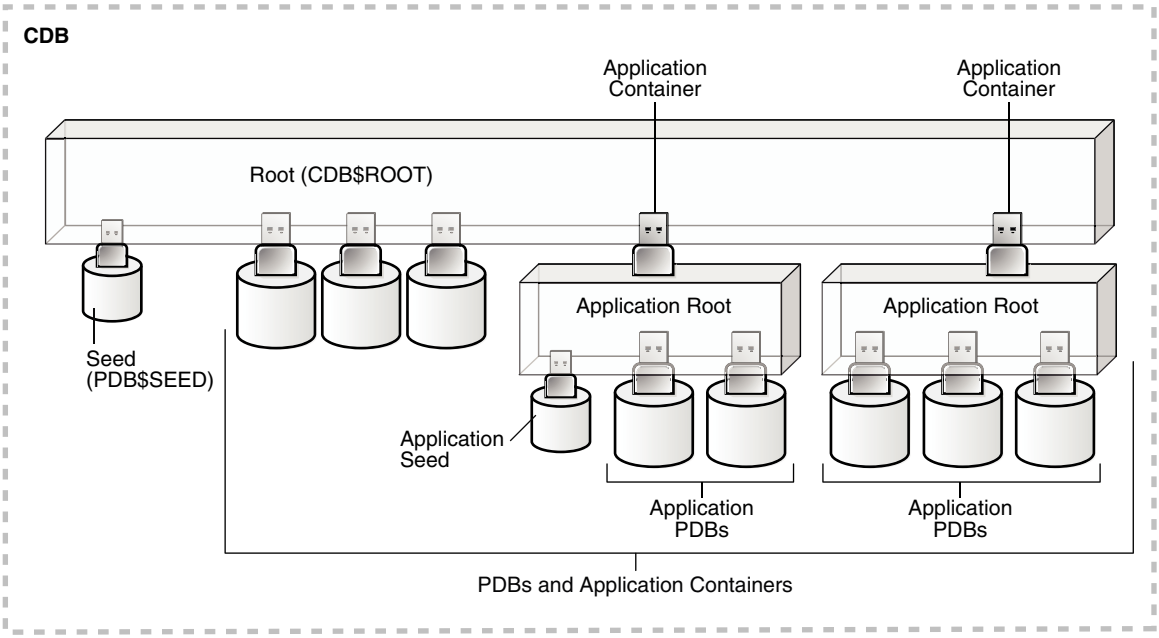
The following graphic shows a CDB with PDBs.

Figure 4-2 CDB with PDBs



You have the option of creating one or more application containers. An application container consists of an application root and application PDBs, and it stores data for one or more applications. An application container can store application common objects, which contain user data that can be shared by the application PDBs in the application container. It can also contain an application seed for fast creation of application PDBs in an application container.

Figure 4-3 Application Containers in a CDB



When you have added the PDBs and application containers to the CDB, the physical structure of a CDB is very similar to the physical structure of a non-CDB. A CDB contains the following files:

- One control file
- One active online redo log for a single-instance CDB, or one active online redo log for each instance of an Oracle RAC CDB
- Sets of temp files

  There is one default temporary tablespace for the root of the CDB and one for each PDB, application root, and application PDB.
- Sets of system data files

  The primary physical difference between a CDB and a non-CDB is in the data files. A non-CDB has only one set of system data files. In contrast, a CDB includes one set of system data files for each container in the CDB, including a set of system data files for each PDB, application root, and application PDB. In addition, a CDB has one set of user-created data files for each container. If the CDB is in local undo mode, then each container also has its own undo tablespace and associated data files.
- Sets of user-created data files

  Each PDB has its own set of non-system data files. These data files contain the user-defined schemas and database objects for the PDB.

For backup and recovery of a CDB, Recovery Manager (RMAN) is recommended. PDB point-in-time recovery (PDB PITR) must be performed with RMAN. By default, RMAN turns on control file autobackup for a CDB. It is strongly recommended that control file autobackup is enabled for a CDB, to ensure that PDB PITR can undo data file additions or deletions.

Video

Because the physical structure of a CDB and a non-CDB are similar, most management tasks are the same for a CDB and a non-CDB. However, some administrative tasks are specific to CDBs. The following chapters describe these tasks:

- "Creating and Removing PDBs and Application Containers"

  This chapter documents the following tasks:

  - Creating a PDB using the PDB seed
  - Creating a PDB by cloning an existing PDB or Non-CDB
  - Creating a PDB by relocating it
  - Creating a PDB as a proxy PDB
  - Plugging in a PDB
  - Unplugging a PDB
  - Dropping a PDB
- "Creating and Removing Application Containers and Seeds"

  This chapter documents the following tasks:

  - Creating application containers
  - Creating application seeds
  - Unplugging application containers
  - Unplugging application seeds
  - Dropping application containers
  - Dropping application seeds
- " Administering a CDB"

  This chapter documents the following tasks:

  - Connecting to a container
  - Switching into a container
  - Modifying a CDB
  - Modifying the root
  - Changing the open mode of a PDB
  - Executing DDL statements in a CDB
  - Shutting down the CDB instance
- " Administering PDBs"

  This chapter documents the following tasks:

  - Connecting to a PDB
  - Modifying a PDB
  - Managing services associated with PDBs
- "Administering Application Containers"

  This chapter documents the following tasks:

  - Managing applications in an application container, including installing, upgrading, and patching applications
  - Managing application common objects
  - Issuing DML statements on containers in an application container
  - Partitioning by PDB with container maps
- "Monitoring CDBs and PDBs"

  This chapter documents the following tasks:
  - Querying views for monitoring a CDB and its PDBs
  - Running sample queries that provide information about a CDB and its PDBs
- "Using Oracle Resource Manager for PDBs"

  This chapter documents the following tasks:
  - Creating resource plans in a CDB
  - Managing resource plans in a CDB
- " Using Oracle Scheduler with a CDB"

  This chapter documents the following topics:
  - `DBMS_SCHEDULER` invocations in a CDB
  - Job coordinator and slave processes in a CDB
  - Using `DBMS_JOB`
  - Processes to close a PDB
  - New and changed views

See Also:

- *Oracle Database Concepts* for a multitenant architecture documentation roadmap
- *Oracle Database Backup and Recovery User's Guide* for information about RMAN

Part III

Creating and Removing PDBs and Application Containers

You can create PDBs, application containers, and application seeds using a variety of techniques.

For example, you can create a PDB from scratch, cloning an existing PDB or non-CDB, or plug in an unplugged PDB. You can also remove PDBs from a CDB.

> **Note:**
>
> You can complete the tasks in this part using SQL*Plus or Oracle SQL Developer.

Related Topics

- Tools for a Multitenant Environment
  You can use various tools to configure and administer a multitenant environment.

5

Overview of PDB Creation

A CDB supports multiple techniques for creating PDBs.

The created PDB automatically includes a full data dictionary including metadata and internal links to system-supplied objects in the CDB root. You must define every PDB from a single root: either the CDB root or an application root.

Each PDB has a globally unique identifier (GUID). The PDB GUID is primarily used to generate names for directories that store the PDB's files, including both Oracle Managed Files directories and non-Oracle Managed Files directories.

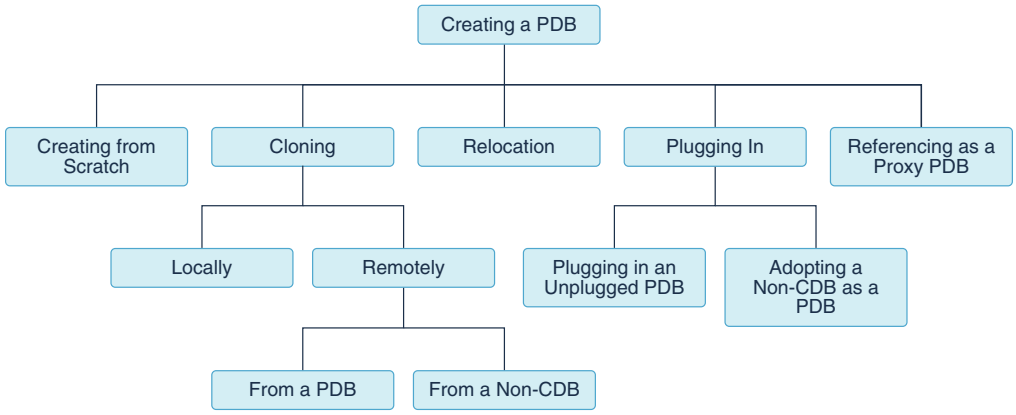
Techniques for Creating a PDB

You can create a PDB with various techniques, all of which require the `CREATE PLUGGABLE DATABASE` statement.

Creating a PDB is the process of associating it with a CDB or an application container.

The following graphic depicts the options for creating a PDB.

Figure 5-1 Options for Creating a PDB



The following table describes the creation techniques. An additional technique, which is not covered in this manual, is to use the `DUPLICATE` command in Recovery Manager to copy a PDB from one CDB to another CDB.

Table 5-1 Techniques for Creating a PDB

Technique	Description	More Information
Create a PDB from scratch	Create a PDB in a CDB using the files of the PDB seed or application seed. This technique copies the files associated with the seed to a new location and associates the copied files with the new PDB. This is the default creation mechanism. The other techniques require either a source database (PDB or non-CDB) or XML.	"Creating a PDB from Scratch"
Clone an existing PDB or non-CDB	Create a PDB by cloning a source PDB or non-CDB. A source can be a PDB in the local CDB, a PDB in a remote CDB, a PDB in a local or remote application container, or a non-CDB. This technique copies the files associated with the source to a new location and associates the copied files with the new PDB.	"Cloning a PDB or Non-CDB"
Relocate a PDB to a different CDB	Create a PDB by relocating it from one CDB to another. This technique moves the files associated with the PDB to a new location.	"Relocating a PDB"
Plug an unplugged PDB into a CDB	Create a PDB by using the XML metadata file that describes the PDB and the files associated with the PDB to plug it into the CDB.	"Plugging In an Unplugged PDB"
Reference a PDB as a proxy PDB	Create a PDB as a proxy PDB by referencing a different PDB with a database link. The referenced PDB can be in the same CDB as the proxy PDB, or it can be in a different CDB.	"Creating a PDB as a Proxy PDB"
Create a PDB from a non-CDB, and then plug the PDB into a CDB	Create a PDB by adopting a non-CDB into a PDB. You can use the `DBMS_PDB` package to create an unplugged PDB from an Oracle Database 12c non-CDB. You can then plug the unplugged PDB into the CDB.	"Options for Creating a PDB from a Non-CDB"

You can unplug a PDB when you want to plug it into a different CDB. You can unplug or drop a PDB when you no longer need it. An unplugged PDB is not usable until it is plugged into a CDB.

See Also:

- "Creating and Removing Application Containers"
- "Unplugging a PDB from a CDB"
- "Dropping a PDB"
- *Oracle Database Backup and Recovery User's Guide* to learn how to copy a PDB using the `DUPLICATE` command
- *Oracle Database SQL Language Reference* for more information about the `CREATE PLUGGABLE DATABASE` statement

Current Container and PDB Creation

You can use the `CREATE PLUGGABLE DATABASE` statement to create PDBs, application containers, application seeds, and application PDBs.

When you create a PDB, the current container—CDB root or application root—determines the association of the PDB. The SQL statements that create PDBs and application PDBs are the same. For example, when you run `CREATE PLUGGABLE DATABASE` statement in the CDB root, the PDB belongs to the CDB root. When you run `CREATE PLUGGABLE DATABASE` statement in an application root, the application PDB belongs to the application root.

When the CDB root is the current container, create an application root by running a `CREATE PLUGGABLE DATABASE` statement with the `AS APPLICATION CONTAINER` clause. When cloning, relocating, or plugging in a PDB to an application container, the application name and version of the PDB must match the application name and version of the application container.

Options for Creating a PDB from a Non-CDB

You have multiple options for moving a non-CDB into a PDB.

You can accomplish this task in the following ways:

- Clone a non-CDB

  This is the simplest way to create a PDB using a non-CDB, but it requires copying the files of the non-CDB to a new location.

  Both the CDB and the non-CDB must be running Oracle Database 12c Release 1 (12.1.0.2) or later. If your current non-CDB uses an Oracle Database release before Oracle Database 12c Release 1 (12.1.0.2), then you must upgrade the non-CDB to a later release to use this technique.

- Generate an XML metadata file by using the `DBMS_PDB` package

  The XML metadata file describes the database files of the non-CDB so that you can plug it into a CDB.

  This method requires more steps than creating a PDB by cloning a non-CDB, but it enables you to create a PDB using a non-CDB without moving the non-CDB files in some situations.

To use this technique, the non-CDB must run Oracle Database 12c or later. If your current non-CDB uses a release before Oracle Database 12c, then you must upgrade to a later release.

- Export the data from the non-CDB and import it into a PDB using Oracle Data Pump

  When you import, specify the connect identifier for the PDB after the user name. For example, if the connect identifier for the PDB is `hrpdb`, then enter the following when you run the Oracle Data Pump Import utility:

  ```
  impdp user_name@hrpdb ...
  ```

  If the Oracle Database release of the non-CDB is Oracle Database 11g Release 2 (11.2.0.3) or later, then you can use full transportable export/import to move the data. When transporting a non-CDB from an Oracle Database 11g Release 2 (11.2.0.3) or later Oracle Database 11g database to Oracle Database 12c or later, the `VERSION` Data Pump export parameter must be set to `12.0.0.0.0` or higher.

  If the Oracle Database release of the non-CDB is before Oracle Database 11g Release 2 (11.2.0.3), then you can use transportable tablespaces to move the data, or you can perform a full database export/import.

- Replicate data from the non-CDB to a PDB using GoldenGate

  When the PDB catches up with the non-CDB, you fail over to the PDB.

  See the Oracle GoldenGate documentation.

See Also:

- "Cloning a PDB or Non-CDB"
- "Adopting a Non-CDB as a PDB"
- *Oracle Database Upgrade Guide* for information about upgrading
- *Oracle Database Administrator's Guide* for information about transporting data

PDB Storage

However you choose to create a PDB, you must decide on the tablespaces and files that will store the data.

Storage Limits

The optional `STORAGE` clause of the `CREATE PLUGGABLE DATABASE` statement specifies storage limits for PDBs.

The `STORAGE` clause specifies the following limits:

- The amount of storage that can be used by all tablespaces that belong to the PDB

  Use `MAXSIZE` and a size clause to specify a limit, or set `MAXSIZE` to `UNLIMITED` to indicate no limit.

- The amount of storage that can be used by unified audit OS spillover (.bin format) files in the PDB

  Use `MAX_AUDIT_SIZE` and a size clause to specify a limit, or set `MAX_AUDIT_SIZE` to `UNLIMITED` to indicate no limit.

- The amount of diagnostics (trace files and incident dumps) in the Automatic Diagnostic Repository (ADR) that can be used by the PDB

  Use `MAX_DIAG_SIZE` and a size clause to specify a limit, or set `MAX_DIAG_SIZE` to `UNLIMITED` to indicate no limit.

If `STORAGE UNLIMITED` is set, or if there is no `STORAGE` clause, then there are no storage limits for the PDB.

The following are examples that use the `STORAGE` clause.

Example 5-1 STORAGE Clause That Specifies a Storage Limit

This `STORAGE` clause specifies that the storage used by all tablespaces that belong to the PDB must not exceed 2 gigabytes.

```
STORAGE (MAXSIZE 2G)
```

Example 5-2 STORAGE Clause That Specifies Unlimited Storage

This `STORAGE` clause specifies unlimited storage for all tablespaces that belong to the PDB.

```
STORAGE (MAXSIZE UNLIMITED)
```

> **See Also:**
>
> *Oracle Database SQL Language Reference* for the syntax of the `STORAGE` clause

Default Tablespace

The `DEFAULT TABLESPACE` clause of the `CREATE PLUGGABLE DATABASE` statement specifies the default tablespace for the new PDB.

Oracle Database will assign the default tablespace to any non-`SYSTEM` users who do not have a different permanent tablespace specified.

When you create the PDB from the PDB seed or an application seed and specify the `DEFAULT TABLESPACE` clause, Oracle Database creates a smallfile tablespace and sets it as the default tablespace for the PDB. When you create the PDB using a method other than the using the PDB seed or application seed, such as cloning a PDB or plugging in an unplugged PDB, the default tablespace must be a tablespace that already exists in the source PDB.

Example 5-3 DEFAULT TABLESPACE Clause

```
DEFAULT TABLESPACE sales
```

User Tablespaces

The USER_TABLESPACES clause of the CREATE PLUGGABLE DATABASE statement specifies which tablespaces are available in the new PDB.

You can use this clause to separate the data for multiple schemas into different PDBs. For example, when you move a non-CDB to a PDB, and the non-CDB had several schemas that each supported a different application, you can use this clause to separate the data belonging to each schema into a separate PDB. This technique assumes that each schema used a separate tablespace in the non-CDB.

You can use this clause to specify one of the following options:

- List one or more tablespaces to include.
- Specify ALL, the default, to include all tablespaces.
- Specify ALL EXCEPT to include all tablespaces, except for the tablespaces listed.
- Specify NONE to exclude all tablespaces.
- If the creation mode of the user tablespaces must be different from the creation mode for the Oracle-supplied tablespaces (such as SYSTEM and SYSAUX), then specify one of the following in the USER_TABLESPACES clause:
  - COPY: The files of the tablespaces are copied to a new location.
  - MOVE: The files of the tablespaces are moved to a new location.
  - NOCOPY: The files of the tablespaces are not copied or moved.
  - SNAPSHOT COPY: The tablespaces are cloned with storage snapshots.
  - NO DATA: The data model definition of the tablespaces is cloned but not the tablespaces' data.

When the compatibility level of the CDB is 12.2.0 or higher, the tablespaces that are excluded by this clause are created offline in the new PDB, and they have no data files associated with them. When the compatibility level of the CDB is lower than 12.2.0, the tablespaces that are excluded by this clause are offline in the new PDB, and all data files that belong to these tablespaces are unnamed and offline.

This clause does not apply to the SYSTEM, SYSAUX, or TEMP tablespaces. Do not include these tablespaces in a tablespace list for this clause.

The following are examples that use the USER_TABLESPACES clause.

Example 5-4 USER_TABLESPACES Clause That Includes One Tablespace

Assume that the non-CDB or PDB from which a PDB is being created includes the following tablespaces: tbs1, tbs2, and tbs3. This USER_TABLESPACES clause includes the tbs2 tablespace, but excludes the tbs1 and tbs3 tablespaces.

```
USER_TABLESPACES=('tbs2')
```

Example 5-5 USER_TABLESPACES Clause That Includes a List of Tablespaces

Assume that the non-CDB or PDB from which a PDB is being created includes the following tablespaces: tbs1, tbs2, tbs3, tbs4, and tbs5. This USER_TABLESPACES

clause includes the tbs1, tbs4, and tbs5 tablespaces, but excludes the tbs2 and tbs3 tablespaces.

```
USER_TABLESPACES=('tbs1','tbs4','tbs5')
```

Example 5-6 USER_TABLESPACES Clause That Includes All Tablespaces Except for Listed Ones

Assume that the non-CDB or PDB from which a PDB is being created includes the following tablespaces: tbs1, tbs2, tbs3, tbs4, and tbs5. This USER_TABLESPACES clause includes the tbs2 and tbs3 tablespaces, but excludes the tbs1, tbs4, and tbs5 tablespaces.

```
USER_TABLESPACES=ALL EXCEPT('tbs1','tbs4','tbs5')
```

Example 5-7 USER_TABLESPACES in a Different Creation Mode

This example shows a full CREATE PLUGGABLE DATABASE statement that plugs in a non-CDB and only includes the tbs3 user tablespace from the non-CDB. The example copies the files for Oracle-supplied tablespaces (such as SYSTEM and SYSAUX) to a new location, but moves the files of the tbs3 user tablespace.

```
CREATE PLUGGABLE DATABASE ncdb USING '/disk1/oracle/ncdb.xml'
  COPY
  FILE_NAME_CONVERT = ('/disk1/oracle/dbs/', '/disk2/oracle/ncdb/')
  USER_TABLESPACES=('tbs3') MOVE;
```

PDB File Locations

In the CREATE PLUGGABLE DATABASE statement, you can specify the locations of files used by the new PDB.

The term "file name" means both the name and the location of a file. The CREATE PLUGGABLE DATABASE statement has the following clauses that indicate the file names of the new PDB being created:

- The FILE_NAME_CONVERT clause specifies the names of the PDB's files after the PDB is created.

  Use this clause when the files are not yet at their ultimate destination, and you want to copy or move them during PDB creation. You can use this clause in any CREATE PLUGGABLE DATABASE statement.

- The CREATE_FILE_DEST clause specifies the default Oracle Managed Files file system directory or Oracle ASM disk group for the PDB's files.

  Use this clause to enable Oracle Managed Files for the new PDB, independent of any Oracle Managed Files default location specified in the root for the CDB. You can use this clause in any CREATE PLUGGABLE DATABASE statement.

When necessary, you can use both clauses in the same CREATE PLUGGABLE DATABASE statement. In addition, the following initialization parameters can control the location of the new PDB files:

- The DB_CREATE_FILE_DEST initialization parameter set in the root

This initialization parameter specifies the default location for Oracle Managed Files for the CDB. When this parameter is set in a PDB, it specifies the default location for Oracle Managed Files for the PDB.

- The `PDB_FILE_NAME_CONVERT` initialization parameter

  This initialization parameter maps names of existing files to new file names when processing a `CREATE PLUGGABLE DATABASE` statement.

The following table shows the precedence order when both clauses are used in the same `CREATE PLUGGABLE DATABASE` statement, and both initialization parameters are set. For each clause and initialization parameter, the table also shows whether the files created by the `CREATE PLUGGABLE DATABASE` statement will use Oracle Managed Files or not.

Table 5-2 Summary of File Location Clauses and Initialization Parameters

Clause or Initialization Parameter	Precedence Order	Will the Files Created by CREATE PLUGGABLE DATABASE Use Oracle Managed Files?
`FILE_NAME_CONVERT` clause	1	No
`CREATE_FILE_DEST` clause	2	Yes
`DB_CREATE_FILE_DEST` initialization parameter	3	Yes
`PDB_FILE_NAME_CONVERT` initialization parameter	4	No

Regarding the use of Oracle Managed Files, the table only applies to files created by the `CREATE PLUGGABLE DATABASE` statement. Files created for the PDB after the PDB has been created might or might not use Oracle Managed Files.

In addition, if `FILE_NAME_CONVERT` and `CREATE_FILE_DEST` are both specified in the `CREATE PLUGGABLE DATABASE` statement, then the `FILE_NAME_CONVERT` setting is used for the files being placed during PDB creation, and the `CREATE_FILE_DEST` setting is used to set the `DB_CREATE_FILE_DEST` initialization parameter in the PDB. In this case, Oracle Managed Files controls the location of the files for the PDB after PDB creation.

Note:

The `PATH_PREFIX` clause does not affect files created by Oracle Managed Files.

See Also:

Oracle Database Reference to learn more about `DB_CREATE_FILE_DEST` and `PDB_FILE_NAME_CONVERT`

FILE_NAME_CONVERT Clause

If the PDB will not use Oracle Managed Files, then the `FILE_NAME_CONVERT` clause of the `CREATE PLUGGABLE DATABASE` statement specifies how to generate the names of files (such as data files) using the names of existing files.

You can use this clause to specify one of the following options:

- One or more file name patterns and replacement file name patterns, in the following form:

  ```
  'string1' , 'string2' , 'string3' , 'string4' , ...
  ```

  The *string2* file name pattern replaces the *string1* file name pattern, and the *string4* file name pattern replaces the *string3* file name pattern. You can use as many pairs of file name pattern and replacement file name pattern strings as required.

  If you specify an odd number of strings (the last string has no corresponding replacement string), then an error is returned. Do not specify more than one pattern/replace string that matches a single file name or directory.

- `NONE` when no files should be copied or moved during PDB creation. Omitting the `FILE_NAME_CONVERT` clause is the same as specifying `NONE`.

You can use the `FILE_NAME_CONVERT` clause in any `CREATE PLUGGABLE DATABASE` statement.

When the `FILE_NAME_CONVERT` clause is not specified in a `CREATE PLUGGABLE DATABASE` statement, either Oracle Managed Files or the `PDB_FILE_NAME_CONVERT` initialization parameter specifies how to generate the names of the files. If you use both Oracle Managed Files and the `PDB_FILE_NAME_CONVERT` initialization parameter, then Oracle Managed Files takes precedence. The `FILE_NAME_CONVERT` clause takes precedence when it is specified.

File name patterns specified in the `FILE_NAME_CONVERT` clause cannot match files or directories managed by Oracle Managed Files.

Example 5-8 FILE_NAME_CONVERT Clause

This `FILE_NAME_CONVERT` clause generates file names for the new PDB in the `/oracle/pdb5` directory using file names in the `/oracle/dbs` directory.

```
FILE_NAME_CONVERT = ('/oracle/dbs/', '/oracle/pdb5/')
```

> **See Also:**
>
> - "Example 19-7"
> - *Oracle Database Administrator's Guide*
> - *Oracle Database SQL Language Reference* for the syntax of the `FILE_NAME_CONVERT` clause
> - *Oracle Database Reference* for information about the `PDB_FILE_NAME_CONVERT` initialization parameter

CREATE_FILE_DEST Clause

The `CREATE_FILE_DEST` clause of the `CREATE PLUGGABLE DATABASE` statement enables Oracle Managed Files for the PDB and specifies the default file system directory or Oracle ASM disk group for the PDB files.

The PDB data files and temp files are restricted to the specified directory and its subdirectories. If a file system directory is specified as the default location in this clause, then the directory must exist. Also, the user who runs the `CREATE PLUGGABLE DATABASE` statement must have the appropriate privileges to create files in the specified directory. Alternatively, you can specify the name of a directory object that exists in the CDB root (`CDB$ROOT`). The directory object points to the file system directory used by `CREATE_FILE_DEST`.

If there is a default Oracle Managed Files location for the CDB set in the CDB root, then the `CREATE_FILE_DEST` setting overrides the CDB root's setting, and the specified `CREATE_FILE_DEST` setting is used for the PDB.

If `CREATE_FILE_DEST=NONE` is specified, then Oracle Managed Files is disabled for the PDB.

When the `CREATE_FILE_DEST` clause is set to a value other than `NONE`, the `DB_CREATE_FILE_DEST` initialization parameter is set implicitly in the PDB with `SCOPE=SPFILE`.

If the CDB root uses Oracle Managed Files, and this clause is not specified, then the PDB inherits the Oracle Managed Files default location from the CDB root.

> **Note:**
>
> This feature is available starting with Oracle Database 12c Release 1 (12.1.0.2).

Example 5-9 CREATE_FILE_DEST Clause

This `CREATE_FILE_DEST` clause specifies /oracle/pdb2/ as the default Oracle Managed Files file system directory for the new PDB.

```
CREATE_FILE_DEST = '/oracle/pdb2/'
```

> **See Also:**
>
> *Oracle Database Administrator's Guide*

Restrictions on PDB File Locations

The `PATH_PREFIX` clause of the `CREATE PLUGGABLE DATABASE` statement ensures that all directory object paths associated with the PDB are restricted to the specified directory or its subdirectories.

This clause also ensures that the following files associated with the PDB are restricted to the specified directory: the Oracle XML repository for the PDB, files created with a `CREATE PFILE` statement, and the export directory for Oracle wallets. Use this clause when you want to ensure that a PDB's files reside in a specific directory and its subdirectories.

You can use this clause to specify one of the following options:

- An absolute path that is used as a prefix for all file paths associated with the PDB.
- The name of a directory object that exists in the CDB root (`CDB$ROOT`). The directory object points to the absolute path to be used for `PATH_PREFIX`.
- `NONE` to indicate that there are no restrictions for the file paths. Omitting the `PATH_PREFIX` clause is the same as specifying `NONE`.

After a PDB is created, its `PATH_PREFIX` setting cannot be modified.

You can use the `PATH_PREFIX` clause in any `CREATE PLUGGABLE DATABASE` statement.

Example 5-10 PATH_PREFIX Clause

This `PATH_PREFIX` clause ensures that all file paths associated with the PDB are restricted to the `/disk1/oracle/dbs/salespdb/` directory.

```
PATH_PREFIX = '/disk1/oracle/dbs/salespdb/'
```

Be sure to specify the path name so that it is properly formed when file names are appended to it. For example, on UNIX systems, be sure to end the path name with a forward slash (/).

Note:

- After the PATH_PREFIX clause is specified for a PDB, existing directory objects might not work as expected, since the PATH_PREFIX string is always added as a prefix to all local directory objects in the PDB.
- The PATH_PREFIX clause does not affect files created by Oracle Managed Files.
- The PATH_PREFIX clause only applies to user-created directory objects. It does not apply to Oracle-supplied directory objects.
- The PATH_PREFIX clause does not apply to data files or temporary files. If you are using Oracle Managed Files, then use the CREATE_FILE_DEST clause to restrict the locations of data files and temporary files.

See Also:

- "About a Multitenant Environment"
- "Viewing Information About the Containers in a CDB"

Service Name Conversion

An important aspect of PDB creation is managing the renaming of database services.

When the service name of a new PDB conflicts with an existing service name in the CDB, plug-in violations can result. The SERVICE_NAME_CONVERT clause of the CREATE PLUGGABLE DATABASE statement renames the user-defined services of the new PDB based on the service names of the source PDB. Using this clause, you can rename services and avoid plug-in violations.

You can use this clause to specify one of the following options:

- One or more service names and replacement service names, in the following form:

  ```
  'string1' , 'string2' , 'string3' , 'string4' , ...
  ```

  The *string2* service name replaces the *string1* service name, and the *string4* service name replaces the *string3* service name. You can use as many pairs of service names and replacement service names as required.

  If you specify an odd number of strings (the last string has no corresponding replacement string), then an error is returned.

- NONE when no service names need to be renamed. Omitting the SERVICE_NAME_CONVERT clause is the same as specifying NONE.

You can use the SERVICE_NAME_CONVERT clause in any CREATE PLUGGABLE DATABASE statement, except for a CREATE PLUGGABLE DATABASE statement that creates a PDB from the PDB seed. The PDB seed cannot have user-defined services. However, you

can use this statement for a `CREATE PLUGGABLE DATABASE` statement that creates an application PDB from an application seed in an application container.

> **Note:**
>
> This clause does not apply to the default service for the PDB. The default service has the same name as the PDB.

Example 5-11 SERVICE_NAME_CONVERT Clause

This `SERVICE_NAME_CONVERT` clause uses renames the `salesrep` service to `salesperson`.

```
SERVICE_NAME_CONVERT = ('salesrep','salesperson')
```

> **See Also:**
>
> *Oracle Database SQL Language Reference*

Summary of Clauses for Creating a PDB

When you create a PDB with the `CREATE PLUGGABLE DATABASE` statement, various clauses are available based on different factors.

One factor is the technique you are using to create the PDB. You can determine which clauses to use by answering a series of questions.

The following table describes which `CREATE PLUGGABLE DATABASE` clauses to specify based on different factors.

Table 5-3 Clauses for Creating a PDB

Question	Yes	No	Clause Can Be Used Only When
Do you want to create an application container instead of a PDB?	Specify the `AS APPLICATION CONTAINER` clause.	Omit the `AS APPLICATION CONTAINER` clause.	Creating an application container in a CDB
Are you plugging a PDB into a CDB that contains one or more PDBs that were created by plugging in the same PDB?	Specify the `AS CLONE` clause to ensure that Oracle Database generates a unique PDB DBID, GUID, and other identifiers expected for the new PDB. The PDB is plugged in as a clone of the unplugged PDB to ensure that all of its identifiers are unique.	Omit the `AS CLONE` clause.	Plugging in an unplugged PDB

Table 5-3 (Cont.) Clauses for Creating a PDB

Question	Yes	No	Clause Can Be Used Only When
Do you want to create an application seed in an application container?	Specify the AS SEED clause.	Omit the AS SEED clause.	Creating an application seed in an application container
Do you want to use a CREATE_FILE_DEST clause to specify the Oracle Managed Files default location for the PDB files? When creating a PDB from the PDB seed or an application seed, the source files are the files associated with the seed.	Include a CREATE_FILE_DEST clause that specifies the default file system directory or Oracle ASM disk group for the PDB's files.	Omit the CREATE_FILE_DEST clause. Use one of these techniques to specify the target locations of the files: • FILE_NAME_CONVERT clause • Enable Oracle Managed Files for the CDB for it to determine the target locations. • Specify the target locations in the PDB_FILE_NAME_CONVERT initialization parameter. See "PDB File Locations".	Creating a PDB from the PDB seed or an application seed Cloning a PDB Relocating a PDB Plugging in an unplugged PDB
Do you want to specify a default permanent tablespace for the PDB?	Specify a DEFAULT TABLESPACE clause with the appropriate limits. Oracle Database will assign to this tablespace any non-SYSTEM users for whom you do not specify a different permanent tablespace. When creating a PDB from the PDB seed or an application seed, Oracle Database creates a smallfile tablespace and sets it as the default tablespace. When using a technique other than creation from the PDB seed or an application seed, the specified tablespace must exist in the source PDB.	Omit the DEFAULT TABLESPACE clause. If you do not specify this clause, then the SYSTEM tablespace is the default permanent tablespace for non-SYSTEM users. Using the SYSTEM tablespace for non-SYSTEM users is not recommended.	Creating a PDB from the PDB seed or an application seed Cloning a PDB Relocating a PDB Plugging in an unplugged PDB

Table 5-3 (Cont.) Clauses for Creating a PDB

Question	Yes	No	Clause Can Be Used Only When
Do you want to use a `FILE_NAME_CONVERT` clause to specify the target locations of the files? When creating a PDB from the PDB seed or an application seed, the source files are the files associated with the seed.	Include a `FILE_NAME_CONVERT` clause that specifies the target locations of the files based on the names of the source files.	Omit the `FILE_NAME_CONVERT` clause. Use one of these techniques to specify the target locations of the files: • `CREATE_FILE_DEST` clause • Enable Oracle Managed Files for the CDB for it to determine the target locations. • Specify the target locations in the `PDB_FILE_NAME_CONVERT` initialization parameter. See "PDB File Locations".	Creating a PDB from the PDB seed or an application seed Cloning a PDB Relocating a PDB Creating a proxy PDB (Only applies to data files in the `SYSTEM` and `SYSAUX` tablespaces.) Plugging in an unplugged PDB
Is the PDB a reference PDB with a dependent proxy PDB, and is the host name of its listener changing?	Include a `HOST` clause and specify the host name of the listener for the PDB being created. For example, you might have a listener network for the physical host name and default port and configure a second listener bound to a virtual host name and virtual IP address with a nondefault port number.	Omit the `HOST` clause.	Creating a PDB from the PDB seed or an application seed Cloning a PDB Relocating a PDB Plugging in an unplugged PDB
Do you want to specify the logging attribute of the tablespaces in the new PDB?	Include the *logging_clause*.	Omit the *logging_clause*.	Creating a PDB from the PDB seed or an application seed Cloning a PDB Plugging in an unplugged PDB

Table 5-3 (Cont.) Clauses for Creating a PDB

Question	Yes	No	Clause Can Be Used Only When
Do you want to copy or move the files to a new location?	Specify `COPY` to copy the files to a new location. `COPY` is the default. Specify `MOVE` to move the files to a new location. Use one of these techniques to specify the target location: • Include a `FILE_NAME_CONVERT` clause that specifies the target locations based on the names of the source files. • Include a `CREATE_FILE_DEST` clause that specifies the Oracle Managed Files default location for the PDB's files. • Enable Oracle Managed Files for it to determine the target locations. • Specify the target locations in the `PDB_FILE_NAME_CONVERT` initialization parameter. See "PDB File Locations".	Specify `NOCOPY`.	Plugging in an unplugged PDB
Do you want to specify that the data model definition of the source PDB is cloned but not the data of the source PDB?	Include the `NO DATA` clause.	Omit the `NO DATA` clause.	Cloning a PDB
Do you want to use multiple parallel execution servers to parallelize PDB creation?	To let the CDB choose the degree of parallelism, include or omit the `PARALLEL` clause. To specify the degree of parallelism, specify the `PARALLEL` clause with an integer. For example, specify `PARALLEL 4` to indicate a degree of parallelism of 4.	Specify `PARALLEL 0` or `PARALLEL 1`.	Creating a PDB from the PDB seed or an application seed Cloning a PDB

Table 5-3 (Cont.) Clauses for Creating a PDB

Question	Yes	No	Clause Can Be Used Only When
Do you want to use a PATH_PREFIX clause to restrict file paths for the PDB for the following: directory objects, the Oracle XML repository for the PDB, files created with a CREATE PFILE statement, and the export directory for Oracle wallets? The PATH_PREFIX clause does not affect files created by Oracle Managed Files.	Include a PATH_PREFIX clause that specifies an absolute path.	Set the PATH_PREFIX clause to NONE or omit it.	Creating a PDB from the PDB seed or an application seed Cloning a PDB Relocating a PDB Plugging in an unplugged PDB
Is the PDB a reference PDB with a dependent proxy PDB, and is the port number of its listener changing to a value other than 1521?	Include a PORT clause and specify the port number of the listener for the PDB being created. For example, you might have a listener network for the physical host name and default port and configure a second listener bound to a virtual host name and virtual IP address with a nondefault port number.	Omit the PORT clause.	Creating a PDB from the PDB seed or an application seed Cloning a PDB Relocating a PDB Plugging in an unplugged PDB
Do you want to be able to refresh the PDB to propagate changes from the source PDB to the clone PDB? A refreshable PDB must be opened in read-only mode.	Include a REFRESH MODE MANUAL or REFRESH MODE EVERY *minutes* clause.	Omit the REFRESH MODE clause or include a REFRESH MODE NONE clause.	Cloning a PDB
Do you want to grant predefined Oracle roles to the PDB_DBA role locally in the PDB? The new administrator for the PDB is granted the PDB_DBA common role locally in the PDB. By default, the CREATE PLUGGABLE DATABASE statement does not grant the administrator or the role any privileges.	Include the ROLES clause and specify the predefined Oracle roles to grant to the PDB_DBA role. The specified roles are granted to the PDB_DBA role locally in the PDB. The user who runs the CREATE PLUGGABLE DATABASE statement does not need to be granted the specified roles. See *Oracle Database Security Guide* for information about predefined Oracle roles.	Omit the ROLES clause.	Creating a PDB from the PDB seed or an application seed Creating a proxy PDB

Table 5-3 (Cont.) Clauses for Creating a PDB

Question	Yes	No	Clause Can Be Used Only When
Do you want to use a SERVICE_NAME_CONVERT clause to rename the user-defined services of the new PDB based on the service names of the source PDB?	Include a SERVICE_NAME_CONVERT clause that specifies the new name of a service and the service name it is replacing. Specify multiple service names and replacement service names if necessary.	Omit the SERVICE_NAME_CONVERT clause.	Creating a PDB from the application seed, but not a PDB seed Cloning a PDB Relocating a PDB Creating a proxy PDB (Only applies to data files in the SYSTEM and SYSAUX tablespaces.) Plugging in an unplugged PDB
Do you want to clone a PDB using a storage-managed snapshot (*not* a snapshot generated by ALTER PLUGGABLE DATABASE SNAPSHOT)?	Specify a SNAPSHOT COPY clause to clone a PDB using storage-managed snapshots. SNAPSHOT COPY is supported only if the underlying file system supports storage snapshots. A snapshot copy is nearly instantaneous because it does not require copying the full data files of the source PDB. However, you cannot unplug a snapshot copy PDB from the CDB root or application root. Also, if a snapshot copy PDB exists, then you cannot drop the storage snapshot on which the snapshot copy PDB is based. The process of **materializing** transforms a snapshot copy PDB, which uses sparse files, into a full PDB. Materialize a PDB by running the ALTER PLUGGABLE DATABASE MATERIALIZE command.	Omit the SNAPSHOT COPY clause.	Cloning a PDB

Table 5-3 (Cont.) Clauses for Creating a PDB

Question	Yes	No	Clause Can Be Used Only When
Do you want to enable PDB-level snapshots using `ALTER PLUGGABLE DATABASE SNAPSHOT`?	Specify a `SNAPSHOT MODE` clause in the `ALTER PLUGGABLE DATABASE SNAPSHOT` command, and specify `MANUAL` or `EVERY` *`snapshot_interval`* `[MINUTES|HOURS]`.	Omit the `SNAPSHOT MODE` clause or specify `SNAPSHOT MODE NONE`.	Creating a PDB from the PDB seed or an application seed Cloning a PDB Relocating a PDB Creating a proxy PDB (Only applies to data files in the `SYSTEM` and `SYSAUX` tablespaces.) Plugging in an unplugged PDB
Are all source files in a single directory with new file names that would require multiple `SOURCE_FILE_NAME_CONVERT` entries?	Specify the `SOURCE_FILE_DIRECTORY` with the full absolute path to the source files.	Omit the `SOURCE_FILE_DIRECTORY` clause.	Plugging in an unplugged PDB using an XML file directly. This clause does not apply to plugging in an unplugged PDB with a .pdb archive file.
Do the contents of the XML file accurately describe the locations of the source files?	Omit the `SOURCE_FILE_NAME_CONVERT` clause.	Use the `SOURCE_FILE_NAME_CONVERT` clause to specify the source file locations.	Plugging in an unplugged PDB using an XML file directly. This clause does not apply to plugging in an unplugged PDB with a .pdb archive file.
Do you want to include the new PDB in one or more standby CDBs?	Specify `ALL`, `ALL EXCEPT`, or a list of standby CDBs. When creating a remote clone, you can set the initialization parameter `STANDBY_PDB_SOURCE_FILE_DBLINK` to the name of the database link that points to the source PDB data files. The operation copies the data files only if the source PDB is open read-only.	Omit the `STANDBYS` clause or specify `NONE`.	Creating a PDB from the PDB seed or an application seed Cloning a PDB Relocating a PDB Plugging in an unplugged PDB

Table 5-3 (Cont.) Clauses for Creating a PDB

Question	Yes	No	Clause Can Be Used Only When
Do you want to limit the amount of storage that the PDB can use?	Specify a STORAGE clause with the appropriate limits.	Omit the STORAGE clause, or specify unlimited storage using the STORAGE clause.	Creating a PDB from the PDB seed or an application seed Cloning a PDB Relocating a PDB Plugging in an unplugged PDB
Do you want to reuse the temp file if a temp file exists in the target location?	Include the TEMPFILE REUSE clause.	Omit the TEMPFILE REUSE clause. Ensure that there is no file with the same name as the new temp file in the target location.	Creating a PDB from the PDB seed or an application seed Cloning a PDB Relocating a PDB Plugging in an unplugged PDB
Do you want to specify which tablespaces are included in the new PDB and which tablespaces are excluded from the new PDB?	Include the USER_TABLESPACES clause and specify the tablespaces that are included in the new PDB.	Omit the USER_TABLESPACES clause.	Plugging in an unplugged PDB
Do you want to plug an unplugged PDB into a CDB?	Include the USING *filename* clause. If you are plugging in a PDB to a primary CDB in a Data Guard scenario, then set the STANDBY_PDB_SOURCE_FILE_DIRECTORY initialization parameter to a standby location that contains the source data files for instantiating the PDB. If not found, then the standby database tries to locate the files in the OMF location. If not found in the OMF location, then copy the data files to the OMF location, and restart redo apply on the standby database.	Omit the USING *filename* clause.	Plugging in an unplugged PDB

Table 5-3 (Cont.) Clauses for Creating a PDB

Question	Yes	No	Clause Can Be Used Only When
Do you want to create a new PDB based on a PDB snapshot?	Include the `USING SNAPSHOT` clause and specify either the PDB snapshot name, SCN, or timestamp. The result is a full, standalone PDB. A **PDB snapshot** is a point-in-time copy of a PDB. The source PDB can be open read-only or read/write while the snapshot is created. To create PDB-level snapshots manually, specify the `SNAPSHOT` clause of `CREATE PLUGGABLE DATABASE` (or `ALTER PLUGGABLE DATABASE`). Specifying the `EVERY` *`interval`* clause configures the PDB to create snapshots automatically. **Note:** PDB-level snapshots are different from storage-managed snapshots.	Exclude the `USING SNAPSHOT` clause.	Cloning a PDB snapshot
Do you want to clone a PDB that resides in Oracle ASM by splitting a mirror?	Include the `USING MIRROR COPY` clause and specify the name of the mirror copy and the source PDB.	Omit the `USING MIRROR COPY` clause.	Cloning a PDB that uses Oracle ASM storage

General Prerequisites for PDB Creation

Before creating a PDB, you must meet certain prerequisites.

Ensure that the following prerequisites are met before creating a PDB.

Table 5-4 Prerequisites for Creating PDBs

Prerequisite	See Also
The CDB must exist.	" Creating and Configuring a CDB"
The CDB must be in read/write mode.	"Modifying the Open Mode of PDBs"
The current user must be a common user whose current container is the CDB root or an application container.	"Common Users in a CDB"
The current user must have the `CREATE PLUGGABLE DATABASE` system privilege.	"How Commonly Granted System Privileges Work"

Table 5-4 (Cont.) Prerequisites for Creating PDBs

Prerequisite	See Also
You must decide on a unique container name for each container. Each container name must be unique in a single CDB, and each container name must be unique within the scope of all the CDBs whose instances are reached through a specific listener. The PDB name distinguishes a PDB from other PDBs in the CDB. PDB names follow the same rules as service names, which includes being case-insensitive.	*Oracle Database Net Services Reference* to learn the rules for service names
If you are creating a PDB in an Oracle Data Guard configuration with a physical standby database, then you must complete additional tasks before creating a PDB.	*Oracle Data Guard Concepts and Administration* for more information
If you are creating a PDB that includes data that was encrypted with Transparent Data Encryption, then you must complete additional tasks.	*Oracle Database Advanced Security Guide* for instructions
If you are creating a Database Vault-enabled PDB, then you must complete additional tasks.	*Oracle Database Vault Administrator's Guide* for instructions
If you are creating a PDB by cloning a non-CDB, and if you want the ability to recover the new PDB using backups of the source non-CDB, then you *must* execute `DBMS_PDB.EXPORTRMANBACKUP` before cloning. When the source database is opened in read-write mode, execute the procedure as the last step before cloning. This procedure captures all backup metadata in the data dictionary. When relocating a PDB to a different CDB, executing `DBMS_PDB.EXPORTRMANBACKUP` is not necessary. Unplugging the PDB automatically exports the backup metadata.	*Oracle Database Backup and Recovery User's Guide* for instructions

See Also:

- "About the Current Container"
- *Oracle Database PL/SQL Packages and Types Reference* to learn more about `DBMS_PDB.EXPORTRMANBACKUP`

6

Creating a PDB from Scratch

Use the `CREATE PLUGGABLE DATABASE` statement to create a PDB in a CDB using the files of the PDB seed (`PDB$SEED`).

You can also use this statement to create an application PDB in an application container using the files of an application seed or the PDB seed.

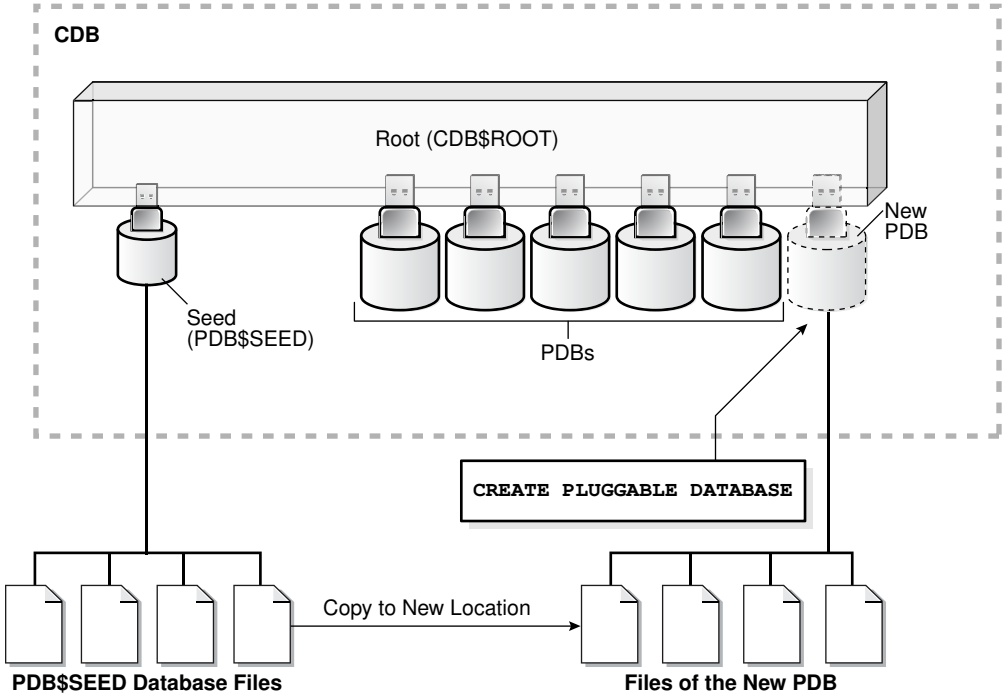
> **See Also:**
>
> *Oracle Database SQL Language Reference* for more information about the `CREATE PLUGGABLE DATABASE` statement

About Creating a PDB from Scratch

Use the `CREATE PLUGGABLE DATABASE` statement to create a new PDB by using the files of the PDB seed or an application PDB from the files of an application seed or the PDB seed.

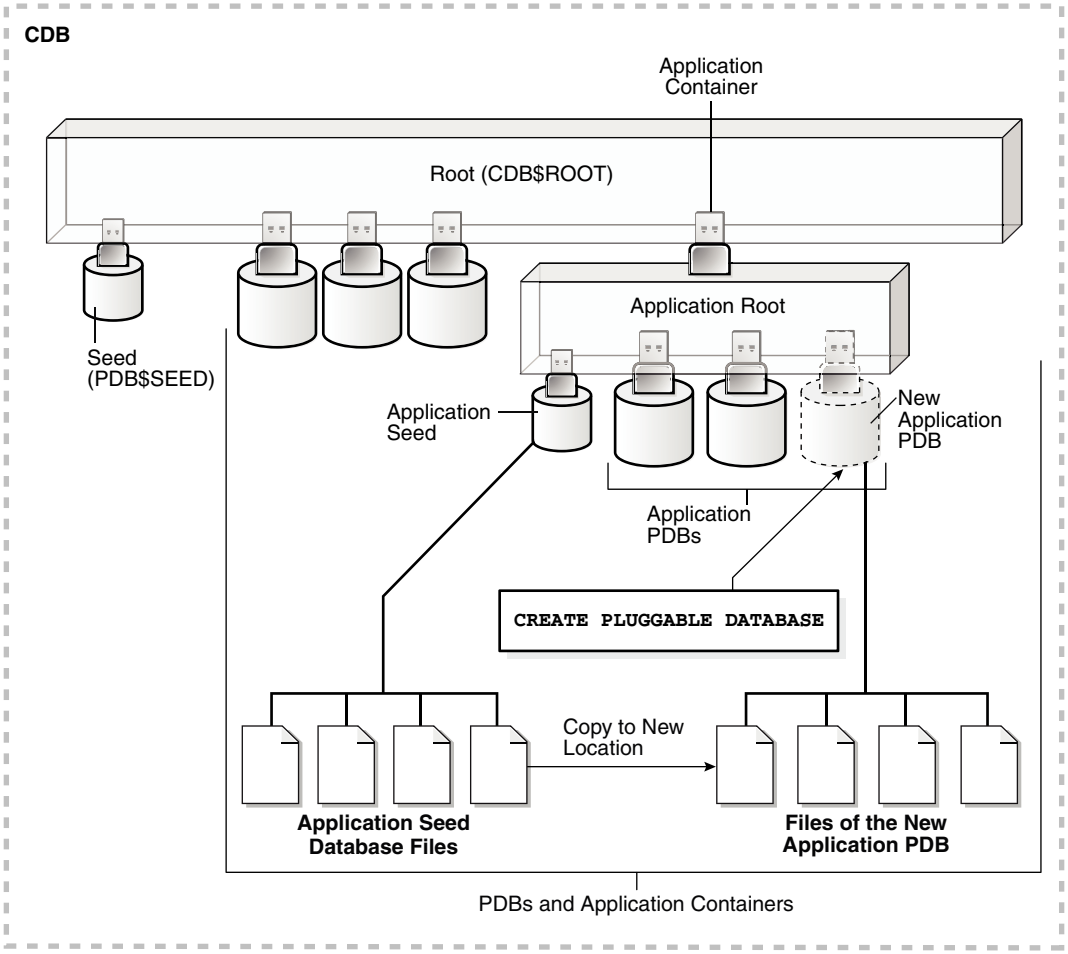
The statement copies these files to a new location and associates them with the new PDB. The following figure illustrates how this technique creates a new PDB in a CDB with the CDB root as the current container.

Figure 6-1 Create a PDB in the CDB Root Using the PDB$SEED Files



The following figure illustrates how this technique creates a new application PDB in an application container with the application root as the current container.

Figure 6-2 Create a PDB in an Application Root Using the Application Seed Files



See Also:

When an application container includes an application seed, and a `CREATE PLUGGABLE DATABASE` statement is run in the application root to create an application PDB from the seed, the application PDB is created using the application seed. However, when an application container does not include an application seed, and a `CREATE PLUGGABLE DATABASE` statement is run in the application root to create an application PDB from the seed, the application PDB is created using the PDB seed (`PDB$SEED`).

When you create a new PDB or application PDB from the seed, you must specify an administrator for the PDB or application PDB in the `CREATE PLUGGABLE DATABASE` statement. The statement creates the administrator as a local user in the PDB and grants the `PDB_DBA` role locally to the administrator.

Before creating a PDB using the PDB seed or an application seed, address the questions that apply to creating a PDB from the seed in Table 5-3. The table describes which CREATE PLUGGABLE DATABASE clauses you must specify based on different factors.

> **See Also:**
>
> - "PDB Storage"
> - "Creating an Application PDB"

Creating a PDB

Using the CREATE PLUGGABLE DATABASE statement, you can create a PDB from the PDB seed, and you can create an application PDB from an application seed or the PDB seed.

Prerequisites

Before creating a PDB from the PDB seed (PDB$SEED) or an application PDB from an application seed or the PDB seed, complete the prerequisites described in "General Prerequisites for PDB Creation".

To create a PDB:

1. In SQL*Plus, ensure that the current container is the CDB root or an application root.

   When the current container is the CDB root, the PDB is created in the CDB using the files of the PDB seed.

   When the current container is an application root, the application PDB is created in the application container using the files of the application seed. If there is no application seed in the application container, then the application PDB is created in the application container using the files of the PDB seed.

2. Run the CREATE PLUGGABLE DATABASE statement, and specify a local administrator for the PDB. Specify other clauses when they are required.

   After you create the PDB, it is in mounted mode, and its status is NEW. You can view the open mode of a PDB by querying the OPEN_MODE column in the V$PDBS view. You can view the status of a PDB by querying the STATUS column of the CDB_PDBS or DBA_PDBS view.

   A new default service is created for the PDB. The service has the same name as the PDB and can be used to access the PDB. Oracle Net Services must be configured properly for clients to access this service.

3. Open the new PDB in read/write mode.

   You must open the new PDB in read/write mode for Oracle Database to complete the integration of the new PDB into the CDB. An error is returned if you attempt to open the PDB in read-only mode. After the PDB is opened in read/write mode, its status is NORMAL.

4. Back up the PDB.

A PDB cannot be recovered unless it is backed up.

A local user with the name of the specified local administrator is created and granted the PDB_DBA common role locally in the PDB. If this user was not granted administrator privileges during PDB creation, then use the SYS and SYSTEM common users to administer to the PDB.

> **Note:**
>
> If an error is returned during PDB creation, then the PDB being created might be in an UNUSABLE state. You can check a PDB's state by querying the CDB_PDBS or DBA_PDBS view, and you can learn more about PDB creation errors by checking the alert log. An unusable PDB can only be dropped, and it must be dropped before a PDB with the same name as the unusable PDB can be created.

> **See Also:**
>
> - "About Container Access in a CDB"
> - "Modifying the Open Mode of PDBs" for more information
> - *Oracle Database Backup and Recovery User's Guide* for information about backing up a PDB

Creating a PDB: Examples

These examples create a new PDB named salespdb and a salesadm local administrator given different factors.

In addition to creating the salespdb PDB, this statement grants the PDB_DBA role to the PDB administrator salesadm and grants the specified predefined Oracle roles to the PDB_DBA role locally in the PDB.

In each example, the root to which the new PDB belongs depends on the current container when the CREATE PLUGGABLE DATABASE statement is run:

- When the current container is the CDB root, the new PDB is created in the CDB root.
- When the current container is an application root in an application container, the new PDB is created as an application PDB in the application root.

Creating a PDB Using No Clauses: Example

This example shows the simplest way to create a PDB.

This example assumes the following factors:

- Storage limits are not required for the PDB. Therefore, the STORAGE clause is not required.

- The PDB does not require a default tablespace.
- The `PATH_PREFIX` clause is not required.
- The `FILE_NAME_CONVERT` clause and the `CREATE_FILE_DEST` clause are not required.

  Either Oracle Managed Files is enabled for the CDB, or the `PDB_FILE_NAME_CONVERT` initialization parameter is set. The files associated with the PDB seed or application seed will be copied to a new location based on the Oracle Managed Files configuration or the initialization parameter setting.
- There is no file with the same name as the new temp file that will be created in the target location. Therefore, the `TEMPFILE REUSE` clause is not required.
- No predefined Oracle roles need to be granted to the `PDB_DBA` role.

The following statement creates the PDB:

```
CREATE PLUGGABLE DATABASE salespdb ADMIN USER salesadm IDENTIFIED BY
pwd;
```

See Also:

- *Oracle Database Administrator's Guide* for information about using Oracle Managed Files
- *Oracle Database Reference* for information about the `PDB_FILE_NAME_CONVERT` initialization parameter
- *Oracle Database Security Guide* for guidelines about choosing passwords

Creating a PDB and Granting Predefined Oracle Roles to the PDB Administrator: Example

This example uses the `ROLES` parameter to grant a predefined role.

This example assumes the following factors:

- Storage limits are not required for the PDB. Therefore, the `STORAGE` clause is not required.
- The PDB does not require a default tablespace.
- The `PATH_PREFIX` clause is not required.
- The `FILE_NAME_CONVERT` clause and the `CREATE_FILE_DEST` clause are not required.

  Either Oracle Managed Files is enabled, or the `PDB_FILE_NAME_CONVERT` initialization parameter is set. The files associated with the PDB seed or application seed will be copied to a new location based on the Oracle Managed Files configuration or the initialization parameter setting.
- There is no file with the same name as the new temp file that will be created in the target location. Therefore, the `TEMPFILE REUSE` clause is not required.

- The PDB_DBA role should be granted the following predefined Oracle role locally: DBA.

The following statement creates the PDB:

```
CREATE PLUGGABLE DATABASE salespdb
  ADMIN USER salesadm IDENTIFIED BY password
    ROLES=(DBA);
```

See Also:

- *Oracle Database Administrator's Guide* for information about using Oracle Managed Files
- *Oracle Database Reference* for information about the PDB_FILE_NAME_CONVERT initialization parameter
- *Oracle Database Security Guide* for guidelines about choosing passwords

Creating a PDB Using Multiple Clauses: Example

This example creating a PDB using the STORAGE, DEFAULT TABLESPACE, PATH_PREFIX, and FILE_NAME_CONVERT clauses.

This example assumes the following factors:

- Storage limits must be enforced for the PDB. Therefore, the STORAGE clause is required. Specifically, all tablespaces that belong to the PDB must not exceed 2 gigabytes.
- A default permanent tablespace is required for any non-administrative users for which you do not specify a different permanent tablespace. Specifically, this example creates a default permanent tablespace named sales with the following characteristics:
  - The single data file for the tablespace is sales01.dbf, and the statement creates it in the /disk1/oracle/dbs/salespdb directory.
  - The SIZE clause specifies that the initial size of the tablespace is 250 megabytes.
  - The AUTOEXTEND clause enables automatic extension for the file.
- The path prefix must be added to the PDB directory object paths. Therefore, the PATH_PREFIX clause is required. In this example, the path prefix /disk1/oracle/dbs/salespdb/ is added to the PDB's directory object paths.
- The CREATE_FILE_DEST clause will not be used, Oracle Managed Files is not enabled, and the PDB_FILE_NAME_CONVERT initialization parameter is not set. Therefore, the FILE_NAME_CONVERT clause is required. Specify the location of the data files for the PDB seed or application seed on your system. In this example, Oracle Database copies the files from /disk1/oracle/dbs/pdbseed to /disk1/oracle/dbs/salespdb.

- There is no file with the same name as the new temp file that will be created in the target location. Therefore, the `TEMPFILE REUSE` clause is not required.
- No predefined Oracle roles need to be granted to the `PDB_DBA` role.

The following statement creates the PDB:

```
CREATE PLUGGABLE DATABASE salespdb
  ADMIN USER salesadm IDENTIFIED BY password
  STORAGE (MAXSIZE 2G)
  DEFAULT TABLESPACE sales
    DATAFILE '/disk1/oracle/dbs/salespdb/sales01.dbf' SIZE 250M
    AUTOEXTEND ON
    PATH_PREFIX = '/disk1/oracle/dbs/salespdb/'
    FILE_NAME_CONVERT = ('/disk1/oracle/dbs/pdbseed/',
                         '/disk1/oracle/dbs/salespdb/');
```

See Also:

- "Example 19-7" to learn how to view the location of the data files for the PDB seed or application seed
- *Oracle Database SQL Language Reference* for more information about the `DEFAULT TABLESPACE` clause
- *Oracle Database Security Guide* for guidelines about choosing passwords

7

Cloning a PDB or Non-CDB

You can create a PDB by cloning a local PDB, a remote PDB, or a non-CDB.

About Cloning a PDB or Non-CDB

Cloning means creating a new PDB from a source PDB or from a non-CDB.

A typical use case is development testing. You can create one or more clones of a PDB or non-CDB and safely test them in isolation. For example, you might test a new or modified application on a cloned PDB before using the application with a production PDB.

> **See Also:**
>
> *Oracle Database Advanced Security Guide* to learn about cloning a source with encrypted data or a keystore set

How Cloning Works

This technique creates a new PDB from a source PDB or non-CDB. The process automatically plugs the new PDB into the CDB.

To use this technique, you must specify the source in a `CREATE PLUGGABLE DATABASE` statement. The source can be any of the following:

- Local PDB
- PDB in a remote CDB
- Non-CDB

The target PDB is the copy of the source PDB or non-CDB. The copy is called a clone PDB.

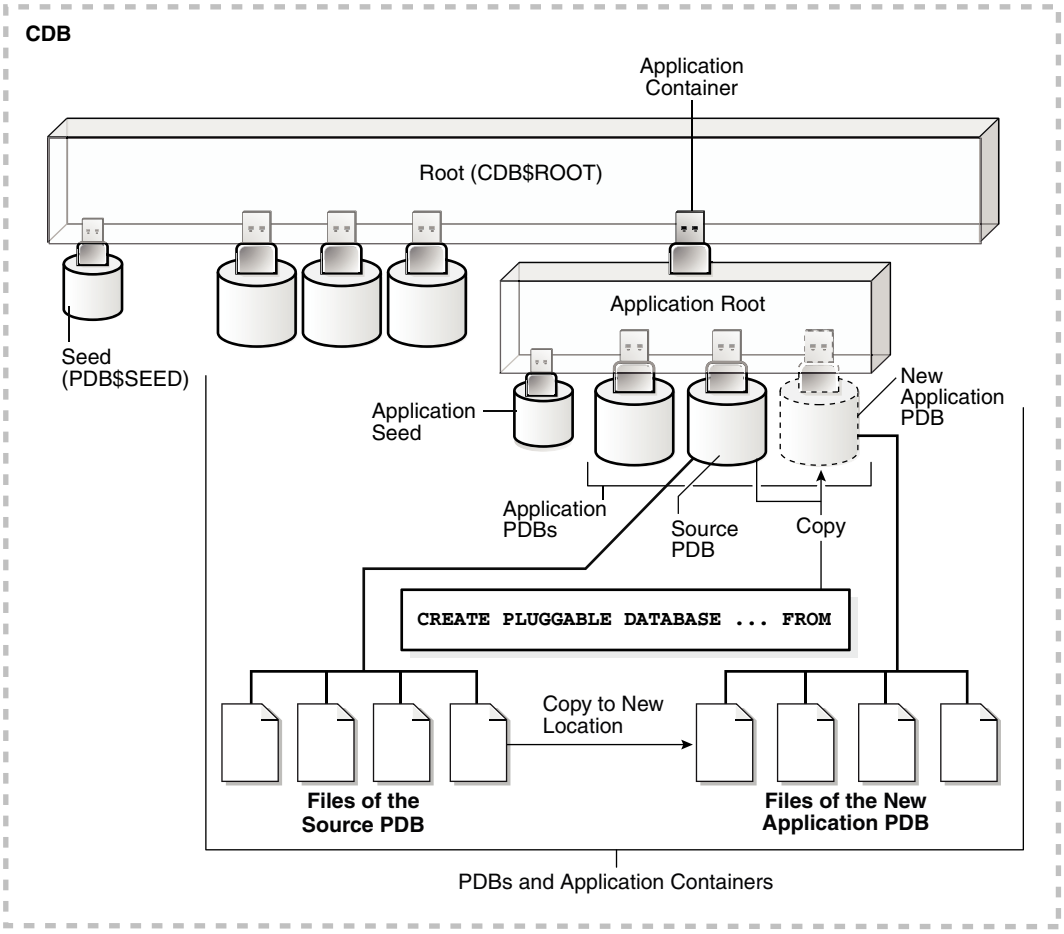
The `CREATE PLUGGABLE DATABASE` statement copies the files associated with the source to a new location and associates the files with the target PDB. When the CDB is in `ARCHIVELOG` mode and local undo mode, the source PDB can be open in read/write mode and operational during the cloning process. This technique is known as **hot cloning**.

> **Note:**
>
> If you clone a PDB, and if the source database has encrypted data or a keystore set, then you must provide the keystore password by including the keystore identified by *keystore_password* clause in the `CREATE PLUGGABLE DATABASE ... FROM` SQL statement. You must provide this password even if the source database is using an auto-login software keystore. You can determine whether the source database has encrypted data or a keystore by querying the `DBA_ENCRYPTED_COLUMNS` data dictionary view.

In all cloning scenarios, when you run the `CREATE PLUGGABLE DATABASE` statement in the application root, the clone PDB is created in the application container. The application name and version of the source PDB must match the application name and version of the application container.

The following graphic illustrates how this technique creates a new application PDB in an application container by cloning a local source application PDB. The source PDB can also be a PDB plugged into the local CDB root, a PDB plugged into a remote CDB root, or an application PDB plugged into a remote application root.

Figure 7-1 Clone a PDB in an Application Container



> **See Also:**
>
> "PDB Storage"

User Interface for PDB Cloning

All forms of PDB cloning use the `CREATE PLUGGABLE DATABASE` statement.

Cloning requires specifying the source PDB in a `FROM` clause. The following table summarizes the most important clauses.

Table 7-1 CREATE PLUGGABLE DATABASE Options for PDB Cloning

Clause	Cloning Operation	See Also
`USING SNAPSHOT`	Creates a clone from a PDB-level snapshot (`ALTER PLUGGABLE DATABASE SNAPSHOT`). Specify the PDB snapshot name, SCN, or timestamp. **Note:** You cannot create a snapshot *copy* PDB that is based on a PDB snapshot by including both the `USING SNAPSHOT` clause and the `SNAPSHOT COPY` clause. However, you can create a clone based on a PDB snapshot with `USING SNAPSHOT`, and then create a `SNAPSHOT COPY` PDB from the clone.	"Clones from PDB Snapshots"
`REFRESH MODE`	Creates a refreshable clone PDB.	"Refreshable Clone PDBs"
`SNAPSHOT COPY`	Creates a snapshot copy PDB from a storage-managed snapshot (*not* `ALTER PLUGGABLE DATABASE SNAPSHOT`). Storage-managed snapshots are only supported on specific file systems. A snapshot copy PDB does not include a complete copy of the source data files. Rather, Oracle Database creates a storage-level snapshot of the underlying file system, and then creates the clone PDB from the snapshot. Unlike a standard clone PDB, the snapshot copy PDB is dependent on the storage snapshot. Therefore, you cannot unplug a snapshot copy PDB from the CDB root or plug it in to an application root. Also, you cannot drop the storage snapshot on which the PDB is based. Instead, you must materialize the snapshot copy PDB, which converts it into a full PDB with non-sparse files.	"Snapshot Copy PDBs"
`USING MIRROR COPY`	Creates a new PDB by splitting the ASM storage mirror specified by *mirror_name*. You can only split one PDB from a prepared mirror copy. If you want to create additional splits, you must prepare a new mirror copy.	"Creating a Split Mirror Clone PDB"

> **See Also:**
>
> - "Materializing a Snapshot Copy PDB"
> - *Oracle Database SQL Language Reference* to learn more about `CREATE PLUGGABLE DATABASE` clauses

Cloning a Local PDB

You can clone a local PDB by running a `CREATE PLUGGABLE DATABASE` statement and specifying a local PDB in the `FROM` statement.

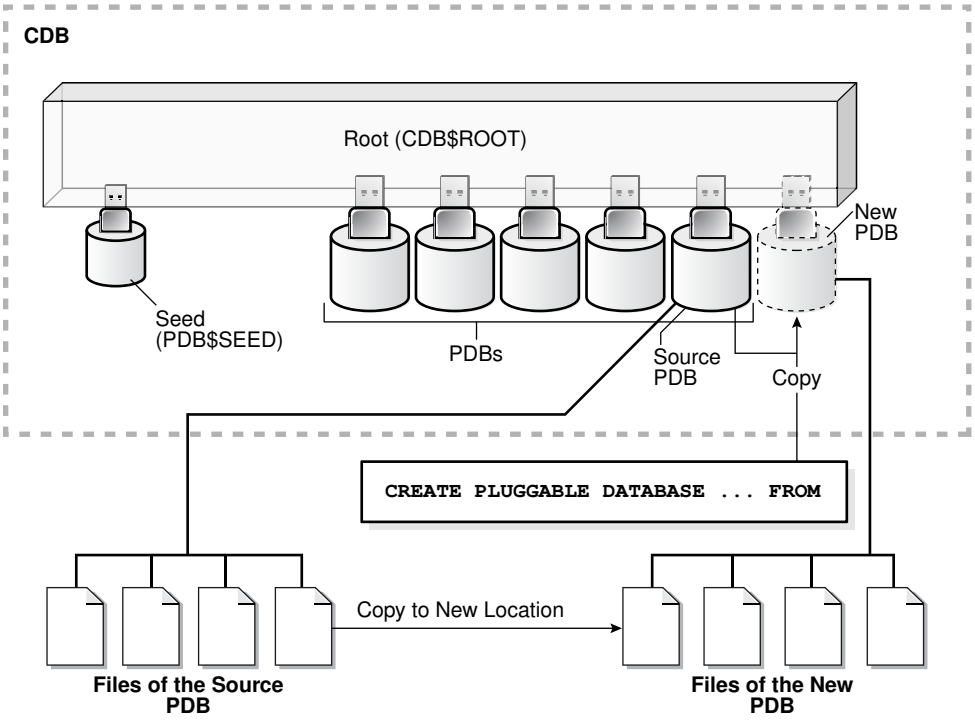
About Cloning a Local PDB

The simplest form of cloning copies a PDB from a CDB into the same CDB.

> **Note:**
>
> You cannot use the `FROM` clause in the `CREATE PLUGGABLE DATABASE` statement to create a PDB from the PDB seed (`PDB$SEED`) or from an application seed.

The following figure illustrates how to clone a local PDB.

Figure 7-2 Clone a Local PDB



Before cloning a PDB, address the questions that apply to cloning a PDB in "Table 5-3". The table describes which `CREATE PLUGGABLE DATABASE` clauses to specify based on different factors.

Starting in Oracle Database 18c, you can clone a local PDB using DBCA.

> **See Also:**
>
> - "Determining the Current Container ID or Name"
> - "Creating a PDB from Scratch" to learn how to create a PDB from the seed
> - "Cloning a Local PDB Using DBCA: Example"
> - "Creating an Application PDB"

Cloning a Local PDB: Basic Steps

You can clone a local PDB by executing `CREATE PLUGGABLE DATABASE` and specify the source PDB in the `FROM` clause.

Prerequisites

You must meet the following prerequisites:

- Complete the prerequisites described in "General Prerequisites for PDB Creation".
- The current user must have the `CREATE PLUGGABLE DATABASE` system privilege in both the root and the source PDB.
- The source PDB cannot be closed.
- If the CDB is not in local undo mode, then the source PDB must be in open read-only mode. This requirement does not apply if the CDB is in local undo mode.
- If the CDB is not in `ARCHIVELOG` mode, then the source PDB must be in open read-only mode. This requirement does not apply if the CDB is in `ARCHIVELOG` mode.
- If you are creating an application PDB, then the application PDB must have the same character set and national character set as the application container.

  If the database character set of the CDB is AL32UTF8, then the character set and national character set of the application container can be different from the CDB. However, all application PDBs in an application container must have same character set and national character set, matching that of the application container.

> **Note:**
>
> You can use the `REFRESH MODE` clause to create a refreshable clone of a local PDB, but only if the database link loops back to the same CDB.

To clone a local PDB:

1. In SQL*Plus, ensure that the current container is the CDB root or an application root.

   When the current container is the CDB root, the PDB is created in the CDB. When the current container is an application root, the application PDB is created in the application container.

2. Run the `CREATE PLUGGABLE DATABASE` statement, and specify the source PDB in the `FROM` clause. Specify other clauses when required.

   After cloning a local PDB, the source and target PDBs are in the same CDB. The new PDB is in mounted mode, and its status is `NEW`. You can view the open mode of a PDB by querying the `OPEN_MODE` column in the `V$PDBS` view. You can view the status of a PDB by querying the `STATUS` column of the `CDB_PDBS` or `DBA_PDBS` view.

   A new default service is created for the PDB. The service has the same name as the PDB and can be used to access the PDB. Oracle Net Services must be configured properly for clients to access this service.

3. Open the new PDB in read/write mode.

   You must open the new PDB in read/write mode for Oracle Database to complete the integration of the new PDB into the CDB. An error is returned if you attempt to open the PDB in read-only mode. After the PDB is opened in read/write mode, its status is `NORMAL`.

4. Back up the new PDB.

   A PDB cannot be recovered unless it is backed up.

> **Note:**
>
> If an error is returned during PDB creation, then the PDB being created might be in an `UNUSABLE` state. You can check the PDB state by querying the `CDB_PDBS` or `DBA_PDBS` view. You can learn more about PDB creation errors by checking the alert log. An unusable PDB can only be dropped, and it must be dropped before you can create a PDB with the same name as the unusable PDB.

> **See Also:**
>
> - "About the Current Container" and "About Container Access in a CDB"
> - "About the CDB Undo Mode"
> - "Modifying the Open Mode of PDBs"
> - *Oracle Database Backup and Recovery User's Guide* to learn how to back up a PDB

After Cloning a Local PDB

Certain rules regarding users and tablespaces apply after cloning a local PDB.

Users in the new PDB who used the default temporary tablespace of the source PDB use the default temporary tablespace of the new PDB. Users who used nondefault temporary tablespaces in the PDB continue to use the same local temporary tablespaces in the cloned PDB.

> **See Also:**
>
> "About Managing Tablespaces in a CDB"

Cloning a Local PDB: Examples

The following examples clone a local source PDB named `pdb1` to a target PDB named `pdb2` given different factors.

In each example, the root to which the new PDB belongs depends on the current container when the `CREATE PLUGGABLE DATABASE` statement is run:

- When the current container is the CDB root, the database creates the PDB in the CDB root.
- When the current container is an application root in an application container, the database creates an application PDB in the application root.

Cloning a Local PDB Using No Clauses: Example

This example shows the simplest way to clone a PDB.

This example assumes the following factors:

- The `PATH_PREFIX` clause is not required.
- The `FILE_NAME_CONVERT` clause and the `CREATE_FILE_DEST` clause are not required.

  Either Oracle Managed Files is enabled, or the `PDB_FILE_NAME_CONVERT` initialization parameter is set. Therefore, the `FILE_NAME_CONVERT` clause is not required. The files will be copied to a new location based on the Oracle Managed Files configuration or the initialization parameter setting.
- Storage limits are not required for the PDB. Therefore, the `STORAGE` clause is not required.
- There is no file with the same name as the new temp file that will be created in the target location. Therefore, the `TEMPFILE REUSE` clause is not required.

The following statement clones the `pdb2` PDB from the `pdb1` PDB:

```
CREATE PLUGGABLE DATABASE pdb2 FROM pdb1;
```

> **See Also:**
>
> - *Oracle Database Administrator's Guide* for more information about Oracle Managed Files
> - *Oracle Database Reference* for information about the `PDB_FILE_NAME_CONVERT` initialization parameter

Cloning a Local PDB Using DBCA: Example

This example clones a PDB using the silent mode of DBCA. Hot cloning is supported.

This example assumes the following factors:

- The source CDB is a single-instance database with the SID `orcl`.
- The source PDB is `pdb1`. You intend for `pdb1` to remain open during the cloning operation, which means that local undo and `ARCHIVELOG` mode are enabled in the CDB. Otherwise, DBCA closes the PDB during the clone operation, and after receiving confirmation, opens the source PDB in read-only mode.
- The new PDB is `pdb2`.
- You are running DBCA in noninteractive mode.

The following command clones the `pdb2` PDB from the `pdb1` PDB:

```
./dbca -silent
  -createpluggabledatabase
  -sourcedb orcl
  -createpdbfrom PDB
  -pdbName pdb2
  -sourcepdb pdb1
```

> **See Also:**
>
> *Oracle Database Administrator's Guide* for the DBCA command reference

Cloning a Local PDB with the PATH_PREFIX Clause: Example

This example explains how to clone a local PDB with the `PATH_PREFIX`, `FILE_NAME_CONVERT`, and `SERVICE_NAME_CONVERT` clauses.

This example assumes the following factors:

- The path prefix must be added to the PDB's directory object paths. Therefore, the `PATH_PREFIX` clause is required. In this example, the path prefix /disk2/oracle/pdb2/ is added to the PDB's directory object paths.
- The `FILE_NAME_CONVERT` clause is required to specify the target locations of the copied files. In this example, the files are copied from `/disk1/oracle/pdb1` to `/disk2/oracle/pdb2`.

  The `CREATE_FILE_DEST` clause is not used, and neither Oracle Managed Files nor the `PDB_FILE_NAME_CONVERT` initialization parameter is used to specify the target locations of the copied files.

  To view the location of the data files for a PDB, run the query in "Example 19-7".
- Storage limits are not required for the PDB. Therefore, the `STORAGE` clause is not required.
- There is no file with the same name as the new temp file that will be created in the target location. Therefore, the `TEMPFILE REUSE` clause is not required.

- The PDB that is being cloned (pdb1) has two user-defined services: salesrep_ca and orders_ca for the sales representatives and order entry personnel in California. The new services will be for the sales representatives and order entry personnel in Oregon, and the service names will be renamed to salesrep_or and orders_or, respectively, in the cloned PDB (pdb2).
- Future tablespaces created within the PDB will be created with the NOLOGGING attribute by default. This feature is available starting with Oracle Database 12c Release 1 (12.1.0.2).

The following statement clones the pdb2 PDB from the pdb1 PDB:

```
CREATE PLUGGABLE DATABASE pdb2 FROM pdb1
  PATH_PREFIX = '/disk2/oracle/pdb2/'
  FILE_NAME_CONVERT = ('/disk1/oracle/pdb1/', '/disk2/oracle/pdb2/')
  SERVICE_NAME_CONVERT =
('salesrep_ca','salesrep_or','orders_ca','orders_or')
  NOLOGGING;
```

Cloning a Local PDB Using the STORAGE Clause: Example

This example clones a local PDB using the FILE_NAME_CONVERT, STORAGE, and SERVICE_NAME_CONVERT clauses.

This example assumes the following factors:

- The PATH_PREFIX clause is not required.
- The FILE_NAME_CONVERT clause is required to specify the target locations of the copied files. In this example, the files are copied from /disk1/oracle/pdb1 to /disk2/oracle/pdb2.

  The CREATE_FILE_DEST clause is not used, and neither Oracle Managed Files nor the PDB_FILE_NAME_CONVERT initialization parameter is used to specify the target locations of the copied files.

  To view the location of the data files for a PDB, run the query in Example 19-7.
- Storage limits must be enforced for the PDB. Therefore, the STORAGE clause is required. Specifically, all tablespaces that belong to the PDB must not exceed 2 gigabytes.
- The source PDB (pdb1) has two user-defined services: salesrep_ca and orders_ca for the sales representatives and order entry personnel in California. The new services will be for the sales representatives and order entry personnel in Oregon, and the service names will be renamed to salesrep_or and orders_or, respectively, in the cloned PDB (pdb2).
- There is no file with the same name as the new temp file that will be created in the target location. Therefore, the TEMPFILE REUSE clause is not required.

The following statement clones the pdb2 PDB from the pdb1 PDB:

```
CREATE PLUGGABLE DATABASE pdb2 FROM pdb1
  FILE_NAME_CONVERT = ('/disk1/oracle/pdb1/', '/disk2/oracle/pdb2/')
  STORAGE (MAXSIZE 2G)
  SERVICE_NAME_CONVERT =
('salesrep_ca','salesrep_or','orders_ca','orders_or');
```

Cloning a Local PDB with the NO DATA Clause: Example

This example clones the data model definition of the PDB, but does not clone the data in the PDB.

This example assumes the following factors:

- The `NO DATA` clause is required because the goal is to clone the data model definition of the source PDB without cloning its data.
- The `PATH_PREFIX` clause is not required.
- The `FILE_NAME_CONVERT` clause and the `CREATE_FILE_DEST` clause are not required.

  Either Oracle Managed Files is enabled, or the `PDB_FILE_NAME_CONVERT` initialization parameter is set. Therefore, the `FILE_NAME_CONVERT` clause is not required. The process copies the files to a new location based on the Oracle Managed Files configuration or the initialization parameter setting.
- Storage limits are not required for the PDB. Therefore, the `STORAGE` clause is not required.
- There is no file with the same name as the new temp file that will be created in the target location. Therefore, the `TEMPFILE REUSE` clause is not required.

Assume that the source PDB `pdb1` has a large amount of data. The following steps illustrate how the clone does not contain the data of the source PDB when the operation is complete:

1. With the source PDB `pdb1` as the current container, query a table with a large amount of data:

   ```
   SELECT COUNT(*) FROM tpch.lineitem;

     COUNT(*)
   ----------
     60001215
   ```

   The table has over sixty million rows.
2. Clone the source PDB with the `NO DATA` clause:

   ```
   CREATE PLUGGABLE DATABASE pdb2 FROM pdb1 NO DATA;
   ```

3. Open the cloned PDB:

   ```
   ALTER PLUGGABLE DATABASE pdb2 OPEN;
   ```

4. With the cloned PDB `pdb2` as the current container, query the table that has a large amount of data in the source PDB:

   ```
   SELECT COUNT(*) FROM tpch.lineitem;

     COUNT(*)
   ```

```
----------
         0
```

The table in the cloned PDB has no rows.

Cloning a Remote PDB

You can clone a local PDB by running a `CREATE PLUGGABLE DATABASE` statement, and specifying a database link to the remote PDB in the `FROM` statement.

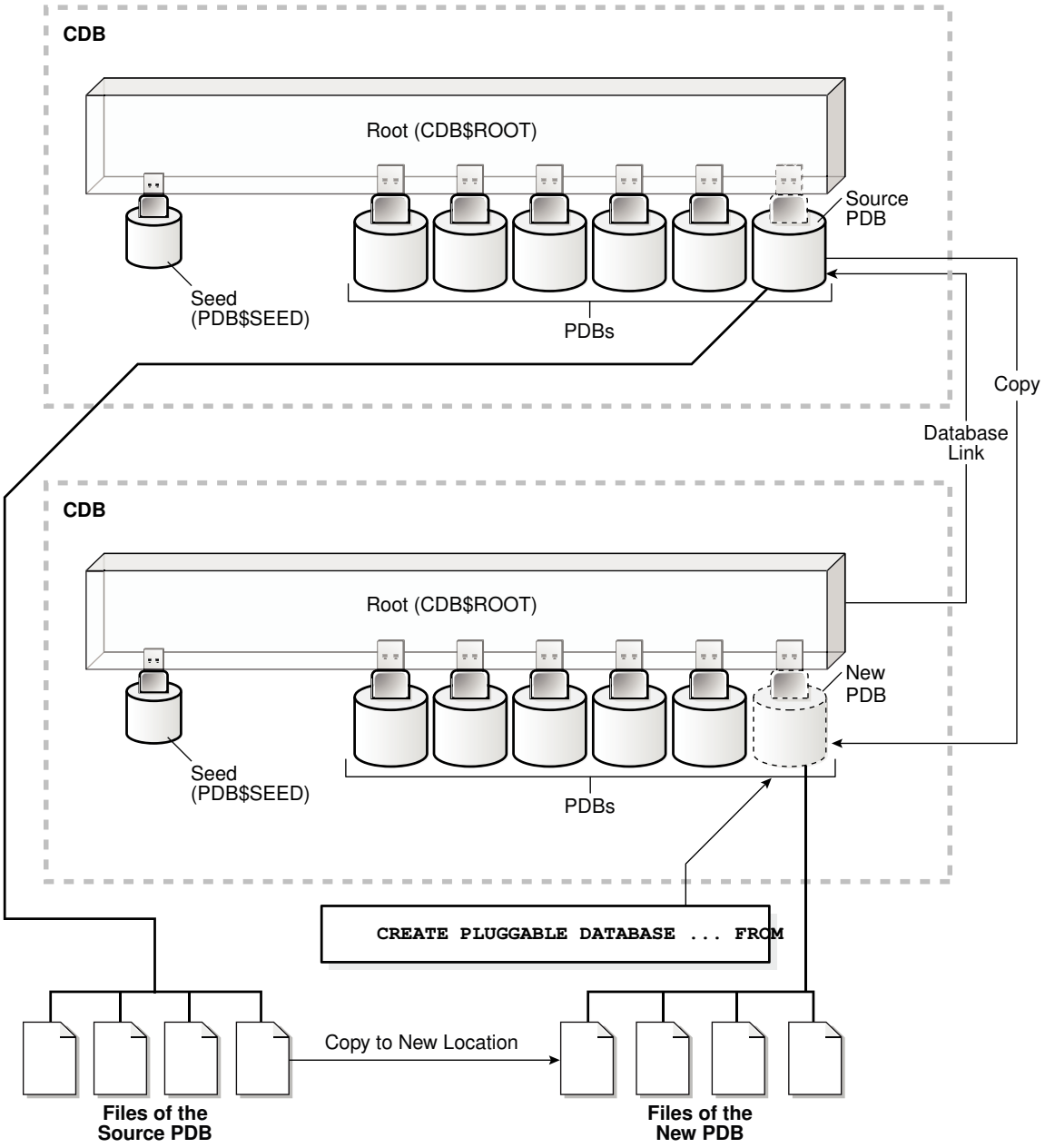
About Cloning a Remote PDB

When the source is a PDB is in a remote CDB, you must use a database link to clone the PDB into the local CDB.

The database link must exist in the *local* CDB (not the remote CDB). When you issue the `CREATE PLUGGABLE DATABASE` statement from the root of the local CDB, you must specify a database link to the remote CDB that contains the PDB being cloned in the `FROM` clause. The database link connects from the local CDB to either to the root of the remote CDB or to the remote source PDB.

The following figure illustrates how this technique creates a new PDB when the source PDB is remote.

Figure 7-3 Creating a PDB by Cloning a Remote PDB



Starting in Oracle Database 19c, you can clone a remote PDB using DBCA in silent mode.

Cloning a Remote PDB: Basic Steps

You can create a PDB by cloning a remote PDB. After the cloning operation, the source and the target PDB are in different locations.

General Prerequisites

The following prerequisites must be met:

- Complete the prerequisites described in "General Prerequisites for PDB Creation".
- The current user must have the `CREATE PLUGGABLE DATABASE` system privilege in the root of the CDB that will contain the target PDB.
- The source and target platforms must meet the following requirements:
  - They must have the same endianness.
  - The database options installed on the source platform must be the same as, or a subset of, the database options installed on the target platform.
- If you are creating an application PDB, then the application name and version of the source PDB must match the application name and version of the target application container.

Prerequisites for Character Sets

- If the character set of the CDB to which the PDB is being cloned is not AL32UTF8, then the source and target must have compatible character sets and national character sets. If the character set of the CDB to which the PDB is being cloned is AL32UTF8, then this requirement does not apply.
- If you are creating an application PDB, then the application PDB must have the same character set and national character set as the application container.

  If the database character set of the CDB is AL32UTF8, then the character set and national character set of the application container can different from the CDB. However, all application PDBs in an application container must have same character set and national character set, matching that of the application container.

> **Note:**
>
> Oracle Multitenant does not support a LOB in one container from being accessed by a container with a different character set using data links, extended data links, or the `CONTAINERS()` clause. For example, if the CDB root and `salespdb` have different character sets, then a `CONTAINERS()` query run in the CDB root should not access LOBs stored in `salespdb`.

Prerequisites for the Open Mode of the Source PDB

- The source PDB must not be closed.
- If the remote CDB is not in local undo mode, then the source PDB must be open in read-only mode.

  See "About the CDB Undo Mode".

- If the remote CDB is not in ARCHIVELOG mode, then the source PDB must be open in read-only mode.
- If you are creating a refreshable PDB, then the source PDB must be in ARCHIVELOG mode and local undo mode.

Prerequisites for the Database Link

The following prerequisites must be met:

- A database link must enable a connection from the destination CDB (the CDB to which the PDB is being cloned) to the PDB in the source CDB.
- The database link can connect as a common user to the root of the source CDB, or as a common or local user to the source PDB. The source PDB can be either a standard PDB or application PDB.
- The user account specified in the database link must have either of the following privileges:
  - The CREATE PLUGGABLE DATABASE privilege, granted either commonly or locally, on the source PDB
  - The SYSOPER privilege
- In an Oracle Data Guard environment, if you are performing a remote clone of a PDB into a primary CDB, then on the standby CDB set the STANDBY_PDB_SOURCE_FILE_DBLINK initialization parameter. This parameter specifies the name of the database link used in CREATE PLUGGABLE DATABASE ... FROM *dblink*. The standby CDB attempts to copy the data files from the source PDB referenced in the database link, but only if the source PDB is open in read-only mode. Otherwise, you must copy data files to the Oracle Managed Files location on the standby CDB.

To clone a remote PDB:

1. In SQL*Plus, ensure that the current container is the root of the target CDB or the application root of the target application container.
2. Run the CREATE PLUGGABLE DATABASE statement, and specify the source PDB in the FROM clause. Specify other clauses when required.

   After you create the PDB, it is in mounted mode, and its status is NEW. You can view the open mode of a PDB by querying the OPEN_MODE column in the V$PDBS view. You can view the status of a PDB by querying the STATUS column of the CDB_PDBS or DBA_PDBS view.

   A new default service is created for the PDB. The service has the same name as the PDB and can be used to access the PDB. Oracle Net Services must be configured properly for clients to access this service.

> **Note:**
>
> If an error is returned during PDB creation, then the PDB being created might be in an `UNUSABLE` state. You can check the PDB state by querying the `CDB_PDBS` or `DBA_PDBS` view, and you can learn more about PDB creation errors by checking the alert log. An unusable PDB can only be dropped, and it must be dropped before a PDB with the same name as the unusable PDB can be created.

3. Open the new PDB in read/write mode.

   You must open the new PDB in read/write mode for Oracle Database to complete the integration of the new PDB into the CDB. An error is returned if you attempt to open the PDB in read-only mode. After the PDB is opened in read/write mode, its status is `NORMAL`.

4. Back up the PDB.

   A PDB cannot be recovered unless it is backed up.

> **See Also:**
>
> - "Refreshing a PDB"
> - "Modifying the Open Mode of PDBs"
> - *Oracle Database Backup and Recovery User's Guide* for information about backing up a PDB
> - *Oracle Data Guard Concepts and Administration* to learn more about plugging in a PDB in an Oracle Data Guard environment
> - *Oracle Database Globalization Support Guide* to learn about the requirements for the compatibility of character sets
> - *Oracle Database Reference* for information about the `PDB_FILE_NAME_CONVERT` initialization parameter

After Cloning a Remote PDB

Certain rules regarding users and tablespaces apply after cloning a remote PDB.

The following applies after cloning a remote PDB:

- Users in the new PDB who used the default temporary tablespace of the source PDB use the default temporary tablespace of the new PDB. Users who used nondefault temporary tablespaces in the PDB continue to use the same local temporary tablespaces in the cloned PDB.
- User-created common user accounts that existed in the source CDB but not in the target CDB do not have privileges granted commonly. However, if the target CDB has a common user account with the same name as a common user account in the PDB, then the latter is linked to the former and has the privileges granted to this common user account in the target CDB.

If the cloned or plugged-in PDB has a common user account that does not exist in the target CDB, and if this user does not own objects in the PDB, then Oracle Database drops the user during the synchronization step; otherwise, the user account is locked in the target PDB. You have the following options regarding locked accounts:

- Close the PDB, connect to the root, and create a common user account with the same name. When the PDB is opened in read/write mode, differences in roles and privileges granted commonly to the user account are resolved, and you can unlock the account. Privileges and roles granted locally to the user account remain unchanged during this process.
- Create a new local user account in the PDB and use Data Pump to export/import the locked user's data into the new local user's schema.
- Leave the user account locked.
- Drop the user account.

See Also:

- "About Managing Tablespaces in a CDB"
- *Oracle Database Security Guide* for information about creating a local user
- *Oracle Database Utilities* for information about using Oracle Data Pump with a CDB

Cloning a Remote PDB: Examples

These examples clone a remote PDB or non-CDB given different factors.

In each example, the root to which the new PDB belongs depends on the current container when the `CREATE PLUGGABLE DATABASE` statement is run:

- When the current container is the CDB root, the new PDB is created in the CDB root.
- When the current container is an application root in an application container, the new PDB is created as an application PDB in the application root.

Cloning a Remote PDB Using No Clauses: Example

This example clones a remote source PDB named `pdb1` to a target PDB named `pdb2` given different factors.

This example assumes the following factors:

- The database link name to the remote PDB is `pdb1_link`.
- The `PATH_PREFIX` clause is not required.
- The `FILE_NAME_CONVERT` clause and the `CREATE_FILE_DEST` clause are not required.

Either Oracle Managed Files is enabled, or the PDB_FILE_NAME_CONVERT initialization parameter is set. The files will be copied to a new location based on the Oracle Managed Files configuration or the initialization parameter setting.

- Storage limits are not required for the PDB. Therefore, the STORAGE clause is not required.
- There is no file with the same name as the new temp file that will be created in the target location. Therefore, the TEMPFILE REUSE clause is not required.

The following statement clones the pdb2 PDB from the pdb1 remote PDB:

```
CREATE PLUGGABLE DATABASE pdb2 FROM pdb1@pdb1_link;
```

> **See Also:**
>
> - *Oracle Database Administrator's Guide* for more information about Oracle Managed Files
> - *Oracle Database Reference* for information about the PDB_FILE_NAME_CONVERT initialization parameter

Cloning a Remote PDB Using DBCA: Example

This example uses DBCA to clone a PDB named pdb1 from a remote CDB to the local CDB, where it will be renamed clonepdb1.

Prerequisites

This scenario assumes the following:

- The user in the local database has the CREATE PLUGGABLE DATABASE privilege in the root container.
- The remote CDB is in local undo mode.
- The remote and local CDBs are in ARCHIVELOG mode.
- The common user in the remote CDB to whom the database link connects has the CREATE PLUGGABLE DATABASE, SESSION, and SYSOPER privilege.
- The local and remote CDBs have the same options installed.

Assumptions

This scenario assumes the following:

- You are running DBCA on the host of the CDB that will contain the cloned PDB. The local CDB is named loccdb1.
- The remote (source) CDB is named remcdb1 and resides on host remcdb1host. The instance name for the remote CDB is reminst.
- The remote PDB, which is the PDB to be cloned, is named rempdb1.
- The common user c##adminuser_remcdb1 resides in remcdb1.

- The administrative user `locSYS` has `SYSDBA` privileges on `loccdb1`, which is the CDB to which the PDB is being cloned.
- The administrative user `remSYS` has `SYSDBA` privileges on `remcdb1`, which is the CDB that contains the PDB to be cloned.
- After cloning to `loccdb1`, the PDB will be renamed `clonepdb1`.

This following silent command clones `rempdb1` to `loccdb1`:

```
./dbca -silent
  -createPluggableDatabase
  -createFromRemotePDB
  -sourceDB remcdb1
  -remotePDBName rempdb1
  -remoteDBConnString remcdb1host:1521/reminst
  -remoteDBSYSDBAUserName remSYS
      -remoteDBSYSDBAUserPassword remsyspwd
  -dbLinkUsername c##adminuser_remcdb1
      -dbLinkUserPassword pwd4dblinkusr
  -sysDBAUserName locSYS
      -sysDBAPassword locsyspwd
  -pdbName clonepdb1
```

> **See Also:**
>
> *Oracle Database Administrator's Guide* for syntax and semantics of DBCA commands

Cloning a Non-CDB

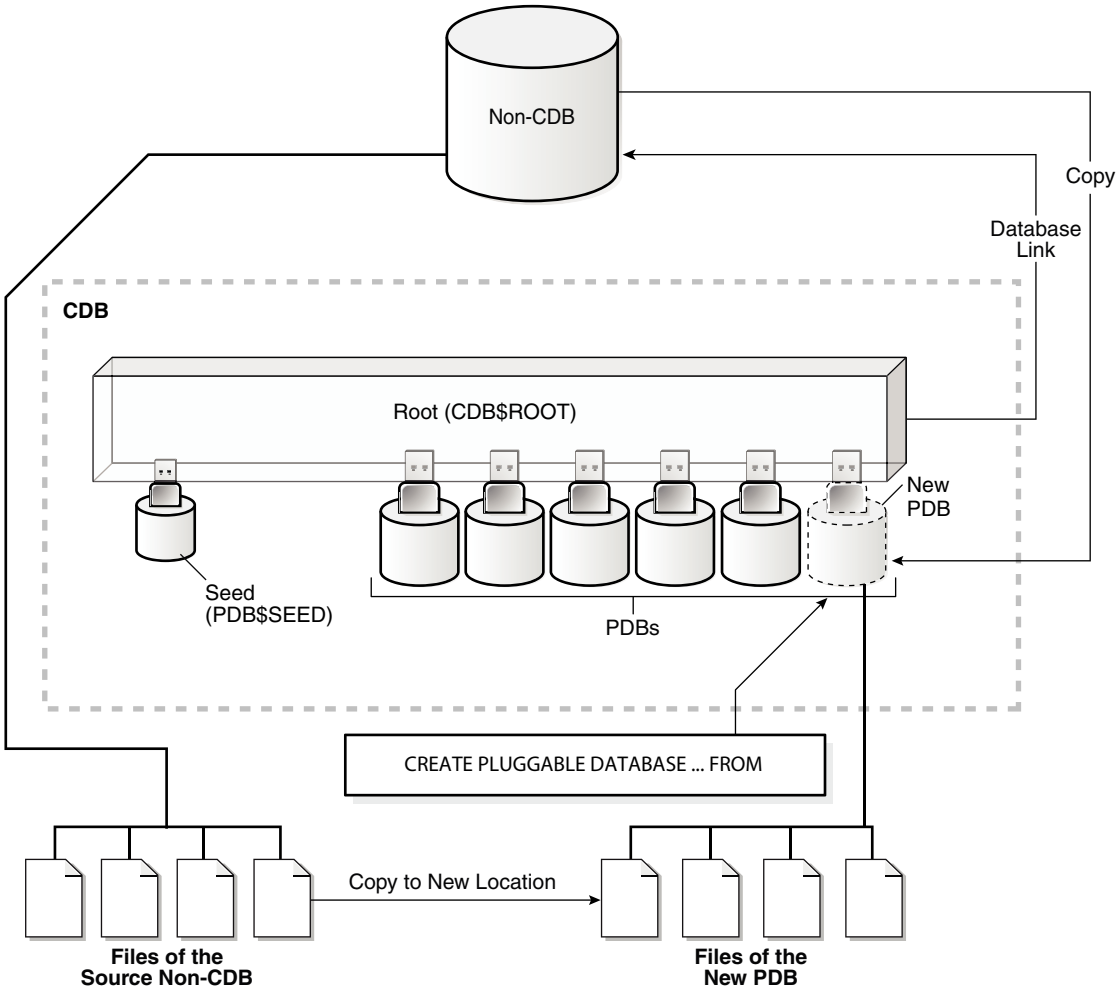
The procedure for cloning a non-CDB is very similar to the procedure for cloning a remote PDB.

About Cloning a Non-CDB

When the source is a non-CDB, you must specify a database link to the non-CDB in the `FROM` clause.

The following figure illustrates how this technique creates a new PDB when the source is a remote non-CDB.

Figure 7-4 Creating a PDB by Cloning a Non-CDB



Cloning a Non-CDB: Basic Steps

You can create a PDB by cloning a non-CDB.

General Prerequisites

The following prerequisites must be met:

- Complete the prerequisites described in "General Prerequisites for PDB Creation".

> **Note:**
>
> If you want to be able to recover the new PDB using backups of the source non-CDB, then you *must* use `DBMS_PDB.EXPORTRMANBACKUP` before cloning.

- The current user must have the `CREATE PLUGGABLE DATABASE` system privilege in the root of the CDB that will contain the target PDB.
- The source and target platforms must meet the following requirements:
  - They must have the same endianness.
  - The database options installed on the source platform must be the same as, or a subset of, the database options installed on the target platform.
- The CDB and the non-CDB must be running Oracle Database 12c Release 1 (12.1.0.2) or later.
- The CDB and the non-CDB must be running the same Oracle Database release.
- The data block size of the newly created PDB must match the CDB.
- If the non-CDB is in `NOARCHIVELOG` mode, then it must be open in read-only mode. If the non-CDB is in `ARCHIVELOG` mode, then it can be open read-only or read/write.

Prerequisites for Character Sets

- If the character set of the CDB is not AL32UTF8, then the source and target must have compatible character sets and national character sets. If the character set of the CDB to which the PDB is being cloned is AL32UTF8, then this requirement does not apply.
- If you are creating an application PDB, then the application PDB must have the same character set and national character set as the application container.

  If the database character set of the CDB is AL32UTF8, then the character set and national character set of the application container can different from the CDB. However, all application PDBs in an application container must have same character set and national character set, matching that of the application container.

> **Note:**
>
> Oracle Multitenant does not support a LOB in one container from being accessed by a container with a different character set using data links, extended data links, or the `CONTAINERS()` clause. For example, if the CDB root and `salespdb` have different character sets, then a `CONTAINERS()` query run in the CDB root should not access LOBs stored in `salespdb`.

Prerequisites for the Database Link

The following prerequisites must be met:

- A database link must enable a connection from the target CDB to the source CDB. The database link can connect to either the root of the CDB, to an application PDB if the source is an application PDB, or to the PDB.
- The user that the database link connects with must have the `CREATE PLUGGABLE DATABASE` system privilege.
- If the database link connects to the root in the CDB of the source PDB, then the user that the database link connects with must be a common user.
- In an Oracle Data Guard environment, if you are performing a remote clone of a PDB into a primary CDB, then on the standby CDB set

the STANDBY_PDB_SOURCE_FILE_DBLINK initialization parameter. This parameter specifies the name of the database link used in CREATE PLUGGABLE DATABASE ... FROM *dblink*. The standby CDB attempts to copy the data files from the source PDB referenced in the database link, but only if the source PDB is open in read-only mode. Otherwise, you must copy data files to the Oracle Managed Files location on the standby CDB.

To clone a remote non-CDB:

1. In SQL*Plus, ensure that the current container is the root of the target CDB or the application root of the target application container.
2. Run the CREATE PLUGGABLE DATABASE statement, and specify the source non-CDB in the FROM clause. Specify other clauses when required.

   After you create the PDB, it is in mounted mode, and its status is NEW. You can view the open mode of a PDB by querying the OPEN_MODE column in the V$PDBS view. You can view the status of a PDB by querying the STATUS column of the CDB_PDBS or DBA_PDBS view.

   A new default service is created for the PDB. The service has the same name as the PDB and can be used to access the PDB. Oracle Net Services must be configured properly for clients to access this service.

   > **Note:**
   >
   > If an error is returned during PDB creation, then the PDB being created might be in an UNUSABLE state. You can check the PDB state by querying the CDB_PDBS or DBA_PDBS view, and you can learn more about PDB creation errors by checking the alert log. An unusable PDB can only be dropped, and it must be dropped before a PDB with the same name as the unusable PDB can be created.

3. Run the *ORACLE_HOME*/rdbms/admin/noncdb_to_pdb.sql script. This script must be run before the PDB can be opened for the first time.

   To run the noncdb_to_pdb.sql script, complete the following steps:

   a. Set the container to the newly created PDB.

      The current user must have SYSDBA administrative privilege, and the privilege must be either commonly granted or locally granted in the PDB. The user must exercise the privilege using AS SYSDBA at connect time.

   b. Run the noncdb_to_pdb.sql script:

      ```
      @$ORACLE_HOME/rdbms/admin/noncdb_to_pdb.sql
      ```

   The script opens the PDB, performs changes, and then closes the PDB.

4. Open the new PDB in read/write mode.

   You must open the new PDB in read/write mode for Oracle Database to complete the integration of the new PDB into the CDB. An error is returned if you attempt to open the PDB in read-only mode. After the PDB is opened in read/write mode, its status is NORMAL.

5. Back up the PDB.

A PDB cannot be recovered unless it is backed up.

See Also:

- My Oracle Support Note 1928653.1 for a detailed example of cloning a PDB from a non-CDB
- "Refreshing a PDB"
- "Modifying the Open Mode of PDBs"
- *Oracle Database Backup and Recovery User's Guide* for information about backing up a PDB
- *Oracle Database Globalization Support Guide* to learn about the requirements for the compatibility of character sets
- *Oracle Database Reference* for information about the `PDB_FILE_NAME_CONVERT` initialization parameter

Cloning a Remote Non-CDB: Example

This example creates a new PDB by cloning a remote source non-CDB named `mydb` to a target PDB named `pdb2` given different factors.

This example assumes the following factors:

- The database link name to the remote non-CDB is `mydb_link`.
- The `PATH_PREFIX` clause is not required.
- The `FILE_NAME_CONVERT` clause and the `CREATE_FILE_DEST` clause are not required.

  Either Oracle Managed Files is enabled, or the `PDB_FILE_NAME_CONVERT` initialization parameter is set. The files will be copied to a new location based on the Oracle Managed Files configuration or the initialization parameter setting.
- Storage limits are not required for the PDB. Therefore, the `STORAGE` clause is not required.
- There is no file with the same name as the new temp file that will be created in the target location. Therefore, the `TEMPFILE REUSE` clause is not required.

The following statement creates the `pdb2` PDB from the remote non-CDB named `mydb`:

```
CREATE PLUGGABLE DATABASE pdb2 FROM mydb@mydb_link;
```

When the source database is a non-CDB, you can substitute `NON$CDB` for the name of the non-CDB. For example, the following statement is equivalent to the previous example:

```
CREATE PLUGGABLE DATABASE pdb2 FROM NON$CDB@mydb_link;
```

> **See Also:**
>
> *Oracle Database Administrator's Guide* for more information about Oracle Managed Files

About Refreshable Clone PDBs

The `CREATE PLUGGABLE DATABASE ... REFRESH MODE` statement clones a source PDB and configures the clone to be refreshable. Refreshing the clone PDB updates it with redo accumulated since the last redo log apply.

Purpose of Refreshable Clone PDBs

The cloning operation for production PDBs can take significant time.

If PDBs are cloned infrequently to avoid a drag on the system, then the cloned data becomes stale. A refreshable clone PDB solves this problem. When a refreshable clone PDB is stale, you can close it and then refresh it with recent redo. When not being refreshed, a refreshable clone PDB can be open read-only. A typical practice is to maintain a "golden master" refreshable clone of a production PDB, take PDB-level snapshots, and then create clones from the PDB snapshots for development and testing.

You can reverse the roles for source and clone PDBs using an `ALTER PLUGGABLE DATABASE ... SWITCHOVER` statement. This capability is useful in the following situations:

- Planned switchover

  The CDB hosting the source PDB may experience significantly more overhead than the CDB hosting the clone PDB. To achieve load balancing, you can reverse the roles, making the clone the new source PDB, and the source PDB the new clone.

- Unplanned switchover

  The source PDB may suffer an unplanned failure. In this case, you can make the clone PDB the new source PDB, and resume normal operations.

> **See Also:**
>
> - "Managing Refreshable Clone PDBs"
> - *Oracle Database SQL Language Reference* to learn more about `ALTER PLUGGABLE DATABASE ... SWITCHOVER`

Automatic and Manual Refresh Modes

You can configure the clone PDB to refresh automatically at set intervals, or you can refresh it manually with the `ALTER PLUGGABLE DATABASE REFRESH` statement.

The `REFRESH MODE` clause is supported only in a `CREATE PLUGGABLE DATABASE ... FROM` statement. You can use this clause to specify one of the following options:

- Specify `REFRESH MODE NONE`, the default, to create a PDB that is not refreshable.

  You can change a refreshable clone PDB into an ordinary PDB by including the `REFRESH MODE NONE` clause in an `ALTER PLUGGABLE DATABASE` statement and then opening the PDB in read/write mode. You cannot change an ordinary PDB into a refreshable clone PDB. After a refreshable clone PDB is converted to an ordinary PDB, you cannot change it back into a refreshable clone PDB.
- Specify `REFRESH MODE MANUAL` to create a refreshable PDB that must be refreshed manually.
- Specify `REFRESH MODE EVERY` *`number_of_minutes`* `MINUTES` to create a refreshable PDB that is refreshed automatically after the specified number of minutes has passed. A refreshable PDB that uses automatic refresh can also be refreshed manually.

> **Note:**
>
> - When you create a refreshable PDB, you can set the `REMOTE_RECOVERY_FILE_DEST` initialization parameter in the PDB. This initialization parameter specifies a directory from which to read archive log files during refresh operations if the source PDB is not available over its database link.
> - If new data files are created in the source PDB, then the `PDB_FILE_NAME_CONVERT` initialization parameter must be set in the CDB to convert the data file paths from the source PDB to the clone PDB.

Example 7-1 A REFRESH MODE Clause That Specifies Automatic Refresh

This refresh mode clause specifies that a refreshable PDB is refreshed automatically every two hours (120 minutes):

```
REFRESH MODE EVERY 120 MINUTES
```

> **See Also:**
>
> - "Cloning a Remote PDB: Basic Steps"
> - "Refreshing a PDB"

Requirements for Refreshable Clone PDBs

Creation of a refreshable clone PDB requires a database link. The database link can point to the same CDB or a different CDB.

A refreshable clone PDB must be in either of the following states:

- Closed

  A refreshable PDB must be closed when a refresh is performed. If it is not closed when automatic refresh is attempted, then the refresh is deferred until the next scheduled refresh. If it is not closed when a user attempts to perform manual refresh, then an error is reported.

- Open in read-only mode

  The refreshable PDB must be kept in read-only mode to prevent out-of-sync changes on the refreshable PDB which do not occur on the source PDB. The refreshable PDB is intended to serve as a clone master and as such must accurately reflect the source PDB at the refreshed point in time.

Creating a Refreshable Clone PDB: Scenario

This scenario creates a refreshable clone named `pdb1_ref_cln` from a remote PDB named `pdb1`.

The clone PDB is a copy of the source PDB. You can refresh the clone PDB periodically to update it with any changes made to the source PDB.

Assumptions

This scenario assumes the following factors:

- The database link name to the remote PDB is `pdb1_link`.
- The `PATH_PREFIX` clause is not required.
- The `FILE_NAME_CONVERT` clause and the `CREATE_FILE_DEST` clause are not required.

  Either Oracle Managed Files is enabled, or the `PDB_FILE_NAME_CONVERT` initialization parameter is set. The files will be copied to a new location based on the Oracle Managed Files configuration or the initialization parameter setting.
- Storage limits are not required for the PDB. Therefore, the `STORAGE` clause is not required.
- There is no file with the same name as the new temp file that will be created in the target location. Therefore, the `TEMPFILE REUSE` clause is not required.
- The refreshable clone will be refreshed automatically every 60 minutes.

> **Note:**
>
> To create a refreshable PDB, the source PDB must be in `ARCHIVELOG` mode and local undo mode.

To create a refreshable clone PDB:

1. In SQL*Plus, ensure that the current container is the CDB root or an application root.

   When the current container is the CDB root, the PDB is created in the CDB. When the current container is an application root, the application PDB is created in the application container.

2. Execute the `CREATE PLUGGABLE DATABASE` statement.

   The following statement creates `pdb1_ref_cln` from `pdb1`:

   ```
   CREATE PLUGGABLE DATABASE pdb1_ref_cln FROM pdb1@pdb1_link REFRESH
   MODE EVERY 60 MINUTES;
   ```

> **See Also:**
>
> "Managing Refreshable Clone PDBs"

Cloning PDBs from PDB Snapshots

You can create PDBs from PDB snapshots by executing the `CREATE PLUGGABLE DATABASE … USING SNAPSHOT` statement.

About Cloning PDBs from PDB Snapshots

A **PDB snapshot** is a point-in-time copy of a PDB. The source PDB can be open read-only or read/write while the snapshot is created. A clone from a PDB snapshot is a full, standalone PDB.

Cloning a PDB from a PDB Snapshot: Scenario

This scenario creates a new PDB from a PDB snapshot by executing `CREATE PLUGGABLE DATABASE ... USING SNAPSHOT`.

Assumptions

This example assumes the following factors:

- A PDB snapshot carousel exists with 8 daily snapshots of source PDB `salespdb`, named after the weekday, day of the month, and time when they were created: `pdb1_mon_2_1201`, `pdb1_tue_3_1201`, `pdb1_wed_4_1201`, and so on.
- All snapshots were created when the source `salespdb` was in read/write mode.
- The new PDB will be a clone of a snapshot named `pdb1_wed_4_1201`, which is a snapshot of `pdb1` taken last Wednesday on the 4th of the month at 12:01 a.m.
- The `PATH_PREFIX` clause is not required.
- The `FILE_NAME_CONVERT` clause and the `CREATE_FILE_DEST` clause are not required.

  Either Oracle Managed Files is enabled, or the `PDB_FILE_NAME_CONVERT` initialization parameter is set. Therefore, the `FILE_NAME_CONVERT` clause is not required. The files will be copied to a new location based on the Oracle Managed Files configuration or the initialization parameter setting.
- Storage limits are not required for the PDB. Therefore, the `STORAGE` clause is not required.

- There is no file with the same name as the new temp file that will be created in the target location. Therefore, the `TEMPFILE REUSE` clause is not required.

To clone a PDB from a PDB snapshot:

1. In SQL*Plus, ensure that the current container is the CDB root or an application root.

   When the current container is the CDB root, the PDB is created in the CDB. When the current container is an application root, the application PDB is created in the application container.

2. Execute the `CREATE PLUGGABLE DATABASE ... USING SNAPSHOT` statement.

   The following statement clones the `pdb1_copy` PDB from the PDB snapshot named `pdb1_wed_4_1201`:

```
CREATE PLUGGABLE DATABASE pdb1_copy FROM pdb1
  USING SNAPSHOT pdb1_wed_4_1201;
```

See Also:

- "Configuring Automatic PDB Snapshots"
- *Oracle Database Licensing Information User Manual* for details on which features are supported for different editions and services

Creating and Materializing Snapshot Copy PDBs

You can clone a PDB from snapshots of the underlying storage. The PDB files are sparse, but you can materialize the files to create a standalone PDB.

About Snapshot Copy PDBs

You can create a **snapshot copy PDB** by executing a `CREATE PLUGGABLE DATABASE ... FROM ... SNAPSHOT COPY` statement. The source PDB is specified in the `FROM` clause.

A snapshot copy reduces the time required to create the clone because it does not include a complete copy of the source data files. Furthermore, the snapshot copy PDB occupies a fraction of the space of the source PDB.

Storage clones are named and tagged using the GUID of the target PDB. To view clone tags for storage clones, query the `DBA_PDB_HISTORY.CLONETAG` column.

Storage Requirements for Snapshot Copy PDBs

If you use `CREATE PLUGGABLE DATABASE ... FROM` *`srcpdb`* `... SNAPSHOT COPY`, then the source PDB data files must reside in the same storage type.

The behavior of the `CREATE PLUGGABLE DATABASE ... FROM ... SNAPSHOT COPY` command depends on the following rules:

1. If the file system supports storage-managed snapshots, then the snapshot copy PDB is based on a storage-level copy of the underlying file system. The snapshot copy PDB contains sparse files. The copy-on-write technology means that only modified blocks require additional storage on disk.
2. If the file system does *not* support storage snapshots, then the algorithm is as follows:
   - If the storage system uses Oracle Exadata sparse disk groups, then Oracle Database creates a snapshot copy PDB. However, the source PDB must remain read/only for the lifetime of the snapshot copy PDB.
   - If the storage system does *not* use Oracle Exadata sparse disk groups, then the behavior is as follows:
     – If `CLONEDB=true`, then the underlying file system for the source PDB files can be any local file system, network file system (NFS), or a clustered file system such as Oracle ACFS. If using a network file system, Direct NFS should be enabled for the CDB. The file system should support sparse files. Most UNIX systems meet these requirements.

       When `CLONEDB=true`, the open mode of the source PDB has the following effects:

       * If the source PDB is open in read-only mode, then Oracle Database creates a snapshot copy PDB using copy-on-write technology. The snapshot copy PDB contains sparse files, not full copies.
       * If the source PDB is *not* open in read-write mode, then Oracle Database issues an error.

     – If `CLONEDB=false`, then Oracle Database issues an error.

Direct NFS Client enables an Oracle database to access network attached storage (NAS) devices directly, rather than using the operating system kernel NFS client. If the files of the source PDB are stored on Direct NFS Client storage, then the following additional requirements must be met:

- The source PDB files must be located on an NFS volume.
- Storage credentials must be stored in a Transparent Data Encryption keystore.
- The storage user must have the privileges required to create and destroy snapshots on the volume that hosts the files of the source PDB.
- Credentials must be stored in the keystore using an `ADMINISTER KEY MANAGEMENT ADD SECRET` SQL statement.

  The following example configures an Oracle Database secret in a software keystore:

```
ADMINISTER KEY MANAGEMENT
  ADD SECRET 'secret' FOR CLIENT 'client_name'
  USING TAG 'storage_user'
  IDENTIFIED BY  keystore_password  WITH BACKUP;
```

  Run this statement to add a separate entry for each storage server in the configuration. In the previous example, the following values must be specified:

  – *secret* is the storage password.

- *client_name* is the storage server. On a Linux or UNIX platform, it is the name entered in /etc/hosts or the IP address of the storage server.
- *tag* is the user name passed to the storage server.
- *keystore_password* is the password for the keystore.

> **Note:**
>
> Snapshot copy behavior and efficiency are vendor specific and may vary between vendors.

> **See Also:**
>
> - *Oracle Automatic Storage Management Cluster File System Administrator's Guide* for more information about Oracle ACFS
> - *Oracle Grid Infrastructure Installation and Upgrade Guide* for your operating system for information about Direct NFS Client
> - *Oracle Database Advanced Security Guide* for more information about Transparent Data Encryption
> - My Oracle Support Note 1597027.1 for more information about supported platforms for snapshot cloning of PDBs
> - *Oracle Exadata System Software User's Guide* for information about Exadata support for PDB clones created using the `SNAPSHOT COPY` clause

Restrictions for Snapshot Copy PDBs

You cannot drop the storage snapshot on which a snapshot copy PDB is based.

You cannot unplug snapshot copy PDBs from the CDB root or application container. Attempting to unplug a snapshot copy PDB results in an error. However, you can materialize the snapshot copy PDB, which turns it into a standalone PDB, and then drop it.

> **Note:**
>
> A PDB created with the `USING SNAPSHOT` clause and a PDB created with the `SNAPSHOT COPY` clause have different properties. You cannot specify both clauses in a single `CREATE PLUGGABLE DATABASE` command. The `CREATE PLUGGABLE DATABASE … FROM … USING SNAPSHOT` clause creates a full, standalone PDB that does not need to be materialized. The `CREATE PLUGGABLE DATABASE … FROM … SNAPSHOT COPY` clause creates a sparse PDB that must be materialized if you want to drop the storage-level snapshot on which it is based.

Creating a Snapshot Copy PDB: Scenario

This scenario create a snapshot copy PDB by specify the `SNAPSHOT COPY` clause in `CREATE PLUGGABLE DATABASE`.

Assumptions

This scenario assumes the following factors:

- The new snapshot copy PDB will be created from a PDB named `pdb1`.
- The underlying file system supports storage snapshots. Thus, you do not need to set the `CLONEDB` initialization parameter.
- The `PATH_PREFIX` clause is not required.
- The `FILE_NAME_CONVERT` clause and the `CREATE_FILE_DEST` clause are not required.

  Either Oracle Managed Files is enabled, or the `PDB_FILE_NAME_CONVERT` initialization parameter is set. Therefore, the `FILE_NAME_CONVERT` clause is not required. The files will be copied to a new location based on the Oracle Managed Files configuration or the initialization parameter setting.
- Storage limits are not required for the PDB. Therefore, the `STORAGE` clause is not required.
- There is no file with the same name as the new temp file that will be created in the target location. Therefore, the `TEMPFILE REUSE` clause is not required.

To create a snapshot copy PDB:

1. In SQL*Plus, ensure that the current container is the CDB root or an application root.

   When the current container is the CDB root, the PDB is created in the CDB. When the current container is an application root, the application PDB is created in the application container.
2. Execute the `CREATE PLUGABBLE DATABASE … SNAPSHOT COPY` statement.

   The following statement clones the `pdb1_snap_copy` PDB from `pdb1`:

   ```
   CREATE PLUGGABLE DATABASE pdb1_snap_copy FROM pdb1 SNAPSHOT COPY;
   ```

   As long as `pdb1_snap_copy` exists, you cannot drop the storage snapshot on which `pdb1_snap_copy` is based.

> **See Also:**
>
> "Materializing a Snapshot Copy PDB"

Materializing a Snapshot Copy PDB

You can materialize a snapshot copy PDB by running an `ALTER PLUGGABLE DATABASE` statement with the `MATERIALIZE` clause. Materializing a snapshot copy PDB copies all data blocks.

Materializing a snapshot copy PDB transforms the snapshot copy PDB, which uses sparse files, into a full PDB, which does *not* use sparse files. The materialized PDB is no longer dependent on the source PDB, which can be dropped or changed to a different open mode.

For example, if `pd1_snap_copy` is a snapshot copy PDB, then you can materialize it into a standalone PDB by running an `ALTER PLUGGABLE DATABASE MATERIALIZE` command. After materialization, `pdb1_snap_copy` no longer depends on the storage-level snapshot, enabling you to drop it.

To materialize a PDB snapshot:

1. In SQL*Plus, ensure that the current container is the snapshot copy PDB that is being materialized.
2. Run an `ALTER PLUGGABLE DATABASE` statement with the `MATERIALIZE` clause.

Example 7-2 Materializing a Snapshot Copy PDB

The following SQL statement materializes a snapshot copy PDB:

```
ALTER PLUGGABLE DATABASE MATERIALIZE;
```

See Also:

- "About Snapshot Copy PDBs" to learn more about snapshot copy PDBs
- "Creating a Snapshot Copy PDB: Scenario"
- My Oracle Support Note 2627975.1 to learn how to revert the source PDB data file permissions after removing all snapshot clone PDBs

Creating a Split Mirror Clone PDB

In Oracle ASM, a split mirror is the process of detaching a point-in-time media copy from a parent copy. After the split, updates to the parent do not affect the child copy.

Starting in Oracle Database 18c, the parent copy can be a PDB rather than a storage volume. The split mirror clone PDB resides on the same media as the parent. The principal use case is to rapidly provision test and development PDBs in an Oracle ASM environment.

> **Note:**
>
> Oracle ASM flex and extended disk groups are required for split mirror clone PDBs.

Mirror refresh is refreshing a split mirror clone PDB with changes from the parent PDB. In effect, this operation is equivalent to deleting the mirror split, and then taking a new mirror split.

To drop a split mirror clone PDB, enter `ALTER PLUGGABLE DATABASE ... DROP MIRROR COPY`.

To create a split mirror clone PDB:

1. Start SQL*Plus, and connect to the CDB root.
2. Prepare the source PDB by issuing the `ALTER PLUGGABLE DATABASE ... PREPARE MIRROR COPY` statement.
3. Create a clone PDB from the source PDB by issuing the `CREATE PLUGGABLE DATABASE ... FROM ... USING MIRROR COPY` statement.
4. Optionally, query `V$ASM_DBCLONE_INFO` view to see the relationship between the source PDB, the cloned PDB, and their file groups.

> **See Also:**
>
> - *Oracle Automatic Storage Management Administrator's Guide* to learn how to create or drop a split mirror clone PDB
> - *Oracle Database Reference* to learn more about `V$ASM_DBCLONE_INFO`

8
Relocating a PDB

You can move a PDB to a different CDB or application container.

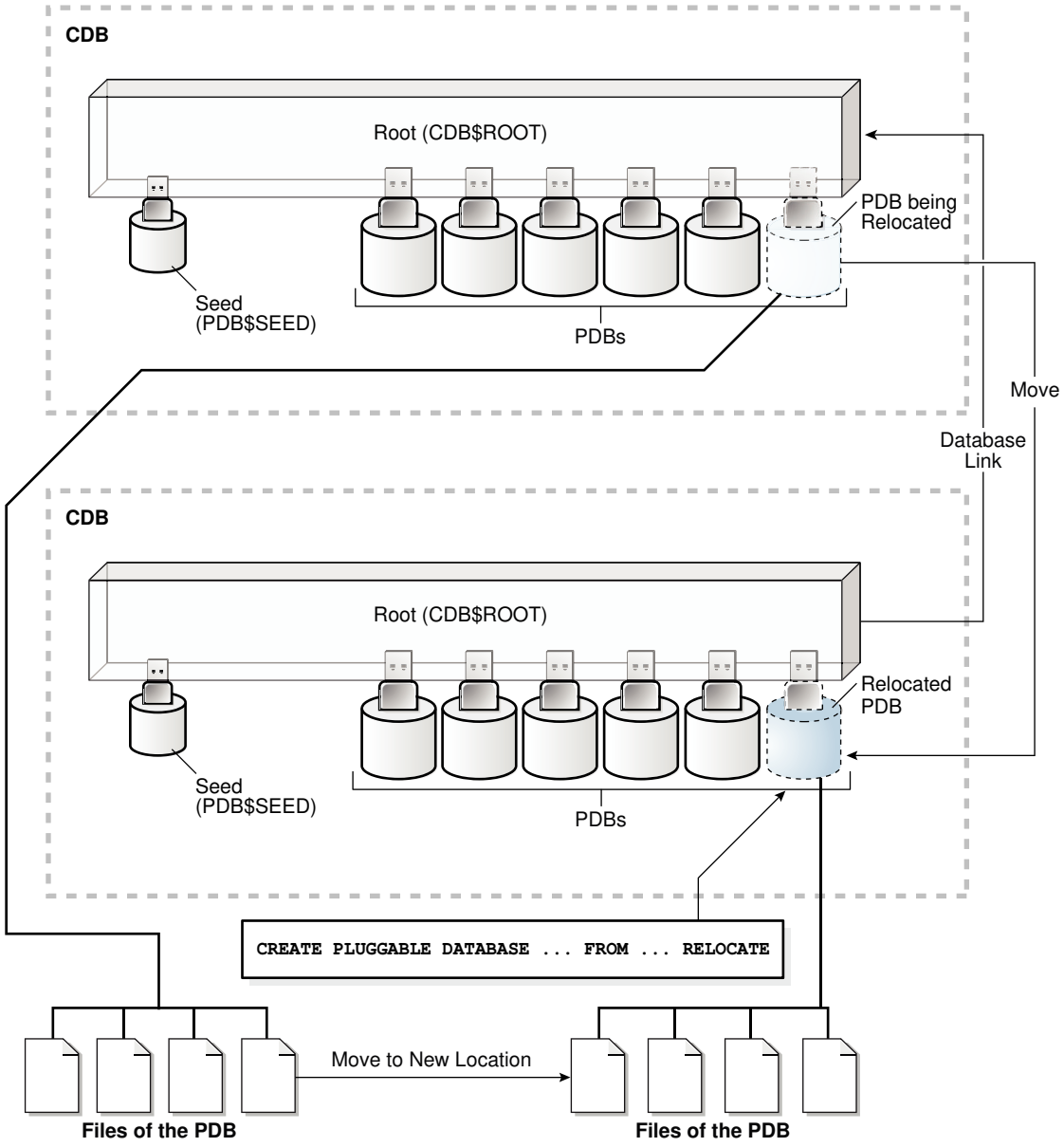
About PDB Relocation

During relocation, the source PDB can be open in read/write mode and fully functional.

PDB relocation executes an online block level copy of the source PDB data files, redo, and undo while the source PDB is open with active sessions. When the target PDB comes online because of an `ALTER PLUGGABLE DATABASE OPEN` statement, Oracle Database terminates the active sessions and closes the source PDB.

The following graphic shows the relocation of a common PDB (that is, not an application PDB) to a new single-instance CDB. The source PDB is plugged in to the CDB root, and the target PDB is plugged in to the CDB root. Note that the `CREATE PLUGGABLE DATABASE ... RELOCATE` statement copies the data blocks, undo blocks, and redo blocks to the new location. A database link is required.

Figure 8-1 Relocate a PDB into the Root Container

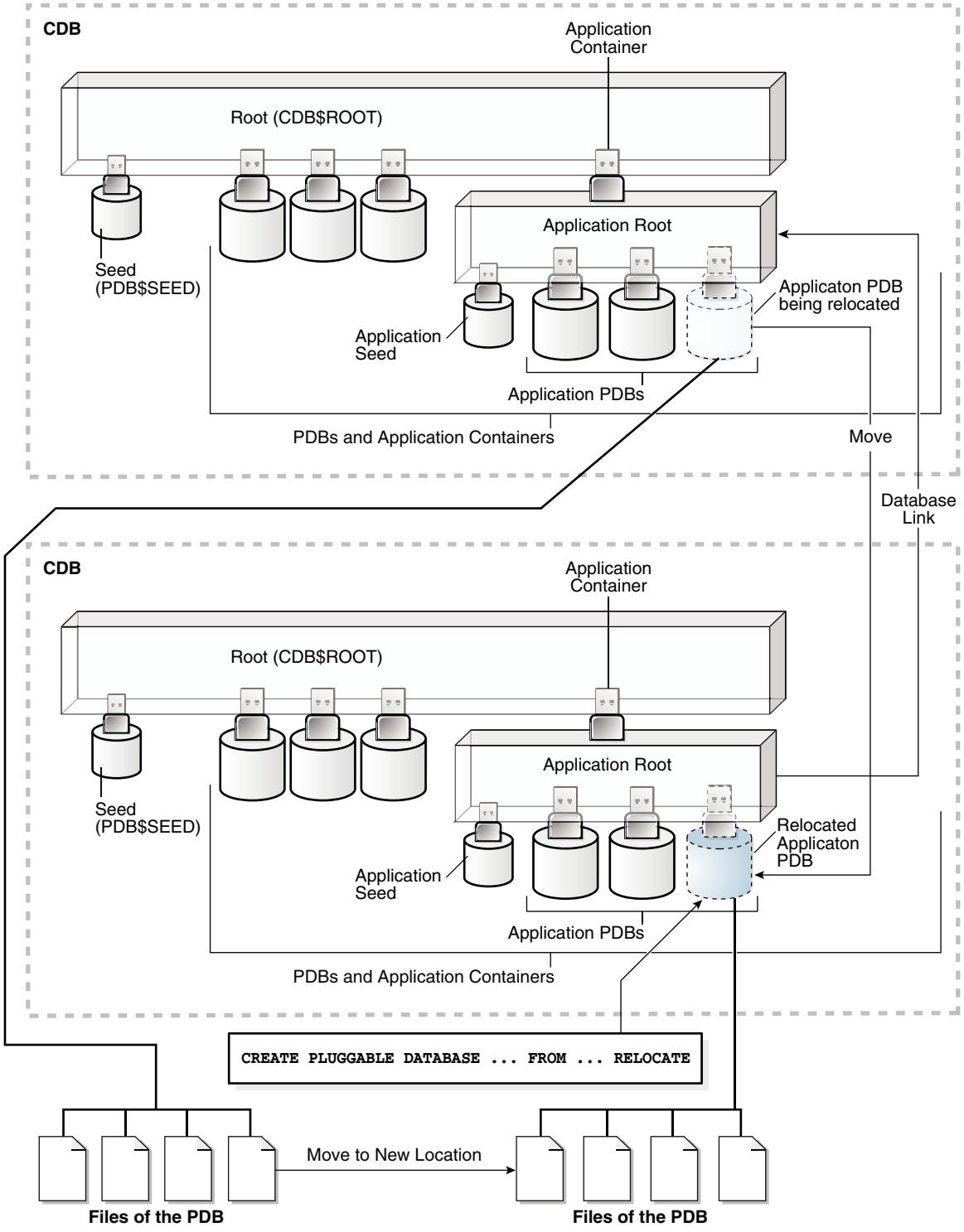


When the target PDB is an application PDB or application root, you have the following options:

- You can relocate a PDB into an application container as an application PDB. The target PDB can be in the same CDB or a different CDB.
- You can relocate an application PDB from one application root to another. The target PDB must be in a different CDB.
- You can relocate an empty application root from one CDB to another, but the application root must not have any hosted application PDBs.

The following graphic illustrates how this technique creates a new application PDB in an application container.

Figure 8-2 Relocate a PDB into an Application Container



When you open the relocated PDB for the first time, Oracle Database drains active sessions on the source PDB and redirects client connections to the relocated PDB services. Opening the relocated PDB initiates the shutdown of the original source PDB. The source and relocated PDBs are never open at the same time.

> **See Also:**
>
> "PDB Storage"

Purpose of PDB Relocation

This technique is the fastest way to move a PDB with minimal or no down time. Otherwise, unplugging the source PDB requires a PDB outage until the PDB is plugged in to the target CDB.

When moving a PDB between data centers, or from an on-premises environment to a cloud environment, all the data must physically move. For large PDBs, this process may take considerable time, possibly violating availability components of an SLA. PDB relocation eliminates the outage completely. You can relocate the PDB without taking the application offline, changing the application, or changing network connection strings.

How PDB Relocation Works

The operation moves the files associated with the PDB to a new location, adds the PDB to the target CDB, and then opens the PDB.

Server Session Draining When Relocating or Stopping PDBs

A key requirement of planned maintenance is draining or failing over PDB sessions so that application work is not interrupted.

Automatic Session Failover

In database-generic session draining, active sessions can exit gracefully under a timer. After the timer has expired, Oracle Database terminates all active sessions, and then reconnects them to the relocated PDB.

Starting in Oracle Database 20c, during planned maintenance, the database may decide that a session is unlikely to drain in the drain window. In this case, the database invokes Application Continuity and fails over the session automatically. The draining feature is enabled by default for all maintenance operations invoked at the database service and PDB levels: stop service, relocate service, relocate PDB, and stop PDB.

> **Note:**
>
> If your application server user a Purge Pool property, then disable this property because it disrupts sessions that are not ready to drain.

Rules for Session Draining

The database uses an extensible set of rules to determine when to drain a database session, which persists until a rule is satisfied. The rules include the following:

- Standard application server tests for validity

- Custom SQL tests for validity
- Request boundaries are in use and no request is active
- Request boundaries are in use and the current request has ended
- The session has one or more session states that are recoverable, and can be recreated at failover

A typical use case is application servers and pooled applications that test connections when borrowing from connection pools, returning connections to the pool, and at batch commits. When draining sessions, the database automatically intercepts the connection test, closes the connection, and then returns a failed status for the test. After receiving the failed status, the application layer can request a different connection. In this way, the application is not disrupted.

Application Continuity with FAN on Oracle RAC

For an optimal configuration that minimizes the impact on the client, consider configuring Application Continuity with FAN on the Oracle RAC database. In Oracle Clusterware, the Fleet Patching and Provisioning feature automates PDB relocation. An example of finer-grained relocation in an Oracle RAC environment is service relocation between PDB instances. Oracle RAC and Oracle Clusterware offer a rich high availability environment that further minimizes the impact on connected clients during relocation. For example, shared storage may minimize or remove the necessity to copy data files. Transparent Application Continuity, a mode of Application Continuity, is enabled by default in Oracle Cloud.

> **Note:**
>
> In an Oracle Clusterware environment, when relocating a PDB between different CDBs, you must create non-database services using SRVCTL.

> **See Also:**
>
> *Oracle Clusterware Administration and Deployment Guide* to learn about Application Continuity, SRVCTL, and Fleet Patching and Provisioning

Stages of PDB Relocation

The details of PDB relocation vary depending on the listener networks.

PDB Relocation in a Common Listener Network

When the source and target location share a common listener network, forwarding client connections is not necessary because the SQL*Net layer forwards client connections implicitly.

AVAILABILITY NORMAL

When the listener network is common, specify the `AVAILABILITY NORMAL` clause in `CREATE PLUGGABLE DATABASE ... RELOCATE`. This option is the default. The following situations are typical use cases for `AVAILABILITY NORMAL`:

- Shared listener

  If you use the same listener for the PDB in its old and new locations, then new connections are automatically routed to the new location when relocation completes. This situation is typical of a relocation between CDBs in the same host. In this case, the PDB is re-registered with the listener in its new location. Additional connection handling is not required.

- Cross-registered listeners

  If the PDBs use different listeners, and if you employ cross-registration of their respective listeners through configuration of the `local_listener` and `remote_listener` parameters, then relocation is seamless. The availability and location of the PDB's services are automatically registered with both listeners. This situation is typical of relocation between hosts within a data center, perhaps for load balancing purposes.

In shared and cross registered listener environments, services from all databases are published to the common listener network. For this reason, services for relocated PDBs are immediately known to the common listener network. To avoid service name space collisions, PDB service definitions must be unique in the common listener network.

Stages of Relocation in a Common Listener Network

1. The user issues `CREATE PLUGGABLE DATABASE ... RELOCATE AVAILABILITY NORMAL`.

   This step executes a hot clone of the source PDB from its original location to its target location. The source PDB copies data files, undo blocks, and redo blocks to the target PDB as of an implicit begin SCN marker.

   When this step completes, two transactionally consistent copies of this PDB exist: one in the source container and one in the target container. For the duration of the operation, processing continues uninterrupted on the source PDB. Users of an application or applications connected to the source PDB are unaware that a relocation is underway.

   All existing application connections, and new connections created during this step, continue to connect to the source PDB.

2. The user issues `ALTER PLUGGABLE DATABASE OPEN`.

   The following actions occur in the background:

   a. The target PDB implicitly sets the end SCN marker, and applies any redo or undo required to complete media recovery to satisfy the implicit end SCN marker.

b. When media recovery occurs on the target PDB, Oracle Database initiates active session draining on the source PDB.

c. PDB services are registered with the listener and are available on the target CDB.

d. The source PDB is closed.

e. The target PDB opens in read/write mode.

   This step completes the relocation of the PDB to the target CDB. At the end of the operation, connections point to the newly relocated PDB.

   After the PDB is opened in read/write mode, its status is `NORMAL`. The database returns an error if you attempt to open the PDB in read-only mode.

See Also:

- *Oracle Database Net Services Administrator's Guide* for more information about listener redirects
- *Oracle Real Application Clusters Administration and Deployment Guide* to learn more about using Application Continuity to drain and migration sessions before planned maintenance

PDB Relocation in Isolated Listener Networks

When independent listeners do not use cross-registration, the listener in the target CDB and source CDB have no knowledge of each other or of their respective published services.

AVAILABILITY MAX

The `AVAILABILITY MAX` clause in `CREATE PLUGGABLE DATABASE ... RELOCATE` implicitly instructs the SQL*Net layer to reconfigure the original listener. This situation may be common when relocating a PDB between data centers. This configuration is intended to be temporary while the Oracle Internet Directory (OID) or LDAP server is updated or the client connections are modified.

If a local listener redirects to a Single Client Access Name (SCAN) listener in an Oracle RAC configuration, then this listener may need to further redirect the client connection request to another cluster node. Multiple redirects are not supported by Oracle Net listeners by default. Because any SCAN listener can route the connection request to any node, set the `ALLOW_MULTIPLE_REDIRECTS_`*`listener_name`* parameter to the *`listener_name`* of every SCAN listener, and set it in every `listener.ora` file in the cluster. For example, if the SCAN listeners are named `listener_scan1`, `listener_scan2`, and `listener_scan3`, then the `listener.ora` file on every destination host should have the following settings:

```
ALLOW_MULTIPLE_REDIRECTS_LISTENER_SCAN1=YES
ALLOW_MULTIPLE_REDIRECTS_LISTENER_SCAN2=YES
ALLOW_MULTIPLE_REDIRECTS_LISTENER_SCAN3=YES
```

> **Caution:**
>
> Do not set the `ALLOW_MULTIPLE_REDIRECTS_listener_name` parameter for node listeners because it may allow infinite redirection loops in certain network configurations.

Stages of Relocation in an Isolated Listener Network

1. The user issues `CREATE PLUGGABLE DATABASE ... RELOCATE AVAILABILITY MAX`.

   This step executes a hot clone of the source PDB from its original location to its target location. The source PDB copies data files, undo blocks, and redo blocks to the target PDB as of an implicit begin SCN marker.

2. The user issues `ALTER PLUGGABLE DATABASE OPEN`.

   The following actions occur in the background:

   a. The target PDB implicitly sets the end SCN marker, and applies any redo or undo required to complete media recovery to satisfy the implicit end SCN marker.

   b. When media recovery occurs on the target PDB, Oracle Database initiates active session draining on the source PDB.

   c. The `LISTENER_NETWORKS` initialization parameter is implicitly updated in the source PDB with the forwarding address, and the listener PDB services for the source CDB are updated with the forwarding address.

   d. The target PDB opens in read-only mode while media recovery completes.

      At this stage, only queries of the target PDB are permitted. Queries behave exactly as if they had been run on the source PDB. However, connections attempting DML do not complete.

   e. Read-only connections are immediately forwarded to the new hosting listener, and new read/write connections are forwarded to the new hosting listener, where they spin until the target PDB is opened in a consistent state.

   f. The source PDB executes a `SHUTDOWN IMMEDIATE`, terminating persistent connections.

   g. The target PDB opens in read/write mode.

      This step completes the relocation of the PDB to the target CDB. At the end of the operation, connections point to the newly relocated PDB.

      After the PDB is opened in read/write mode, its status is `NORMAL`. The database returns an error if you attempt to open the PDB in read-only mode.

> **Note:**
>
> An artifact known as a *tombstone PDB* remains in the source CDB to protect the PDB's namespace and preserve the listener forwarding configuration until the updates are complete. In the root of the source CDB, the tombstone PDB is visible in `V$CONTAINERS` with a status of `RELOCATED`. When you change the application connect strings to provide direct connections to the target PDB, you can drop the tombstone PDB from the source CDB.

> **See Also:**
>
> - "Creating an Application PDB"
> - *Oracle Database Net Services Administrator's Guide* for more information about listener redirects
> - *Oracle Real Application Clusters Administration and Deployment Guide* to learn more about using Application Continuity to drain and migration sessions before planned maintenance

User Interface for PDB Relocation

You can relocate PDBs on the command line using SQL, the DBCA utility, or the Fleet Patching and Provisioning utility.

SQL Statement

The form of the SQL statement is as follows:

```
CREATE PLUGGABLE DATABASE ... FROM src_pdb_name@link2src ... RELOCATE
AVAILABILITY [MAX | NORMAL]
```

The `FROM` clause identifies the location of the source PDB. For *`src_pdb_name`*, specify the name of the source PDB. For `link2src`, specify a database link that indicates the location of the source PDB. The database link must have been created in the target CDB, which is the CDB to which the PDB will be relocated. The link can connect either to the root of the remote CDB or to the remote PDB.

The `AVAILABILITY` clause determines how the database handles client connections.

DBCA

You can relocate a PDB by running DBCA in silent mode. The `relocatePDB` command performs the relocation.

Table 8-1 relocatePDB Parameters

Parameter	Description
`-remotePDBName` *remote_pdb_name*	The name of the PDB that you intend to relocate.
`-remoteDBConnString` *remote_db_conn_string*	The net service connection to the remote CDB.
`-sysDBAUserName` *sysdbusername*	The name of the SYS user in the local CDB.
`-sysDBAPassword` *sysdbapassowrd*	The password of the SYS user in the local CDB.
`-remoteDBSYSDBAUserName` *sysdbusername*	The name of the SYS user in the remote CDB.
`-remoteDBSYSDBAPassword` *sysdbapassowrd*	The password of the SYS user in the remote CDB.
`-dbLinkUsername` *dblink_common_user_name*	The name of the common user in the remote CDB.
`-dbLinkUserPassword` *dblink_common_username_pwd*	The password of the common user in the remote CDB.
`-sourceDB` *dbname_pdb_toberelocated*	The name of the source PDB.
`-pdbName` *pdbtoberecreated*	The name of the PDB after relocation.

Fleet Patching and Provisioning Control (RHPCTL)

In Oracle Grid Infrastructure, you can use Fleet Patching and Provisioning to automate relocation of a PDB from one CDB to another.

See Also:

- *Oracle Database SQL Language Reference* for `CREATE PLUGGABLE DATABASE` syntax and semantics
- *Oracle Database Administrator's Guide* for the DBCA command reference for silent mode
- *Oracle Clusterware Administration and Deployment Guide* to learn more about Fleet Patching and Provisioning

Relocating a PDB Using CREATE PLUGGABLE DATABASE

The `CREATE PLUGGABLE DATABASE ... RELOCATE` statement moves a PDB to a different container.

The *target CDB* (also called the *destination CDB*) is the CDB to which the PDB is being relocated. The *target PDB* is the PDB being relocated. After the `CREATE PLUGGABLE DATABASE ... RELOCATE` operation completes, Oracle Database moves the PDB from the source CDB to the destination CDB.

General Prerequisites

Address the questions that apply to relocating a PDB in "Table 5-3". The table describes which `CREATE PLUGGABLE DATABASE` clauses you must specify based on different factors. Also, complete the prerequisites described in "General Prerequisites for PDB Creation".

Database Mode and State Prerequisites

You must meet the following prerequisites:

- The source CDB must be in local undo mode.
- In the source CDB, you must save the service and open state of the PDBs in all database instances. Log in to the CDB root as an administrator and issue the following statement:

  ```
  ALTER PLUGGABLE DATABASE ALL SAVE STATE INSTANCES=ALL;
  ```

  This step ensures that the PDB relocation operation automatically starts the PDB services in the target CDB.
- If the target CDB is not in `ARCHIVELOG` mode, then the target PDB must be opened read-only during the operation. This requirement does not apply if the target CDB is in `ARCHIVELOG` mode.

User Privilege Prerequisites

You must meet the following prerequisites:

- In the target CDB, the current user must have the `CREATE PLUGGABLE DATABASE` system privilege in the CDB root.
- The following prerequisites apply to the database link:
  - A database link must enable a connection from the destination CDB to the source CDB.
  - If the target is a standard PDB, then the database link must connect to the root of the source CDB. If the target PDB is an application PDB, then the database link must connect to its application root.
  - If the database link user connects to the CDB root in the source CDB, then this user must be a common user. If the database link connects to the application root, then this user can be either a CDB-wide common user or an application common user.
  - The database link user must have either the `CREATE PLUGGABLE DATABASE` system privilege or the `SYSOPER` administrative privilege.

Platform and Character Set Prerequisites

You must meet the following prerequisites:

- The platforms of the source CDB and the destination CDB must meet the following requirements:
  - They must have the same endianness.

  - The database options installed on the source platform must be the same as, or a subset of, the database options installed on the destination platform.

- If the character set of the destination CDB is not AL32UTF8, then the source CDB and destination CDB must have compatible character sets and national character sets.

  If the character set of the destination CDB is AL32UTF8, then this requirement does not apply.

> **Note:**
>
> Oracle Multitenant does not support a LOB in one container from being accessed by a container with a different character set using data links, extended data links, or the `CONTAINERS()` clause. For example, if the CDB root and `salespdb` have different character sets, then a `CONTAINERS()` query run in the CDB root should not access LOBs stored in `salespdb`.

Application Name and Version Prerequisites

If you are creating an application PDB, then the source PDB and target application container must have the same application name and version.

To relocate a PDB:

1. In SQL*Plus, log in to the target CDB as a user with the `CREATE PLUGGABLE DATABASE` system privilege.
2. Ensure that the current container is the root of the target CDB or target application container.
3. Run the `CREATE PLUGGABLE DATABASE ... RELOCATE` statement with the `FROM` clause.

   Specify the source PDB in the `FROM` clause, and include the `RELOCATE` clause. To redirect connections from the old location of the PDB to the new location, specify the `AVAILABILITY MAX` clause. Specify other clauses when they are required.

   After you relocate the PDB, it is in mounted mode, and its status is `RELOCATING`. You can view the open mode of a PDB by querying the `OPEN_MODE` column in the `V$PDBS` view. You can view the status of a PDB by querying the `STATUS` column of the `CDB_PDBS` or `DBA_PDBS` view.

   A new default service is created for the PDB. The service has the same name as the PDB and can be used to access the PDB. Oracle Net Services must be configured properly for clients to access this service.
4. Optionally, to determine the status of the file copy operation, query `V$SESSION_LONGOPS`.

   The `OPNAMES` column shows `kpdbfCopyTaskCbk` for the data file copy and `kcrfremnoc` for the redo file copy.
5. Open the new PDB in read/write mode.

This step is required to complete the integration of the new PDB into the CDB. After the PDB is opened in read/write mode, its status is `NORMAL`. An error is returned if you attempt to open the PDB in read-only mode.

6. Back up the PDB.

   A PDB cannot be recovered unless it is backed up.

> **Note:**
>
> If an error is returned during PDB relocation, then the PDB being created might be in an `UNUSABLE` state. You can check the PDB state by querying the `CDB_PDBS` or `DBA_PDBS` view, and you can learn more about PDB creation errors by checking the alert log. An unusable PDB can only be dropped, and it must be dropped before a PDB with the same name as the unusable PDB can be created.

> **See Also:**
>
> - "About the CDB Undo Mode"
> - "Modifying the Open Mode of PDBs"
> - *Oracle Database Globalization Support Guide* for the compatibility requirements for character sets and national character sets
> - *Oracle Database Backup and Recovery User's Guide* for information about backing up a PDB

Relocating a PDB: Examples

The examples in this section demonstration relocation using SQL and DBCA.

Relocating a PDB from a Remote CDB

This example relocates a PDB named `pdb1` from a remote CDB to the current CDB.

In this example, the root to which the new PDB belongs depends on the current container when the `CREATE PLUGGABLE DATABASE` statement is run:

- When the current container is the CDB root, the new PDB is created in the CDB root.
- When the current container is an application root in an application container, the new PDB is created as an application PDB in the application root.

This example relocates a PDB named `pdb1` from a remote CDB given different factors. This example assumes the following factors:

- The current user has the `CREATE PLUGGABLE DATABASE` system privilege in the root of the target CDB.

- The database link name to the source CDB is `lnk2src`. This database link was created with the following SQL statement:

  ```
  CREATE PUBLIC DATABASE LINK lnk2src CONNECT TO c##myadmin IDENTIFIED
  BY password USING 'MYCDB';
  ```

  The common user `c##myadmin` has `SYSOPER` administrative privilege and `CREATE PLUGGABLE DATABASE` system privilege in the source CDB.
- The `PATH_PREFIX` clause is not required.
- The `FILE_NAME_CONVERT` clause and the `CREATE_FILE_DEST` clause are not required.

  Either Oracle Managed Files is enabled, or the `PDB_FILE_NAME_CONVERT` initialization parameter is set. The files will be moved to a new location based on the Oracle Managed Files configuration or the initialization parameter setting.
- Storage limits are not required for the PDB. Therefore, the `STORAGE` clause is not required.
- There is no file with the same name as the new temp file that will be created in the target location. Therefore, the `TEMPFILE REUSE` clause is not required.
- Connections should be relocated automatically from the source PDB to the relocated PDB. Therefore, the `AVAILABILITY MAX` clause is included.

The following statement relocates the `pdb1` PDB from the source CDB to the current CDB:

```
CREATE PLUGGABLE DATABASE pdb1 FROM pdb1@lnk2src RELOCATE AVAILABILITY
MAX;
```

Relocating a PDB Using DBCA: Example

This example uses DBCA to relocate a PDB named `pdb1` from a remote CDB to the local CDB, where it will be renamed `relpdb1`.

Prerequisites

This scenario assumes the following:

- The user in the local database has the `CREATE PLUGGABLE DATABASE` privilege in the root container.
- The remote CDB is in local undo mode.
- The remote and local CDBs are in `ARCHIVELOG` mode.
- The common user in the remote CDB to whom the database link connects has the `CREATE PLUGGABLE DATABASE`, `SESSION`, and `SYSOPER` privilege.
- The local and remote CDBs have the same options installed.

Assumptions

This scenario assumes the following:

- You are running DBCA on the host of the CDB that will contain the relocated PDB. The local CDB is named `loccdb1`.

- The remote (source) CDB is named `remcdb1` and resides on host `remcdb1host`. The instance name for the remote CDB is `reminst`.
- The remote PDB, which is the PDB to be relocated, is named `rempdb1`.
- The common user `c##adminuser_remcdb1` resides in `remcdb1`.
- The administrative user `locSYS` has `SYSDBA` privileges on `loccdb1`, which is the CDB to which the PDB is being relocated.
- The administrative user `remSYS` has `SYSDBA` privileges on `remcdb1`, which is the CDB that contains the PDB to be relocated.
- After relocation to `loccdb1`, the PDB will be renamed `relpdb1`.

This following silent command relocates `rempdb1` to `loccdb1`:

```
./dbca -silent
  -relocatePDB
  -sourceDB remcdb1
  -remotePDBName rempdb1
  -remoteDBConnString remcdb1host:1521/reminst
  -remoteDBSYSDBAUserName remSYS
      -remoteDBSYSDBAUserPassword remsyspwd
  -dbLinkUsername c##adminuser_remcdb1
      -dbLinkUserPassword pwd4dblinkusr
  -sysDBAUserName locSYS
      -sysDBAPassword locsyspwd
  -pdbName relpdb1
```

> **See Also:**
>
> *Oracle Database Administrator's Guide* for syntax and semantics of DBCA commands

9
Plugging In an Unplugged PDB

You can create a PDB by plugging an unplugged PDB into a CDB.

About PDB Plugin Operations

To plug in a PDB, specify the `USING` clause of `CREATE PLUGGABLE DATABASE`. This clause specifies a XML metadata file or a compressed archive file (`.pdb` file).

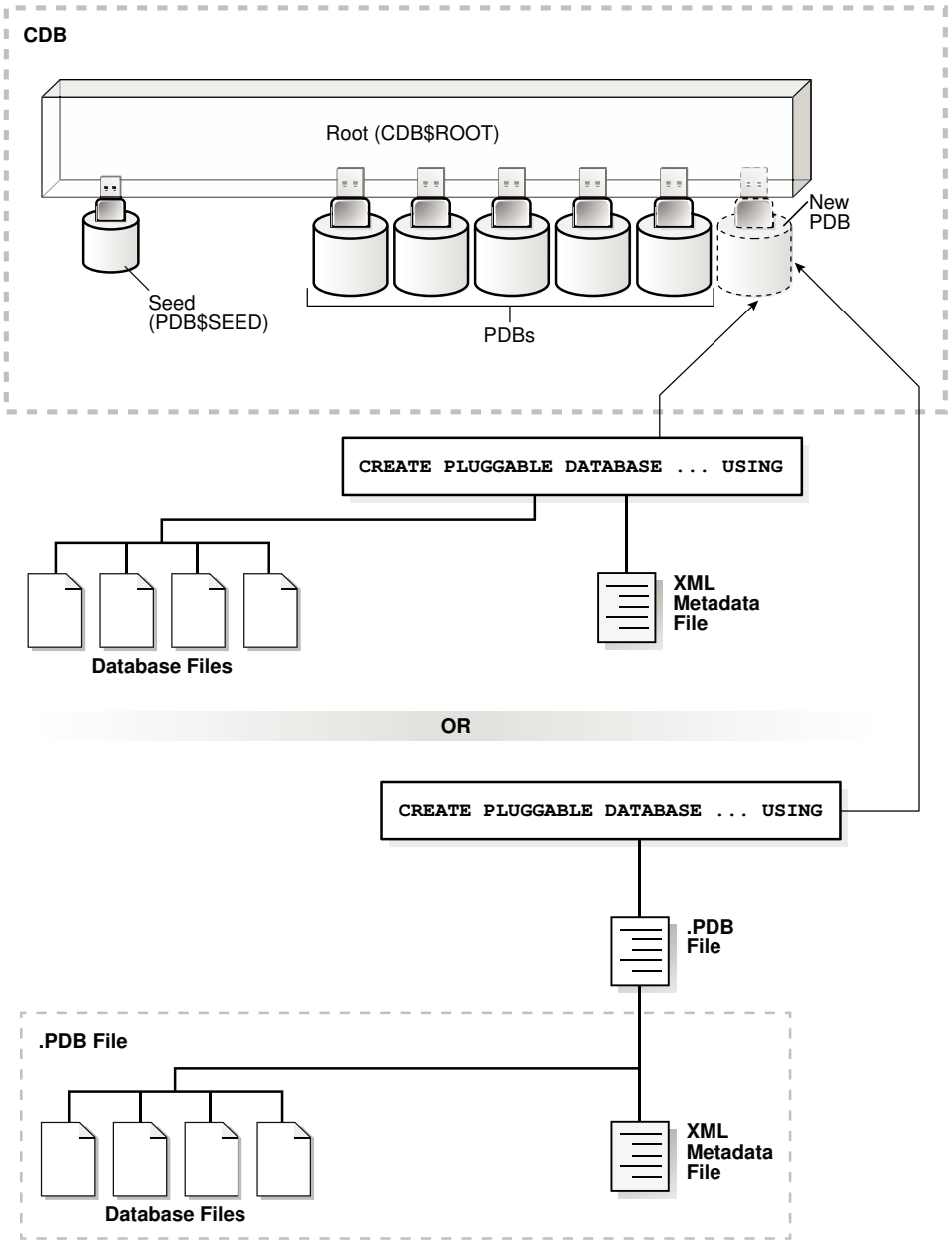
About the XML File and Archive File

An XML metadata file describes the unplugged PDB and the files associated with the PDB (such as the data files and wallet file). An archive file includes both the XML metadata file and the PDB files.

When the XML metadata file is specified, the XML file includes the full paths of the PDB files. When the .pdb archive file is specified, the XML metadata file contains the relative file names only.

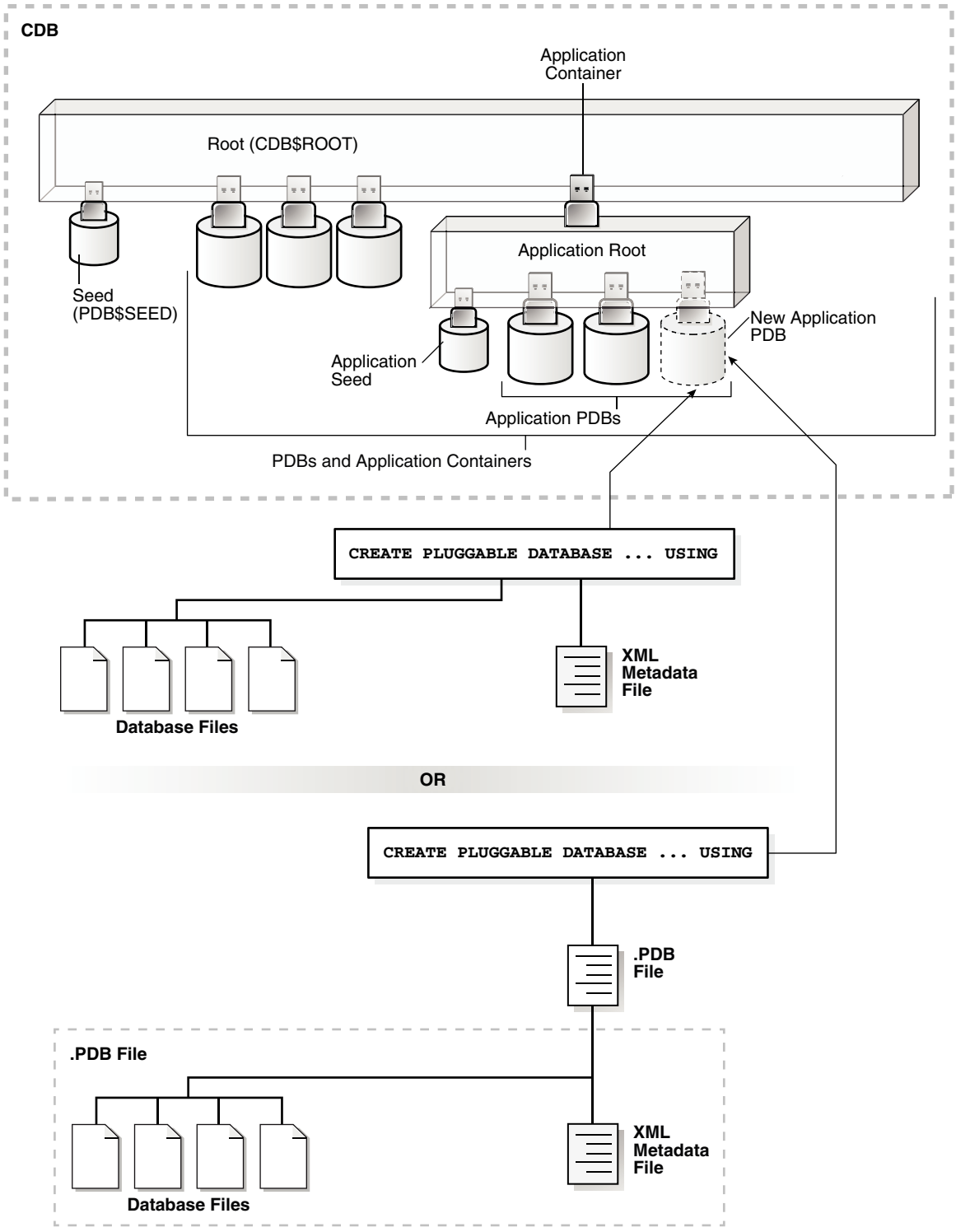
The following figure illustrates how to plug in an unplugged PDB.

Figure 9-1 Plugging an Unplugged PDB Into a CDB Root



The following figure illustrates how this technique creates a new application PDB in an application container.

Figure 9-2 Plugging an Unplugged PDB Into an Application Root



> **Note:**
>
> Automatic downgrade of a PDB is not supported. Therefore, you cannot plug in a PDB if the source CDB is a higher Oracle Database release than the target CDB.

When you plug in an unplugged PDB, you must address the questions that apply to plugging in an unplugged PDB in Table 5-3. The table describes which `CREATE PLUGGABLE DATABASE` clauses you must specify based on different factors.

> **See Also:**
>
> - "PDB Storage"
> - "Creating an Application PDB"

Source File Locations When Plugging In an Unplugged PDB

Use the `CREATE PLUGGABLE DATABASE ... USING` statement to plug an unplugged PDB into a CDB.

When you use a .pdb archive file when plugging in a PDB, Oracle Database extracts this file when you plug in the PDB, and places the PDB files in the same directory as the .pdb archive file. Therefore, the clauses that specify the source file locations are not required when you use a .pdb archive file.

When you specify an XML metadata file when plugging in a PDB, this file describes the names and locations of an unplugged PDB source files. The XML file might not describe the locations of these files accurately if you transported the unplugged files from one storage system to a different one. The files are in a new location, but the file paths in the XML file still indicate the old location.

When plugging in an unplugged PDB using an XML metadata file (not a .pdb archive file), use either the `SOURCE_FILE_NAME_CONVERT` clause or the `SOURCE_FILE_DIRECTORY` clause. These clauses are mutually exclusive.

SOURCE_FILE_NAME_CONVERT Clause

The `SOURCE_FILE_NAME_CONVERT` clause specifies how to locate PDB files when they reside in a location different from that specified in the XML file.

You can use this clause to specify one of the following options:

- One or more file name patterns and replacement file name patterns, in the following form:

  ```
  'string1' , 'string2' , 'string3' , 'string4' , ...
  ```

  The *string2* file name pattern replaces the *string1* file name pattern, and the *string4* file name pattern replaces the *string3* file name pattern. You can use as many pairs of file name pattern and replacement file name pattern strings as required.

When you use this clause, ensure that the files you want to use for the PDB reside in the replacement file name patterns. Move or copy the files to these locations if necessary.

- `NONE` when no file names need to be located because the PDB's XML file describes the file names accurately. Omitting the `SOURCE_FILE_NAME_CONVERT` clause is the same as specifying `NONE`.

You can use the `SOURCE_FILE_NAME_CONVERT` clause only in a `CREATE PLUGGABLE DATABASE` statement with a `USING` clause that specifies an XML metadata file. Therefore, you can use this clause only when you are plugging in an unplugged PDB with an XML metadata file. You cannot use this clause when you are plugging in a PDB with a .pdb archive file.

Example 9-1 SOURCE_FILE_NAME_CONVERT Clause

This `SOURCE_FILE_NAME_CONVERT` clause uses the files in the /disk2/oracle/pdb7 directory instead of the /disk1/oracle/pdb7 directory. In this case, the XML file describing a PDB specifies the /disk1/oracle/pdb7 directory, but the PDB should use the files in the /disk2/oracle/pdb7 directory.

```
SOURCE_FILE_NAME_CONVERT = ('/disk1/oracle/pdb7/', '/disk2/oracle/pdb7/')
```

See Also:

- Plugging In an Unplugged PDB
- *Oracle Database SQL Language Reference* for the syntax of the `SOURCE_FILE_NAME_CONVERT` clause

SOURCE_FILE_DIRECTORY Clause

The `SOURCE_FILE_DIRECTORY` clause specifies the source directory of the files that will be used to create the new PDB.

The clause specifies a directory that contains all of the files listed in the XML file. Using this clause is convenient when you have many data files and specifying a `SOURCE_FILE_NAME_CONVERT` pattern for each file is not feasible.

When you plug in a PDB, if the source files are all present in a single directory, then you can specify the directory name in this clause. The directory is scanned to find the appropriate files based on the unplugged PDB's XML file.

You can use this clause to specify one of the following options:

- The absolute path of the source file directory.
- `NONE` when no files should be copied or moved during PDB creation. Omitting the `SOURCE_FILE_DIRECTORY` clause is the same as specifying `NONE`.

You can use the `SOURCE_FILE_DIRECTORY` clause only in a `CREATE PLUGGABLE DATABASE` statement with a `USING` clause that specifies an XML metadata file. Therefore, you can use this clause only when you are plugging in an unplugged PDB with an XML metadata file. You cannot use this clause when you are plugging in a PDB with a .pdb archive file.

You can specify this clause for configurations that use Oracle Managed Files and for configurations that do not use Oracle Managed Files.

Example 9-2 SOURCE_FILE_DIRECTORY Clause

This `SOURCE_FILE_DIRECTORY` clause generates file names for the new PDB by using the source files in the /oracle/pdb5/ directory.

```
SOURCE_FILE_DIRECTORY = '/oracle/pdb5/'
```

See Also:

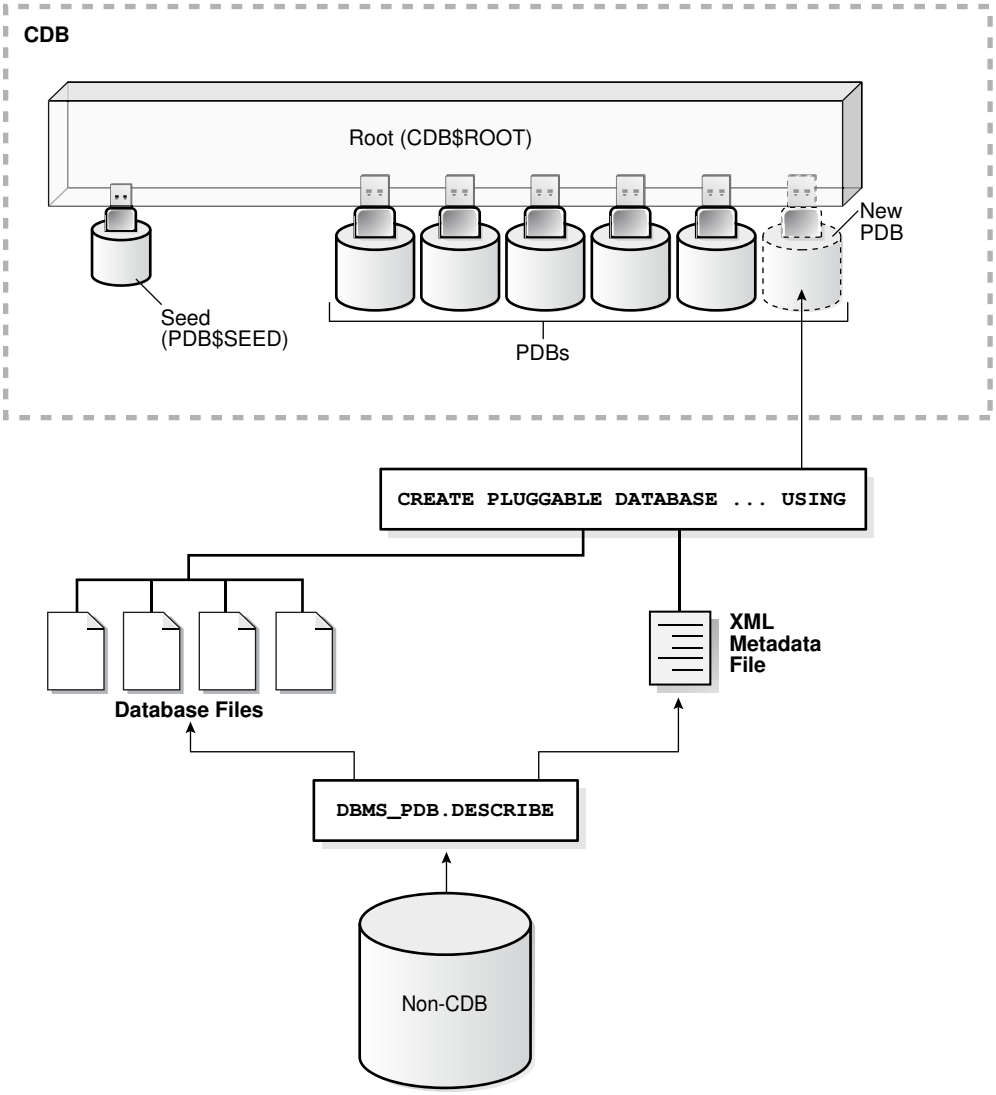
- Plugging In an Unplugged PDB
- *Oracle Database SQL Language Reference* for the syntax of the `SOURCE_FILE_DIRECTORY` clause

About Adopting a Non-CDB as a PDB

To generate an XML file that describes a non-CDB, use the `DBMS_PDB.DESCRIBE` procedure. Afterward, plug in the non-CDB just as you would an unplugged PDB.

Create the PDB with the `CREATE PLUGGABLE DATABASE ... USING` statement. When the non-CDB is plugged in to a CDB, it is a PDB.

Figure 9-3 Plug In a Non-CDB Using the DBMS_PDB.DESCRIBE Procedure



You can use the same technique to create a new application PDB in an application container.

> **Note:**
>
> To use this technique, the non-CDB must be at release Oracle Database 12c or later.

> **See Also:**
>
> "Plugging In an Unplugged PDB"

Plugging In an Unplugged PDB

Plug in a PDB with the `CREATE PLUGGABLE DATABASE ... USING` statement.

General Prerequisites

To plug in an unplugged PDB, the following prerequisites must be met:

- Complete the prerequisites described in "General Prerequisites for PDB Creation".
- Either the XML file that describes the PDB or the .pdb archive file must exist in a location that is accessible to the CDB.

  The `USING` clause must specify the XML file or the .pdb archive file. If the PDB's XML file is unusable or cannot be located, then use the `DBMS_PDB.RECOVER` procedure to generate an XML file using the PDB's data files.
- If an XML file (not a .pdb file) is specified in the `USING` clause, then the files associated with the PDB (such as the data files and wallet file) must exist in a location that is accessible to the CDB.
- If the target database for the plugin operation is the primary database in an Oracle Data Guard configuration, then ensure that the standby database can locate the files for the plugged-in PDB.

  On the standby database, set the `STANDBY_PDB_SOURCE_FILE_DIRECTORY` initialization parameter to a location that contains the source data files for instantiating the PDB. If the files are not found, then the standby database tries to locate the files in the OMF location. If not found in the OMF location, then you must copy the data files to the OMF location on the standby database, and restart redo apply on the standby database.
- The source and target CDB platforms must meet the following requirements:
  - They must have the same endianness.
  - The database options installed on the source platform must be the same as, or a subset of, the database options installed on the target platform.
- If you are creating an application PDB, then the application name and version of the unplugged PDB must match the application name and version of the application container into which the application PDB is being plugged.

Character Set Prerequisites

You must meet the following prerequisites for matching the character sets:

- If the character set of the CDB into which the PDB is being plugged is not AL32UTF8, then the CDB that contained the unplugged PDB and the target CDB must have compatible character sets and national character sets. To be compatible, the character sets and national character sets must meet the requirements specified in *Oracle Database Globalization Support Guide*.

  If the character set of the CDB into which the PDB is being plugged is AL32UTF8, then this requirement does not apply.

> **Note:**
>
> Oracle Multitenant does not support a LOB in one container from being accessed by a container with a different character set using data links, extended data links, or the `CONTAINERS()` clause. For example, if the CDB root and `salespdb` have different character sets, then a `CONTAINERS()` query run in the CDB root should not access LOBs stored in `salespdb`.

- If you are creating an application PDB, then the application PDB must have the same character set and national character set as the application container.

  If the database character set of the CDB is AL32UTF8, then the character set and national character set of the application container can be different from the CDB. However, all application PDBs in an application container must have same character set and national character set, matching that of the application container.

To determine whether the preceding requirements are met, use the `DBMS_PDB.CHECK_PLUG_COMPATIBILITY` function. Step 2 in the following procedure describes using this function.

To plug in a PDB:

1. In SQL*Plus, ensure that the current container is the CDB root or application root of the target CDB.

   When the current container is the CDB root, the PDB is created in the CDB. When the current container is an application root, the application PDB is created in the application container.

2. (Optional) Run the `DBMS_PDB.CHECK_PLUG_COMPATIBILITY` function to determine whether the unplugged PDB is compatible with the CDB.

   a. If the PDB is not yet unplugged, then run the `DBMS_PDB.DESCRIBE` procedure to produce an XML file that describes the PDB.

      If the PDB is already unplugged, then proceed to Step 2b.

      For example, to generate an XML file named `salespdb.xml` in the /disk1/oracle directory, run the following procedure:

      ```
      BEGIN
        DBMS_PDB.DESCRIBE(
          pdb_descr_file => '/disk1/oracle/salespdb.xml',
          pdb_name       => 'SALESPDB');
      END;
      /
      ```

      If the PDB is in a remote CDB, then you can include *`@database_link_name`* in the `pdb_name` parameter, where *`database_link_name`* is the name of a valid database link to the remote CDB or to the PDB. For example, if the database link name to the remote CDB is `rcdb`, then set the `pdb_name` value to `SALESPDB@rcdb`.

   b. Run the `DBMS_PDB.CHECK_PLUG_COMPATIBILITY` function.

      When you run the function, set the following parameters:

- `pdb_descr_file` - Set this parameter to the full path to the XML file.
- `pdb_name` - Specify the name of the new PDB. If this parameter is omitted, then the PDB name in the XML file is used.

For example, to determine whether a PDB described by the `/disk1/usr/salespdb.xml` file is compatible with the current CDB, run the following PL/SQL block:

```
SET SERVEROUTPUT ON
DECLARE
  compatible CONSTANT VARCHAR2(3) :=
    CASE DBMS_PDB.CHECK_PLUG_COMPATIBILITY(
           pdb_descr_file => '/disk1/usr/salespdb.xml',
           pdb_name       => 'SALESPDB')
    WHEN TRUE THEN 'YES'
    ELSE 'NO'
END;
BEGIN
  DBMS_OUTPUT.PUT_LINE(compatible);
END;
/
```

If the output is `YES`, then the PDB is compatible, and you can continue with the next step. If the output is `NO`, then the PDB is not compatible: check the `PDB_PLUG_IN_VIOLATIONS` view to see why it is not compatible.

> **Note:**
>
> You can specify a .pdb archive file in the `pdb_descr_file` parameter.

3. If the PDB is not unplugged, then unplug it.
4. Run the `CREATE PLUGGABLE DATABASE ... USING` statement, specifying the XML file or the .pdb archive file in the `USING` clause. Specify other clauses when they are required.

   After you create the PDB, it is in mounted mode, and its status is `NEW`. You can view the open mode of a PDB by querying the `OPEN_MODE` column in the `V$PDBS` view. You can view the status of a PDB by querying the `STATUS` column of the `CDB_PDBS` or `DBA_PDBS` view.

   A new default service is created for the PDB. The service has the same name as the PDB and can be used to access the PDB. Oracle Net Services must be configured properly for clients to access this service.
5. Open the new PDB in read/write mode.

   You must open the new PDB in read/write mode for Oracle Database to complete the integration of the new PDB into the CDB. An error is returned if you attempt to open the PDB in read-only mode. After the PDB is opened in read/write mode, its status is `NORMAL`.
6. Back up the PDB.

   A PDB cannot be recovered unless it is backed up.

> **Note:**
>
> If an error is returned during PDB creation, then the PDB being created might be in an `UNUSABLE` state. You can check a PDB's state by querying the `CDB_PDBS` or `DBA_PDBS` view, and you can learn more about PDB creation errors by checking the alert log. An unusable PDB can only be dropped, and it must be dropped before a PDB with the same name as the unusable PDB can be created.

> **See Also:**
>
> - "Unplugging a PDB from a CDB"
> - "Modifying the Open Mode of PDBs" for more information.
> - *Oracle Database Backup and Recovery User's Guide* for information about backing up a PDB.
> - *Oracle Data Guard Concepts and Administration* to learn more about plugging in a PDB in an Oracle Data Guard environment
> - *Oracle Database Reference* for information about the `PDB_FILE_NAME_CONVERT` initialization parameter
> - *Oracle Database PL/SQL Packages and Types Reference* for more information about this procedure.

Adopting a Non-CDB as a PDB

You can adopt (move) a non-CDB into a PDB by using the `DBMS_PDB.DESCRIBE` procedure.

To adopt a non-CDB as a PDB using the `DBMS_PDB` package:

1. Create the CDB if it does not exist.
2. Ensure that the non-CDB is in a transactionally consistent state.
3. Place the non-CDB in read-only mode.
4. Connect to the non-CDB, and run the `DBMS_PDB.DESCRIBE` procedure to construct an XML file that describes the non-CDB.

   The current user must have `SYSDBA` administrative privilege. The user must exercise the privilege using `AS SYSDBA` at connect time.

   For example, to generate an XML file named `ncdb.xml` in the `/disk1/oracle` directory, run the following procedure:

   ```
   BEGIN
     DBMS_PDB.DESCRIBE(
       pdb_descr_file => '/disk1/oracle/ncdb.xml');
   ```

```
END;
/
```

After the procedure completes successfully, you can use the XML file and the non-CDB database files to plug the non-CDB into a CDB.

5. Run the `DBMS_PDB.CHECK_PLUG_COMPATIBILITY` function to determine whether the non-CDB is compatible with the CDB.

   When you run the function, set the following parameters:

   - `pdb_descr_file` - Set this parameter to the full path to the XML file.
   - `pdb_name` - Specify the name of the new PDB. If this parameter is omitted, then the PDB name in the XML file is used.

   For example, to determine whether a non-CDB described by the `/disk1/oracle/ncdb.xml` file is compatible with the current CDB, run the following PL/SQL block:

```
SET SERVEROUTPUT ON
DECLARE
  compatible CONSTANT VARCHAR2(3) :=
    CASE DBMS_PDB.CHECK_PLUG_COMPATIBILITY(
           pdb_descr_file => '/disk1/oracle/ncdb.xml',
           pdb_name       => 'NCDB')
    WHEN TRUE THEN 'YES'
    ELSE 'NO'
END;
BEGIN
  DBMS_OUTPUT.PUT_LINE(compatible);
END;
/
```

   If the output is `YES`, then the non-CDB is compatible, and you can continue with the next step. If the output is `NO`, then the non-CDB is not compatible, and you can check the `PDB_PLUG_IN_VIOLATIONS` view to see why it is not compatible. All violations must be corrected before you continue. For example, any version or patch mismatches should be resolved by running an upgrade or the datapatch utility. After correcting the violations, run `DBMS_PDB.CHECK_PLUG_COMPATIBILITY` again to ensure that the non-CDB is compatible with the CDB.

6. Shut down the non-CDB.

7. Plug in the non-CDB.

   For example, the following SQL statement plugs in a non-CDB, copies its files to a new location, and includes only the `tbs3` user tablespace from the non-CDB:

```
CREATE PLUGGABLE DATABASE ncdb USING '/disk1/oracle/ncdb.xml'
  COPY
  FILE_NAME_CONVERT = ('/disk1/oracle/dbs/', '/disk2/oracle/ncdb/')
  USER_TABLESPACES=('tbs3');
```

   If there are no violations, then do not open the new PDB. You will open it in the following step.

The USER_TABLESPACES clause enables you to separate data that was used for multiple tenants in a non-CDB into different PDBs. You can use multiple CREATE PLUGGABLE DATABASE statements with this clause to create other PDBs that include the data from other tablespaces that existed in the non-CDB.

8. Run the *ORACLE_HOME*/rdbms/admin/noncdb_to_pdb.sql script. This script must be run before the PDB can be opened for the first time.

   If the PDB was not a non-CDB, then running the noncdb_to_pdb.sql script is not required. To run the noncdb_to_pdb.sql script, complete the following steps:

   a. Access the PDB.

      The current user must have SYSDBA administrative privilege, and the privilege must be either commonly granted or locally granted in the PDB. The user must exercise the privilege using AS SYSDBA at connect time.

   b. Run the noncdb_to_pdb.sql script:

      ```
      @$ORACLE_HOME/rdbms/admin/noncdb_to_pdb.sql
      ```

   The script opens the PDB, performs changes, and closes the PDB when the changes are complete.

9. Open the new PDB in read/write mode.

   You must open the new PDB in read/write mode for Oracle Database to complete the integration of the new PDB into the CDB. An error is returned if you attempt to open the PDB in read-only mode. After the PDB is opened in read/write mode, its status is NORMAL.

10. Back up the PDB.

    A PDB cannot be recovered unless it is backed up.

> **Note:**
>
> If an error is returned during PDB creation, then the PDB being created might be in an UNUSABLE state. You can check the PDB state by querying the CDB_PDBS or DBA_PDBS view, and you can learn more about PDB creation errors by checking the alert log. An unusable PDB can only be dropped, and you must drop it before a PDB with the same name as the unusable PDB can be created.

Related Topics

- Plugging In an Unplugged PDB
  Plug in a PDB with the CREATE PLUGGABLE DATABASE ... USING statement.
- Creating and Configuring a CDB
  Creating and configuring a multitenant container database (CDB) includes tasks such as planning, creating the CDB, and optionally configuring EM Express.
- After Plugging in an Unplugged PDB
  Certain rules regarding users and tablespaces apply after plugging in an unplugged PDB.
- *Oracle Database Backup and Recovery User's Guide*

After Plugging in an Unplugged PDB

Certain rules regarding users and tablespaces apply after plugging in an unplugged PDB.

The following applies after plugging in an unplugged PDB:

- User accounts in the PDB who used the default temporary tablespace of the source PDB use the default temporary tablespace of the target PDB. User accounts who used nondefault temporary tablespaces in the source PDB continue to use the same local temporary tablespaces in the target PDB.
- Manually created common user accounts that existed in the source CDB but not in the target CDB do not have privileges granted commonly. However, if the target CDB has a common user with the same name as a common user in the PDB, then the latter is linked to the former and has the privileges granted to this common user in the target CDB.

  If the cloned or plugged-in PDB has a common user account that does not exist in the target CDB, and if this user does not own objects in the PDB, then Oracle Database drops the user during the synchronization step; otherwise, the user account is locked in the target PDB. You have the following options regarding locked accounts:

  – Close the PDB, connect to the root, and create a common user account with the same name. When the PDB is opened in read/write mode, differences in roles and privileges granted commonly to the user account are resolved, and you can unlock the account. Privileges and roles granted locally to the user account remain unchanged during this process.
  – Create a new local user account in the PDB and use Data Pump to export/import the locked user's data into the new local user's schema.
  – Leave the user account locked.
  – Drop the user account.

See Also:

- "Managing Services for PDBs"
- "About Managing Tablespaces in a CDB"
- *Oracle Database Concepts* for information about common users and local users
- *Oracle Database Security Guide* for information about creating common users and local users in a CDB
- *Oracle Database Utilities* for information about using Oracle Data Pump with a CDB

Plugging in an Unplugged PDB: Examples

These examples plug in an unplugged PDB named `salespdb` using the `/disk1/usr/salespdb.xml` file or the `/disk1/usr/sales.pdb` file given different factors.

In each example, the root to which the new PDB belongs depends on the current container when the `CREATE PLUGGABLE DATABASE` statement is run:

- When the current container is the CDB root, the new PDB is created in the CDB.
- When the current container is an application root, the new application PDB is created in the application root's application container.

Example 9-3 Plugging In an Unplugged PDB Using the NOCOPY Clause

This example assumes the following factors:

- The new PDB is not based on the same unplugged PDB that was used to create an existing PDB in the CDB. Therefore, the `AS CLONE` clause is not required.
- The `PATH_PREFIX` clause is not required.
- The XML file accurately describes the current locations of the files. Therefore, the `SOURCE_FILE_NAME_CONVERT` clause or `SOURCE_FILE_DIRECTORY` clause is not required.
- The files are in the correct location. Therefore, `NOCOPY` is included.
- Storage limits are not required for the PDB. Therefore, the `STORAGE` clause is not required.
- A file with the same name as the temp file specified in the XML file exists in the target location. Therefore, the `TEMPFILE REUSE` clause is required.

Given the preceding factors, the following statement plugs in the PDB:

```
CREATE PLUGGABLE DATABASE salespdb USING '/disk1/usr/salespdb.xml'
  NOCOPY
  TEMPFILE REUSE;
```

Example 9-4 Plugging In an Unplugged PDB Using the AS CLONE and NOCOPY Clauses

This example assumes the following factors:

- The new PDB is based on the same unplugged PDB that was used to create an existing PDB in the CDB. Therefore, the `AS CLONE` clause is required. The `AS CLONE` clause ensures that the new PDB has unique identifiers.
- The `PATH_PREFIX` clause is not required.
- The XML file accurately describes the current locations of the files. Therefore, the `SOURCE_FILE_NAME_CONVERT` clause or `SOURCE_FILE_DIRECTORY` clause is not required.
- The files are in the correct location. Therefore, `NOCOPY` is included.
- Storage limits are not required for the PDB. Therefore, the `STORAGE` clause is not required.

- A file with the same name as the temp file specified in the XML file exists in the target location. Therefore, the `TEMPFILE REUSE` clause is required.

Given the preceding factors, the following statement plugs in the PDB:

```
CREATE PLUGGABLE DATABASE salespdb AS CLONE USING '/disk1/usr/
salespdb.xml'
  NOCOPY
  TEMPFILE REUSE;
```

Example 9-5 Plugging In an Unplugged PDB Using the SOURCE_FILE_NAME_CONVERT, NOCOPY, and STORAGE Clauses

This example assumes the following factors:

- The new PDB is not based on the same unplugged PDB that was used to create an existing PDB in the CDB. Therefore, the `AS CLONE` clause is not required.
- The `PATH_PREFIX` clause is not required.
- The XML file does not accurately describe the current locations of the files. Therefore, the `SOURCE_FILE_NAME_CONVERT` clause or `SOURCE_FILE_DIRECTORY` clause is required. In this example, the XML file indicates that the files are in /disk1/oracle/sales, but the files are in /disk2/oracle/sales, and the `SOURCE_FILE_NAME_CONVERT` clause is used.
- The files are in the correct location. Therefore, `NOCOPY` is included.
- Storage limits must be enforced for the PDB. Therefore, the `STORAGE` clause is required. Specifically, all tablespaces that belong to the PDB must not exceed 2 gigabytes.
- A file with the same name as the temp file specified in the XML file exists in the target location. Therefore, the `TEMPFILE REUSE` clause is required.

Given the preceding factors, the following statement plugs in the PDB:

```
CREATE PLUGGABLE DATABASE salespdb USING '/disk1/usr/salespdb.xml'
  SOURCE_FILE_NAME_CONVERT = ('/disk1/oracle/sales/', '/disk2/oracle/
sales/')
  NOCOPY
  STORAGE (MAXSIZE 2G)
  TEMPFILE REUSE;
```

Example 9-6 Plugging In an Unplugged PDB With the COPY, PATH_PREFIX, and FILE_NAME_CONVERT Clauses

This example assumes the following factors:

- The new PDB is not based on the same unplugged PDB that was used to create an existing PDB in the CDB. Therefore, the `AS CLONE` clause is not required.
- The path prefix must be added to the PDB's directory object paths. Therefore, the `PATH_PREFIX` clause is required. In this example, the path prefix /disk2/oracle/sales/ is added to the PDB's directory object paths.
- The XML file accurately describes the current locations of the files. Therefore, the `SOURCE_FILE_NAME_CONVERT` clause or `SOURCE_FILE_DIRECTORY` clause is not required.

- The files are not in the correct location. Therefore, COPY or MOVE must be included. In this example, the files are copied.

  The CREATE_FILE_DEST clause is not used, Oracle Managed Files is not enabled, and the PDB_FILE_NAME_CONVERT initialization parameter is not set. Therefore, the FILE_NAME_CONVERT clause is required. In this example, the files are copied from /disk1/oracle/sales to /disk2/oracle/sales.

- Storage limits are not required for the PDB. Therefore, the STORAGE clause is not required.
- There is no file with the same name as the new temp file that will be created in the target location. Therefore, the TEMPFILE REUSE clause is not required.

Given the preceding factors, the following statement plugs in the PDB:

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

Example 9-7 Plugging In an Unplugged PDB Using the SOURCE_FILE_NAME_CONVERT, MOVE, FILE_NAME_CONVERT, and STORAGE Clauses

This example assumes the following factors:

- The new PDB is not based on the same unplugged PDB that was used to create an existing PDB in the CDB. Therefore, the AS CLONE clause is not required.
- The PATH_PREFIX clause is not required.
- The XML file does not accurately describe the current locations of the files. Therefore, the SOURCE_FILE_NAME_CONVERT clause or SOURCE_FILE_DIRECTORY clause is required. In this example, the XML file indicates that the files are in /disk1/oracle/sales, but the files are in /disk2/oracle/sales, and the SOURCE_FILE_NAME_CONVERT clause is used.
- The files are not in the correct final location for the PDB. Therefore, COPY or MOVE must be included. In this example, MOVE is specified to move the files.

  The CREATE_FILE_DEST clause is not used, Oracle Managed Files is not enabled, and the PDB_FILE_NAME_CONVERT initialization parameter is not set. Therefore, the FILE_NAME_CONVERT clause is required. In this example, the files are moved from /disk2/oracle/sales to /disk3/oracle/sales.

- Storage limits must be enforced for the PDB. Therefore, the STORAGE clause is required. Specifically, all tablespaces that belong to the PDB must not exceed 2 gigabytes.
- There is no file with the same name as the new temp file that will be created in the target location. Therefore, the TEMPFILE REUSE clause is not required.

Given the preceding factors, the following statement plugs in the PDB:

```
CREATE PLUGGABLE DATABASE salespdb USING '/disk1/usr/salespdb.xml'
  SOURCE_FILE_NAME_CONVERT = ('/disk1/oracle/sales/', '/disk2/oracle/
sales/')
  MOVE
```

```
  FILE_NAME_CONVERT = ('/disk2/oracle/sales/', '/disk3/oracle/sales/')
  STORAGE (MAXSIZE 2G);
```

Example 9-8 Plugging In an Unplugged PDB Using the SOURCE_FILE_DIRECTORY, MOVE, FILE_NAME_CONVERT, and STORAGE Clauses

This example assumes the following factors:

- The new PDB is not based on the same unplugged PDB that was used to create an existing PDB in the CDB. Therefore, the `AS CLONE` clause is not required.
- The `PATH_PREFIX` clause is not required.
- The XML file does not accurately describe the current locations of the files. Therefore, the `SOURCE_FILE_NAME_CONVERT` clause or `SOURCE_FILE_DIRECTORY` clause is required. In this example, the XML file indicates that the files are in /disk1/oracle/sales, but the files are in /disk2/oracle/sales, and the `SOURCE_FILE_DIRECTORY` clause is used.
- The files are not in the correct final location for the PDB. Therefore, `COPY` or `MOVE` must be included. In this example, `MOVE` is specified to move the files.

  The `CREATE_FILE_DEST` clause is not used, Oracle Managed Files is not enabled, and the `PDB_FILE_NAME_CONVERT` initialization parameter is not set. Therefore, the `FILE_NAME_CONVERT` clause is required. In this example, the files are moved from /disk2/oracle/sales to /disk3/oracle/sales.
- Storage limits must be enforced for the PDB. Therefore, the `STORAGE` clause is required. Specifically, all tablespaces that belong to the PDB must not exceed 2 gigabytes.
- There is no file with the same name as the new temp file that will be created in the target location. Therefore, the `TEMPFILE REUSE` clause is not required.

Given the preceding factors, the following statement plugs in the PDB:

```
CREATE PLUGGABLE DATABASE salespdb USING '/disk1/usr/salespdb.xml'
  SOURCE_FILE_DIRECTORY = '/disk2/oracle/sales/'
  MOVE
  FILE_NAME_CONVERT = ('/disk2/oracle/sales/', '/disk3/oracle/sales/')
  STORAGE (MAXSIZE 2G);
```

Example 9-9 Plugging In an Unplugged PDB Using an Archive File

This example assumes the following factors:

- The unplugged PDB is in a .pdb archive file named `sales.pdb`. The archive file includes the XML metadata file and the PDB's files (such as the data files and wallet file) in compressed form, and these files are extracted to the current directory of the .pdb archive file when the `CREATE PLUGGABLE DATABASE` statement is run.
- The new PDB is not based on the same unplugged PDB that was used to create an existing PDB in the CDB. Therefore, the `AS CLONE` clause is not required.
- The `PATH_PREFIX` clause is not required.

- Storage limits must be enforced for the PDB. Therefore, the `STORAGE` clause is required. Specifically, all tablespaces that belong to the PDB must not exceed 2 gigabytes.
- There is no file with the same name as the new temp file that will be created in the target location. Therefore, the `TEMPFILE REUSE` clause is not required.

Given the preceding factors, the following statement plugs in the PDB using an archive file:

```
CREATE PLUGGABLE DATABASE salespdb USING '/disk1/usr/sales.pdb'
  STORAGE (MAXSIZE 2G);
```

10

Creating a PDB as a Proxy PDB

You can create a PDB as a proxy PDB by referencing it in a remote CDB.

About Creating a Proxy PDB

A **proxy PDB** provides access to a PDB in a remote CDB. It is analogous to a symbolic link.

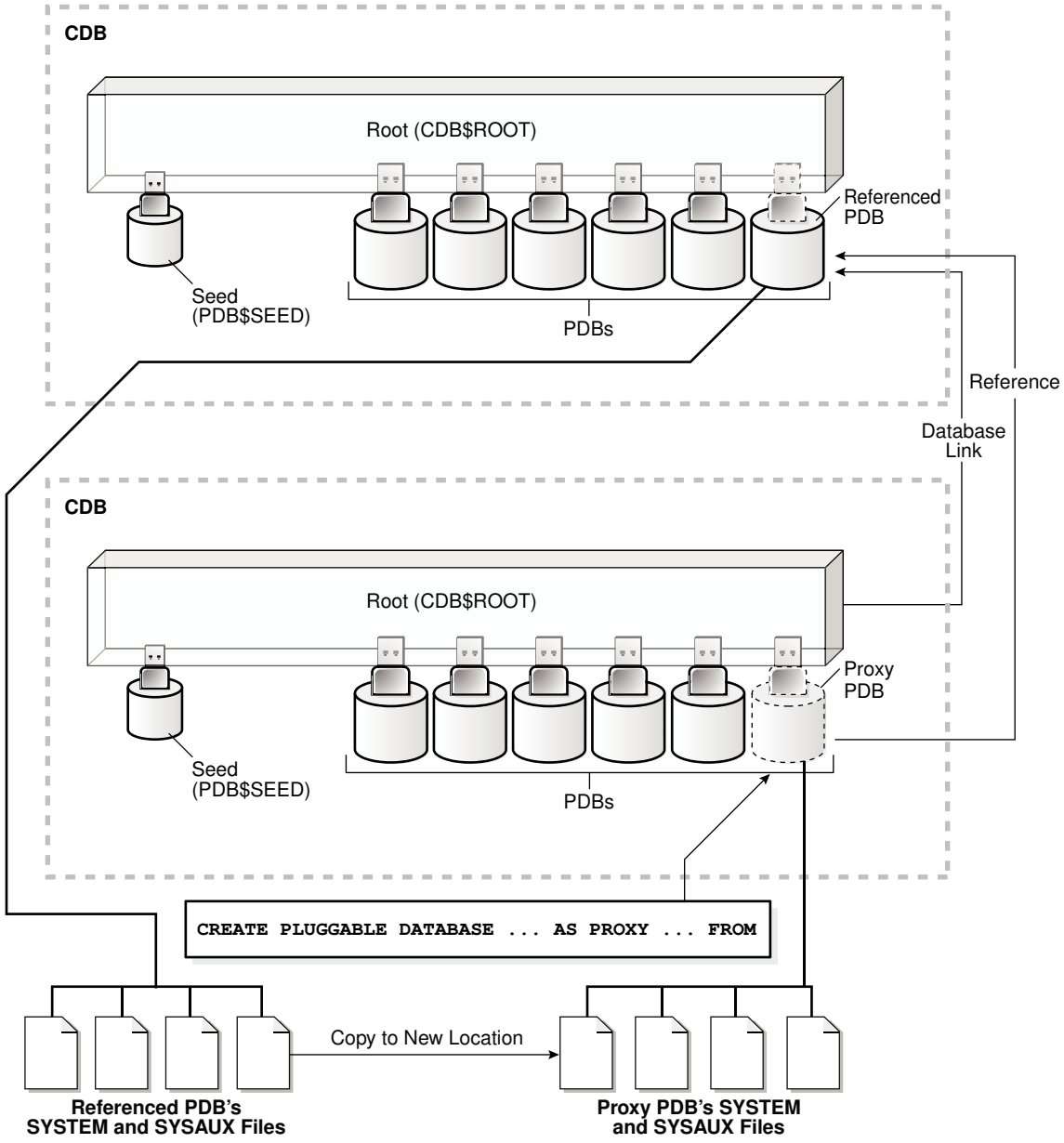
The `CREATE PLUGGABLE DATABASE` statement creates a proxy PDB by referencing a PDB in a different CDB, which is called the **referenced PDB**. You can use a proxy PDB when you want a local context for a remote PDB. In addition, when application containers in different CDBs have the same application, you can keep their application roots synchronized with a proxy PDB.

To use this technique, run the `CREATE PLUGGABLE DATABASE` statement in the CDB that will contain the proxy PDB. You must include:

- The `AS PROXY` clause to specify that you are creating a proxy PDB.
- A `FROM` clause that specifies the PDB that the proxy PDB is referencing.
- A database link to the current location of the referenced PDB in the `FROM` clause. The database link must be created in the root of the CDB that will contain the proxy PDB, and the database link connects either to the root of remote CDB or to the remote referenced PDB.

The following figure illustrates how this technique creates a proxy PDB that references a PDB in a remote CDB.

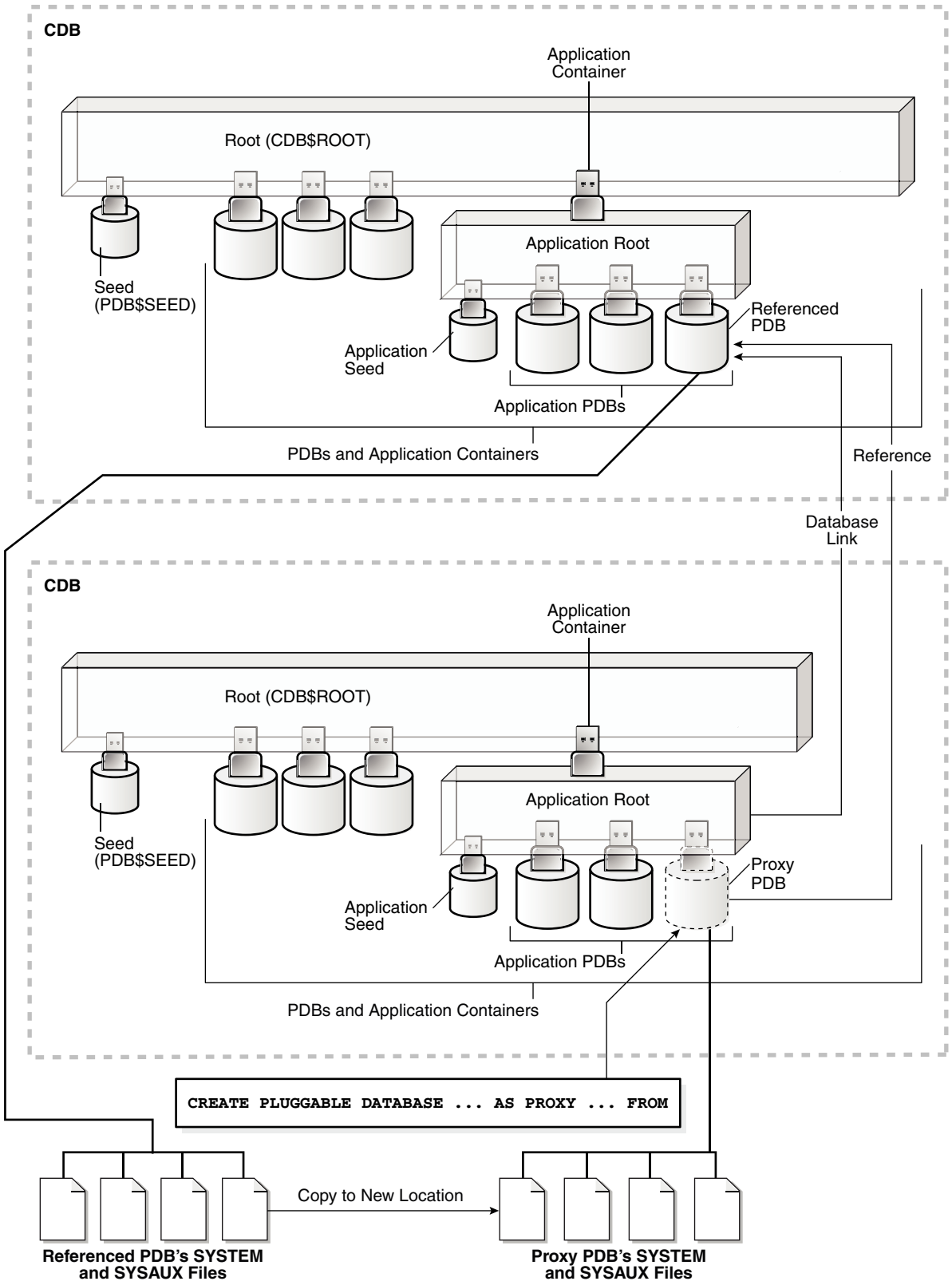
Figure 10-1 Create a Remote Proxy PDB



You can create a proxy PDB in an application container. To do so, the referenced PDB must be an application root or an application PDB in an application container in a different CDB. The database link must be created in the root of the application container that will contain the proxy PDB, and the database link connects either to the root of remote application container or to the remote referenced application PDB.

The following graphic illustrates how this technique creates a proxy PDB in an application container based on a remote referenced PDB in an application container.

Figure 10-2 Create a Remote Proxy PDB in an Application Container



Before creating a proxy PDB, address the questions that apply to creating a proxy PDB in "Table 5-3". The table describes which `CREATE PLUGGABLE DATABASE` clauses you must specify based on different factors.

> **See Also:**
>
> - "PDB Storage"
> - "Synchronizing an Application Root Replica with a Proxy PDB"

Proxy PDBs and SQL Statements

As a rule, when the proxy PDB is the current container, SQL statements submitted for execution in the proxy PDB are executed in the referenced PDB.

The results of the remote execution are returned to the proxy PDB. For example, data definition language (DDL) statements, data manipulation language (DML) statements, and queries executed in the proxy PDB are sent to the referenced PDB for execution, and the results are returned to the proxy PDB.

There is one exception to the rule. When the proxy PDB is the current container, and when you execute `ALTER PLUGGABLE DATABASE` and `ALTER DATABASE` statements, these statements only affect the proxy PDB. They are not sent to the referenced PDB for execution. Similarly, when the current container is the root to which the proxy PDB belongs, `ALTER PLUGGABLE DATABASE` statements only affect the proxy PDB. For example, an `ALTER PLUGGABLE DATABASE` statement executed in a CDB root, application root, or proxy PDB can open or close a proxy PDB, but this statement does not open or close the referenced PDB.

Proxy PDBs and Database Links

A database link is required when you create a proxy PDB.

After the proxy PDB is created, the database link specified during creation is no longer used by the proxy PDB. Instead, the proxy PDB communicates directly with the referenced PDB.

This direct communication requires the port number and host name of the listener of the CDB that contains the referenced PDB. During proxy PDB creation, the proxy PDB uses the following values by default:

- **Listener port number:** 1521

  If the referenced PDB's listener does not use the default port number, then you must use the `PORT` clause to specify the listener's port number. You can specify the port number when you create the proxy PDB, or you can alter the proxy PDB to change the port number.

- **Listener host name:** The host name of the CDB that contains the referenced PDB

  If the referenced PDB's listener does not use the default host name, then you must use the `HOST` clause to specify the listener's host name. You can specify the host name when you create the proxy PDB, or you can alter the proxy PDB to change the host name.

Related Topics

- Proxy PDBs and the Listener
  The host name and port number settings for a PDB are important only if proxy PDBs will reference the PDB.
- Modifying the Listener Settings of a Referenced PDB
  A PDB that is referenced by a proxy PDB is called a referenced PDB.

Proxy PDBs and Authentication

Only password authentication is supported for sessions in a proxy PDB.

Proxy PDBs and the Listener

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

HOST Clause

The `HOST` clause of the `CREATE PLUGGABLE DATABASE` statement specifies the host name of the listener for the PDB being created.

By default, the host name of the listener is the same as the host name of the PDB being created. Specify the `HOST` clause when both of the following conditions are true:

- The host name of the listener is different from the host name of the PDB being created.
- You plan to create proxy PDBs that reference the PDB being created.

A proxy PDB uses a database link to establish communication with its referenced PDB. After communication is established, the proxy PDB communicates directly with the referenced PDB without using a database link. The host name of the listener must be correct for the proxy PDB to function properly.

Example 10-1 HOST Clause

```
HOST='myhost.example.com'
```

See Also:

- "About Creating a Proxy PDB"
- "Altering the Listener Host Name of a Referenced PDB"
- *Oracle Database SQL Language Reference* to learn more about the `HOST` clause

PORT Clause

The `PORT` clause of the `CREATE PLUGGABLE DATABASE` statement specifies the port number of the listener for the PDB being created.

By default, the port number of the listener for the PDB being created is 1521. Specify the `PORT` clause when both of the following conditions are true:

- The port number of the listener is not 1521.
- You plan to create proxy PDBs that reference the PDB being created.

A proxy PDB uses a database link to establish communication with its referenced PDB. After communication is established, the proxy PDB communicates directly with the referenced PDB without using a database link. The port number of the listener must be correct for the proxy PDB to function properly.

Example 10-2 PORT Clause

```
PORT=1599
```

> **Note:**
>
> - "About Creating a Proxy PDB"
> - "Altering the Listener Host Name of a Referenced PDB"
> - *Oracle Database SQL Language Reference* to learn more about the `PORT` clause

Creating a Proxy PDB

Create a proxy PDB by referencing a PDB in a different CDB.

Prerequisites

The following prerequisites must be met:

- Complete the prerequisites described in "General Prerequisites for PDB Creation".
- The current user must have the `CREATE PLUGGABLE DATABASE` system privilege in the root of the CDB in which the proxy PDB is being created.
- The CDB that contains the referenced PDB must be in local undo mode.
- The CDB that contains the referenced PDB must be in `ARCHIVELOG` mode.
- The referenced PDB must be in open read/write mode when the proxy PDB is created. The open mode of the referenced PDB can be changed after the proxy PDB is created.
- A database link must enable a connection from the root of the CDB in which the proxy PDB is being created to the location of the referenced PDB. The database link can connect to either the root of the remote CDB or to the remote PDB.
- If the database link connects to the root in a remote CDB that contains the referenced PDB, then the user that the database link connects with must be a common user.
- If the database link connects to the referenced PDB, then the user that the database link connects with in the referenced PDB must have the `CREATE PLUGGABLE DATABASE` system privilege.

- If you are creating a proxy PDB in an application container, then the following prerequisites apply:
  - The referenced PDB must be an application root or an application PDB in an application container.
  - The application name and version of the proxy PDB's application container must match the application name and version of the referenced PDB.
  - When the proxy PDB is being created in an application container, a database link must enable a connection from the root of the application container in which the proxy PDB is being created to the location of the referenced PDB. The database link can connect to either the root of the remote application container or to the remote application PDB.
  - If the database link connects to the root in a remote application container that contains the referenced PDB, then the user that the database link connects with must be an application common user.
  - If the database link connects to the referenced application PDB, then the user that the database link connects with in the referenced application PDB must have the `CREATE PLUGGABLE DATABASE` system privilege.

> **Note:**
>
> You can create a proxy PDB in a CDB root that is based on a referenced PDB in an application container.

To create a proxy PDB:

1. In SQL*Plus, ensure that the current container is the CDB root or application root in which the proxy PDB is being created.

   When the current container is the CDB root, the proxy PDB is created in the CDB. When the current container is an application root, the proxy PDB is created in the application container.

2. Run the `CREATE PLUGGABLE DATABASE` statement. Specify the `AS PROXY` clause, and specify the referenced PDB with the database link name in the `FROM` clause. Specify other clauses when they are required.

   After you create the proxy PDB, it is in mounted mode, and its status is `NEW`. You can view the open mode of a PDB by querying the `OPEN_MODE` column in the `V$PDBS` view. You can view the status of a PDB by querying the `STATUS` column of the `CDB_PDBS` or `DBA_PDBS` view.

   A new default service is created for the PDB. The service has the same name as the PDB and can be used to access the PDB. Oracle Net Services must be configured properly for clients to access this service.

3. Open the new PDB in read/write mode.

   You must open the new PDB in read/write mode for Oracle Database to complete the integration of the new PDB into the CDB. An error is returned if you attempt to open the PDB in read-only mode. After the PDB is opened in read/write mode, its status is `NORMAL`.

4. Back up the PDB.

   A PDB cannot be recovered unless it is backed up.

> **Note:**
>
> If an error is returned during creation of the proxy PDB, then the PDB being created might be in an `UNUSABLE` state. You can check a PDB's state by querying the `CDB_PDBS` or `DBA_PDBS` view, and you can learn more about PDB creation errors by checking the alert log. An unusable PDB can only be dropped, and it must be dropped before a PDB with the same name as the unusable PDB can be created.

Example 10-3 Creating a Remote Proxy PDB

In this example, the root to which the new PDB belongs depends on the current container when the `CREATE PLUGGABLE DATABASE` statement is run:

- When the current container is the CDB root, the new PDB is created in the CDB root.
- When the current container is an application root in an application container, the new PDB is created as an application PDB in the application root.

This example creates a remote proxy PDB named `pdb1` given different factors. This example assumes the following factors:

- The database link name to the referenced PDB's CDB is `pdb1_link`.
- The `FILE_NAME_CONVERT` clause and the `CREATE_FILE_DEST` clause are not required.

  Either Oracle Managed Files is enabled, or the `PDB_FILE_NAME_CONVERT` initialization parameter is set. The `SYSTEM` and `SYSAUX` files will be copied to a new location based on the Oracle Managed Files configuration or the initialization parameter setting.

Given the preceding factors, the following statement creates the `pdb1` proxy PDB:

```
CREATE PLUGGABLE DATABASE pdb1 AS PROXY FROM pdb1@pdb1_link;
```

> **See Also:**
>
> - "About the CDB Undo Mode"
> - "About Container Access in a CDB"
> - "Modifying the Open Mode of PDBs"
> - *Oracle Database Backup and Recovery User's Guide* for information about backing up a PDB

11
Removing a PDB

You can remove a plugged-in PDB from a CDB by unplugging it, dropping it, or relocating it.

> **See Also:**
>
> "Relocating a PDB"

Unplugging a PDB from a CDB

Just as you can plug a PDB into a CDB, you can unplug a PDB from a CDB.

About Unplugging a PDB

Unplugging a PDB disassociates the PDB from a CDB. A PDB is usable only when it is plugged into a CDB.

Unplug a PDB when you want to do any of the following:

- Move the PDB to a different CDB
- Archive the PDB for later use
- Make the PDB unavailable for use

To unplug a PDB, connect to its CDB root or application root and use the `ALTER PLUGGABLE DATABASE` statement to specify either of the following:

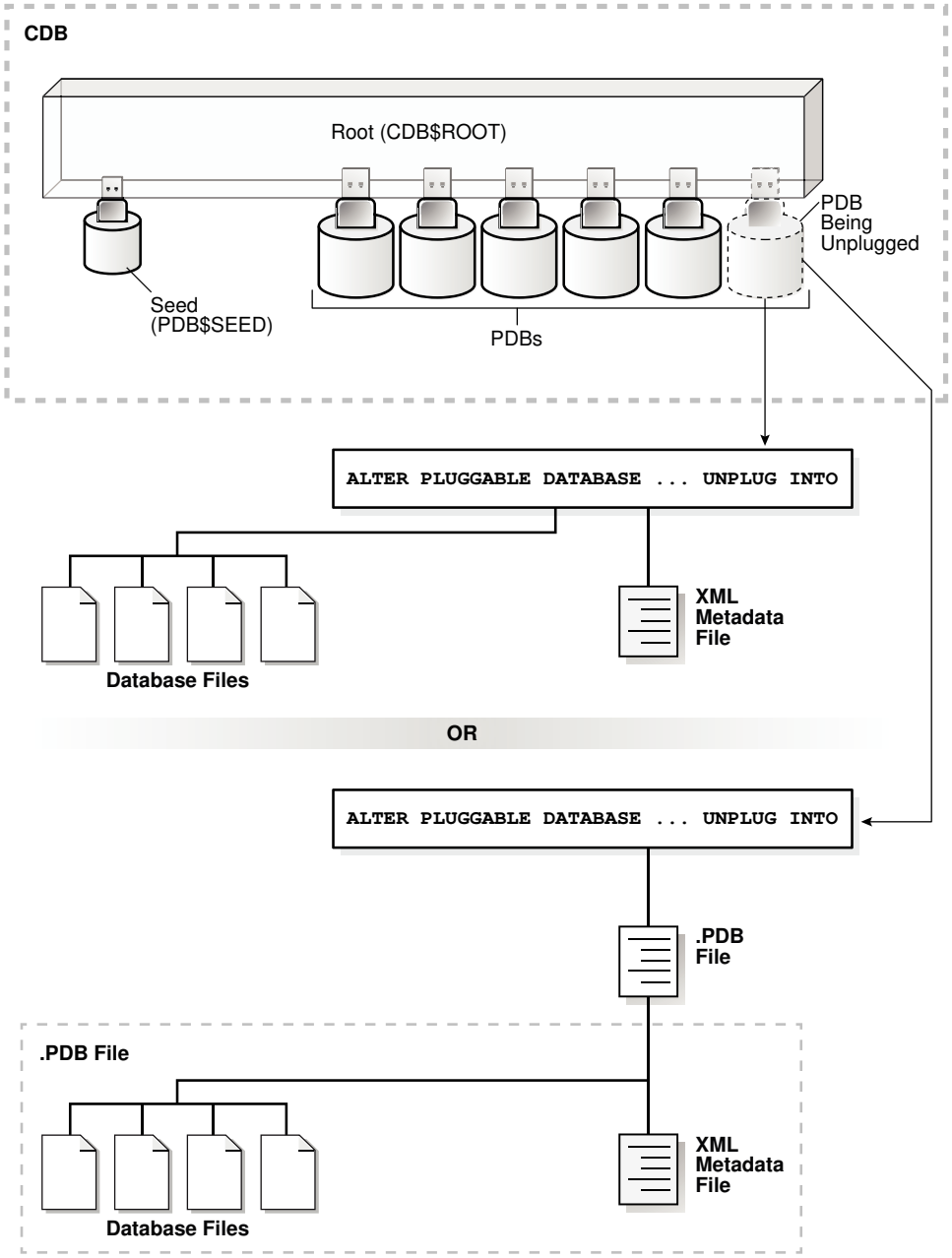
- XML file

  An XML file (.xml extension) contains metadata about the PDB after it is unplugged. This metadata contains the required information to enable a `CREATE PLUGGABLE DATABASE` statement on a target CDB to plug in the PDB.

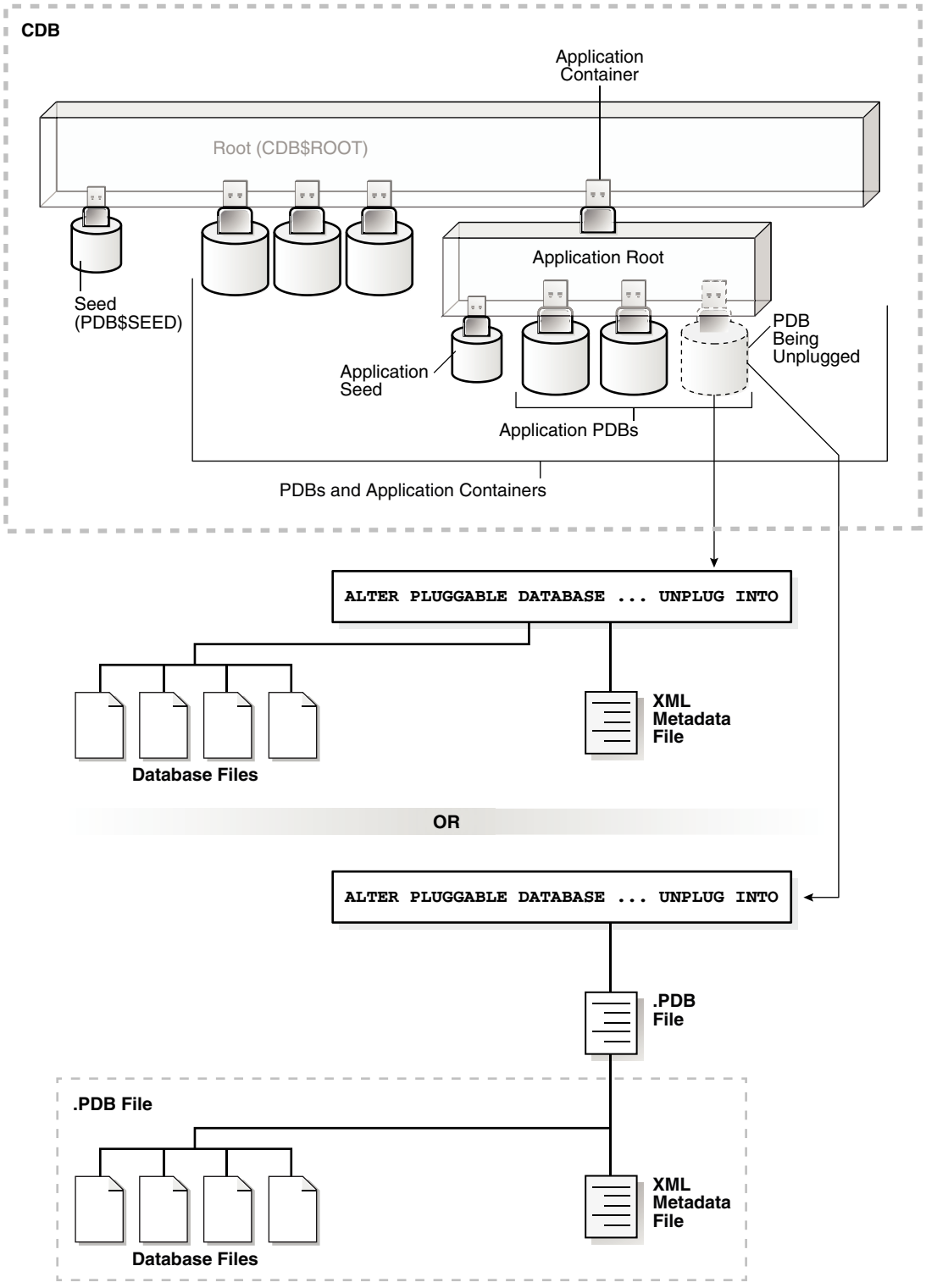
- .pdb file

  A .pdb file contains a compressed archive of the XML file that describes the PDB and the files used by the PDB (such as the data files and wallet file). A .pdb file enables you to copy a single, compressed file (instead of multiple files) to a new location to plug the PDB into a CDB.

Figure 11-1 Unplug a PDB



The following illustration shows how this technique unplugs an application PDB from an application container.



The PDB must be closed before it can be unplugged. When you unplug a PDB, the unplugged PDB is in mounted mode. The unplug operation makes some changes in the PDB's data files to record, for example, that the PDB was successfully unplugged. Because it is still part of the CDB, the unplugged PDB is included in an RMAN backup

of the entire CDB. Such a backup provides a convenient way to archive the unplugged PDB in case it is needed in the future.

To completely remove the PDB from the CDB, drop the PDB. The only operation supported on an unplugged PDB is dropping the PDB. The PDB must be dropped from the CDB before it can be plugged back into the same CDB.

> **Note:**
>
> You can unplug an application container only if no application PDBs belong to it.

> **See Also:**
>
> - "Dropping a PDB"
> - "Modifying the Open Mode of PDBs" for information about closing a PDB
> - "Modifying a PDB at the System Level" for information about initialization parameters and unplugged PDBs
> - *Oracle Database Security Guide* for information about common users and local users

Unplugging a PDB

Unplug a PDB with a `ALTER PLUGGABLE DATABASE ... UNPLUG INTO` statement.

Prerequisites

The following prerequisites must be met:

- The current user must have `SYSDBA` or `SYSOPER` administrative privilege, and the privilege must be either commonly granted or locally granted in the PDB. The user must exercise the privilege using `AS SYSDBA` or `AS SYSOPER` at connect time.
- The PDB must have been opened at least once.

> **Note:**
>
> If you are unplugging a PDB that includes data that was encrypted with Transparent Data Encryption, then follow the instructions in *Oracle Database Advanced Security Guide*.

To unplug a PDB:

1. In SQL*Plus, ensure that the current container is the root of the PDB.

If the PDB is plugged into the CDB root, then the current container must be the CDB root. If the PDB is plugged into an application root, then the current container must be the application root.

If you are unplugging an application container, then the current container must be the CDB root, and the application container must not have any application PDBs plugged into it.

2. Close the PDB.

   In an Oracle Real Application Clusters (Oracle RAC) environment, the PDB must be closed on all instances.

3. Run the `ALTER PLUGGABLE DATABASE` statement with the `UNPLUG INTO` clause, and specify the PDB to unplug and the name and location of the PDB's XML metadata file or .pdb file.

Example 11-1 Unplugging PDB salespdb Into an XML Metadata File

This `ALTER PLUGGABLE DATABASE` statement unplugs the PDB `salespdb` and creates the `salespdb.xml` metadata file in the `/oracle/data/` directory:

```
ALTER PLUGGABLE DATABASE salespdb UNPLUG INTO '/oracle/data/
salespdb.xml';
```

Example 11-2 Unplugging PDB salespdb Into an Archive File

This `ALTER PLUGGABLE DATABASE` statement unplugs the PDB `salespdb` and creates the `sales.pdb` archive file in the `/oracle/data/` directory. The `sales.pdb` archive file is a compressed file that includes the XML metadata file and the PDB's files (such as the data files and wallet file).

```
ALTER PLUGGABLE DATABASE salespdb UNPLUG INTO '/oracle/data/sales.pdb';
```

Dropping a PDB

Drop a PDB when you want to move the PDB to a new CDB or when you no longer need it.

When you drop a PDB, the control file of the CDB is modified to eliminate all references to the dropped PDB. Archived redo log files and backups associated with the PDB are not removed, but you can use Oracle Recovery Manager (RMAN) to remove them.

When dropping a PDB, you can either keep or delete the PDB's data files by using one of the following clauses of the `DROP PLUGGABLE DATABASE` statement:

- `KEEP DATAFILES`, the default, retains the data files.

  The PDB temp file is removed even when `KEEP DATAFILES` is specified because the temp file is no longer needed.

  When `KEEP DATAFILES` is specified, the PDB must be unplugged.

- `INCLUDING DATAFILES` removes the data files from disk.

  If a PDB was created with the `SNAPSHOT COPY` clause, then you must specify `INCLUDING DATAFILES` when you drop the PDB.

Prerequisites

The following prerequisites must be met:

- The PDB must be in mounted mode, or it must be unplugged.

  See "Modifying the Open Mode of PDBs".

  See "Unplugging a PDB from a CDB".

- The current user must have `SYSDBA` or `SYSOPER` administrative privilege, and the privilege must be either commonly granted or locally granted in the PDB. The user must exercise the privilege using `AS SYSDBA` or `AS SYSOPER` at connect time.

> **Note:**
>
> This operation is destructive.

To drop a PDB:

1. In SQL*Plus, ensure that the current container is the CDB root, or, for an application PDB, the application root that contains the application PDB.

   If the PDB is plugged into the CDB root, then the current container must be the CDB root. If the PDB is plugged into an application root, then the current container must be that application root or the CDB root.

   If you are dropping an application container, then the current container must be the CDB root, and the application container must not have any application PDBs plugged into it.

2. Run the `DROP PLUGGABLE DATABASE` statement and specify the PDB to drop.

Example 11-3 Dropping PDB salespdb While Keeping Its Data Files

```
DROP PLUGGABLE DATABASE salespdb
  KEEP DATAFILES;
```

Example 11-4 Dropping PDB salespdb and Its Data Files

```
DROP PLUGGABLE DATABASE salespdb
  INCLUDING DATAFILES;
```

> **See Also:**
>
> - "Unplugging a PDB from a CDB"
> - "Storage Requirements for Snapshot Copy PDBs"
> - *Oracle Database SQL Language Reference*
> - *Oracle Database Backup and Recovery User's Guide* for information about RMAN

12

Creating and Removing Application Containers and Seeds

You can create application containers and application seeds in several different ways. You can also remove application containers from a CDB, and you can remove application seeds from application containers.

> **See Also:**
>
> - "About Application Containers"
> - "Administering Application Containers"

Creating and Removing Application Containers

You can create application containers in several different ways, including using the PDB seed, cloning an existing PDB or non-CDB, and plugging in an unplugged PDB. You can also remove application containers from a CDB.

Creating Application Containers

You can use the `CREATE PLUGGABLE DATABASE` statement to create an application container in a CDB.

About Creating an Application Container

The `CREATE PLUGGABLE DATABASE ... AS APPLICATION CONTAINER` statement creates a new application container.

An application container consists of an application root and a collection of application PDBs that store data for one or more applications. The application PDBs are plugged into the application root, and you can optionally create an application seed for quick and easy creation of new application PDBs. The application PDBs and application root can share application common objects.

There are three types of application common objects:

- Metadata-linked application common objects store the metadata for specific objects, such as tables, so that the containers that share the application common object have the same structure but different data.
- Data-linked application common objects are defined once in the application root and shared as read-only objects in the context of hosted application PDBs.

- Extended data-linked application common objects store shared data in the application root but also allow application PDBs to store data appended to that object. The appended data is local data that is unique to each application PDB.

You create an application container by including the `AS APPLICATION CONTAINER` clause in the `CREATE PLUGGABLE DATABASE` statement. You can use the following techniques to create an application container:

- Using the PDB seed
- Cloning an existing PDB or non-CDB
- Relocating a PDB
- Plugging in an unplugged PDB

To create an application container, the current container must be the CDB root and you must specify the `AS APPLICATION CONTAINER` clause in the `CREATE PLUGGABLE DATABASE` statement. You must create the application container using Oracle Managed Files.

> **Note:**
>
> An application container cannot be unplugged or dropped if any application PDBs belong to it.

Migrating Existing Applications to an Application Container

You can migrate an application to an application root by creating an application root using an existing PDB. You must complete additional tasks when you are migrating an existing application to an application container. The PDBs that you plug in must contain the application objects, including their data, and you must run procedures in the `DBMS_PDB` package to specify which objects are shared. Also, when application common users, roles, or profiles exist in the application root, you must run procedures in the `DBMS_PDB` package to specify that they are common.

After the application is migrated to the application root, you can create application PDBs in the application root, and create application PDBs using existing PDBs.

> **See Also:**
>
> "Migrating an Existing Application to an Application Container"

Preparing for Application Containers

Prerequisites must be met before creating an application container.

- The CDB must exist.
- The CDB must be in read/write mode.
- The current user must be a common user whose current container is the CDB root.
- The current user must have the `CREATE PLUGGABLE DATABASE` system privilege.

- You must decide on a unique application container name for every application container. Every application container name must be unique with respect to all containers in a single CDB, and every application container name must be unique within the scope of all the CDBs whose database instances are reached through a specific listener.

  The application container name is used to distinguish an application container from other containers in the CDB. Application container names follow the same rules as service names, which includes being case-insensitive.
- You must create the containing using Oracle Managed Files.
- If you are creating an application container in an Oracle Data Guard configuration with a physical standby database, then additional tasks must be completed before creating an application container.
- If you are migrating an existing application to an application container using installation scripts, then the scripts must be available to run.
- If you are migrating an existing application to an application container using a PDB, then it must be possible to clone the PDB to the application root or plug in the PDB into the application root.

See Also:

- "About the Current Container"
- "Migrating an Existing Application to an Application Container"
- *Oracle Data Guard Concepts and Administration*

Creating an Application Container

You can create an application container using the `CREATE PLUGGABLE DATABASE` statement with the `AS APPLICATION CONTAINER` clause.

Before creating an application container, complete the prerequisites described in "Preparing for Application Containers".

1. In SQL*Plus, ensure that the current container is the CDB root.
2. Run the `CREATE PLUGGABLE DATABASE` statement, and include the `AS APPLICATION CONTAINER` clause. Specify other clauses when they are required.

   After you create the application container, it is in mounted mode, and its status is `NEW`. You can view the open mode of an application container by querying the `OPEN_MODE` column in the `V$PDBS` view. You can view the status of an application container by querying the `STATUS` column of the `CDB_PDBS` or `DBA_PDBS` view.

   A new default service is created for the application container. The service has the same name as the application container and can be used to access the application container. Oracle Net Services must be configured properly for clients to access this service.
3. Open the new application container in read/write mode.

   You must open the new application container in read/write mode for Oracle Database to complete the integration of the new application container into the

CDB. An error is returned if you attempt to open the application container in read-only mode. After the application container is opened in read/write mode, its status is NORMAL.

4. Back up the application container.

   A application container cannot be recovered unless it is backed up.

> **Note:**
>
> If an error is returned during application container creation, then the application container being created might be in an UNUSABLE state. You can check an application container's state by querying the CDB_PDBS or DBA_PDBS view, and you can learn more about application container creation errors by checking the alert log. An unusable application container can only be dropped, and it must be dropped before an application container or PDB with the same name as the unusable application container can be created.

5. If you are migrating an existing application to the application container, then follow the instructions in "Migrating an Existing Application to an Application Container".

The application container is created with an application root. You can create application PDBs in the application container.

Example 12-1 Creating an Application Container Using the PDB seed

This example assumes the following factors:

- Storage limits are not required for the application container. Therefore, the STORAGE clause is not required.
- The application container does not require a default tablespace.
- The PATH_PREFIX clause is not required.
- The FILE_NAME_CONVERT clause and the CREATE_FILE_DEST clause are not required.

  Either Oracle Managed Files is enabled for the CDB, or the PDB_FILE_NAME_CONVERT initialization parameter is set. The files associated with the PDB seed will be copied to a new location based on the Oracle Managed Files configuration or the initialization parameter setting.
- There is no file with the same name as the new temp file that will be created in the target location. Therefore, the TEMPFILE REUSE clause is not required.
- No predefined Oracle roles need to be granted to the PDB_DBA role.

The following statement creates the application container from the PDB seed:

```
CREATE PLUGGABLE DATABASE salesact AS APPLICATION CONTAINER
  ADMIN USER salesadm IDENTIFIED BY password;
```

Example 12-2 Creating an Application Container by Cloning a Local PDB

This example assumes the following factors:

- The PATH_PREFIX clause is not required.

- The FILE_NAME_CONVERT clause is required to specify the target locations of the copied files. In this example, the files are copied from /disk1/oracle/pdb1/ to /disk2/oracle/hract/.

  The CREATE_FILE_DEST clause is not used, and neither Oracle Managed Files nor the PDB_FILE_NAME_CONVERT initialization parameter is used to specify the target locations of the copied files.

  To view the location of the data files for a PDB, run the query in "Example 19-7".
- Storage limits must be enforced for the application root. Therefore, the STORAGE clause is required. Specifically, all tablespaces that belong to the application root must not exceed 2 gigabytes. This storage limit does not apply to the application PDBs that are plugged into the application root.
- There is no file with the same name as the new temp file that will be created in the target location. Therefore, the TEMPFILE REUSE clause is not required.

Given the preceding factors, the following statement clones hract as an application container from pdb1:

```
CREATE PLUGGABLE DATABASE hract AS APPLICATION CONTAINER FROM pdb1
  FILE_NAME_CONVERT = ('/disk1/oracle/pdb1/', '/disk2/oracle/hract/')
  STORAGE (MAXSIZE 2G);
```

> **Note:**
>
> If you are migrating an existing application to the new application container, then follow the instructions in "Migrating an Existing Application to an Application Container".

Example 12-3 Creating an Application Container by Plugging In an Unplugged PDB

This example assumes the following factors:

- The new application container is not based on the same unplugged PDB that was used to create an existing PDB or application container in the CDB. Therefore, the AS CLONE clause is not required.
- The PATH_PREFIX clause is not required.
- The XML file does not accurately describe the current locations of the files. Therefore, the SOURCE_FILE_NAME_CONVERT clause or SOURCE_FILE_DIRECTORY clause is required. In this example, the XML file indicates that the files are in /disk1/oracle/payroll/, but the files are in /disk2/oracle/payroll/, and the SOURCE_FILE_NAME_CONVERT clause is used.
- The files are in the correct location. Therefore, NOCOPY is included.
- Storage limits must be enforced for the application container. Therefore, the STORAGE clause is required. Specifically, all tablespaces that belong to the application container must not exceed 2 gigabytes.
- A file with the same name as the temp file specified in the XML file exists in the target location. Therefore, the TEMPFILE REUSE clause is required.

The following statement plugs in the PDB:

```
CREATE PLUGGABLE DATABASE payrollact AS APPLICATION CONTAINER
  USING '/disk1/usr/payrollpdb.xml'
  SOURCE_FILE_NAME_CONVERT = ('/disk1/oracle/payroll/',
                              '/disk2/oracle/payroll/')
  NOCOPY
  STORAGE (MAXSIZE 2G)
  TEMPFILE REUSE;
```

> **Note:**
>
> If you are migrating an existing application to the new application container, then follow the instructions in "Migrating an Existing Application to an Application Container".

Related Topics

- About the Current Container
  The data dictionary in each container in a CDB is separate, and the current container is the container whose data dictionary is used for name resolution and for privilege authorization.
- Administering Application Containers
  You can administer application containers, including application roots and application PDBs. You can also administer the applications installed in application containers.
- Accessing a Container in a CDB
  Access a container in a CDB with SQL*Plus by issuing a `CONNECT` or `ALTER SESSION` command.
- Modifying the Open Mode of PDBs
  You can modify the open mode of a PDB by using the `ALTER PLUGGABLE DATABASE` SQL statement or the SQL*Plus `STARTUP` command.

Unplugging an Application Container from a CDB

You can unplug an application container from a CDB.

About Unplugging an Application Container

Unplugging an application container disassociates the application container from a CDB.

Typically, you unplug an application container when you want to move the application container to a different CDB. Also, you can unplug the application container when you no longer want it to be available.

Unplugging an application container is similar to unplugging a PDB. To unplug an application container, connect to its CDB root and use the `ALTER PLUGGABLE DATABASE` statement to specify an XML file or a `.pdb` file. When you specify an XML file (`.xml` extension), it will contain metadata about the application container after it is

unplugged. The SQL statement creates the XML file, and it contains the required information to enable a `CREATE PLUGGABLE DATABASE` statement on a target CDB to plug in the application container. When you specify a `.pdb` file, it contains a compressed archive of the XML file that describes the application container and the files used by the application container (such as the data files and wallet file). A `.pdb` file enables you to copy a single, compressed file (instead of multiple files) to a new location to plug the application container into a CDB.

Before it can be unplugged, the application container must not have any application PDBs plugged into it, and it must be closed. When you unplug an application container, the unplugged application container is in mounted mode. The unplug operation makes some changes in the application container's data files to record, for example, that the application container was successfully unplugged. Because it is still part of the CDB, the unplugged application container is included in an RMAN backup of the entire CDB. Such a backup provides a convenient way to archive the unplugged application container in case it is needed in the future.

To completely remove the application container from the CDB, you can drop it. The only operation supported on an unplugged application container is dropping the application container. The application container must be dropped from the CDB before it can be plugged back into the same CDB. An application container is usable only when it is plugged into a CDB.

See Also:

- "Unplugging a PDB from a CDB"
- "Dropping an Application Container"
- "Modifying the Open Mode of PDBs" for information about closing a PDB
- "Modifying a PDB at the System Level" for information about initialization parameters and unplugged PDBs
- *Oracle Database Security Guide* for information about common users and local users

Unplugging an Application Container

Unplug an application container by using an `ALTER PLUGGABLE DATABASE ... UNPLUG INTO` statement.

Prerequisites

You must meet the following prerequisites:

- The current user must have `SYSDBA` or `SYSOPER` administrative privilege, and the privilege must be either commonly granted or locally granted in the PDB. The user must exercise the privilege using `AS SYSDBA` or `AS SYSOPER` at connect time.
- The application container must have been opened at least once.
- The application container must not have any application PDBs plugged into it.
- The application container must not have an application seed plugged into it.

> **Note:**
>
> If you are unplugging an application container that includes data that was encrypted with Transparent Data Encryption, then follow the instructions in *Oracle Database Advanced Security Guide*.

To unplug an application container:

1. In SQL*Plus, ensure that the current container is the root of the CDB.
2. Close the application container.

   In an Oracle Real Application Clusters (Oracle RAC) environment, the application container must be closed on all instances.
3. Run the `ALTER PLUGGABLE DATABASE` statement with the `UNPLUG INTO` clause, and specify the application container to unplug and the name and location of the application container's XML metadata file or .pdb file.

Example 12-4 Unplugging Application Container salesact

This `ALTER PLUGGABLE DATABASE` statement unplugs the application container `salesact` and creates the `salesact.xml` metadata file in the `/oracle/data/directory`:

```
ALTER PLUGGABLE DATABASE salesact UNPLUG INTO '/oracle/data/
saleact.xml';
```

Dropping an Application Container

You can drop an application container when you want to move the application container from one CDB to another or when you no longer need the application container.

Dropping an application container is very similar to dropping a PDB. When you drop an application container, the control file of the CDB is modified to eliminate all references to the dropped application container. Archived redo log files and backups associated with the application container are not removed, but you can use Oracle Recovery Manager (RMAN) to remove them.

When dropping an application container, you can either keep or delete the application container's data files by using one of the following clauses in the `DROP PLUGGABLE DATABASE` statement:

- `KEEP DATAFILES`, the default, retains the data files.

  The application container's temp file is removed even when `KEEP DATAFILES` is specified because the temp file is no longer needed.
- `INCLUDING DATAFILES` removes the data files from disk.

  If an application container was created with the `SNAPSHOT COPY` clause, then you must specify `INCLUDING DATAFILES` when you drop the application container.

The following prerequisites must be met:

- The application container must be in mounted mode, or it must be unplugged.

See "Modifying the Open Mode of PDBs".

See "Unplugging an Application Container".

- The current user must have SYSDBA or SYSOPER administrative privilege, and the privilege must be either commonly granted or locally granted in the application container. The user must exercise the privilege using AS SYSDBA or AS SYSOPER at connect time.
- The application container must not have any application PDBs plugged into it.
- The application container must not have an application seed plugged into it.

> **Note:**
>
> This operation is destructive.

To drop an application container:

1. In SQL*Plus, ensure that the current container is the CDB root.

   See "About the Current Container" and "Accessing a Container in a CDB with SQL*Plus".

2. Run the DROP PLUGGABLE DATABASE statement and specify the application container to drop.

Example 12-5 Dropping Application Container salesact While Keeping Its Data Files

```
DROP PLUGGABLE DATABASE salesact
  KEEP DATAFILES;
```

Example 12-6 Dropping Application Container salesact and Its Data Files

```
DROP PLUGGABLE DATABASE saleact
  INCLUDING DATAFILES;
```

> **See Also:**
>
> - "Unplugging an Application Container"
> - "Dropping a PDB"
> - "Storage Requirements for Snapshot Copy PDBs"
> - *Oracle Database SQL Language Reference*
> - *Oracle Database Backup and Recovery User's Guide* for information about RMAN

Creating and Removing Application Seeds

You can create application seeds in several different ways, including using the PDB seed, cloning an existing PDB or non-CDB, and plugging in an unplugged PDB. You can also remove application seeds from application containers.

Creating Application Seeds

You can use the `CREATE PLUGGABLE DATABASE` statement to create an application seed in an application container.

About Creating an Application Seed

To create a new application seed in an application container, use the `CREATE PLUGGABLE DATABASE` statement with the `AS SEED` clause.

You can use an application seed to provision an application container with application PDBs that have the application root's applications installed. Typically, the application container's applications are installed in the application root before seed creation. After the application seed is created, it is synchronized with the application root so that the applications are installed in the application seed. When that is complete, any PDBs created using the application seed have the applications installed. When an application in the application root is upgraded or patched, the application seed must be synchronized with the application root to apply these changes.

An application container can have zero or one application seeds. When you create an application seed using the `AS SEED` clause of `CREATE PLUGGABLE DATABASE`, you do not specify its name. The application seed name is always *`application_container_name`*`$SEED`, where *`application_container_name`* is the name of the application seed's application container. For example, an application seed in the `salesact` application container must be named `salesact$SEED`.

When you create a new application seed, you must specify an administrator for the application container in the `CREATE PLUGGABLE DATABASE` statement. The statement creates the administrator as a local user in the application container and grants the `PDB_DBA` role locally to the administrator.

See Also:

- "Creating a PDB from Scratch"
- "Managing Applications in an Application Container"
- "Synchronizing Applications in an Application PDB"
- *Oracle Database SQL Language Reference* for syntax and semantics of the `AS SEED` clause

Preparing for an Application Seed

Prerequisites must be met before creating an application seed.

Ensure that the following prerequisites are met before creating an application seed:

- The CDB must exist.

  See Creating and Configuring a CDB.
- The CDB must be in read/write mode.
- The application container to which the application seed will belong must be in read/write mode.
- The current user must be a common user whose current container is the application root to which the application seed will belong.
- The current user must have the `CREATE PLUGGABLE DATABASE` system privilege.
- For the application seed to include the application for the application container, the application must be installed in the application root.

> **See Also:**
>
> - "About the Current Container"
> - Managing Applications in an Application Container

Creating an Application Seed

You create an application seed by including the `AS SEED` clause in the `CREATE PLUGGABLE DATABASE` statement.

An application seed in an application container is similar to the seed in a CDB. An application seed enables you to create application PDBs that meet the requirements of an application container quickly and easily.

Before creating an application seed, complete the prerequisites described in "Preparing for an Application Seed".

1. In SQL*Plus, ensure that the current container is the application root.
2. Run the `CREATE PLUGGABLE DATABASE` statement, and include the `AS SEED` clause, to create the application seed. Specify other clauses when they are required.

   After you create the application seed, it is in mounted mode, and its status is `NEW`. You can view the open mode of an application seed by querying the `OPEN_MODE` column in the `V$PDBS` view. You can view the status of an application seed by querying the `STATUS` column of the `CDB_PDBS` or `DBA_PDBS` view.

   A new default service is created for the application seed. The service has the same name as the application seed and can be used to access the application seed. Oracle Net Services must be configured properly for clients to access this service.
3. Open the new application seed in read/write mode.

4. You must open the new application seed in read/write mode for Oracle Database to complete the integration of the new application seed into the application container. An error is returned if you attempt to open the application seed in read-only mode. After the application seed is opened in read/write mode, its status is `NORMAL`.
5. Perform one or more of the following actions:
   - If the application seed was created from the PDB seed, then switch container to the application seed, and use an `ALTER PLUGGABLE DATABASE` statement with the `SYNC` clause to synchronize the application seed. Synchronizing with the application root instantiates one or more of the application root's applications in the application seed.
   - If the application seed was created from an application root, then switch container to the application seed, and run the `pdb_to_apppdb.sql` script to convert the application root to an application PDB.

   These actions are not required when the application seed is created by cloning an application PDB.
6. Close the application seed, and then open it in open read-only mode.
7. Back up the application seed.

   An application seed cannot be recovered unless it is backed up.

> **Note:**
>
> - If an error is returned during application seed creation, then the application seed being created might be in an `UNUSABLE` state. You can check an application seed's state by querying the `CDB_PDBS` or `DBA_PDBS` view, and you can learn more about application seed creation errors by checking the alert log. An unusable application seed can only be dropped.
> - When an application in the application root is upgraded or patched in the application root, the application seed must synchronize with the application root to include the changes.

Example 12-7 Creating an Application Seed from the PDB seed

This example assumes the following factors:

- The application seed is being created in an application container named `salesact`.
- Storage limits are not required for the application seed. Therefore, the `STORAGE` clause is not required.
- The application seed does not require a default tablespace.
- The `PATH_PREFIX` clause is not required.
- The `FILE_NAME_CONVERT` clause and the `CREATE_FILE_DEST` clause are not required.

  Either Oracle Managed Files is enabled for the CDB, or the `PDB_FILE_NAME_CONVERT` initialization parameter is set. The files associated with

the PDB seed will be copied to a new location based on the Oracle Managed Files configuration or the initialization parameter setting.

- There is no file with the same name as the new temp file that will be created in the target location. Therefore, the `TEMPFILE REUSE` clause is not required.
- No predefined Oracle roles need to be granted to the `PDB_DBA` role.

The following statement creates the application seed from the PDB seed, opens the application seed, switches containers to the application seed, synchronizes the application seed with the applications in the application root, closes the application seed, and then opens the application seed in open read-only mode:

```
CREATE PLUGGABLE DATABASE AS SEED
  ADMIN USER actseedadm IDENTIFIED BY password;
ALTER PLUGGABLE DATABASE salesact$SEED OPEN;
ALTER SESSION SET CONTAINER=salesact$SEED;
ALTER PLUGGABLE DATABASE APPLICATION ALL SYNC;
ALTER PLUGGABLE DATABASE CLOSE IMMEDIATE;
ALTER PLUGGABLE DATABASE OPEN READ ONLY;
```

Because the application container name is `salesact`, the application seed name is `salesact$SEED`.

A local user with the name of the specified local administrator is created and granted the `PDB_DBA` common role locally in the application seed. If this user was not granted administrator privileges during application seed creation, then use the `SYS` and `SYSTEM` common users to administer to the application seed.

The application seed was synchronized with the application root when it was created. Therefore, the application seed includes the applications installed in the application root and the application common objects that are part of those applications. When a new application PDB is created using the application seed, the application PDB also includes the installed applications and application common objects.

Example 12-8 Creating an Application Seed From an Application PDB

This example assumes the following factors:

- The application seed is being created in an application container named `salesact`.
- The application seed is being created in an application PDB in the application container named `salesapppdb`.
- Storage limits are not required for the application seed. Therefore, the `STORAGE` clause is not required.
- The application seed does not require a default tablespace.
- The `PATH_PREFIX` clause is not required.
- The `FILE_NAME_CONVERT` clause and the `CREATE_FILE_DEST` clause are not required.

  Either Oracle Managed Files is enabled for the CDB, or the `PDB_FILE_NAME_CONVERT` initialization parameter is set. The files associated with the application root will be copied to a new location based on the Oracle Managed Files configuration or the initialization parameter setting.
- There is no file with the same name as the new temp file that will be created in the target location. Therefore, the `TEMPFILE REUSE` clause is not required.

Given the preceding factors, the following statement creates the application seed from the application root, opens the application seed, closes the application seed, and opens the application seed in open read-only mode:

```
CREATE PLUGGABLE DATABASE AS SEED FROM salesapppdb;
ALTER PLUGGABLE DATABASE salesact$SEED OPEN;
ALTER PLUGGABLE DATABASE CLOSE IMMEDIATE;
ALTER PLUGGABLE DATABASE OPEN READ ONLY;
```

Because the application container name is `salesact`, the application seed name is `salesact$SEED`.

The application seed was created from an application PDB. Therefore, the application seed includes the applications installed in the application root and the application common objects that are part of those applications. When a new application PDB is created using the application seed, the application PDB also includes the installed applications and application common objects.

Example 12-9 Creating an Application Seed From an Application Root

This example assumes the following factors:

- The application seed is being created in an application container named `salesact`. The application seed is cloned from the root of the application container.
- Storage limits are not required for the application seed. Therefore, the `STORAGE` clause is not required.
- The application seed does not require a default tablespace.
- The `PATH_PREFIX` clause is not required.
- The `FILE_NAME_CONVERT` clause and the `CREATE_FILE_DEST` clause are not required.

  Either Oracle Managed Files is enabled for the CDB, or the `PDB_FILE_NAME_CONVERT` initialization parameter is set. The files associated with the application root will be copied to a new location based on the Oracle Managed Files configuration or the initialization parameter setting.
- There is no file with the same name as the new temp file that will be created in the target location. Therefore, the `TEMPFILE REUSE` clause is not required.

Given the preceding factors, the following statement creates the application seed from the application root, opens the application seed, switches containers to the application seed, runs the `pdb_to_apppdb.sql` script to convert the application root to an application PDB, closes the application seed, and opens the application seed in open read-only mode:

```
CREATE PLUGGABLE DATABASE AS SEED FROM salesact;
ALTER PLUGGABLE DATABASE salesact$SEED OPEN;
ALTER SESSION SET CONTAINER=salesact$SEED;
@$ORACLE_HOME/rdbms/admin/pdb_to_apppdb.sql
ALTER PLUGGABLE DATABASE CLOSE IMMEDIATE;
ALTER PLUGGABLE DATABASE OPEN READ ONLY;
```

Because the application container name is `salesact`, the application seed name is `salesact$SEED`.

The application seed was created from the application root. Therefore, the application seed includes the applications installed in the application root and the application common objects that are part of those applications. When a new application PDB is created using the application seed, the application PDB also includes the installed applications and application common objects.

Unplugging an Application Seed from an Application Container

You can unplug an application seed from an application container.

About Unplugging an Application Seed

Unplugging an application seed disassociates the application seed from an application container. You unplug an application seed when you no longer want the application seed to be available.

Unplugging an application seed is similar to unplugging a PDB. To unplug an application seed, connect to its application root and use the `ALTER PLUGGABLE DATABASE` statement to specify an XML file or a `.pdb` file. When you specify an XML file (`.xml` extension), it will contain metadata about the application seed after it is unplugged. The SQL statement creates the XML file, and it contains the required information to enable a `CREATE PLUGGABLE DATABASE` statement on a target CDB to plug it in as a PDB or an application PDB. When you specify a .pdb file, it contains a compressed archive of the XML file that describes the application seed and the files used by the application seed (such as the data files and wallet file). A `.pdb` file enables you to copy a single, compressed file (instead of multiple files) to a new location to plug in as a PDB or an application PDB.

Before it can be unplugged, the application seed must be closed. When you unplug an application seed, the unplugged application seed is in mounted mode. The unplug operation makes some changes in the application seed's data files to record, for example, that the application seed was successfully unplugged. Because it is still part of the application container, the unplugged application seed is included in an RMAN backup of the entire CDB. Such a backup provides a convenient way to archive the unplugged application seed in case it is needed in the future.

To completely remove the application seed from the application container, you can drop it. The only operation supported on an unplugged application seed is dropping the application seed. The application seed must be dropped from the application container before it can be plugged back into the same application container. An application seed is usable only when it is plugged into an application container.

> **See Also:**
>
> - "Unplugging a PDB from a CDB"
> - "Dropping an Application Seed"
> - "Modifying the Open Mode of PDBs" for information about closing a PDB
> - "Modifying a PDB at the System Level" for information about initialization parameters and unplugged PDBs
> - *Oracle Database Security Guide* for information about common users and local users

Unplugging an Application Seed

To unplug an application seed, run the `ALTER PLUGGABLE DATABASE ... UNPLUG INTO` statement.

Prerequisites

The following prerequisites must be met:

- The current user must have `SYSDBA` or `SYSOPER` administrative privilege, and the privilege must be either commonly granted or locally granted in the application container. The user must exercise the privilege using `AS SYSDBA` or `AS SYSOPER` at connect time.
- The application seed must have been opened at least once.

> **Note:**
>
> If you are unplugging an application seed that includes data that was encrypted with Transparent Data Encryption, then follow the instructions in *Oracle Database Advanced Security Guide*.

To unplug an application seed:

1. In SQL*Plus, ensure that the current container is the application root of the application container to which the application seed belongs.
2. Close the application seed.

   In an Oracle Real Application Clusters (Oracle RAC) environment, the application seed must be closed on all instances.
3. Run the `ALTER PLUGGABLE DATABASE` statement with the `UNPLUG INTO` clause, and specify the application seed to unplug and the name and location of the application seed's XML metadata file or .pdb file.

Example 12-10 Unplugging Application Seed salesact$SEED

This `ALTER PLUGGABLE DATABASE` statement unplugs the application seed `salesact$SEED` and creates the `salesact$SEED.xml` metadata file in the `/oracle/data/` directory:

```
ALTER PLUGGABLE DATABASE salesact$SEED
  UNPLUG INTO '/oracle/data/saleact$SEED.xml';
```

Dropping an Application Seed

You can use the `DROP PLUGGABLE DATABASE` statement to drop an application seed. You can drop an application seed when you no longer need it.

When you drop an application seed, the control file of the CDB is modified to eliminate all references to the dropped application seed. Archived redo log files and backups associated with the application seed are not removed, but you can use Oracle Recovery Manager (RMAN) to remove them.

When dropping an application seed, you can either keep or delete the application seed's data files by using one of the following clauses:

- `KEEP DATAFILES`, the default, retains the data files.

  The application seed's temp file is removed even when `KEEP DATAFILES` is specified because the temp file is no longer needed.

- `INCLUDING DATAFILES` removes the data files from disk.

  If an application seed was created with the `SNAPSHOT COPY` clause, then you must specify `INCLUDING DATAFILES` when you drop the application seed.

The following prerequisites must be met:

- The application seed must be in mounted mode, or it must be unplugged.
- The current user must have `SYSDBA` or `SYSOPER` administrative privilege, and the privilege must be either commonly granted or locally granted in the application container. The user must exercise the privilege using `AS SYSDBA` or `AS SYSOPER` at connect time.

> **Note:**
>
> This operation is destructive.

To drop an application seed:

1. In SQL*Plus, ensure that the current container is the application root of the application container to which the application seed belongs.
2. Run the `DROP PLUGGABLE DATABASE` statement and specify the application seed.

Example 12-11 Dropping Application Seed salesact$SEED While Keeping Its Data Files

```
DROP PLUGGABLE DATABASE salesact$SEED
  KEEP DATAFILES;
```

Example 12-12 Dropping Application Seed salesact$SEED and Its Data Files

```
DROP PLUGGABLE DATABASE saleact$SEED
  INCLUDING DATAFILES;
```

See Also:

- "About Container Access in a CDB"
- "Modifying the Open Mode of PDBs"
- "Unplugging an Application Seed"
- "Storage Requirements for Snapshot Copy PDBs"
- *Oracle Database SQL Language Reference*
- *Oracle Database Backup and Recovery User's Guide* for information about RMAN

Creating an Application PDB

You create an application PDB by running the `CREATE PLUGGABLE DATABASE` statement with an application root as the current container.

You can create application PDBs using the same SQL statements that you use to create PDBs in the CDB root. The newly created PDB is an application PDB when the `CREATE PLUGGABLE DATABASE` statement is run in an application root. The statement must be run in an application root and has an explicit dependency on the application database defined in that application root.

Before creating an application PDB, complete the prerequisites described in "General Prerequisites for PDB Creation". You must also complete the prerequisites for the specific type of PDB you are creating. For example, if you are cloning a PDB, then you must meet the prerequisites PDB cloning.

1. In SQL*Plus, ensure that the current container is the application root.
2. Run a `CREATE PLUGGABLE DATABASE` statement.

   After you create the application PDB, it is in mounted mode, and its status is `NEW`. You can view the open mode of an application PDB by querying the `OPEN_MODE` column in the `V$PDBS` view. You can view the status of an application PDB by querying the `STATUS` column of the `CDB_PDBS` or `DBA_PDBS` view.

   A new default service is created for the application PDB. The service has the same name as the application PDB and can be used to access the application PDB. Oracle Net Services must be configured properly for clients to access this service.

3. Open the new application PDB in read/write mode.
4. You must open the new application PDB in read/write mode for Oracle Database to complete the integration of the new application PDB into the application container. An error is returned if you attempt to open the application PDB in read-only mode. After the application PDB is opened in read/write mode, its status is `NORMAL`.
5. Switch container to the application PDB.
6. Use an `ALTER PLUGGABLE DATABASE` statement with the `SYNC` clause to synchronize the application PDB.

   Synchronizing with the application PDB instantiates one or more of the application root's applications in the application PDB.
7. Close the application PDB, and then open it in open read-only mode.
8. Back up the application PDB.

   An application PDB cannot be recovered unless it is backed up.

> **Note:**
>
> - If an error is returned during application PDB creation, then the application PDB being created might be in an `UNUSABLE` state. You can check an application PDB's state by querying the `CDB_PDBS` or `DBA_PDBS` view, and you can learn more about application PDB creation errors by checking the alert log. An unusable application PDB can only be dropped.
> - When an application in the application root is upgraded or patched in the application root, the application PDB must synchronize with the application root to include the changes.

Related Topics

- Creating and Removing PDBs and Application Containers
  You can create PDBs, application containers, and application seeds using a variety of techniques.
- Administering Application Containers
  You can administer application containers, including application roots and application PDBs. You can also administer the applications installed in application containers.

Part IV
Administering a Multitenant Environment

You can administer a multitenant environment using SQL*Plus or Enterprise Manager Cloud Control (Cloud Control).

13
Administering a CDB

Administering a multitenant container database (CDB) includes tasks such as accessing a container, modifying a CDB, executing DDL statements, and running Oracle-supplied SQL scripts.

> **Note:**
>
> You can complete the tasks in this chapter using SQL*Plus or Oracle SQL Developer.

> **See Also:**
>
> "Tools for a Multitenant Environment"

About CDB Administration

Administering a CDB is similar to administering a non-CDB, with some differences.

Most differences occur because some administrative tasks apply to the entire CDB, whereas others apply to specific containers.

About the Current Container

The data dictionary in each container in a CDB is separate, and the current container is the container whose data dictionary is used for name resolution and for privilege authorization.

The current container can be the CDB root, an application root, a PDB, or an application PDB. Each session has exactly one current container at any point in time. However, a session can switch from one container to another.

Each container has a unique ID and name in a CDB. You can use the `CON_ID` and `CON_NAME` parameters in the `USERENV` namespace to determine the current container ID and name with the `SYS_CONTEXT` function. For example, the following query returns the current container name:

```
SELECT SYS_CONTEXT ('USERENV', 'CON_NAME') FROM DUAL;
```

You can access a container in various ways. For example, you can use the SQL*Plus `CONNECT` command, and you can use an `ALTER SESSION SET CONTAINER` statement to switch the container of the current session.

The following rules apply to the current container in a CDB:

- The current container can be CDB$ROOT (CDB root) only for common users.
- The current container can be a specific PDB for common users and local users.
- The current container can be an application root only for common users or for application common users created in the application root.
- The current container can be a specific application PDB for common users, application common users, and local users.
- The current container must be the CDB root or an application root when a SQL statement includes CONTAINER = ALL.

  You can include the CONTAINER clause in several SQL statements, such as the CREATE USER, ALTER USER, CREATE ROLE, GRANT, REVOKE, and ALTER SYSTEM statements. Note the following rules about CONTAINER = ALL:

  – When a SQL statement includes CONTAINER = ALL and the current container is the CDB root, the SQL statement affects all containers in the CDB, including all PDBs, application roots, and application PDBs.

  – When a SQL statement includes CONTAINER = ALL and the current container is an application root, the SQL statement affects all containers in the application container, including the application root and all the application PDBs that belong to the application root. The SQL statement does not affect the CDB root or any PDBs or application PDBs that do not belong to the current application root.

  – Only a common user or application common user with the commonly granted SET CONTAINER privilege can run a SQL statement that includes CONTAINER = ALL.

See Also:

- "About Container Access in a CDB"
- "Executing Code in Containers Using the DBMS_SQL Package"
- "Determining the Current Container ID or Name"
- "Namespaces in a CDB"
- *Oracle Database SQL Language Reference*
- *Oracle Database Security Guide*

About Administrative Tasks in a CDB

Common users perform administrative tasks for a CDB.

A common user has a single identity and can log in to the CDB root, any application root, PDB, or application PDB in which it has privileges. Some tasks, such as starting up a CDB instance, can be performed only by a common user.

Other administrative tasks are the same for a CDB and a non-CDB. The following table describes some of these tasks and provides pointers to the relevant documentation.

Table 13-1 Administrative Tasks Common to CDBs and Non-CDBs

Task	Description	Additional Information
Starting up a CDB instance	To start a CDB instance, the current user must be a common user whose current container is the CDB root. When you open a CDB, the CDB root is opened, but its other containers are mounted. Use the `ALTER PLUGGABLE DATABASE` statement to modify the open mode of one or more containers.	*Oracle Database Administrator's Guide* for information about starting up a database "Modifying the Open Mode of PDBs" "Modifying a PDB with the ALTER PLUGGABLE DATABASE Statement" "About the Current Container"
Managing processes	A CDB has one set of background processes shared by the CDB root and all containers.	*Oracle Database Administrator's Guide* for information about managing processes
Managing memory	A CDB has a single system global area (SGA) and a single aggregate program global area (PGA). The memory required by a CDB is the sum of the memory requirements for all containers that will be part of the CDB.	*Oracle Database Administrator's Guide* for information about managing memory
Managing security	You can create and drop common users, application common users, and local users in a CDB. You can also grant privileges to and revoke privileges from these users. You can also manage the `CONTAINER_DATA` attributes of common users and application common users. In addition, grant the following roles to the appropriate users: • Grant the `CDB_DBA` role to CDB administrators. • Grant the `PDB_DBA` role to application container administrators and PDB administrators.	*Oracle Database Security Guide*
Monitoring errors and alerts	A CDB has one alert log for the entire CDB. The name of an application container, PDB, or application PDB is included in records in trace files, when appropriate.	*Oracle Database Administrator's Guide* for information about monitoring errors and alerts
Managing diagnostic data	In a CDB, you can use the Oracle Database fault diagnosability infrastructure and the Automatic Diagnostic Repository (ADR).	*Oracle Database Administrator's Guide* for information about managing diagnostic data
Managing control files	A CDB has one control file.	*Oracle Database Administrator's Guide* for information about managing control files

Table 13-1 (Cont.) Administrative Tasks Common to CDBs and Non-CDBs

Task	Description	Additional Information
Managing the online redo log and the archived redo log files	A CDB has one online redo log and one set of archived redo log files.	*Oracle Database Administrator's Guide* for information about managing the redo log *Oracle Database Administrator's Guide* for information about managing archived redo log files
Managing tablespaces	You can create, modify, and drop tablespaces and temporary tablespaces for the CDB root and for individual containers. You can also specify a default tablespace, default tablespace type, and a default temporary tablespace for the CDB root. The CDB root has its own set of Oracle-supplied tablespaces, such as the `SYSTEM` tablespace, and other containers have their own set of Oracle-supplied tablespaces.	*Oracle Database Administrator's Guide* for information about managing tablespaces "About Container Modification When Connected to CDB Root"
Managing data files and temp files	The CDB root has its own data files, and other containers have their own data files. In a CDB, you can manage data files and temp files in basically the same way you would manage them for a non-CDB. However, the following exceptions apply to CDBs: • You can limit the amount of storage used by the data files for a container by using the `STORAGE` clause in a `CREATE PLUGGABLE DATABASE` or `ALTER PLUGGABLE DATABASE` statement. • There is a default temporary tablespace for the CDB root and for individual containers.	*Oracle Database Administrator's Guide* for information about managing data files and temp files "About Container Modification When Connected to CDB Root" "Storage Limits" "Modifying a PDB at the Database Level"
Managing undo	A CDB can run in local undo mode or shared undo mode. Local undo mode means that every container in the CDB uses local undo. Shared undo mode means that there is one active undo tablespace for a single-instance CDB, or for an Oracle RAC CDB, there is one active undo tablespace for each instance. In a CDB, the `UNDO_MANAGEMENT` initialization parameter must be set to `AUTO`, and an undo tablespace is required to manage the undo data.	"Setting the Undo Mode in a CDB Using ALTER DATABASE" *Oracle Database Administrator's Guide* for information about managing undo "About the Current Container"

Table 13-1 (Cont.) Administrative Tasks Common to CDBs and Non-CDBs

Task	Description	Additional Information
Moving data between containers	You can move data between containers using the same methods that you would use to move data between non-CDBs. For example, you can transport the data or use Data Pump export/import to move the data.	*Oracle Database Administrator's Guide* for information about transporting data *Oracle Database Utilities*
Using Oracle Managed Files	Using Oracle Managed files can simplify administration for both a CDB and a non-CDB.	*Oracle Database Administrator's Guide* for information about using Oracle Managed Files
Using Transparent Data Encryption	Transparent Data Encryption is a feature that enables encryption of individual table columns before storing them in the data file, or enables encryption of entire tablespaces. In a CDB, each container has its own master key for Transparent Data Encryption, and, where applicable, the `ADMINISTER KEY MANAGEMENT` SQL statement enables key management at the CDB level and for individual containers.	*Oracle Database Advanced Security Guide* "About the Current Container"
Using a standby database	Oracle Data Guard can configure a physical standby or a logical standby of a CDB. Data Guard operates on the entire CDB, not on individual containers in a CDB.	*Oracle Data Guard Concepts and Administration*
Using Oracle Database Vault	Oracle Database Vault common realms can be scoped to an application root on common objects. Database Vault common command rules can be scoped to either the CDB or an application root. Local realms and command rules can be locally scoped to individual PDBs or application PDBs. When Oracle Database Vault security objects are in the CDB root or an application root, enforcement of the security objects only applies to the containers that have Oracle Database Vault enabled.	*Oracle Database Vault Administrator's Guide*
Dropping a database	When you drop a CDB, all containers in the CDB are dropped along with their data. These containers include the CDB root and PDB seed and all application containers, application seeds, PDBs, and application PDBs. You can also drop individual application containers, application seeds, PDBs, and application PDBs with the `DROP PLUGGABLE DATABASE` statement.	*Oracle Database Administrator's Guide* for information about dropping a database "Dropping a PDB"

> **See Also:**
>
> "Overview of the Multitenant Architecture " for more information about the architecture of a CDB

About Using Manageability Features in a CDB

For each of Oracle Database's manageability features in a CDB, it is important to understand the data location and the data visibility.

When feature data resides in the CDB root, the data is not included when a PDB is unplugged. When the data resides in a PDB, however, the data remains both when the PDB is unplugged and when it is plugged in.

Generally, in a CDB, a common user can view data for the CDB root and for multiple PDBs when the common user's current container is the CDB root. A common user can view this data by querying container data objects. The specific data that is visible varies for the manageability features. A user whose current container is a PDB can view data for that PDB only.

The following table describes how the manageability features work in a CDB.

Table 13-2 Manageability Features in a CDB

Manageability Feature	Data Location	Data Visibility	Additional Information
Active Session History (ASH) ASH collects information about active database sessions. You can use this information to analyze and identify performance issues.	Most of the ASH data is stored in memory. A small percentage of the ASH data samples are stored in the CDB root. ASH data related to a PDB is not included if the PDB is unplugged.	A common user whose current container is the CDB root can view ASH data for the CDB root and for PDBs. A user whose current container is a PDB can view ASH data for the PDB only.	*Oracle Database 2 Day + Performance Tuning Guide* *Oracle Database Performance Tuning Guide*
Alerts An alert is a notification of a possible problem.	Threshold settings that pertain to a PDB are stored in the PDB. Alerts posted when thresholds are violated are enqueued into the alert queue in the CDB root. Threshold settings that pertain to a PDB are included if the PDB is unplugged. Alerts related to a PDB are not included if the PDB is unplugged.	A common user whose current container is the CDB root can view alerts for the CDB root and for PDBs. A user whose current container is a PDB can view alert thresholds and alerts for the PDB only.	*Oracle Database Administrator's Guide* for information about monitoring errors and alerts

Table 13-2 (Cont.) Manageability Features in a CDB

Manageability Feature	Data Location	Data Visibility	Additional Information
Automated Database Maintenance Tasks Automated database maintenance tasks are tasks that are started automatically at regular intervals to perform maintenance operations on the database. Automated tasks include automatic optimizer statistics collection, Automatic Segment Advisor tasks, and Automatic SQL Tuning Advisor tasks. The `ENABLE_AUTOMATIC_MAINTENANCE_PDB` initialization parameter can enable or disable the running of automated maintenance tasks for all the PDBs in a CDB or for individual PDBs in a CDB. The `AUTOTASK_MAX_ACTIVE_PDBS` initialization parameter limits the number of PDBs that can schedule automated maintenance tasks at the same time (during a maintenance window).	A user can schedule maintenance windows and enable or disable maintenance tasks for the current container only. If the current container is the CDB root, then the changes only apply to the CDB root. If the current container is a PDB, then the changes only apply to the PDB. Data related to a PDB is stored in the PDB for automatic optimizer statistics collection and the Automatic Segment Advisor. This data is included if the PDB is unplugged. Automatic SQL Tuning Advisor runs only in the CDB root. See the SQL Tuning Advisor row in this table for information about data collected by Automatic SQL Tuning Advisor.	See the appropriate row in this table for data visibility information about the following manageability features: automatic optimizer statistics collection, Optimizer Statistics Advisor, Automatic Segment Advisor, and Automatic SQL Tuning Advisor.	*Oracle Database Administrator's Guide* for information about managing automated database maintenance tasks *Oracle Database Reference* for information about the `ENABLE_AUTOMATIC_MAINTENANCE_PDB` initialization parameter *Oracle Database Reference* for information about the `AUTOTASK_MAX_ACTIVE_PDBS` initialization parameter

Table 13-2 (Cont.) Manageability Features in a CDB

Manageability Feature	Data Location	Data Visibility	Additional Information
Automatic Database Diagnostic Monitor (ADDM) ADDM can diagnose the performance of a CDB or PDB and determine how identified problems can be resolved.	ADDM executions occur in a PDB or in the CDB root. ADDM analyzes data using one of the following sources: • AWR data stored inside the PDB through an AWR snapshot taken inside the PDB • AWR data from a non-CDB, CDB root, or PDB that is imported into the AWR storage of a PDB • AWR data stored in the root container through an AWR snapshot taken in root Before the start of the analysis, ADDM determines the source of the AWR data (PDB or CDB root) and applies the rules applicable to each data type. **Note:** Automatic ADDM for a PDB is enabled only when automatic snapshots are enabled for the PDB.	A common user whose current container is the CDB root can review results for the entire CDB. The ADDM results can include information about multiple PDBs. ADDM results related to a PDB are not included if the PDB is unplugged. The ADDM results cannot be viewed when the current container is a PDB. A user whose current container is a PDB can view ADDM results data for the current PDB only. The results exclude findings that apply to the CDB as a whole, for example, I/O problems relating to the buffer cache size.	*Oracle Database 2 Day DBA* *Oracle Database Performance Tuning Guide*
Automatic Optimizer Statistics Collection Automatic optimizer statistics collection gathers optimizer statistics for all schema objects in the database for which there are no statistics or only stale statistics. The statistics gathered by this task are used by the SQL query optimizer to improve the performance of SQL execution.	When an automatic optimizer statistics collection task gathers data for a PDB, it stores this data in the PDB. This data is included if the PDB is unplugged.	A common user whose current container is the CDB root can view optimizer statistics data for PDBs. A user whose current container is a PDB can view optimizer statistics data for the PDB only.	"Using Oracle Resource Manager for PDBs" *Oracle Database SQL Tuning Guide*

Table 13-2 (Cont.) Manageability Features in a CDB

Manageability Feature	Data Location	Data Visibility	Additional Information
Automatic Segment Advisor The Automatic Segment Advisor identifies segments that have space available for reclamation and makes recommendations on how to defragment those segments.	When Automatic Segment Advisor gathers data for a PDB, it stores this data in the PDB. This data is included if the PDB is unplugged.	A common user whose current container is the CDB root can view Automatic Segment Advisor data for PDBs. A user whose current container is a PDB can view the Automatic Segment Advisor data for the PDB only.	*Oracle Database Administrator's Guide* for information about reclaiming unused space "Using Oracle Resource Manager for PDBs"
Automatic Workload Repository (AWR) The AWR collects, processes, and maintains performance statistics for problem detection and self-tuning purposes. This data is stored in the database. The gathered data can be displayed in both reports and views.	AWR reports can be generated in the CDB root or in any PDB. AWR reports generated in the CDB root pertain to the entire CDB, while AWR reports generated when a PDB is the current container only pertain to that PDB. AWR data generated in the CDB root is stored in the CDB root. AWR data generated in a PDB is stored in the PDB. When a PDB is unplugged, AWR data stored in the CDB root is not included. When a PDB is unplugged, AWR data stored in the PDB is included.	A common user whose current container is the CDB root can view AWR data for the CDB root and for PDBs. A user whose current container is a PDB can view AWR data for the PDB only.	*Oracle Database Performance Tuning Guide*
Database Replay Database Replay is a feature of Oracle Real Application Testing. Database Replay captures the workload for a CDB or PDB and replays it exactly on a test database.	Capture files are always stored in operating system files, regardless of whether the capture and replay is at the CDB level or PDB level.	For CDB-level workloads, a common user whose current container is the CDB root can view database capture and replay information. For PDB-level workloads, a local or common PDB administrator with the `SELECT_CATALOG_ROLE` privilege can view this information in `DBA_WORKLOAD_CAPTURES` and `DBA_WORKLOAD_REPLAYS`.	*Oracle Database Testing Guide*

Table 13-2 (Cont.) Manageability Features in a CDB

Manageability Feature	Data Location	Data Visibility	Additional Information
Optimizer Statistics Advisor Optimizer Statistics Advisor analyzes how statistics are being gathered and suggests changes that can be made to fine tune statistics collection.	Data related to a PDB is stored in the PDB for Optimizer Statistics Advisor. This data is included if the PDB is unplugged.	A common user whose current container is the CDB root can view Optimizer Statistics Advisor data for PDBs. A user whose current container is a PDB can view the Optimizer Statistics Advisor data for the PDB only.	*Oracle Database SQL Tuning Guide*
SQL Management Base (SMB) SMB stores statement logs, plan histories, SQL plan baselines, and SQL profiles in the data dictionary.	SMB data related to a PDB is stored in the PDB. The SMB data related to a PDB is included if the PDB is unplugged.	A common user whose current container is the CDB root can view SMB data for PDBs. A user whose current container is a PDB can view the SMB data for the PDB only.	*Oracle Database SQL Tuning Guide*
SQL Performance Analyzer (SPA) SPA can analyze the SQL performance impact of SQL tuning and other system changes. SPA is often used with Database Replay.	A common user whose current container is the CDB root can run SPA for any PDB. In this case, the SPA results data is stored in the CDB root and is not included if the PDB is unplugged. A user whose current container is a PDB can run SPA on the PDB. In this case, the SPA results data is stored in the PDB and is included if the PDB is unplugged.	A common user whose current container is the CDB root can view SPA results data for PDBs. A user whose current container is a PDB can view the SPA results data for the PDB only.	*Oracle Database Testing Guide*
SQL Tuning Sets (STS) An STS is a database object that includes one or more SQL statements along with their execution statistics and execution context, and could include a user priority ranking. You can use an STS to tune a group of SQL statements or test their performance using SPA.	An STS can be stored in the CDB root or in any PDB. If it is stored in the CDB root, then you can load SQL statements from any PDB into it. When a PDB is unplugged, an STS stored in the CDB root is not included, even if the STS contains SQL statements from the PDB. When a PDB is unplugged, an STS stored in the PDB is included.	A common user whose current container is the CDB root can view STS data stored in the CDB root only. A user whose current container is a PDB can view STS data for the PDB only.	*Oracle Database SQL Tuning Guide*

Table 13-2 (Cont.) Manageability Features in a CDB

Manageability Feature	Data Location	Data Visibility	Additional Information
SQL Tuning Advisor SQL Tuning Advisor optimizes SQL statements that have been identified as high-load SQL statements.	Automatic SQL Tuning Advisor data is stored in the CDB root. It might have results about SQL statements executed in a PDB that were analyzed by the advisor, but these results are not included if the PDB is unplugged. A common user whose current container is the CDB root can run SQL Tuning Advisor manually for SQL statements from any PDB. When a statement is tuned, it is tuned in any container that runs the statement. A user whose current container is a PDB can also run SQL Tuning Advisor manually for SQL statements from the PDB. When SQL Tuning Advisor is run manually from a PDB, the results are stored in the PDB from which it is run. In this case, a statement is tuned only for the current PDB, and the results related to a PDB are included if the PDB is unplugged.	When SQL Tuning Advisor is run automatically, the results are visible only to a common user whose current container is the CDB root. These results cannot be viewed when the current container is a PDB. When SQL Tuning Advisor is run manually by a user whose current container is a PDB, the results are only visible to a user whose current container is that PDB.	*Oracle Database 2 Day + Performance Tuning Guide* *Oracle Database SQL Tuning Guide*

To run SPA or SQL Tuning Advisor for SQL statements from a PDB, a common user must have the following privileges:

- Common `SET CONTAINER` privilege or local `SET CONTAINER` privilege in the PDB
- The privileges required to execute the SQL statements in the PDB

See Also:

- "About the Current Container"
- "About CDB and Container Information in Views" for an overview of container data objects
- *Oracle Database Security Guide* for detailed information about container data objects

About Managing Tablespaces in a CDB

A tablespace is a logical storage container for database objects, such as tables and indexes, that consume storage space.

At the physical level, a tablespace stores data in one or more data files or temp files. You can use the `ALTER DATABASE` statement to manage tablespaces in a CDB.

The following are considerations for tablespaces in a CDB:

- A permanent tablespace can be associated with exactly one container.
- When you create a tablespace in a container, the tablespace is associated with that container.
- When local undo is disabled for a CDB, the CDB has only one active undo tablespace, or one active undo tablespace for each instance of an Oracle RAC CDB. When local undo is enabled for a CDB, each container in the CDB has its own undo tablespace.
- A local undo tablespace is required for each node in an Oracle Real Application Clusters (Oracle RAC) cluster in which the PDB is open.
- There is one default temporary tablespace each container in the CDB, including the CDB root, each PDB, each application root, and each application PDB.

About Managing Permanent Tablespaces in a CDB

A permanent tablespace can be associated with only one container. Therefore, a permanent tablespace can be associated with the root or with one PDB.

Each container in a CDB must have its own default permanent tablespace, and default permanent tablespaces cannot be shared between containers. Users connected to the container who are not explicitly assigned a tablespace use the default permanent tablespace for the container.

About Managing Temporary Tablespaces in a CDB

Each container in a CDB has its own default temporary tablespace (or tablespace group).

You also can create additional temporary tablespaces for individual containers, and you can assign specific users in containers to these temporary tablespaces. When you unplug a PDB, its temporary tablespaces are also unplugged.

When a user is not assigned a temporary tablespace explicitly in a container, the user's temporary tablespace is the default temporary tablespace for the container.

See Also:

- *Oracle Database Administrator's Guide* for information about managing tablespaces
- "Unplugging a PDB from a CDB"
- "Modifying an Entire CDB Using ALTER DATABASE"
- "Modifying the CDB Root Using ALTER DATABASE"

About Managing Database Objects in a CDB

In a CDB, different containers can contain different database objects.

An Oracle database stores database objects, such as tables, indexes, and directories. Database objects that are owned by a schema are called schema objects, while database objects that are not owned by a schema are called nonschema objects. The root and PDBs contain schemas, and schemas contain schema objects. The root and PDBs can also contain nonschema objects, such as users, roles, tablespaces, directories, and editions.

In a CDB, the root contains Oracle-supplied schemas and database objects. Oracle-supplied common users, such as `SYS` and `SYSTEM`, own these schemas and common database objects. They can also own local objects, both in the root and in a PDB.

You can create common users in the root to administer containers in the CDB. User-created common users can create database objects in the root. Oracle recommends that, in the root, schemas owned by user-created common users contain only database triggers and the objects used in their definitions. A user-created common user can also own any type of local object in a PDB.

You can create local users in a PDB. A local user in a PDB can create schema objects and nonschema objects in the PDB. You cannot create local users in the root.

Name resolution in a CDB is similar to name resolution in a non-CDB, except that names are resolved in the context of the dictionary of the user's current container.

See Also:

- "About the Current Container"
- "Overview of Common and Local Users in a CDB"
- *Oracle Database Administrator's Guide* for information about managing schema objects
- *Oracle Database SQL Language Reference* for information about schema objects and nonschema objects
- *Oracle Database Security Guide* for information about creating common users and local users

About Flashing Back a PDB

You can use the `FLASHBACK PLUGGABLE DATABASE` statement to return a PDB to a past time or system change number (SCN).

You can create restore points for a PDB and flash back the PDB to the restore point without affecting the CDB or other PDBs.

> **Note:**
>
> *Oracle Database Backup and Recovery User's Guide*

About Restricting PDB Users for Enhanced Security

There are several ways to restrict PDB users for enhanced security.

A PDB lockdown profile restricts the features and options available to users in a PDB. The `PDB_OS_CREDENTIAL` initialization parameter can specify a unique operating system user for a PDB to limit operating system access. Also, when the `PATH_PREFIX` and `CREATE_FILE_DEST` clauses are specified during PDB creation, they limit file system access.

PDB Lockdown Profiles

When identities are shared between PDBs, elevated privileges might exist. You can use lockdown profiles to prevent this elevation of privileges.

Identities can be shared in the following situations:

- At the operating system level, when the database interacts with operating system resources such as files or processes
- At the network level, when the database communicates with other systems
- Inside the database, as PDBs access or create common objects or communicate across container boundaries using features such as database links

To increase security, a CDB administrator can use PDB lockdown profiles to restrict users in particular PDBs. A PDB lockdown profile can disable users from running specified SQL statements, such as `ALTER SYSTEM` statements, or disable access to a package that can access the network, such as `UTL_SMTP`. A PDB lockdown profile can also restrict access to common users, common objects, administrative tools such as Oracle XML DB, administrative features such as cursor sharing, and database options such as Oracle Database Advanced Queuing. PDB lockdown profiles can prohibit the use of the XDB protocols (FTP, HTTP, HTTPS) by a PDB with the `XDB_PROTOCOLS` feature.

When logged in to the CDB root or application root, create a lockdown profile by issuing the `CREATE LOCKDOWN PROFILE` statement, which supports the following optional clauses:

- `FROM static_base_profile` creates a new lockdown profile by using the values from an existing profile. Any subsequent changes to the existing profile will not affect the new profile.
- `INCLUDING dynamic_base_profile` creates a new lockdown profile by using the values from an existing profile, except that this new lockdown profile inherits the `DISABLE STATEMENT` rules that comprise the base profile, and any subsequent changes to the base profile.

The user issuing the statement must have the `CREATE LOCKDOWN PROFILE` system privilege in the current container. You can add and remove restrictions with the `ALTER LOCKDOWN PROFILE` statement. The user must issue the `ALTER` statement in the CDB root or application root and must have the have `ALTER LOCKDOWN PROFILE` system privilege in the current container.

Specify a lockdown profile by using the `PDB_LOCKDOWN` initialization parameter. This parameter determines whether the PDB lockdown profile applies to a given PDB. You can set this parameter at the following levels:

- PDB

  The profile applies only to the PDB in which it is set.
- Application container

  The profile applies to all application PDBs in the application container. The value can be modified only by an application common user who has application common `SYSDBA` or common `ALTER SYSTEM` privileges or a CDB common user who has common `SYSDBA` or common `ALTER SYSTEM` privileges.
- CDB

  The profile applies to all PDBs. A common user who has common `SYSDBA` or common `ALTER SYSTEM` privileges can override a CDB-wide setting for a specific PDB.

If the `PDB_LOCKDOWN` parameter in a PDB is set to the name of a lockdown profile different from the container for this PDB (CDB or application container), then a set of rules govern the interaction between restrictions.

See Also:

- *Oracle Database Security Guide* for complete information about lockdown profiles
- *Oracle Database SQL Language Reference* for more information about the `CREATE LOCKDOWN PROFILE` statement
- *Oracle Database Reference* for more information about the `PDB_LOCKDOWN` initialization parameter

The PDB_OS_CREDENTIAL Initialization Parameter

When the database accesses an external procedure with the `extproc` agent, the `PDB_OS_CREDENTIAL` initialization parameter determines the identity of the operating system user employed when interacting with the operating system from a PDB.

Using an OS user described by a credential whose name is specified as a value of the PDB_OS_CREDENTIAL initialization parameter can ensure that operating system interactions are performed as a less powerful user. In this way, the feature protects data belonging to one PDB from being accessed by users connected to another PDB. A credential is an object that is created using the CREATE_CREDENTIAL procedure in the DBMS_CREDENTIAL package.

The Oracle OS user is usually a highly privileged user. Using this account for operating system interactions is not recommended. Also, using the same OS user for operating system interactions from different PDBs might compromise data belonging to a given PDB.

See Also:

- *Oracle Database Administrator's Guide* for information about managing processes for external procedures
- *Oracle Database Reference* for more information about the PDB_OS_CREDENTIAL initialization parameter
- *Oracle Database PL/SQL Packages and Types Reference*

The PATH_PREFIX and CREATE_FILE_DEST PDB Creation Clauses

The PATH_PREFIX clause of CREATE PLUGGABLE DATABASE ensures that all directory object paths associated with the PDB are restricted to the specified directory or its subdirectories.

PATH_PREFIX also ensures that the following files associated with the PDB are restricted to specified directory:

- The Oracle XML repository for the PDB
- Files created with a CREATE PFILE statement
- The export directory for Oracle wallets
- Library object created with a CREATE LIBRARY statement

Note:

The library must use a directory object. If a PDB uses a predefined PATH_PREFIX, attempts to use a library object that does not use a directory object result in an ORA-65394 error. The library object is not invalidated, but to make it usable you must recreate it using a directory object.

The CREATE_FILE_DEST clause of the CREATE PLUGGABLE DATABASE statement ensures that all the database files belonging to the PDB are restricted to the specified directory and its subdirectories. The clause enables Oracle Managed Files for the PDB and specifies the default file system directory or Oracle ASM disk group for the PDB files.

> **See Also:**
>
> - "Restrictions on PDB File Locations"
> - "CREATE_FILE_DEST Clause"

Overview of Oracle Multitenant with Oracle RAC

Oracle Multitenant is an option with Oracle Database 12*c* that simplifies consolidation, provisioning, upgrades, and more.

It is based on an architecture that allows a multitenant container database (CDB) to hold several pluggable databases (PDBs). You can adopt an existing database as a PDB without having to change the application tier. In this architecture, Oracle RAC provides the local high availability that is required when consolidating various business-critical applications on one system.

When using PDBs with Oracle RAC, the multitenant CDB is based on Oracle RAC. You can make each PDB available on either every instance of the Oracle RAC CDB or a subset of instances. In either case, access to and management of the PDBs are regulated using dynamic database services, which will also be used by applications to connect to the respective PDB, as they would in a single instance Oracle database using Oracle Net Services for connectivity.

You can isolate PDBs to prevent certain operations from being performed on or within a particular PDB that may interfere with other PDBs sharing the same Oracle RAC database or database instance. PDB isolation allows for a higher degree of consolidation using Oracle Multitenant.

If you create an Oracle RAC database as a CDB and plug one or more PDBs into the CDB, then, by default, a PDB is not started automatically on any instance of the Oracle RAC CDB. With the first dynamic database service assigned to the PDB (other than the default database service which has the same name as the database name), the PDB is made available on those instances on which the service runs.

Whether a PDB is available on more than one instance of an Oracle RAC CDB, the CDB is typically managed by the services running on the PDB. You can manually enable PDB access on each instance of an Oracle RAC CDB by starting the PDB manually on that instance.

Related Topics

- *Oracle Real Application Clusters Administration and Deployment Guide*

Accessing Containers in a CDB

You can connect to a container by using the SQL*Plus `CONNECT` command. Alternatively, you can switch into a container with an `ALTER SESSION SET CONTAINER` SQL statement.

About Container Access in a CDB

You can use SQL*Plus to access the root or a PDB in a CDB.

> **Note:**
>
> This section assumes that you understand how to connect to a non-CDB in SQL*Plus.

> **See Also:**
>
> - *Oracle Database Administrator's Guide* for information about submitting commands and SQL to the database
> - *Oracle Database Net Services Administrator's Guide* for information about configuring Oracle Net Services

Services in a CDB

Clients access the root or a PDB through database services.

Database services have an optional `PDB` property. When a PDB is created, a new default service for the PDB is created automatically. The service has the same name as the PDB. With the service name, you can access the PDB using the easy connect syntax or the net service name from the `tnsnames.ora` file. Oracle Net Services must be configured properly for clients to access this service.

When a user connects using a service with a non-null `PDB` property, the user name is resolved in the context of the specified PDB. When a user connects without specifying a service or using a service name with a null `PDB` property, the user name is resolved in the context of the root. You can view the `PDB` property for a service by querying the `CDB_SERVICES` data dictionary view or by running the `config service` command in the SRVCTL utility.

> **Note:**
>
> When two or more CDBs on the same computer system use the same listener and two or more PDBs have the same service name in these CDBs, a connection that specifies this service name connects randomly to one of the PDBs with the service name. To avoid incorrect connections, ensure that all service names for PDBs are unique on the computer system, or configure a separate listener for each CDB on the computer system.

> **See Also:**
>
> - "Overview of Services in a CDB"
> - "Managing Services for PDBs"
> - "Example 19-9"

Session Limits in a CDB

The setting for the SESSIONS initialization parameter limits the total number of sessions available in a CDB, including the sessions connected to PDBs.

If the limit is reached for the CDB, then users cannot connect to PDBs. To ensure that one PDB does not use too many sessions, you can limit the number of sessions available to a PDB by setting the SESSIONS initialization parameter in the PDB.

> **See Also:**
>
> "Listing the Modifiable Initialization Parameters in PDBs"

User Names in a Multitenant Environment

Within each PDB, a user name must be unique with respect to other user names and roles in that PDB.

Note the following restrictions:

- For common user names, names for user-created common users must begin with a common user prefix. By default, for CDB common users, this prefix is C##. For application common users, this prefix is an empty string. This means that there are no restrictions on the name that can be assigned to an application common user other than that it cannot start with the prefix reserved for CDB common users. For example, you could name a CDB common user c##hr_admin and an application common user hr_admin.

  The COMMON_USER_PREFIX parameter in CDB$ROOT defines the common user prefix. You can change this setting, but do so only with great care.
- For local user names, the name cannot start with C## (or c##).
- A user and a role cannot have the same name.

Related Topics

- *Oracle Database Security Guide*

How the Multitenant Option Affects Password Files for Administrative Users

In a multitenant environment, the password information for the local and common administrative users is stored in different locations.

- **For CDB administrative users:** The password information (hashes of the password) for the CDB common administrative users to whom administrative privileges were granted in the CDB root is stored in the password file.
- **For all users in a CDB to whom administrative privileges were granted outside the CDB root:** To view information about the password hash information of these users, query the $PWFILE_USERS dynamic view.

Related Topics

- *Oracle Database Security Guide*

Accessing a Container in a CDB

Access a container in a CDB with SQL*Plus by issuing a `CONNECT` or `ALTER SESSION` command.

Connecting to a Container Using the SQL*Plus CONNECT Command

You can use the SQL*Plus `CONNECT` command to connect to the root or to a PDB.

Connecting to the Root Using the SQL*Plus CONNECT Command

You can connect to the root in the same way that you connect to a non-CDB.

Specifically, you can use the following techniques to connect to the root with the SQL*Plus `CONNECT` command:

- Local connection
- Local connection with operating system authentication
- Database connection using easy connect
- Database connection using a net service name
- Remote database connection using external authentication

The following prerequisites must be met for the user connecting to the root:

- The user must be a common user.
- The user must be granted `CREATE SESSION` privilege in the root.

To connect to the root using the SQL*Plus CONNECT command:

1. Configure your environment so that you can open SQL*Plus.
2. Start SQL*Plus with the `/NOLOG` argument:

   ```
   sqlplus /nolog
   ```

3. Issue a SQL*Plus `CONNECT` command to connect to the root, as shown in the following examples.

Example 13-1 Connecting to the Root with a Local Connection

This example connects to the root in the local CDB as user `SYSTEM`. SQL*Plus prompts for the `SYSTEM` user password.

```
connect system
```

Example 13-2 Connecting to the Root with Operating System Authentication

This example connects locally to the root with the `SYSDBA` administrative privilege with operating system authentication.

```
connect / as sysdba
```

Example 13-3 Connecting to the Root with a Net Service Name

Assume that clients are configured to have a net service name for the root in the CDB. For example, the net service name can be part of an entry in a `tnsnames.ora` file.

This example connects as common user `c##dba` to the database service designated by the net service name `mycdb`. SQL*Plus prompts for the `c##dba` user password.

```
connect c##dba@mycdb
```

> **See Also:**
>
> *Oracle Database Administrator's Guide* for information about submitting commands and SQL to the database

Connecting to a PDB Using the SQL*Plus CONNECT Command

To connect to a PDB with the SQL*Plus `CONNECT` command, you can use easy connect or a net service name.

To connect to a PDB, a user must be one of the following:

- A common user with a `CREATE SESSION` privilege granted commonly or granted locally in the PDB
- A local user defined in the PDB with `CREATE SESSION` privilege

Only a user with `SYSDBA`, `SYSOPER`, `SYSBACKUP`, or `SYSDG` privilege can connect to a PDB that is in mounted mode. To change the open mode of a PDB, see "Modifying the Open Mode of PDBs".

To connect to a PDB using the SQL*Plus CONNECT command:

1. Configure your environment so that you can open SQL*Plus.
2. Start SQL*Plus with the `/NOLOG` argument:

   ```
   sqlplus /nolog
   ```

3. Issue a SQL*Plus `CONNECT` command using easy connect or a net service name to connect to the PDB.

Example 13-4 Connecting to a PDB

Assume that clients are configured to have a net service name for each PDB that matches each PDB name. For example, the net service name can be part of an entry in a `tnsnames.ora` file.

The following command connects to the `sh` local user in the `salespdb` PDB:

```
CONNECT sh@salespdb
```

The following command connects to the SYSTEM common user in the salespdb PDB:

```
CONNECT system@salespdb
```

> **See Also:**
>
> *Oracle Database Administrator's Guide* for information about submitting the SQL*Plus CONNECT command

Switching to a Container Using the ALTER SESSION Statement

When you are connected to a container as a common user, you can switch to a different container and application service using the ALTER SESSION statement.

You can use the following statement to switch to a different container and application service:

```
ALTER SESSION SET CONTAINER = container_name [SERVICE = service_name]
```

For *container_name*, specify one of the following:

- CDB$ROOT to switch to the CDB root
- PDB$SEED to switch to the PDB seed
- A PDB name to switch to the PDB

  When the current container is the root, you can view the names of the PDBs in a CDB by querying the DBA_PDBS view.

For *service_name*, specify a service that is running in the PDB. You can list the services running in the containers of a CDB, excluding the CDB root, by issuing the following query with the CDB root as the current container:

```
COL NAME FORMAT A30
COL CON_NAME FORMAT A20

SELECT NAME,CON_NAME, CON_ID
  FROM V$ACTIVE_SERVICES
  WHERE UPPER(NAME) != CON_NAME
  AND CON_ID !=1
  ORDER BY CON_ID;
```

By default, when you switch to a container, the session uses the default service for the container. However, the default PDB service does not support all service attributes and features such as service metrics, Fast Application Notification (FAN), load balancing, Resource Manager, Transaction Guard, Application Continuity, and so on. It is best practice to use a nondefault service for the container by specifying SERVICE = *service_name*, where *service_name* is the name of the service.

With this new capability, connection pools can switch the service, and, when needed the PDB, on a connection when a connection is borrowed from the pool. Starting with Oracle Database 12c Release 2 (12.2.0.1), connection pools support more than

one database service with universal connection pools (UCPs). It can also be used standalone.

When switching to a service, applications can consolidate to a CDB, while keeping the database services identified, prioritized, measured, and highly available. Switching to a nondefault service provides the following benefits:

- It preserves the service attributes and features.
- It eliminates too many connection pools with too many connections serving these tenants.
- It allows applications to use more database services for workload control without consuming too many connection pools. Customers can identify and prioritize workloads using services without over sizing the database connections.

The following are considerations for using the `ALTER SESSION SET CONTAINER` statement:

- After the statement completes successfully, the current schema of the session is set to the schema owned by the common user in the specified container.
- After the statement completes successfully, the security context is reset to that of the schema owned by the common user in the specified container.
- After the statement completes successfully, login triggers for the specified container do not fire.

  If you require a trigger, then you can define a before or after `SET CONTAINER` trigger in a PDB to fire before or after the `ALTER SESSION SET CONTAINER` statement is executed.
- After the statement completes successfully and the `SERVICE` clause specifies a nondefault service for the PDB, the session is using a new service with attributes set, including metrics, FAN, TAF, Application Continuity, Transaction Guard, `drain_timeout`, and `stop_option` for the new service.
- Package states are not shared across containers.
- When closing a PDB, sessions that switched into the PDB and sessions that connected directly to the PDB are handled identically.
- A transaction cannot span multiple containers. If you start a transaction and use `ALTER SESSION SET CONTAINER` to switch to a different container, then you cannot issue DML, DDL, `COMMIT`, or `ROLLBACK` statements until you switch back to the container in which you started the transaction.
- If you open a cursor and use `ALTER SESSION SET CONTAINER` to switch to different container, then you cannot fetch data from that cursor until you switch back to the container in which the cursor was opened.
- You can use the `ALTER SESSION SET CONTAINER` statement with the `SERVICE` clause for connection pooling as well as advanced CDB administration.

  For example, you can use this statement for connection pooling with PDBs for a multitenant application. A multitenant application uses a single instance of the software on a server to serve multiple customers (tenants). In a non-CDB, multitenant is typically supported by adding an extra column that identifies the tenant to every table used by the application, and tenants check out connections from a connection pool. In a CDB with PDBs, each tenant can have its own PDB, and you can use the `ALTER SESSION SET CONTAINER` statement in a connection pooling configuration.

- When working with connection pools that serve applications, the applications may be using data sources with different services. Using the `ALTER SESSION SET CONTAINER` statement with the `SERVICE` clause enables the connection pool to use the same connections for many applications, sharing the services.

The following prerequisites must be met to use the `ALTER SESSION SET CONTAINER` statement:

- The current user must be a common user. The initial connection must be made using the SQL*Plus `CONNECT` command.
- When altering a session to switch to a PDB as a common user that was not supplied with Oracle Database, the current user must be granted the `SET CONTAINER` privilege commonly or must be granted this privilege locally in the PDB.

> **Note:**
>
> When an `ALTER SESSION SET CONTAINER` statement is used to switch to the current container, these prerequisites are not enforced, and no error message is returned if they are not met.

Before issuing an `ALTER SESSION SET CONTAINER` statement with the `SERVICE` clause, the following prerequisites must be met:

- The service switched to must be active. You cannot switch to a service that is not running.
- When switching between services, the service attributes of the service being switched from and the service being switched to must match. For example, the services switched from and to must all have TAF, or must all use Application Continuity, or must all have `drain_timeout` set.

To switch to a container using the ALTER SESSION statement:

1. In SQL*Plus, connect to a container as a common user with the required privileges.
2. Check the current open mode of the container to which you are switching.

   To check the current open mode of the root or a PDB, query the `OPEN_MODE` column in the `V$CONTAINERS` view when the current container is the root.

   If the open mode of the root should be changed, then follow the instructions in *Oracle Database Administrator's Guide* about altering database availability to change the open mode.

   If the open mode of the PDB should be changed, then follow the instructions in "Modifying the Open Mode of PDBs" to change the open mode.

   The open mode of the root imposes limitations on the open mode of PDBs. For example, the root must be open before any PDBs can be open. Therefore, you might need to change the open mode of the root before changing the open mode of a PDB.
3. If you are switching to a specific service, then ensure that the service is running.

   To check the active status of the service, query the `V$ACTIVE_SERVICES` view when the current container is the CDB root.

If the service is not running, then use the SRVCTL utility or the `DBMS_SERVICE` package to start the service.

4. Run the `ALTER SESSION SET CONTAINER` statement and specify the container to which you want to switch.

   Include the `SERVICE` clause to switch to a specific application service.

The following examples switch to various containers using `ALTER SESSION`.

Example 13-5 Switching to the PDB salespdb and Using the salesrep Service

```
ALTER SESSION SET CONTAINER = salespdb SERVICE = salesrep;
```

Example 13-6 Switching to the PDB salespdb and Using the Default Service

```
ALTER SESSION SET CONTAINER = salespdb;
```

Example 13-7 Switching to the CDB Root

```
ALTER SESSION SET CONTAINER = CDB$ROOT;
```

Example 13-8 Switching to the PDB Seed

```
ALTER SESSION SET CONTAINER = PDB$SEED;
```

Example 13-9 Switching Services Using a Dummy Service in the CDB Root

To design connection pooling that switches the container and the service, one method is to create a dummy service in the CDB root and set all required service attributes on this dummy service (for example, `drain_timeout`, TAF or Application Continuity). The service attributes must match across the CDB root and the PDB. To use this method, complete the following steps:

1. Connect to the dummy service when first creating the connection pool and when creating new connections.
2. As services are added to each PDB, set the same attributes on these real services.
3. When an application requires a connection, complete one of the following actions:
   - Create a new connection to the dummy service, and switch to the PDB and service.
   - Borrow a free connection in the pool and switch to the PDB and service.

     You do not need to return to the CDB root when switching across PDBs.

You do not need to return to the CDB root when switching across PDBs.

> **See Also:**
>
> *Oracle Database Administrator's Guide* for information about database resident connection pooling

Modifying a CDB at the System Level

You can set initialization parameters at the CDB level. In some cases, you can override these parameters at the PDB level.

About System-Level Modifications of a CDB

The `ALTER SYSTEM SET` statement dynamically sets an initialization parameter in one or more containers.

A CDB uses an inheritance model for initialization parameters in which PDBs inherit initialization parameter values from the root. In this case, inheritance means that the value of a specific parameter in the root applies to a specific PDB.

A PDB can override the root setting for some parameters. In such cases, a PDB has an inheritance property for each initialization parameter that is either true or false. The inheritance property is true for a parameter when the PDB inherits the root's value for the parameter; otherwise, the property is false.

The inheritance property for some parameters must be true. For other parameters, when the current container is the PDB, you can change the inheritance property by running the `ALTER SYSTEM SET` statement. If `V$SYSTEM_PARAMETER.ISPDB_MODIFIABLE` is `TRUE` for an initialization parameter, then the inheritance property can be false for the parameter.

When the current container is the root, the `CONTAINER` clause of the `ALTER SYSTEM SET` statement controls which PDBs inherit the parameter value being set. The `CONTAINER` clause has the following syntax:

```
CONTAINER = { CURRENT | ALL }
```

The following settings are possible:

- `CURRENT`

  The parameter setting applies only to the current container. This is the default setting for `CONTAINER`. When the current container is the root, the parameter setting applies to the root and to any PDB with an inheritance property of true for the parameter.

- `ALL`

  The parameter setting applies to all containers in the CDB, including the root and all PDBs. Specifying `ALL` sets the inheritance property to true for the parameter in all PDBs.

> **See Also:**
>
> "About the Current Container" for more information about the `CONTAINER` clause and rules that apply to it

Modifying a CDB with ALTER SYSTEM

To modify a CDB at the system level, use the `ALTER SYSTEM` statement.

Prerequisites

The current user must have the commonly granted `ALTER SYSTEM` privilege.

To use ALTER SYSTEM SET in the root in a CDB:

1. In SQL*Plus, ensure that the current container is the root.
2. Run the `ALTER SYSTEM SET` statement.

> **Note:**
>
> To change the inheritance property for a parameter in a PDB from false to true, run the `ALTER SYSTEM RESET` statement to reset the parameter when the current container is the PDB. The following sample statement resets the `OPEN_CURSORS` parameter:
>
> ```
> ALTER SYSTEM RESET OPEN_CURSORS SCOPE = SPFILE;
> ```

Example 13-10 Setting an Initialization Parameter for All Containers

This `ALTER SYSTEM SET` statement sets the `OPEN_CURSORS` initialization parameter to `200` for the all containers and sets the inheritance property to `TRUE` in each PDB.

```
ALTER SYSTEM SET OPEN_CURSORS = 200 CONTAINER = ALL;
```

Example 13-11 Setting an Initialization Parameter for the Root

This `ALTER SYSTEM SET` statement sets the `OPEN_CURSORS` initialization parameter to `200` for the root and for PDBs with an inheritance property of true for the parameter.

```
ALTER SYSTEM SET OPEN_CURSORS = 200 CONTAINER = CURRENT;
```

> **See Also:**
>
> - "Modifying a PDB at the System Level"
> - *Oracle Database SQL Language Reference* for more information about the `ALTER SYSTEM SET` statement

Modifying Containers When Connected to the CDB Root

You can modify the entire CDB or the root with the `ALTER DATABASE` statement.

About Container Modification When Connected to CDB Root

The ALTER DATABASE statement modifies a CDB. When you are connected to the CDB root, the ALTER PLUGGABLE DATABASE statement can modify the open mode of one or more PDBs.

The behavior of ALTER DATABASE and ALTER PLUGGABLE DATABASE depends on which container you are connected to when you use the statement:

- Connected as a common user to CDB root

  In this case, the ALTER DATABASE statement works the same as in a non-CDB. When an ALTER DATABASE statement with the RENAME GLOBAL_NAME clause modifies the domain of a CDB, it affects the domain of each PDB with a domain that defaults to that of the CDB. The ALTER PLUGGABLE DATABASE statement with the *pdb_change_state* clause modifies the open mode of one or more PDBs.

- Connected to a PDB

  In this case, the ALTER DATABASE and ALTER PLUGGABLE DATABASE statements modify the current PDB only.

The following table lists which containers are modified by clauses in ALTER DATABASE and ALTER PLUGGABLE DATABASE statements.

Table 13-3 Statements That Modify Containers in a CDB

Modify Entire CDB	Modify Root Only	Modify One or More PDBs
When connected as a common user whose current container is the root, ALTER DATABASE statements with the following clauses modify the entire CDB: • *startup_clauses* • *recovery_clauses* • *logfile_clauses* • *controlfile_clauses* • *standby_database_clauses* • *instance_clauses* • *security_clause* • RENAME GLOBAL_NAME clause • ENABLE BLOCK CHANGE TRACKING clause • DISABLE BLOCK CHANGE TRACKING clause	When connected as a common user whose current container is the root, ALTER DATABASE statements with the following clauses modify the root only: • *database_file_clauses* • DEFAULT EDITION clause • DEFAULT TABLESPACE clause • DEFAULT TEMPORARY TABLESPACE clause ALTER DATABASE statements with the following clauses modify the root and set default values for PDBs: • *flashback_mode_clause* • SET DEFAULT {BIGFILE|SMALLFILE} TABLESPACE clause • *set_time_zone_clause* You can use these clauses to set nondefault values for specific PDBs.	When connected as a common user whose current container is the root, ALTER PLUGGABLE DATABASE statements with the following clause can modify the open mode of one or more PDBs: • *pdb_change_state* When the current container is a PDB, ALTER PLUGGABLE DATABASE statements with this clause can modify the open mode of the current PDB. When connected as a common user whose current container is the root, ALTER PLUGGABLE DATABASE statements with the following clause can preserve or discard the open mode a PDB when the CDB restarts: • *pdb_save_or_discard_state*

> **See Also:**
>
> - "About the Current Container"
> - "Modifying a PDB at the Database Level"
> - *Oracle Database SQL Language Reference*

Modifying an Entire CDB Using ALTER DATABASE

You can use the `ALTER DATABASE` statement to modify an entire CDB, including the root and all PDBs. Most `ALTER DATABASE` statements modify the entire CDB.

For a list of statements that modify the entire CDB rather than the root or individual PDBs, see the "Modify Entire CDB" column of "About Container Modification When Connected to CDB Root".

Prerequisites

To modify an entire CDB, the following prerequisites must be met:

- The current user must be a common user with the `ALTER DATABASE` privilege.
- To use an `ALTER DATABASE` statement with a *recovery_clause*, the current user must have the `SYSDBA` administrative privilege commonly granted. In this case, you must exercise this privilege using `AS SYSDBA` at connect time.

To modify an entire CDB:

1. In SQL*Plus, ensure that the current container is the root.
2. Use an `ALTER DATABASE` statement with a clause that modifies an entire CDB.

Example 13-12 Backing Up the Control File for a CDB

The following `ALTER DATABASE` statement uses a *recovery_clause* to back up a control file.

```
ALTER DATABASE BACKUP CONTROLFILE TO '+DATA/dbs/backup/control.bkp';
```

Example 13-13 Adding a Redo Log File to a CDB

The following `ALTER DATABASE` statement uses a *logfile_clause* to add redo log files.

```
ALTER DATABASE cdb ADD LOGFILE
  GROUP 4 ('/u01/logs/orcl/redo04a.log','/u02/logs/orcl/redo04b.log')
  SIZE 100M BLOCKSIZE 512 REUSE;
```

> **See Also:**
>
> *Oracle Database SQL Language Reference*

Setting the Undo Mode in a CDB Using ALTER DATABASE

When local undo is enabled, each container has its own undo tablespace for every instance in which it is open. When local undo is disabled, there is one undo tablespace for the entire CDB.

About the CDB Undo Mode

You can configure a CDB to use local undo in every container or to use shared undo (default) for the entire CDB.

A CDB runs either in local or shared undo mode. The undo mode applies to the entire CDB. Therefore, every container either uses shared undo or local undo.

You can specify the undo mode of a CDB during CDB creation in the `ENABLE PLUGGABLE DATABASE` clause of the `CREATE DATABASE` statement. If you do not specify the `UNDO` clause, then shared undo mode is the default. You can change the undo mode of a CDB after it is created by issuing an `ALTER DATABASE` statement and restarting the CDB.

To determine the current CDB undo mode, run the following query in the CDB root:

```
SELECT PROPERTY_NAME, PROPERTY_VALUE
FROM   DATABASE_PROPERTIES
WHERE  PROPERTY_NAME = 'LOCAL_UNDO_ENABLED';
```

If the query returns `TRUE` for the `PROPERTY_VALUE`, then the CDB is in local undo mode. Otherwise, the CDB is in shared undo mode.

About Local Undo Mode

Local undo mode means that each container has its own undo tablespace for every instance in which it is open.

In this mode, Oracle Database automatically creates an undo tablespace for every container in the CDB. For an Oracle RAC CDB, there is one active undo tablespace for each instance for each PDB in local undo mode.

Local undo mode provides increased isolation for each container and improves the efficiency of some operations, such as unplugging the container or performing point-in-time recovery on the container. In addition, local undo mode is required for some operations to be supported, such as relocating a PDB or cloning a PDB that is in open read/write mode.

When a CDB is in local undo mode, the following applies:

- Any user who has the appropriate privileges for the current container can create an undo tablespace for the container.
- Undo tablespaces are visible in static data dictionary views and dynamic performance (V$) views in every container in the CDB.

> **See Also:**
>
> *Oracle Database SQL Language Reference* for information about the required privileges

About Shared Undo Mode

Shared undo mode means that only one active undo tablespace exists for a single-instance CDB. For an Oracle RAC CDB, there is one active undo tablespace for each instance.

When a CDB is in shared undo mode, the following applies:

- Only a common user who has the appropriate privileges and whose current container is the CDB root can create an undo tablespace.
- When the current container is not the CDB root, an attempt to create an undo tablespace fails and returns an error.
- Undo tablespaces are visible in static data dictionary views and dynamic performance (V$) views when the current container is the CDB root. Undo tablespaces are visible only in dynamic performance views when the current container is a PDB, an application root, or an application PDB.

> **Note:**
>
> - When you change the undo mode of a CDB, the new undo mode applies to an individual container the first time the container is opened after the change.
> - When you change the undo mode of a CDB, containers in the CDB cannot flash back to a time or SCN that is prior to the change.

Configuring a CDB to Use Local Undo Mode

You can change a CDB to local undo mode by issuing an `ALTER DATABASE LOCAL UNDO ON` statement and restarting the database.

When a CDB is in local undo mode, each container has its own undo tablespace for every instance in which it is open. Oracle Database automatically creates an undo tablespace in any container in the CDB that does not have one. If a PDB without an undo tablespace is cloned, relocated, or plugged into a CDB that is configured to use local undo mode, then Oracle Database automatically creates an undo tablespace for the PDB the first time it is opened.

When a CDB is changed from shared undo mode to local undo mode, Oracle Database creates the required undo tablespaces automatically.

1. If the CDB instance is open, then shut it down.
2. Start up the CDB instance in `OPEN UPGRADE` mode. For example:

   ```
   STARTUP UPGRADE
   ```

3. In SQL*Plus, ensure that the current container is the CDB root. For example, enter the following:

```
SHOW CON_NAME

CON_NAME
------------------------------
CDB$ROOT
```

4. Query the current undo mode of the CDB:

```
SELECT PROPERTY_NAME, PROPERTY_VALUE
FROM   DATABASE_PROPERTIES
WHERE  PROPERTY_NAME = 'LOCAL_UNDO_ENABLED';
```

5. To enable local undo, issue the following SQL statement:

```
ALTER DATABASE LOCAL UNDO ON;
```

6. Shut down and restart the CDB instance.
7. Optional: Manually create an undo tablespace in the PDB seed.

   While Oracle Database creates an undo tablespace in the PDB seed automatically in local undo mode, you might want to control the size and configuration of the undo tablespace by creating an undo tablespace manually. To ensure the PDBs created from the PDB seed use the manually-created undo tablespace and not the automatically-created undo tablespace, you must set the `UNDO_TABLESPACE` initialization parameter to the manually-created undo tablespace, or drop the automatically-created undo tablespace.

   a. In SQL*Plus, ensure that the current container is the root.
   b. Place the PDB seed in open read/write mode:

      ```
      ALTER PLUGGABLE DATABASE PDB$SEED OPEN READ WRITE FORCE;
      ```

   c. Switch container to the PDB seed:

      ```
      ALTER SESSION SET CONTAINER=PDB$SEED;
      ```

   d. Create an undo tablespace in the PDB seed. For example:

      ```
      CREATE UNDO TABLESPACE seedundots1
         DATAFILE 'seedundotbs_1a.dbf'
         SIZE 10M AUTOEXTEND ON
         RETENTION GUARANTEE;
      ```

   e. Switch container to the root:

      ```
      ALTER SESSION SET CONTAINER=CDB$ROOT;
      ```

   f. Place the PDB seed in open read-only mode:

      ```
      ALTER PLUGGABLE DATABASE PDB$SEED OPEN READ ONLY FORCE;
      ```

Configuring a CDB to Use Shared Undo Mode

To change a CDB to use shared undo mode, use an `ALTER DATABASE LOCAL UNDO OFF` statement.

1. If the CDB instance is open, then shut it down.
2. Start up the CDB instance in `OPEN UPGRADE` mode. For example:

   ```
   STARTUP UPGRADE
   ```

3. In SQL*Plus, ensure that the current container is the CDB root. For example, enter the following:

   ```
   SHOW CON_NAME

   CON_NAME
   ------------------------------
   CDB$ROOT
   ```

4. Optionally, query the current undo mode of the CDB:

   ```
   SELECT PROPERTY_NAME, PROPERTY_VALUE
   FROM   DATABASE_PROPERTIES
   WHERE  PROPERTY_NAME = 'LOCAL_UNDO_ENABLED';
   ```

5. To turn off local undo, issue the following SQL statement:

   ```
   ALTER DATABASE LOCAL UNDO OFF;
   ```

6. Shut down and restart the CDB instance.

When in shared undo mode, the CDB ignores any local undo tablespaces that were created when it was in local undo mode. Oracle recommends that you delete the unused local undo tablespaces.

Modifying the CDB Root Using ALTER DATABASE

To modify only the root of a CDB, use the `ALTER DATABASE` statement.

When the current container is the root, some `ALTER DATABASE` statements modify the root without directly modifying any of the PDBs. See the "Modify Root Only" column of Table 13-3 for a list of these statements.

Some statements set the defaults for the PDBs in the CDB. You can overwrite these defaults for a PDB by using the `ALTER PLUGGABLE DATABASE` statement.

Prerequisites

To modify the root, the current user must have the `ALTER DATABASE` privilege in the root.

To modify the root:

1. In SQL*Plus, ensure that the current container is the root.
2. Run an `ALTER DATABASE` statement with a clause that modifies the root.

The following examples modify the root.

A user whose current container is the root that is not explicitly assigned a tablespace uses the default permanent tablespace for the root. The tablespace specified in the `ALTER DATABASE` statement must exist in the root.

After executing this statement, the default type of subsequently created tablespaces in the root is bigfile. This setting is also the default for PDBs.

The tablespace or tablespace group specified in the `ALTER DATABASE` statement must exist in the root.

Example 13-14 Changing the Default Permanent Tablespace for the Root

This `ALTER DATABASE` statement uses a `DEFAULT TABLESPACE` clause to set the default permanent tablespace to `root_tbs` for the root.

```
ALTER DATABASE DEFAULT TABLESPACE root_tbs;
```

Example 13-15 Bringing a Data File Online for the Root

This `ALTER DATABASE` statement uses a *database_file_clause* to bring the /u02/oracle/cdb_01.dbf data file online.

```
ALTER DATABASE DATAFILE '/u02/oracle/cdb_01.dbf' ONLINE;
```

Example 13-16 Changing the Default Tablespace Type for the Root

This `ALTER DATABASE` statement uses a `SET DEFAULT TABLESPACE` clause to change the default tablespace type to bigfile for the root.

```
ALTER DATABASE SET DEFAULT BIGFILE TABLESPACE;
```

Example 13-17 Changing the Default Temporary Tablespace for the Root

This `ALTER DATABASE` statement uses a `DEFAULT TEMPORARY TABLESPACE` clause to set the default temporary tablespace to `root_temp` for the root.

```
ALTER DATABASE DEFAULT TEMPORARY TABLESPACE root_temp;
```

See Also:

- "Modifying a PDB at the Database Level"
- *Oracle Database SQL Language Reference*

Executing SQL in a Different Container

To execute SQL in a different container, use the `CONTAINERS` clause for DML or the `CONTAINER` clause for DDL.

Issuing DML Statements on a Container in a CDB

A DML (data manipulation language) statement issued in a CDB or application root can modify a different container in the CDB. In addition, you can specify a default container target for DML statements.

About Issuing DML Statements on a Container in a CDB

DML statements can affect database objects in a specified container in a CDB.

The container is specified by container ID. Because the container ID can appear in more than one location, the database uses the following order of precedence:

1. The CON_ID specified in the WHERE clause of a DML statement
2. The CONTAINERS_DEFAULT_TARGET database property
3. The current container, which is either the CDB root or application root

In a CDB root or an application root, a DML statement that includes the CONTAINERS clause can modify a table or view in a single container in the CDB or application container. To use the CONTAINERS clause, specify the table or view being modified in the CONTAINERS clause and the container ID affected in the WHERE clause.

You can specify a target container in an INSERT VALUES statement by specifying a value for CON_ID in the VALUES clause. Also, you can specify a target container in an UPDATE or DELETE statement by specifying a CON_ID predicate in the WHERE clause. For example, the following DML statement updates the sales.customers table in the container with a CON_ID of 7:

```
UPDATE CONTAINERS(sales.customers) ctab
   SET ctab.city_name='MIAMI'
   WHERE ctab.CON_ID=7
   AND CUSTOMER_ID=3425;
```

The following restrictions apply to the CONTAINERS clause:

- The specified schema must exist both in the container specified by CON_ID and in the CDB or application root where the statement is executed.
- The value specified for the CON_ID in the WHERE clause must refer to a PDB, application root, or application PDB within the CDB.
- INSERT as SELECT statements where the target of the INSERT is in CONTAINERS( ) is not supported.
- A multitable INSERT statement where the target of the INSERT is in CONTAINERS( ) is not supported.
- DML statements using the CONTAINERS clause require that the database listener is configured using TCP (instead of IPC) and that the PORT and HOST values are specified for each target PDB using the PORT and HOST clauses, respectively.

Specifying the Default Container for DML Statements in a CDB

To specify the default container for DML statements in a CDB, issue the ALTER DATABASE statement with the CONTAINERS DEFAULT TARGET clause.

When a DML statement is issued in a CDB root without specifying containers in the WHERE clause, the DML statement affects the default container for the CDB. The default container can be any container in the CDB, including the CDB root, a PDB, an application root, or an application PDB. Only one default container is allowed.

The CONTAINERS_DEFAULT_TARGET database property sets the default container. By default, this property is not set. You can determine the default target containers for a CDB by running the following query:

```
SELECT PROPERTY_VALUE
FROM   DATABASE_PROPERTIES
WHERE  PROPERTY_NAME='CONTAINERS_DEFAULT_TARGET';
```

1. In SQL*Plus, ensure that the current container is the CDB root or application root.

   The current user must have the commonly granted ALTER DATABASE privilege.
2. Run the ALTER DATABASE statement with the CONTAINERS DEFAULT TARGET clause.

Example 13-18 Specifying the Default Container for DML Statements in a CDB

This example specifies that PDB1 is the default container for DML statements in the CDB.

```
ALTER DATABASE CONTAINERS DEFAULT TARGET = (PDB1);
```

Example 13-19 Clearing the Default Container

This example clears the default container setting. When it is not set, the default container is the CDB root.

```
ALTER DATABASE CONTAINERS DEFAULT TARGET = NONE;
```

Executing DDL Statements in a CDB

In a CDB, you can execute a data definition language (DDL) statement in the current container or in all containers.

About Executing DDL Statements in a CDB

In a CDB, some DDL statements can apply to all containers or to the current container only.

To specify which containers are affected, use the CONTAINER clause:

```
CONTAINER = { CURRENT | ALL }
```

The following settings are possible:

- CURRENT means that the statement applies only to the current container.
- ALL means that the statement applies to all containers in the CDB, including the root and all PDBs.

The following restrictions apply to the CONTAINER clause in DDL statements:

- The restrictions described in "About the Current Container".
- You can use the CONTAINER clause only with the DDL statements listed in Table 13-4.

Table 13-4 DDL Statements and the CONTAINER Clause in a CDB

DDL Statement	CONTAINER = CURRENT	CONTAINER = ALL
`CREATE USER`	Creates a local user in the current PDB.	Creates a common user.
`ALTER USER`	Alters a local user in the current PDB.	Alters a common user.
`CREATE ROLE`	Creates a local role in the current PDB.	Creates a common role.
`GRANT`	Grants a privilege in the local container to a local user, common user, or local role. The `SET CONTAINER` privilege can be granted to a user-created common user in the current PDB.	Grants a system privilege or object privilege on a common object to a common user or common role. The specified privilege is granted to the user or role across the entire CDB.
`REVOKE`	Revokes a privilege in the local container from a local user, common user, or local role. This statement can revoke only a privilege granted with `CURRENT` specified in the `CONTAINER` clause from the specified user or role in the local container. The statement does not affect privileges granted with `ALL` specified in the `CONTAINER` clause. The `SET CONTAINER` privilege can be revoked from a user-created common user in the current PDB.	Revokes a system privilege or object privilege on a common object from a common user or common role. The specified privilege is revoked from the user or role across the entire CDB. This statement can revoke only a privilege granted with `ALL` specified in the `CONTAINER` clause from the specified common user or common role. The statement does not affect privileges granted with `CURRENT` specified in the `CONTAINER` clause. However, any privileges granted locally that depend on the privilege granted commonly that is being revoked are also revoked.

All other DDL statements apply to the current container only.

In addition to the usual rules for user, role, and profile names, the following rules and best practices apply when you create a user, role, or profile in a CDB:

- It is best practice for common user, role, and profile names to start with a prefix to avoid naming conflicts between common users, roles, and profiles and local users, roles, and profiles. You specify this prefix with the `COMMON_USER_PREFIX` initialization parameter in the CDB root. By default, the prefix is `C##` or `c##` in the CDB root.
- In an application container, it is best practice for application common user, role, and profile names to start with a prefix to avoid naming conflicts between application common users, roles, and profiles and local users, roles, and profiles. You specify this prefix with the `COMMON_USER_PREFIX` initialization parameter in the application root. By default, the prefix is `NULL` in an application root.
- When the `COMMON_USER_PREFIX` initialization parameter is set in an application root, the setting applies to the application common user, role, and profile names in the application container. The prefix can be different in the CDB root and in an application root, and the prefix can be different in different application containers.

- Common user, role, and profile names must consist only of ASCII characters. This restriction does not apply to application common user, role, and profile names.
- Local user, role, and profile names cannot start with the prefix specified for common users with the `COMMON_USER_PREFIX` initialization parameter.
- Local user, role, and profile names cannot start with `C##` or `c##`.
- Regardless of the value of `COMMON_USER_PREFIX` in the CDB root, application common user, role, and profile names cannot start with `C##` or `c##`.
- Application common user, role, and profile names cannot start with the prefix specified for common users with the `COMMON_USER_PREFIX` initialization parameter.

See Also:

- "Modifying a CDB with ALTER SYSTEM" for information about using the `ALTER SYSTEM` statement in a CDB
- *Oracle Database SQL Language Reference*
- *Oracle Database Concepts*
- *Oracle Database Security Guide* for more information about managing users in a CDB
- *Oracle Database Reference* for more information about the `COMMON_USER_PREFIX` initialization parameter

Executing a DDL Statement in the Current Container

Specify `CURRENT` in the `CONTAINER` clause of a DDL statement to execute the statement in the current container.

The supported DDL statements are listed in Table 13-4.

The current user must be granted the required privileges to execute the DDL statement in the current container. For example, to create a user, the current user must be granted the `CREATE USER` system privilege in the current container.

To execute a DDL statement in the current container:

1. In SQL*Plus, access a container.

   See "Accessing a Container in a CDB with SQL*Plus".

2. Execute the DDL statement with `CONTAINER` set to `CURRENT`.

A local user's user name cannot start with the prefix specified by the `COMMON_USER_PREFIX` initialization parameter. By default, in the CDB root, the prefix is `C##` or `c##`. An application root can specify its own prefix for an application container. In addition, a common user's name must consist only of ASCII characters. The specified tablespace must exist in the PDB.

Example 13-20 Creating Local User in a PDB

This example creates the local user `testpdb` in the current PDB.

```
CREATE USER testpdb IDENTIFIED BY password
   DEFAULT TABLESPACE pdb1_tbs
   QUOTA UNLIMITED ON pdb1_tbs
   CONTAINER = CURRENT;
```

Executing a DDL Statement in All Containers in a CDB

Specify `ALL` in the `CONTAINER` clause of a DDL statement to execute the statement in all containers in a CDB.

The supported DDL statements are listed in Table 13-4.

The following prerequisites must be met:

- The current user must be a common user.
- The current user must be granted the required privileges commonly to execute the DDL statement. For example, to create a user, the current user must be granted the `CREATE USER` system privilege commonly.

To execute a DDL statement in all containers in a CDB:

1. In SQL*Plus, ensure that the current container is the root.

   See "About Container Access in a CDB".

2. Execute the DDL statement with `CONTAINER` set to `ALL`.

A common user's user name must start with the prefix specified by the `COMMON_USER_PREFIX` initialization parameter. By default, in the CDB root, the prefix is `C##` or `c##`. An application root can specify its own prefix for an application container. In addition, a common user's name must consist only of ASCII characters. The specified tablespace must exist in the root and in all PDBs.

Example 13-21 Creating Common User in a CDB

This example creates the common user `c##testcdb`.

```
CREATE USER c##testcdb IDENTIFIED BY password
   DEFAULT TABLESPACE cdb_tbs
   QUOTA UNLIMITED ON cdb_tbs
   CONTAINER = ALL;
```

Running Oracle-Supplied SQL Scripts in a CDB

You can use the `catcon.pl` script to run Oracle-supplied SQL or SQL scripts within a CDB. You can run the script against any specified containers.

About Running Oracle-Supplied SQL Scripts in a CDB

In a CDB, the `catcon.pl` script is the best way to run SQL scripts and SQL statements.

An Oracle Database installation includes several SQL scripts. These scripts perform operations such as creating data dictionary views and installing options.

The `catcon.pl` script can run scripts in the root and in specified PDBs in the correct order, and 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.

Note:

- Unless you exclude the PDB seed when you run `catcon.pl`, the SQL script or SQL statement is run on the PDB seed.
- You can use the `catcon.pl` script to run scripts on both CDBs and non-CDBs.

Syntax and Parameters for catcon.pl

The `catcon.pl` script is a Perl script that must be run at an operating system prompt.

The `catcon.pl` script has the following syntax and parameters:

```
$ORACLE_HOME/perl/bin/perl $ORACLE_HOME/rdbms/admin/catcon.pl
[--usr username[/password]]
[--int_usr username[/password]]
[--script_dir directory]
[--log_dir directory]
[{--incl_con|--excl_con} container]
[--echo]
[--spool]
[--error_logging { ON | errorlogging-table-other-than-SPERRORLOG } ]
[--app_con application_root]
[--no_set_errlog_ident]
[--diag]
[-ignore_unavailable_pdbs]
[--verbose]
[--force_pdb_mode pdb_mode]
[--recover]
--log_file_base log_file_name_base
-- { SQL_script [arguments] | --x'SQL_statement' }
```

Ensure that `--x SQL_statement` is preceded by `--` if it follows any single-letter parameter. If `--x SQL_statement` is preceded by a script name or another `--x SQL_statement`, then do not precede it with `--`. Also, note that the SQL statement must be inside single quotation marks.

Command line parameters to SQL scripts can be introduced using `--p`. Interactive (or secret) parameters to SQL scripts can be introduced using `--P`.

To view the help for the `catcon.pl` script, change directories to `$ORACLE_HOME/perl/bin/`, and then run the following command:

```
perl $ORACLE_HOME/rdbms/admin/catcon.pl --help
```

The following table describes the `catcon.pl` parameters. A parameter is optional unless it is indicated that it is required.

The short parameter names in the following table are for backward compatibility. Some parameters do not have short names.

Table 13-5 catcon.pl Parameters

Parameter	Short Name	Description
`--usr`	`-u`	Specifies the user name and password to connect to the root and the specified PDBs. Specify a common user with the required privileges to run the SQL script or the SQL statement. The default is "`/ AS SYSDBA`". If no password is supplied, then `catcon.pl` prompts for a password.
`--int_usr`	`-U`	Specifies the user name and password to connect to the root and the specified PDBs. Specify a common user with the required privileges to perform internal tasks, such as querying CDB metadata. The default is `/ AS SYSDBA`. If no password is supplied, then `catcon.pl` prompts for a password.
`--script_dir`	`-d`	Directory that contains the SQL script. The default is the current directory.
`--log_dir`	`-l`	Directory into which `catcon.pl` writes log files. The default is the current directory.
`{--incl_con|--excl_con}`	`{-c|-C}`	The containers in which the SQL script is run or is not run. The `--incl_con` parameter lists the containers in which the SQL script is run. The `--excl_con` parameter lists the containers in which the SQL script is not run. Specify containers in a space-delimited list of PDB names enclosed in single quotation marks. The `--incl_con` and `--excl_con` parameters are mutually exclusive. When this parameter is used, the `--app_con` parameter cannot be used.
`--echo`	`-e`	Sets echo `ON` while running the script. The default is echo `OFF`.
`--spool`	`-s`	Spools the output of every script into a file with the following name: *`log-file-name-base_script-name-without-extension_[container-name-if-any].default-extension`*

Table 13-5 (Cont.) catcon.pl Parameters

Parameter	Short Name	Description
`--error_logging`	`-E`	When set to `ON`, the default error logging table is used. `ON` is the default setting. When set to `ON`, errors are written to the table `SPERRORLOG` in the current schema in each container in which the SQL script runs. If this table does not exist in a container, then it is created automatically. When a table other than `SPERRORLOG` is specified, errors are written to the specified table. The table must exist in each container in which the SQL script runs, and the current user must have the necessary privileges to perform DML operations on the table in each of these containers. See *SQL*Plus User's Guide and Reference* for more information about the error logging table.
`--app_con`	`-F`	Specify an application root. The scripts are run in the application root and in the application PDBs that are plugged into the application root. When this parameter is used, the `--incl_con` and `--excl_con` parameters cannot be used.
`--no_set_errlog_ident`	`-I`	Do not issue a `SET ERRORLOGGING` identifier. This option is intended for cases in which the `SET ERRORLOGGING` identifier is already set and should not be overwritten.
`--diag`	`-g`	Turns on the generation of debugging information.
`--verbose`	`-v`	Turns on verbose output.
`--ignore_unavailable_pdbs`	`-f`	Ignore PDBs that are closed or, if the `--incl_con` or `--excl_con` option is used, do not exist and process only open PDBs that were specified explicitly or implicitly. When this option is not specified and some specified PDBs do not exist or are not open, an error is returned and none of the containers are processed.
`--force_pdb_mode`	n/a	The required open mode for all PDBs against which the scripts are run. Specify one of the following values: • `UNCHANGED` • `READ WRITE` • `READ ONLY` • `UPGRADE` • `DOWNGRADE` When a value other than `UNCHANGED` is specified, all of the PDBs against which the script is run are changed to the specified open mode. If a PDB is open in a different mode, then the PDB is closed and re-opened in the specified mode. After all of the scripts are run, each PDB is restored to its original open mode. When `UNCHANGED`, the default, is specified, the open mode of the PDBs is not changed.
`--recover`	`-R`	Causes `catcon.pl` to attempt to recover if a SQL*Plus process that it spawned ends unexpectedly. When this parameter is not specified, `catcon.pl` does not attempt to recover the process and closes.
`--log_file_base`	`-b`	(Required) The base name for log file names.

Running the catcon.pl Script

Examples illustrate running the `catcon.pl` script.

If a SQL script or SQL statement run by `catcon.pl` performs data manipulation language (DML) or data definition language (DDL) operations, then the containers being modified must be in read/write mode.

To run the catcon.pl script:

1. Open a command line prompt.
2. Run the `catcon.pl` script and specify one or more SQL scripts or SQL statements:

```
cd $ORACLE_HOME/perl/bin/
perl $ORACLE_HOME/rdbms/admin/catcon.pl parameters SQL_script
perl $ORACLE_HOME/rdbms/admin/catcon.pl parameters -- --
xSQL_statement
```

Example 13-22 Running the catblock.sql Script in All Containers in a CDB

The following example runs the `catblock.sql` script in all of the containers of a CDB (the backslash indicates line continuation):

```
$ORACLE_HOME/perl/bin/perl $ORACLE_HOME/rdbms/admin/catcon.pl \
--usr SYS --script_dir $ORACLE_HOME/rdbms/admin \
--log_file_base catblock_output catblock.sql
```

The following parameters are specified:

- The `--usr` parameter specifies that `SYS` user runs the script in each container.
- The `--script_dir` parameter specifies that the SQL script is in the `$ORACLE_HOME/rdbms/admin` directory.
- The `--log_file_base` parameter specifies that the base name for log file names is `catblock_output`.

Default parameter values are used for all other parameters. Neither the `--incl_con` nor the `--excl_con` parameter is specified. Therefore, catcon.pl runs the script in all containers by default.

Example 13-23 Running the catblock.sql Script in Specific PDBs

The following example runs the `catblock.sql` script in the `hrpdb` and `salespdb` PDBs in a CDB.

```
$ORACLE_HOME/perl/bin/perl $ORACLE_HOME/rdbms/admin/catcon.pl \
--usr SYS --int_usr SYS --script_dir $ORACLE_HOME/rdbms/admin \
--log_dir '/disk1/script_output' --incl_con 'HRPDB SALESPDB' \
--log_file_base catblock_output catblock.sql
```

The following parameters are specified:

- The `--usr` parameter specifies that `SYS` user runs the script in each container.
- The `--int_usr` parameter specifies that `SYS` user performs internal tasks.

- The `--script_dir` parameter specifies that the SQL script is in the `$ORACLE_HOME/rdbms/admin` directory.
- The `--log_dir` parameter specifies that the output files are placed in the `/disk1/script_output` directory.
- The `--incl_con` parameter specifies that the SQL script is run in the `hrpdb` and `salespdb` PDBs. The script is not run in any other containers in the CDB.
- The `--log_file_base` parameter specifies that the base name for log file names is `catblock_output`.

Example 13-24 Running the catblock.sql Script in All Containers Except for Specific PDBs

The following example runs the `catblock.sql` script in all of the containers in a CDB except for the `hrpdb` and `salespdb` PDBs.

```
$ORACLE_HOME/perl/bin/perl $ORACLE_HOME/rdbms/admin/catcon.pl \
--usr SYS --script_dir $ORACLE_HOME/rdbms/admin \
--log_dir '/disk1/script_output' --excl_con 'HRPDB SALESPDB' \
--log_file_base catblock_output catblock.sql
```

The following parameters are specified:

- The `--usr` parameter specifies that `SYS` user runs the script in each container.
- The `--script_dir` parameter specifies that the SQL script is in the `$ORACLE_HOME/rdbms/admin` directory.
- The `--log_dir` parameter specifies that the output files are placed in the `/disk1/script_output` directory.
- The `--excl_con` parameter specifies that the SQL script is run in all of the containers in the CDB except for the `hrpdb` and `salespdb` PDBs.
- The `--log_file_base` parameter specifies that the base name for log file names is `catblock_output`.

Example 13-25 Running a SQL Script with Command Line Parameters

The following example runs the `custom_script.sql` script in all of the containers of a CDB.

```
cd $ORACLE_HOME/perl/bin/
perl $ORACLE_HOME/rdbms/admin/catcon.pl --usr SYS --script_dir /u01/
scripts \
--log_file_base custom_script_output custom_script.sql '--phr' \
'--PEnter password for user hr:'
```

The following parameters are specified:

- The `--usr` parameter specifies that `SYS` user runs the script in each container.
- The `--script_dir` parameter specifies that the SQL script is in the /u01/scripts directory.
- The `--log_file_base` parameter specifies that the base name for log file names is `custom_script_output`.

- The `--p` parameter specifies `hr` for a command line parameter
- The `--P` parameter specifies an interactive parameter that prompts for the password of user `hr`.

Default parameter values are used for all other parameters. Neither the `-incl_con` nor the `-excl_con` parameter is specified. Therefore, catcon.pl runs the script in all containers by default.

Example 13-26 Running a SQL Statement in All Containers in a CDB

The following example runs a SQL statement in all of the containers of a CDB.

```
cd $ORACLE_HOME/perl/bin/
perl $ORACLE_HOME/rdbms/admin/catcon.pl --usr SYS --echo \
--log_file_base select_output -- --x"SELECT * FROM DUAL"
```

The following parameters are specified:

- The `--usr` parameter specifies that `SYS` user runs the script in each container.
- The `--echo` parameter shows output for the SQL statement.
- The `--log_file_base` parameter specifies that the base name for log file names is `select_output`.
- The SQL statement `SELECT * FROM DUAL` is inside quotation marks and is preceded by `--x`. Because `--x` is preceded by a parameter (`--log_file_base`), it must be preceded by `--`.

Default parameter values are used for all other parameters. Neither the `-incl_con` nor the `-excl_con` parameter is specified. Therefore, catcon.pl runs the SQL statement in all containers by default.

See Also:

- "Modifying the Open Mode of PDBs"
- *Oracle Database Administrator's Guide* for information about the `catblock.sql` script
- *Oracle Database SQL Language Reference* for more information about SQL scripts

Executing Code in Containers Using the DBMS_SQL Package

When you are executing PL/SQL code in a container in a CDB, and you want to execute one or more SQL statements in a different container, use the `DBMS_SQL` package to switch containers.

For example, you can use the `DBMS_SQL` package to switch containers when you need to perform identical actions in more than one container.

The following are considerations for using `DBMS_SQL` to switch containers:

- A transaction cannot span multiple containers.

If the set of actions you must perform in the target container requires a transaction, then consider using an autonomous transaction and perform a commit or rollback as the last action.

- `SET ROLE` statements are not allowed.

Example 13-27 Performing Identical Actions in More Than One Container

This example includes a PL/SQL block that creates the `identact` table in the `hr` schema in two PDBs (`pdb1` and `pdb2`). The example also inserts a row into the `identact` table in both PDBs.

```
DECLARE
  c1 INTEGER;
  rowcount INTEGER;
  taskList VARCHAR2(32767) :=
    'DECLARE
      PRAGMA AUTONOMOUS TRANSACTION;
     BEGIN
       -- Create the hr.identact table.
       EXECUTE IMMEDIATE
         ''CREATE TABLE hr.identact
             (actionno NUMBER(4) NOT NULL,
              action VARCHAR2 (10))'';
       EXECUTE IMMEDIATE
         ''INSERT INTO identact VALUES(1, 'ACTION1')'';
       -- A commit is required if the tasks include DML.
       COMMIT;
       EXCEPTION
         WHEN OTHERS THEN
           -- If there are errors, then drop the table.
           BEGIN
             EXECUTE IMMEDIATE ''DROP TABLE identact'';
           EXCEPTION
            WHEN OTHERS THEN
              NULL;
            END;
      END;';
  TYPE containerListType IS TABLE OF VARCHAR2(128) INDEX BY PLS_INTEGER;
  containerList  containerListType;
BEGIN
  containerList(1) := 'PDB1';
  containerList(2) := 'PDB2';
  c1 := DBMS_SQL.OPEN_CURSOR;
  FOR conIndex IN containerList.first..containerList.last LOOP
    DBMS_OUTPUT.PUT_LINE('Creating in container: ' ||
containerList(conIndex));
    DBMS_SQL.PARSE(
      c => c1 ,
      statement                  => taskList,
      language_flag              => DBMS_SQL.NATIVE,
      edition                    => NULL,
      apply_crossedition_trigger => NULL,
      fire_apply_trigger         => NULL,
      schema                     => 'HR',
      container                  => containerList(conIndex));
```

```
    rowcount := DBMS_SQL.EXECUTE(c=>c1);
  END LOOP;
  DBMS_SQL.CLOSE_CURSOR(c=>c1);
END;
/
```

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_SQL` package
- *Oracle Database PL/SQL Language Reference* for more information about autonomous transactions

Shutting Down a CDB Instance

You can shut down a CDB instance in the same way that you shut down a non-CDB instance.

Prerequisites

The following prerequisites must be met:

- The CDB instance must be mounted or open.
- The current user must be a common user with `SYSDBA`, `SYSOPER`, `SYSBACKUP`, or `SYSDG` administrative privilege. To shut down a CDB, you must exercise this privilege using `AS SYSDBA`, `AS SYSOPER`, `AS SYSBACKUP`, or `AS SYSDG`, respectively, at connect time.

To shut down a CDB:

1. In SQL*Plus, ensure that the current container is the root.

   See "Connecting to a Container Using the SQL*Plus CONNECT Command".
2. Shut down the CDB instance.

See Also:

- "Modifying the Open Mode of PDBs"
- "About the Current Container"
- *Oracle Database Administrator's Guide* to learn how to shut down an instance

14

Administering a CDB Fleet

A **CDB fleet** is a collection of CDBs and hosted PDBs that you can manage as one logical CDB.

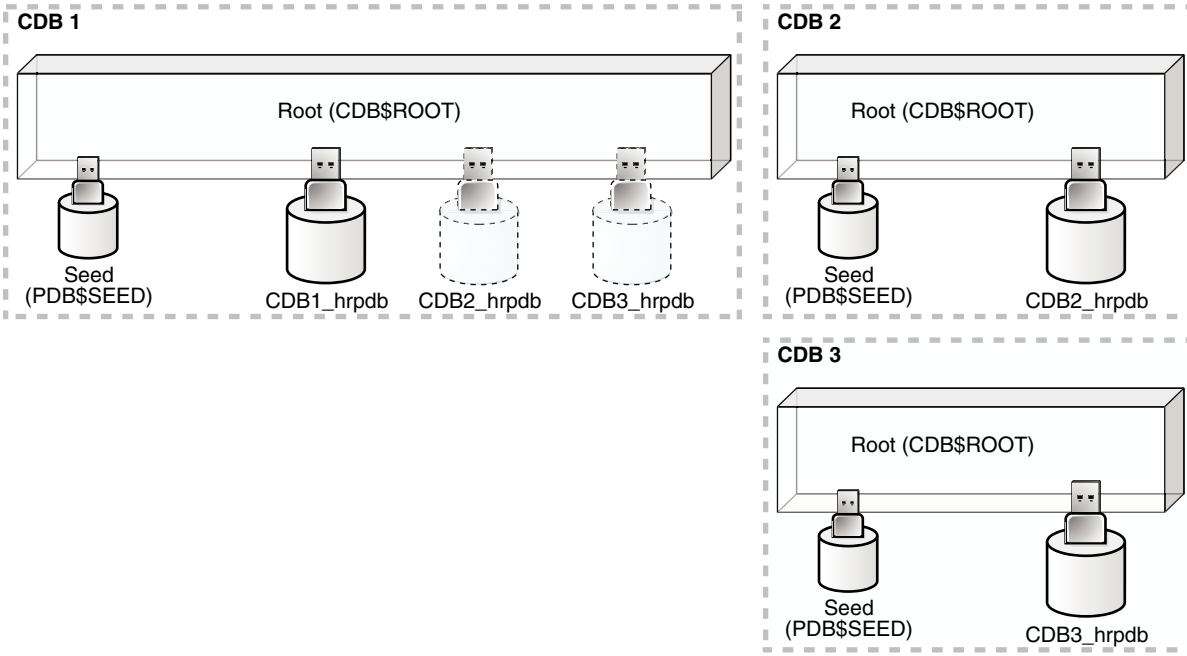
About CDB Fleets

A **lead CDB** is the central location for monitoring and managing the CDBs in the fleet.

Designate one CDB in the fleet to be the lead CDB by setting its `LEAD_CDB` database property to `TRUE`. The other CDBs in the fleet point to the lead CDB by setting the `LEAD_CDB_URI` database property. After you configure the CDB fleet, PDB information from the various CDBs is synchronized with the lead CDB. All PDBs in the CDBs are now “visible” in the lead CDB, enabling you to access the PDBs in the fleet as a single, logical CDB from the lead CDB.

The following figure shows a CDB fleet consisting of CDB1, CDB2, and CDB3. The lead CDB is CDB1. CDB2_hrpdb, which resides in CDB2, is visible in CDB1. CDB3_hrpdb, which resides in CDB3, is also visible in CDB1.

Figure 14-1 CDB Fleet



All Oracle Database features, such as Oracle Real Application Cluster (Oracle RAC), RMAN, point-in-time recovery, and flashback features, are supported for CDBs in the fleet.

You can use the following cross-container features to access the CDBs and PDBs in a CDB fleet:

- `CDB` views
- `GV$` views
- The `CONTAINERS` clause
- Container maps

If a common application schema is configured with application containers, then these cross-container features enable query and data aggregation across PDBs in different CDBs managed in the fleet.

> **Note:**
>
> - Each PDB name must be unique across all CDBs in a CDB fleet.
> - You can create a PDB in any CDB in the fleet, but you can only open a PDB in the CDB where it was created.

> **See Also:**
>
> - "Monitoring CDBs and PDBs"
> - "Partitioning by PDB with Container Maps"

Purpose of a CDB Fleet

A CDB fleet provides the database infrastructure for scalability and centralized management of many CDBs.

A CDB fleet is useful in the following situations:

- The number of PDBs you must provision exceeds the `MAX_PDBS` initialization parameter setting, requiring you to create multiple CDBs.
- Different PDBs in a single configuration require different types of servers to function optimally.

  For example, some PDBs might process a large transaction load, while other PDBs are used mainly for monitoring, and you want the appropriate server resources for these PDBs, such as CPU, memory, I/O rate, and storage systems.
- Different PDBs that use the same application must reside in different locations.

Monitoring and Diagnostic Collection Across CDBs

The lead CDB can run monitoring and reporting applications that execute across the CDBs in the fleet. You can install a monitoring application in one container, and then use `CDB` views and `GV$` views to monitor and process diagnostic data for the entire CDB fleet. A cross-container query issued in the lead CDB can automatically execute in all PDBs across the CDB fleet.

Software as a Service (SaaS) Applications

Using a common schema and common application objects in different application containers across the CDB fleet, you can use the `CONTAINERS` clause or a container map to run queries across all PDBs in the CDB fleet. To ensure a common application schema across the CDBs, the application can be installed in an application root.

A typical use case involves installing the master application root in the lead CDB. An application root clone resides in every other CDB in the fleet. Proxy PDBs for the application root clones reside in the master application root.

Database as a Service (DBaaS) Applications

The lead CDB can serve as a central location where you can collect and view usage metrics and status of all or a subset of the PDBs provisioned in the CDB fleet.

Microservices

Microservices are a specialization of service-oriented architectures used to build flexible, independently deployable software systems. With microservices, each team can deploy and manage a CDB fleet with customized scaling and availability SLAs. The CDBs can use different storage systems and configuration settings and cater to different types of workloads. The lead CDB can help the central DBA manage the collection of CDBs associated with each individual microservice.

See Also:

- "Monitoring CDBs and PDBs"
- "Partitioning by PDB with Container Maps"
- *Oracle Database Reference* to learn more about `MAX_PDBS`

Setting the Lead CDB in a CDB Fleet

Set the lead CDB in a CDB fleet by setting the `LEAD_CDB` database property to `true`.

To set the lead CDB in a CDB fleet:

1. In SQL*Plus, ensure that the current container is the root of the CDB that will be the lead CDB.
2. Optionally, check the current `LEAD_CDB` database property by running the following query:

```
SELECT PROPERTY_VALUE
FROM   DATABASE_PROPERTIES
WHERE  PROPERTY_NAME='LEAD_CDB';
```

3. Set the `LEAD_CDB` database property to `TRUE`.

Example 14-1 Setting the Lead CDB Database Property to true

1. Access the CDB root:

```
ALTER SESSION SET CONTAINER = CDB$ROOT;
```

2. Run the following SQL statement:

```
ALTER DATABASE SET LEAD_CDB = TRUE;
```

> **See Also:**
>
> "About Container Access in a CDB"

Designating a CDB Fleet Member

Designate a fleet member by setting the `LEAD_CDB_URI` database property to a database link that points to the lead CDB.

Prerequisites

You must use a database link with fixed user semantics, which means that the user name and password are in the link definition. The link cannot use connected user semantics, in which case the user name and password are not in the link definition.

To designate a CDB fleet member:

1. In SQL*Plus, ensure that the current container is the root of the CDB that you want to designate as a fleet member.
2. Optionally, check the current `LEAD_CDB_URI` database property by running the following query:

```
SELECT PROPERTY_VALUE FROM DATABASE_PROPERTIES WHERE
PROPERTY_NAME='LEAD_CDB_URI';
```

3. If a database link does not exist, then create a link to the root of the lead CDB in the fleet.

   The database link must be a fixed common user database link.
4. Set the `LEAD_CDB_URI` database property to the name of the database link to the lead CDB.

Example 14-2 Designating a CDB Fleet Member

This example assumes that the lead CDB is `cdb1` and that the database link to the lead CDB does not exist. It also assumes that the network is configured so that the current CDB can connect to `cdb1` using the `lead_pod` service name.

1. Access the root of the CDB that you want to designate as a fleet member:

```
ALTER SESSION SET CONTAINER = CDB$ROOT;
```

2. Create the database link to `cdb1`:

```
CREATE PUBLIC DATABASE LINK lead_link
   CONNECT TO C##CF1 IDENTIFIED BY password
   USING 'lead_pod';
```

3. Set the `LEAD_CDB_URI` property to the name of the database link:

```
ALTER DATABASE SET LEAD_CDB_URI = 'dblink:LEAD_LINK';
```

See Also:

- "About Container Access in a CDB"
- *Oracle Database Administrator's Guide* for information about fixed user database links

15
Administering PDBs

Administering PDBs includes tasks such as connecting to a PDB, modifying a PDB, and managing services associated with PDBs.

Related Topics

- Tools for a Multitenant Environment
  You can use various tools to configure and administer a multitenant environment.

About PDB Administration

Administering a pluggable database (PDB) involves a subset of the tasks required to administer a non-CDB.

In this subset of tasks, most are the same for a PDB and a non-CDB, but differences exist. For example, there are differences when you modify the open mode of a PDB. Also, a PDB administrator is limited to managing a single PDB and cannot manage other PDBs in the multitenant container database (CDB).

> **See Also:**
>
> "Modifying a PDB at the Database Level" for more information about changing the open mode of the current PDB

Tasks Common to PDBs and Non-CDBs

Most administrative tasks are the same for a PDB and a non-CDB.

When you are administering a PDB, you can modify the PDB with an `ALTER DATABASE`, `ALTER PLUGGABLE DATABASE`, or `ALTER SYSTEM` statement. You can also execute DDL statements on the PDB. The following table describes some of these tasks common to a PDB and non-CDB.

Table 15-1 Administrative Tasks Common to PDBs and Non-CDBs

Task	Description	Additional Information
Managing tablespaces	You can create, modify, and drop tablespaces for a PDB. You can specify a default tablespace and default tablespace type for each PDB. Also, there is a default temporary tablespace for each PDB. You optionally can create additional temporary tablespaces for use by individual PDBs.	"Modifying a PDB at the Database Level" *Oracle Database Administrator's Guide* for information about managing tablespaces

Table 15-1 (Cont.) Administrative Tasks Common to PDBs and Non-CDBs

Task	Description	Additional Information
Managing data files and temp files	Each PDB has its own data files. You can manage data files and temp files in the same way that you would manage them for a non-CDB. You can also limit the amount of storage used by the data files for a PDB by using the `STORAGE` clause in a `CREATE PLUGGABLE DATABASE` or `ALTER PLUGGABLE DATABASE` statement.	"Modifying a PDB at the Database Level" *Oracle Database Administrator's Guide* for information about managing data files and temp files
Managing schema objects	You can create, modify, and drop schema objects in a PDB in the same way that you would in a non-CDB. You can also create triggers that fire for a specific PDB. When you manage database links in a CDB, the root has a unique global database name, and so does each PDB. The global name of the root is defined by the `DB_NAME` and `DB_DOMAIN` initialization parameters. The global database name of a PDB is defined by the PDB name and the `DB_DOMAIN` initialization parameter. The global database name of each PDB must be unique within the domain.	*Oracle Database Administrator's Guide* for more information about schema objects *Oracle Database Administrator's Guide* *Oracle Database PL/SQL Language Reference* for information about creating triggers in a CDB

Tasks Specific to CDBs

Some administrative tasks cannot be performed when the current container is a PDB.

The following tasks are performed by a common user for the entire CDB or for the CDB root when the current container is the root:

- Starting up and shutting down a CDB instance
- Modifying the CDB or the root with an `ALTER DATABASE` statement
- Modifying the CDB or the root with an `ALTER SYSTEM` statement
- Executing data definition language (DDL) statements on a CDB or the root
- Managing the following components:
  - Processes
  - Memory
  - Errors and alerts
  - Diagnostic data
  - Control files
  - The online redo log and the archived redo log files

  - Undo
- Creating, plugging in, unplugging, and dropping PDBs

A common user whose current container is the root can also change the open mode of one or more PDBs. Similarly, a common user or local user whose current container is a PDB can change the open mode of the current PDB.

> **See Also:**
>
> - "About the Current Container"
> - " Administering a CDB" for more information about this task and other tasks related to administering a CDB or the root

Managing Connections to a PDB

You manage connections for a PDB in the same way as for a non-CDB, with some special considerations.

Connecting to a PDB

You can use several techniques to connect to a PDB with the SQL*Plus `CONNECT` command.

This section assumes that you understand how to connect to a non-CDB in SQL*Plus.

You can use the following techniques to connect to a PDB with the SQL*Plus `CONNECT` command:

- Local connection with operating system authentication
- Database connection using easy connect
- Database connection using a net service name

Prerequisites

The following prerequisites must be met:

- The user connecting to the PDB must be granted the `CREATE SESSION` privilege in the PDB.
- To connect to a PDB as a user that does not have `SYSDBA`, `SYSOPER`, `SYSBACKUP`, or `SYSDG` administrative privilege, the PDB must be open.

> **Note:**
>
> This section assumes that the user connecting to the PDB using a local user account. You can also connect to the PDB as a common user, and you can connect to the root as a common user and switch to the PDB.

To connect to a PDB using the SQL*Plus CONNECT command:

1. Configure your environment so that you can open SQL*Plus.
2. Start SQL*Plus with the `/NOLOG` argument:

   ```
   sqlplus /nolog
   ```

3. Issue a `CONNECT` command using easy connect or a net service name to connect to the PDB.

   To connect to a PDB, connect to a service with a `PDB` property.

Example 15-1 Connecting to a PDB in SQL*Plus Using the PDB's Net Service Name

The following command connects to the `hr` user using the `hrapp` service. The `hrapp` service has a `PDB` property for the `hrpdb` PDB. This example assumes that the client is configured to have a net service name for the `hrapp` service.

```
CONNECT hr@hrapp
```

See Also:

- "Modifying the Open Mode of PDBs" and "Modifying a PDB at the Database Level" for information about changing the open mode of a PDB.
- "About Container Access in a CDB" for information about connecting to a PDB as a common user
- "Managing Services for PDBs"
- *Oracle Database Administrator's Guide* for information about connecting to the database with SQL*Plus

Managing Services for PDBs

You can create, modify, or remove services for a PDB.

See Also:

Oracle Database Administrator's Guide

About Services for PDBs

Each PDB has a default service, but you can create your own using SRVCTL or `DBMS_SERVICE`.

The PDB Property

The PDB property associates a service with a PDB. When a client connects to a service with a PDB property, the current container for the connection is the PDB.

The PDB property is required only when you do either of the following:

- Create a service
- Modify the PDB property of a service

You do not specify a PDB property when you start, stop, or remove a service. Also, you do not need to specify a PDB property when you modify a service without modifying its PDB property.

You can view the PDB property for a service by querying the ALL_SERVICES data dictionary view. Alternatively, when using the SRVCTL utility, you can use the `srvctl config service` command.

> **See Also:**
>
> "About the Current Container"

Default and User-Defined Services

Creating a PDB creates a new default service for the PDB automatically.

Each database service name must be unique in a CDB, and each database service name must be unique within the scope of all the CDBs whose instances are reached through a specific listener. The default service has the same name as the PDB. You cannot manage this service, which you should only use for administrative tasks.

Always use user-defined services for applications. The reason is that you can customize user-defined services to fit the requirements of your applications. Oracle recommends that you not use the default PDB service for applications.

> **Note:**
>
> Do not associate a service with a proxy PDB.

In an Oracle Clusterware environment, you must create an Oracle Clusterware resource for each service that is created for the PDB. When your database is being managed by Oracle Restart or Oracle Clusterware, and when you use the SRVCTL utility to start a service with a PDB property for a PDB that is closed, the PDB is opened in read/write mode on the nodes where the service is started. However, stopping a PDB service does not change the open mode of the PDB.

When you unplug or drop a PDB, the services of the unplugged or dropped PDB are not removed automatically. You can remove these services manually.

> **See Also:**
>
> - "Modifying a PDB with the ALTER PLUGGABLE DATABASE Statement" for information about changing the open mode of a PDB
> - "Creating a Proxy PDB That References an Application Root Replica"

Tools for Managing Services

Oracle recommends using the SRVCTL utility to create and modify services. Alternatively, you can use the `DBMS_SERVICE` package.

SRVCTL

If your single-instance database is being managed by Oracle Restart or your Oracle RAC database is being managed by Oracle Clusterware, then use the Server Control (SRVCTL) utility to create, modify, or remove the service.

To create a service for a PDB using the SRVCTL utility, use the `add service` command and specify the PDB in the `-pdb` parameter. If you do not specify `-pdb`, then the service is associated with the root.

To modify the `PDB` property of a service using the SRVCTL utility, use the `modify service` command and specify the PDB in the `-pdb` parameter. To remove a service for a PDB using the SRVCTL utility, use the `remove service` command.

You can use other SRVCTL commands to manage the service, such as the `start service`, `stop service`, and `relocate service` commands, even if they do not include the `-pdb` parameter.

The PDB name is not validated when you create or modify a service with the SRVCTL utility. However, an attempt to start a service with invalid PDB name results in an error.

DBMS_SERVICE

If your database is not being managed by Oracle Restart or Oracle Clusterware, then use the `DBMS_SERVICE` package to create or remove a database service.

`DBMS_SERVICE` exists at the root level and in each PDB. It is owned and executed by `SYS` at each level. A PDB administrator cannot stop, relocate, or test the connection for a service that is owned by another PDB.

When you create a service with the `DBMS_SERVICE` package, the `PDB` property of the service is set to the current container. Therefore, to create a service with a `PDB` property set to a specific PDB using the `DBMS_SERVICE` package, run the `CREATE_SERVICE` procedure when the PDB is the current container. If you create a service using the `CREATE_SERVICE` procedure when the current container is the root, then the service is associated with the root.

You cannot modify the `PDB` property of a service with the `DBMS_SERVICE` package. However, you can remove a service in one PDB and create a similar service in a different PDB. In this case, the new service has the `PDB` property of the PDB in which it was created.

You can also use other DBMS_SERVICE subprograms to manage the service, such as the START_SERVICE and STOP_SERVICE procedures. You can use DBMS_SERVICE.*_CONNECTION_TEST procedures to check the health of a database connection during planned maintenance. Use the DELETE_SERVICE procedure to remove a service.

See Also:

- "Example 19-9"
- *Oracle Database Administrator's Guide* for information about configuring automatic restart of an Oracle database
- *Oracle Database PL/SQL Packages and Types Reference* for information about the DBMS_SERVICE package
- *Oracle Real Application Clusters Administration and Deployment Guide* for information about creating services in an Oracle Real Application Clusters (Oracle RAC) environment

Managing Services for a PDB Using SRVCTL and DBMS_SERVICE

You can create, modify, or remove a service with a PDB property.

To manage a service with a PDB property using the SRVCTL utility:

1. Log in to the host computer with the correct user account.
2. Ensure that you run SRVCTL from the correct Oracle home.
3. Perform one of the following operations:
   - To create or modify a service, run the `add service` command, and specify the PDB in the `-pdb` parameter.
   - To modify the PDB property of a service, run the `modify service` command, and specify the PDB in the `-pdb` parameter.
   - To remove a service, run the `remove service` command.

To create or remove a service for a PDB using the DBMS_SERVICE package:

1. In SQL*Plus, ensure that the current container is a PDB.

   See "Connecting to a PDB".
2. Run the appropriate subprogram in the DBMS_SERVICE package.

Note:

If your database is being managed by Oracle Restart or Oracle Clusterware, then use the SRVCTL utility to manage services. Do not use the DBMS_SERVICE package.

Example 15-2 Creating a Service for a PDB Using the SRVCTL Utility

This example adds the `salesrep` service for the PDB `salespdb` in the CDB with `DB_UNIQUE_NAME mycdb`:

```
srvctl add service -db mycdb -service salesrep -pdb salespdb
```

Example 15-3 Modifying the PDB Property of a Service Using the SRVCTL Utility

This example modifies the `salesrep` service in the CDB with `DB_UNIQUE_NAME mycdb` to associate the service with the `hrpdb` PDB:

```
srvctl modify service
  -db mycdb
  -service salesrep
  -pdb hrpdb
```

Example 15-4 Relocating a Service in Oracle RAC Using the SRVCTL Utility

You can use the `relocate service` command to relocate a service from one Oracle RAC instance, where the service is currently running, to another instance, where it can run. This technique applies both to services for administrator-managed databases as well as singleton services for policy-managed databases.

The following command relocates service `svc1` from Oracle RAC instance `cdb_inst1`, where it is currently running, to instance `cdb_inst2`, where it is currently not running:

```
srvctl relocate service
  db cdb
  service svc1
  oldinst cdb_inst1
  newinst cdb_inst2
  –drain_timeout NNN
  –stopoption immediate
```

The following command performs the same operation for a policy-managed database:

```
srvctl relocate service
  db cdb
  service svc1
  currentnode cdb_inst1
  targetnode cdb_inst2
  –drain_timeout NNN
  –stopoption immediate
```

Example 15-5 Removing a Service Using the SRVCTL Utility

This example removes the `salesrep` service in the CDB with `DB_UNIQUE_NAME mycdb`:

```
srvctl remove service
  -db mycdb
  -service salesrep
```

Example 15-6 Creating a Service for a PDB Using the DBMS_SERVICE Package

This example creates the `salesrep` service for the current PDB:

```
BEGIN
  DBMS_SERVICE.CREATE_SERVICE(
    service_name => 'salesrep',
    network_name => 'salesrep.example.com');
END;
/
```

The `PDB` property of the service is set to the current container. For example, if the current container is the `salespdb` PDB, then the PDB property of the service is `salespdb`.

Example 15-7 Removing a Service Using the DBMS_SERVICE Package

This example removes the `salesrep` service in the current PDB.

```
BEGIN
  DBMS_SERVICE.DELETE_SERVICE(
    service_name => 'salesrep');
END;
/
```

See Also:

- "Example 19-9"
- *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_SERVICE` package
- *Oracle Real Application Clusters Administration and Deployment Guide* for information about managing services in an Oracle Real Application Clusters (Oracle RAC) environment

Modifying the Listener Settings of a Referenced PDB

A PDB that is referenced by a proxy PDB is called a referenced PDB.

When the port or host name changes for the listener of the referenced PDB, you must modify the listener settings of the referenced PDB so that its proxy PDBs continue to function properly.

Related Topics

- Creating a PDB as a Proxy PDB
  You can create a PDB as a proxy PDB by referencing it in a remote CDB.

Altering the Listener Host Name of a Referenced PDB

When the host name of the listener for a referenced PDB changes, you must run an `ALTER PLUGGABLE DATABASE CONTAINERS HOST` statement to reset the host name of the referenced PDB so that its proxy PDBs continue to function properly.

A proxy PDB uses a database link to establish communication with its referenced PDB during PDB creation. After communication is established, the proxy PDB communicates directly with the referenced PDB without using a database link, and the database link can be dropped. When the listener host name changes for the referenced PDB, each proxy PDB must reestablish communication with its referenced PDB.

The listener host name of a referenced PDB is a database property. When it is set, you can view the current setting by querying the `DATABASE_PROPERTIES` data dictionary view.

The current user must have the `ALTER DATABASE` system privilege, and the privilege must be either commonly granted or locally granted in the PDB.

1. In SQL*Plus, ensure that the current container is the referenced PDB.

   See "Connecting to a PDB".

2. Run an `ALTER PLUGGABLE DATABASE CONTAINERS HOST` statement and specify the new host name, or include the `RESET` keyword to return the host name to its default setting, which is the host name of the referenced PDB.
3. Drop and re-create the proxy PDBs that reference the referenced PDB to reestablish communication for each proxy PDB and its referenced PDB.

Example 15-8 Altering the Listener Host Name of a Referenced PDB

This example changes the host name for the referenced PDB to `myhost.example.com`.

```
ALTER PLUGGABLE DATABASE CONTAINERS HOST='myhost.example.com';
```

Example 15-9 Resetting the Listener Host Name to the Default Value

This example resets the host name for the referenced PDB to its default value. The default value is the host name of the referenced PDB.

```
ALTER PLUGGABLE DATABASE CONTAINERS HOST RESET;
```

See Also:

- "Creating a PDB as a Proxy PDB"
- "HOST Clause"

Altering the Listener Port Number of a Referenced PDB

When the port number of the listener for a referenced PDB changes, you must run an `ALTER PLUGGABLE DATABASE CONTAINERS PORT` statement to reset the port number of the referenced PDB so that its proxy PDBs continue to function properly.

A proxy PDB uses a database link to establish communication with its referenced PDB during PDB creation. After communication is established, the proxy PDB communicates directly with the referenced PDB without using a database link, and the database link can be dropped. When the listener port number changes for the referenced PDB, each proxy PDB must re-establish communication with its referenced PDB.

The listener port number of a referenced PDB is a database property. When it is set, you can view the current setting by querying the `DATABASE_PROPERTIES` data dictionary view.

The current user must have the `ALTER DATABASE` system privilege, and the privilege must be either commonly granted or locally granted in the PDB.

1. In SQL*Plus, ensure that the current container is the referenced PDB.
2. Run an `ALTER PLUGGABLE DATABASE CONTAINERS PORT` statement and specify the new port number, or include the `RESET` keyword to return the port number to its default setting, which is 1521.
3. Drop and re-create the proxy PDBs that reference the referenced PDB to re-establish communication for each proxy PDB and its referenced PDB.

Example 15-10 Altering the Listener Port Number of a Referenced PDB

This example changes the port number for the referenced PDB to `1543`.

```
ALTER PLUGGABLE DATABASE CONTAINERS PORT=1543;
```

Example 15-11 Resetting the Listener Port Number to the Default Value

This example resets the port number for the referenced PDB to its default value. The default value for the port number is 1521.

```
ALTER PLUGGABLE DATABASE CONTAINERS PORT RESET;
```

Related Topics

- Connecting to a PDB
  You can use several techniques to connect to a PDB with the SQL*Plus `CONNECT` command.
- Creating a PDB as a Proxy PDB
  You can create a PDB as a proxy PDB by referencing it in a remote CDB.
- PORT Clause
  The `PORT` clause of the `CREATE PLUGGABLE DATABASE` statement specifies the port number of the listener for the PDB being created.

Modifying a PDB at the System Level

You can use the ALTER SYSTEM statement to modify a PDB.

About System-Level Modifications of a PDB

The ALTER SYSTEM statement can dynamically alter a PDB. You can issue an ALTER SYSTEM statement when you want to change the way a PDB operates.

When the current container is a PDB, you can run the following ALTER SYSTEM statements:

- ALTER SYSTEM FLUSH { SHARED_POOL | BUFFER_CACHE | FLASH_CACHE }
- ALTER SYSTEM { ENABLE | DISABLE } RESTRICTED SESSION
- ALTER SYSTEM SET USE_STORED_OUTLINES
- ALTER SYSTEM { SUSPEND | RESUME }
- ALTER SYSTEM CHECKPOINT
- ALTER SYSTEM CHECK DATAFILES
- ALTER SYSTEM REGISTER
- ALTER SYSTEM { KILL | DISCONNECT } SESSION
- ALTER SYSTEM SET *initialization_parameter* (for a subset of initialization parameters)

All other ALTER SYSTEM statements affect the entire CDB and must be run by a common user in the root.

The ALTER SYSTEM SET *initialization_parameter* statement can modify only some initialization parameters for PDBs. All initialization parameters can be set for the root. For any initialization parameter that is not set explicitly for a PDB, the PDB inherits the parameter value from the root.

You can modify an initialization parameter for a PDB when the ISPDB_MODIFIABLE column is TRUE for the parameter in the V$SYSTEM_PARAMETER view. The following query lists all initialization parameters that are modifiable for a PDB:

```
SELECT NAME
FROM   V$SYSTEM_PARAMETER
WHERE  ISPDB_MODIFIABLE='TRUE'
ORDER BY NAME;
```

When the current container is a PDB, run the ALTER SYSTEM SET *initialization_parameter* statement to modify the PDB. The statement does not affect the root or other PDBs. The following table describes the behavior of the SCOPE clause when you use a server parameter file (SPFILE) and run the ALTER SYSTEM SET statement on a PDB.

SCOPE Setting	Behavior
MEMORY	The initialization parameter setting is changed in memory and takes effect immediately in the PDB. The new setting affects only the PDB. The setting reverts to the value set in the root in the any of the following cases: • An ALTER SYSTEM SET statement sets the value of the parameter in the root with SCOPE equal to BOTH or MEMORY, and the PDB is closed and re-opened. The parameter value in the PDB is not changed if SCOPE is equal to SPFILE, and the PDB is closed and re-opened. • The PDB is closed and re-opened. • The CDB is shut down and re-opened.
SPFILE	The initialization parameter setting is changed for the PDB and stored persistently. The new setting takes effect in any of the following cases: • The PDB is closed and re-opened. • The CDB is shut down and re-opened. In these cases, the new setting affects only the PDB.
BOTH	The initialization parameter setting is changed in memory, and it is changed for the PDB and stored persistently. The new setting takes effect immediately in the PDB and persists after the PDB is closed and re-opened or the CDB is shut down and re-opened. The new setting affects only the PDB.

When a PDB is unplugged from a CDB, the values of the initialization parameters that were specified for the PDB with SCOPE=BOTH or SCOPE=SPFILE are added to the PDB's XML metadata file. These values are restored for the PDB when it is plugged in to a CDB.

Note:

A text initialization parameter file (PFILE) cannot contain PDB-specific parameter values.

See Also:

- "Unplugging a PDB from a CDB"
- "About the Current Container"
- "Modifying a CDB with ALTER SYSTEM"
- *Oracle Database SQL Language Reference*

Modifying a PDB with ALTER SYSTEM

To modify a PDB at the system level, use the ALTER SYSTEM statement (just as for a non-CDB).

Prerequisites

The current user must be granted the following privileges, which must be either commonly granted or locally granted in the PDB:

- `CREATE SESSION`
- `ALTER SYSTEM`

To use ALTER SYSTEM to modify a PDB:

1. In SQL*Plus, ensure that the current container is a PDB.

   See "Connecting to a PDB".
2. Run the `ALTER SYSTEM` statement.

Example 15-12 Enable Restricted Sessions in a PDB

To restrict sessions in a PDB, issue the following statement:

```
ALTER SYSTEM ENABLE RESTRICTED SESSION;
```

Example 15-13 Changing the Statistics Gathering Level for the PDB

This `ALTER SYSTEM` statement sets the `STATISTICS_LEVEL` initialization parameter to `ALL` for the current PDB:

```
ALTER SYSTEM SET STATISTICS_LEVEL = ALL SCOPE = MEMORY;
```

See Also:

- "Modifying a CDB with ALTER SYSTEM"
- *Oracle Database SQL Language Reference*

Modifying a PDB at the Database Level

You can modify a PDB using the `ALTER PLUGGABLE DATABASE` statement.

About Database-Level Modifications of a PDB

The `ALTER PLUGGABLE DATABASE` for a PDB is analogous to the `ALTER DATABASE` for a non-CDB.

> **Note:**
>
> An `ALTER DATABASE` statement issued when the current container is a PDB that includes clauses that are supported for an `ALTER PLUGGABLE DATABASE` statement have the same effect as the corresponding `ALTER PLUGGABLE DATABASE` statement. However, these statements cannot include clauses that are specific to PDBs, such as the *pdb_storage_clause*, the *pdb_change_state_clause*, the *logging_clause*, and the *pdb_recovery_clause*.

Storage Clauses

Use `ALTER PLUGGABLE DATABASE` to configure storage at the PDB level.

The following clauses of `ALTER PLUGGABLE DATABASE` modify PDB storage:

- *database_file_clauses*

  These clauses work the same as they would in an `ALTER DATABASE` statement, but the statement applies to the current PDB.

- `DEFAULT TABLESPACE` clause

  For users created while the current container is a PDB, this clause specifies the default tablespace for the user if the default tablespace is not specified in the `CREATE USER` statement.

- `DEFAULT TEMPORARY TABLESPACE` clause

  For users created while the current container is a PDB, this clause specifies the default temporary tablespace for the user if the default temporary tablespace is not specified in the `CREATE USER` statement.

- `SET DEFAULT { BIGFILE | SMALLFILE } TABLESPACE` clause

  This clause changes the default type of subsequently created tablespaces in the PDB to either bigfile or smallfile. This clause works the same as it would in an `ALTER DATABASE` statement, but it applies to the current PDB.

- *pdb_storage_clause*

  This clause sets a limit on the amount of storage used by all tablespaces that belong to a PDB. This limit applies to the total size of all data files and temp files comprising tablespaces that belong to the PDB.

  This clause can also set a limit on the amount of storage that can be used by unified audit OS spillover (.bin format) files in the PDB. If the limit is reached, then no additional storage is available for these files.

  This clause can also set a limit on the amount of storage in a shared temporary tablespace that can be used by sessions connected to the PDB. If the limit is reached, then no additional storage in the shared temporary tablespace is available to sessions connected to the PDB.

Logging and Recovery Clauses

Use `ALTER PLUGGABLE DATABASE` to set logging and recovery and recovery modes at the PDB level.

logging_clause

> **Note:**
>
> This clause is available starting with Oracle Database 12c Release 1 (12.1.0.2).

This clause specifies the logging attribute of the PDB. The logging attribute controls whether certain DML operations are logged in the redo log file (`LOGGING`) or not (`NOLOGGING`).

You can use this clause to specify one of the following attributes:

- `LOGGING` indicates that any future tablespaces created within the PDB will be created with the `LOGGING` attribute by default. You can override this default logging attribute by specifying `NOLOGGING` at the schema object level, in a `CREATE TABLE` statement for example.
- `NOLOGGING` indicates that any future tablespaces created within the PDB will be created with the `NOLOGGING` attribute by default. You can override this default logging attribute by specifying `LOGGING` at the schema object level, in a `CREATE TABLE` statement for example.

The specified attribute is used to establish the logging attribute of tablespaces created within the PDB if the *logging_clause* is not specified in the `CREATE TABLESPACE` statement.

The `DBA_PDBS` view shows the current logging attribute for a PDB.

> **Note:**
>
> The PDB must be open in restricted mode to use this clause.

pdb_force_logging_clause

> **Note:**
>
> This clause is available starting with Oracle Database 12c Release 1 (12.1.0.2).

This clause places a PDB into force logging or force nologging mode or takes a PDB out of force logging or force nologging mode.

You can use this clause to specify one of the following attributes:

- `ENABLE FORCE LOGGING` places the PDB in force logging mode, which causes all changes in the PDB, except changes in temporary tablespaces and temporary segments, to be logged. Force logging mode cannot be overridden at the schema object level.

  PDB-level force logging mode takes precedence over and is independent of any `NOLOGGING` or `FORCE LOGGING` settings you specify for individual tablespaces in the PDB and any `NOLOGGING` settings you specify for individual database objects in the PDB.

  `ENABLE FORCE LOGGING` cannot be specified if a PDB is in force nologging mode. `DISABLE FORCE NOLOGGING` must be specified first.

- `DISABLE FORCE LOGGING` takes a PDB which is currently in force logging mode out of that mode. If the PDB is not in force logging mode currently, then specifying `DISABLE FORCE LOGGING` results in an error.

- `ENABLE FORCE NOLOGGING` places the PDB in force nologging mode, which causes no changes in the PDB to be logged. Force nologging mode cannot be overridden at the schema object level.

  CDB-wide force logging mode supersedes PDB-level force nologging mode. PDB-level force nologging mode takes precedence over and is independent of any `LOGGING` or `FORCE LOGGING` settings you specify for individual tablespaces in the PDB and any `LOGGING` settings you specify for individual database objects in the PDB.

  `ENABLE FORCE NOLOGGING` cannot be specified if a PDB is in force logging mode. `DISABLE FORCE LOGGING` must be specified first.

- `DISABLE FORCE NOLOGGING` takes a PDB that is currently in force nologging mode out of that mode. If the PDB is not in force nologging mode currently, then specifying `DISABLE FORCE NOLOGGING` results in an error.

The `DBA_PDBS` view shows whether a PDB is in force logging or force nologging mode.

> **Note:**
>
> The PDB must be open in restricted mode to use this clause.

pdb_recovery_clause

> **Note:**
>
> This clause is available starting with Oracle Database 12c Release 1 (12.1.0.2).

`ALTER PLUGGABLE DATABASE DISABLE RECOVERY` takes the data files that belong to the PDB offline and disables recovery of the PDB. The PDB data files are not part of any recovery session until it is enabled again. Any new data files created while recovery is disabled are created as unnamed files for the PDB.

`ALTER PLUGGABLE DATABASE ENABLE RECOVERY` brings the data files that belong to the PDB online and marks the PDB for active recovery. Recovery sessions include these files.

Check the recovery status of a PDB by querying the `RECOVERY_STATUS` column in the `V$PDBS` view.

> **See Also:**
>
> - *Oracle Data Guard Concepts and Administration* for more information about the *pdb_recovery_clause*.
> - *Oracle Database Administrator's Guide* for information about controlling the writing of redo records
> - *Oracle Database SQL Language Reference* for more information about the logging attribute

Miscellaneous Clauses

You can use `ALTER PLUGGABLE DATABASE` to modify the open mode, global name, time zone, and default edition.

When the current container is a PDB, an `ALTER PLUGGABLE DATABASE` statement with any of the following clauses modifies the PDB:

- *pdb_change_state_clause*

  This clause changes the open mode of the current PDB.

  If you specify the optional `RESTRICTED` keyword, then the PDB is accessible only to users with the `RESTRICTED SESSION` privilege in the PDB.

  Specifying `FORCE` in this clause changes semantics of the `ALTER PLUGGABLE DATABASE` statement so that, in addition to opening a PDB that is currently closed, it can be used to change the open mode of a PDB that is already open.

- `RENAME GLOBAL_NAME` clause

  This clause changes the unique global database name for the PDB. The new global database name must be different from that of any container in the CDB. When you change the global database name of a PDB, the PDB name is changed to the name before the first period in the global database name.

  You must change the `PDB` property of database services used to connect to the PDB when you change the global database name.

- *set_time_zone_clause*

  This clause works the same as it would in an `ALTER DATABASE` statement, but it applies to the current PDB.

- `DEFAULT EDITION` clause

  This clause works the same as it would in an `ALTER DATABASE` statement, but it applies to the current PDB. Each PDB can use edition-based redefinition, and editions in one PDB do not affect editions in other PDBs. In a multitenant

environment in which each PDB has its own application, you can use edition-based redefinition independently for each distinct application.

> **See Also:**
>
> - "Managing Services for PDBs"
> - "Modifying the Open Mode of PDBs with ALTER PLUGGABLE DATABASE"

Modifying a PDB with the ALTER PLUGGABLE DATABASE Statement

To modify the attributes of a single PDB, use the `ALTER PLUGGABLE DATABASE` statement.

When the current container is a PDB, an `ALTER PLUGGABLE DATABASE` statement modifies the PDB. The modifications overwrite the defaults set for the root in the PDB. The modifications do not affect the CDB root or other PDBs.

Prerequisites

The following prerequisites must be met:

- To change the open mode of the PDB from mounted to opened or from opened to mounted, the current user must have `SYSDBA`, `SYSOPER`, `SYSBACKUP`, or `SYSDG` administrative privilege. The privilege must be either commonly granted or locally granted in the PDB. The user must exercise the privilege using `AS` *`sys_privilege_name`* at connect time.
- For all other operations performed using the `ALTER PLUGGABLE DATABASE` statement, the current user must have the `ALTER DATABASE` system privilege, and the privilege must be either commonly granted or locally granted in the PDB.
- To close a PDB, the PDB must be open.

> **Note:**
>
> This section does not cover changing the global database name of a PDB using the `ALTER PLUGGABLE DATABASE` statement.

To modify a PDB:

1. In SQL*Plus, ensure that the current container is a PDB.
2. Run an `ALTER PLUGGABLE DATABASE` statement.

Example 15-14 Changing the Open Mode of a PDB

This `ALTER PLUGGABLE DATABASE` statement changes the open mode of the current PDB to mounted.

```
ALTER PLUGGABLE DATABASE CLOSE IMMEDIATE;
```

The following statement changes the open mode of the current PDB to open read-only.

```
ALTER PLUGGABLE DATABASE OPEN READ ONLY;
```

A PDB must be in mounted mode to change its open mode to read-only or read/write unless you specify the `FORCE` keyword.

The following statement changes the open mode of the current PDB from mounted or open read-only to open read/write.

```
ALTER PLUGGABLE DATABASE OPEN FORCE;
```

The following statement changes the open mode of the current PDB from mounted to migrate.

```
ALTER PLUGGABLE DATABASE OPEN UPGRADE;
```

Example 15-15 Bringing a Data File Online for a PDB

This `ALTER PLUGGABLE DATABASE` statement uses a *database_file_clause* to bring the `/u03/oracle/pdb1_01.dbf` data file online.

```
ALTER PLUGGABLE DATABASE DATAFILE '/u03/oracle/pdb1_01.dbf' ONLINE;
```

Example 15-16 Changing the Default Tablespaces for a PDB

This `ALTER PLUGGABLE DATABASE` statement uses a `DEFAULT TABLESPACE` clause to set the default permanent tablespace to `pdb1_tbs` for the PDB.

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

This `ALTER PLUGGABLE DATABASE` statement uses a `DEFAULT TEMPORARY TABLESPACE` clause to set the default temporary tablespace to `pdb1_temp` for the PDB.

```
ALTER PLUGGABLE DATABASE DEFAULT TEMPORARY TABLESPACE pdb1_temp;
```

The tablespace or tablespace group specified in the `ALTER PLUGGABLE DATABASE` statement must exist in the PDB. Users whose current container is a PDB that are not explicitly assigned a default tablespace or default temporary tablespace use the default tablespace or default temporary tablespace for the PDB.

Example 15-17 Changing the Default Tablespace Type for a PDB

This `ALTER DATABASE` statement uses a `SET DEFAULT TABLESPACE` clause to change the default tablespace type to bigfile for the PDB.

```
ALTER PLUGGABLE DATABASE SET DEFAULT BIGFILE TABLESPACE;
```

Example 15-18 Setting Storage Limits for a PDB

This statement sets the storage limit for all tablespaces that belong to a PDB to two gigabytes.

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

This statement specifies that there is no storage limit for the tablespaces that belong to the PDB.

```
ALTER PLUGGABLE DATABASE STORAGE(MAXSIZE UNLIMITED);
```

This statement specifies that there is no storage limit for the tablespaces that belong to the PDB and that there is no storage limit for the shared temporary tablespace that can be used by sessions connected to the PDB.

```
ALTER PLUGGABLE DATABASE STORAGE UNLIMITED;
```

Example 15-19 Setting the Logging Attribute of a PDB

With the PDB open in restricted mode, this statement specifies the `NOLOGGING` attribute for the PDB:

```
ALTER PLUGGABLE DATABASE NOLOGGING;
```

Example 15-20 Setting the Force Logging Mode of a PDB

This statement enables force logging mode for the PDB:

```
ALTER PLUGGABLE DATABASE ENABLE FORCE LOGGING;
```

Example 15-21 Setting the Default Edition for a PDB

This example sets the default edition for the current PDB to `PDB1E3`.

```
ALTER PLUGGABLE DATABASE DEFAULT EDITION = PDB1E3;
```

See Also:

- "About Database-Level Modifications of a PDB" for information about the clauses that modify the attributes of a single PDB
- "Changing the Global Database Name of a PDB"
- *Oracle Database SQL Language Reference* for more information about the `ALTER PLUGGABLE DATABASE` statement
- *Oracle Database Development Guide* for a complete discussion of edition-based redefinition

Changing the Global Database Name of a PDB

You can change the global database name of a PDB with the `ALTER PLUGGABLE DATABASE RENAME GLOBAL_NAME TO` statement.

When you change the global database name of a PDB, the new global database name must be different from that of any container in the CDB.

Prerequisites

The following prerequisites must be met:

- The current user must have the `ALTER DATABASE` system privilege, and the privilege must be either commonly granted or locally granted in the PDB.
- For an Oracle Real Application Clusters (Oracle RAC) database, the PDB must be open on the current instance only. The PDB must be closed on all other instances.
- The PDB being modified must be opened on the current instance in read/write mode with `RESTRICTED` specified so that it is accessible only to users with `RESTRICTED SESSION` privilege in the PDB.

To change the global database name of a PDB:

1. In SQL*Plus, ensure that the current container is a PDB.
2. Run an `ALTER PLUGGABLE DATABASE RENAME GLOBAL_NAME TO` statement.

   The following example changes the global database name of the PDB to `salespdb.example.com`:

   ```
   ALTER PLUGGABLE DATABASE RENAME GLOBAL_NAME TO salespdb.example.com;
   ```

3. Close the PDB.
4. Open the PDB in read/write mode.

When you change the global database name of a PDB, the PDB name is changed to the first part of the new global name, which is the part before the first period. Also, Oracle Database changes the name of the default database service for the PDB automatically. Oracle Database also changes the `PDB` property of all database services in the PDB to the new global name of the PDB. You must close the PDB and open it in read/write mode for Oracle Database to complete the integration of the new PDB service name into the CDB.

Oracle Net Services must be configured properly for clients to access database services. You might need to alter your Oracle Net Services configuration because of the PDB name change.

See Also:

- "Connecting to a PDB"
- "Managing Services for PDBs" for information about PDBs and database services

Managing Refreshable Clone PDBs

A **refreshable clone PDB** is a read-only clone that can periodically synchronize with its source PDB.

Refreshing a PDB

You can refresh a PDB that was created as a refreshable clone.

When you refresh a PDB manually, changes made to the source PDB since the last refresh are propagated to the PDB being refreshed. You can manually refresh a PDB that is configured for automatic refresh.

Prerequisites

To refresh a PDB, the PDB must have been created as a clone with the `REFRESH MODE MANUAL` or `REFRESH MODE EVERY` *`minutes`* clause included.

1. In SQL*Plus, ensure that the current container is the PDB you want to refresh.
2. If the PDB is not closed, then close the PDB. For example, issue the following SQL statement:

   ```
   ALTER PLUGGABLE DATABASE CLOSE IMMEDIATE;
   ```

3. Issue the following SQL statement:

   ```
   ALTER PLUGGABLE DATABASE REFRESH;
   ```

Related Topics

- About Refreshable Clone PDBs
  The `CREATE PLUGGABLE DATABASE ... REFRESH MODE` statement clones a source PDB and configures the clone to be refreshable. Refreshing the clone PDB updates it with redo accumulated since the last redo log apply.

Switching Over a Refreshable Clone PDB

You can switch the roles of a source PDB and its refreshable clone PDB.

The following statement performs a switchover:

```
ALTER PLUGGABLE DATABASE refresh_mode FROM clonepdb@dblink SWITCHOVER;
```

You must not specify `REFRESH MODE NONE` for *`refresh_mode`*. The database link specified in the `FROM` clause must point to the root of the CDB in which the clone PDB resides.

After the switchover completes, the source PDB becomes the refreshable clone PDB, which can only be opened in `READ ONLY` mode.

Prerequisites

You must meet the following prerequisites:

- You must be connected to the source PDB when you issue `ALTER PLUGGABLE DATABASE ... SWITCHOVER`.

- If the source PDB and clone PDB are in separate CDBs, then the user specified in the database link must have the same name and password in the source PDB and clone PDB.

To switch the roles of the source and clone PDBs:

1. In SQL*Plus or SQL Developer, log in to the source PDB.
2. Execute the `ALTER PLUGGABLE DATABASE` *`refresh_mode`* `FROM` *`clonepdb@dblink`* `SWITCHOVER` statement.

   After the statement completes, the currently connected PDB is now the refreshable clone PDB.
3. Optionally, refresh the clone PDB:

```
ALTER PLUGGABLE DATABASE REFRESH;
```

Example 15-22 Switching Over a Refreshable Clone PDB

This example assumes that your data center contains CDBs named `cdb1` and `cdb2`. The PDB named `cdb1_pdb1` resides in `cdb1`. You want to create a refreshable clone of this PDB in `cdb2` and name it `cdb1_pdb1_ref`. Your goal is to switch over `cdb1_pdb1_ref` so that it becomes the source PDB and `cdb1_pdb1` becomes the clone PDB.

1. In SQL*Plus, connect to `cdb1` as a user with administrator privileges, and then ensure sure that `cdb1_pdb1` is open in read/write mode (sample output included):

```
CONNECT SYS@cdb1 AS SYSDBA
Enter password: *******

ALTER PLUGGABLE DATABASE ALL CLOSE;
ALTER PLUGGABLE DATABASE cdb1_pdb1 OPEN READ WRITE;
SHOW PDBS;

    CON_ID CON_NAME                       OPEN MODE  RESTRICTED
---------- ------------------------------ ---------- ----------
         2 PDB$SEED                       READ ONLY  NO
         3 CDB1_PDB1                      READ WRITE NO
```

2. Create a common user named `c##u1` (replace *pwd* with a user-specified password):

```
DROP USER c##u1 CASCADE;
CREATE USER c##u1 IDENTIFIED BY pwd;
GRANT CREATE SESSION, RESOURCE, CREATE ANY TABLE, UNLIMITED
TABLESPACE TO c##u1 CONTAINER=ALL;
GRANT CREATE PLUGGABLE DATABASE TO c##u1 CONTAINER=ALL;
GRANT SYSOPER TO c##u1 CONTAINER=ALL;
```

3. Set the container to `cdb1_pdb1`, and then create a table `t1` to use for testing (sample output included):

```
ALTER SESSION SET CONTAINER = cdb1_pdb1;
CREATE TABLE t1(n1 NUMBER);
INSERT INTO t1 VALUES(1);
```

```
COMMIT;
SELECT * FROM t1;

        N1
----------
         1
```

4. Connect to `cdb2` as a user with administrator privileges, and then create the common user named `c##u1` (replace *pwd* with a user-specified password):

```
CONNECT SYS@cdb2 AS SYSDBA
Enter password: *******

DROP USER c##u1 CASCADE;
CREATE USER c##u1 IDENTIFIED BY pwd;
GRANT CREATE SESSION, RESOURCE, CREATE ANY TABLE, UNLIMITED
TABLESPACE TO c##u1 CONTAINER=ALL;
GRANT CREATE PLUGGABLE DATABASE TO c##u1 CONTAINER=ALL;
GRANT SYSOPER TO c##u1 CONTAINER=ALL;
```

   Now `cdb1` and `cdb2` both have a common user with the same name (`c##u1`) and password.

5. Create a database link to `cdb1`.

   The following command specifies user `c##u1`, password *pwd*, and service name `cdb1`:

```
CREATE DATABASE LINK cdb1_datalink CONNECT TO c##u1 IDENTIFIED BY
pwd USING 'cdb1';
```

6. Create the manually refreshable PDB named `cdb1_pdb1_ref`.

   The following statement specifies the database link `cdb1_datalink` and the file destination `/dsk1/df`:

```
CREATE PLUGGABLE DATABASE cdb1_pdb1_ref FROM
cdb1_pdb1@cdb1_datalink
  CREATE_FILE_DEST='/dsk1/df'
  REFRESH MODE MANUAL;
```

7. Refresh `cdb1_pdb1_ref`:

```
ALTER SESSION SET CONTAINER = cdb1_pdb1_ref;
ALTER PLUGGABLE DATABASE REFRESH;
```

8. Query `t1` to check that the refreshable clone PDB contains the correct contents (sample output included):

```
ALTER PLUGGABLE DATABASE OPEN READ ONLY;
SELECT * FROM t1;

        N1
```

```
----------
         1
```

9. Connect to `cdb1` as a user with administrator privileges, and then create a database link to `cdb2`:

```
CONNECT SYS@cdb1 AS SYSDBA
Enter password: *******

CREATE DATABASE LINK cdb2_datalink CONNECT TO c##u1 IDENTIFIED BY
pwd USING 'cdb2';
```

   The preceding statement specifies user `c##u1`, password *pwd*, and service name `cdb2`.

10. Set the container to `cdb1_pdb1`, and then switch over so that `cdb1_pdb1_ref` is the primary PDB and the current PDB is the clone:

```
ALTER SESSION SET CONTAINER = cdb1_pdb1;
ALTER PLUGGABLE DATABASE
  REFRESH MODE MANUAL
  FROM cdb1_pdb1_ref@cdb2_datalink
  SWITCHOVER;
```

11. Query `t1` to check that the current PDB, which is now the refreshable clone PDB, contains the correct contents (sample output included):

```
ALTER PLUGGABLE DATABASE OPEN READ ONLY;
SELECT * FROM t1;

        N1
----------
         1
```

12. Connect to `cdb2` as a user with administrator privileges, set the container to the new source PDB `cdb1_pdb1_ref`, and then insert a new row into table `t1` (sample output included):

```
CONNECT SYS@cdb2 AS SYSDBA
Enter password: *******

ALTER SESSION SET CONTAINER = cdb1_pdb1_ref;
SELECT * FROM t1;

        N1
----------
         1

INSERT INTO t1 VALUES(2);
COMMIT;
SELECT * FROM t1;

        N1
----------
```

```
         1
         2
```

13. Connect to `cdb1` as a user with administrator privileges, set the container to `cdb1_pdb1` (which is the new clone), refresh it, and then query `t1`:

```
CONNECT SYS@cdb1 AS SYSDBA
Enter password: *******

ALTER SESSION SET CONTAINER = cdb1_pdb1;
ALTER PLUGGABLE DATABASE CLOSE IMMEDIATE;
ALTER PLUGGABLE DATABASE REFRESH;
ALTER PLUGGABLE DATABASE OPEN READ ONLY;
SELECT * FROM t1;

        N1
----------
         1
         2
```

The preceding output shows that the clone `cdb1_pdb1` was refreshed from the source `cdb1_pdb1_ref`.

Modifying the Open Mode of PDBs

You can modify the open mode of a PDB by using the `ALTER PLUGGABLE DATABASE` SQL statement or the SQL*Plus `STARTUP` command.

About the Open Mode of a PDB

When a PDB is mounted, you can open it in read/write, read-only, or `MIGRATE` mode. You can also mount a PDB without opening it.

Open Modes of a PDB

You can view the open mode of a PDB by querying the `OPEN_MODE` column of the `V$PDBS` view.

The following table describes the possible open modes.

Table 15-2 PDB Mount and Open Modes

Mode	Description	Notes
`OPEN READ WRITE`	A PDB in open read/write mode allows queries and user transactions to proceed and allows users to generate redo logs. This is the default open mode *except* when a PDB belongs to a physical standby database.	If you specify the optional `RESTRICTED` keyword, then the PDB is accessible only to users with the `RESTRICTED SESSION` privilege in the PDB. If you also specify `FORCE`, then all sessions connected to the PDB that do not have the `RESTRICTED SESSION` privilege in the PDB are terminated, and their transactions are rolled back.

Table 15-2 (Cont.) PDB Mount and Open Modes

Mode	Description	Notes
`OPEN READ ONLY`	A PDB in open read-only mode allows queries but does not allow user changes. This is the default open mode when a PDB belongs to a physical standby database.	Database administrators can create, modify, or drop common users and roles in the CDB. The CDB applies these changes to the PDB when its open mode is changed to read/write mode. Before the changes are applied, descriptions of common users and roles in the PDB might be different from the descriptions in the rest of the CDB. If you specify the optional `RESTRICTED` keyword, then the PDB is accessible only to users with the `RESTRICTED SESSION` privilege in the PDB. If you also specify `FORCE`, then all sessions connected to the PDB that do not have the `RESTRICTED SESSION` privilege in the PDB are terminated, and their transactions are rolled back.
`OPEN MIGRATE`	When a PDB is in open migrate mode, you can run database upgrade scripts on the PDB. A PDB is in this mode after you run an `ALTER DATABASE OPEN UPGRADE` statement.	If you specify the optional `RESTRICTED` keyword, then the PDB is accessible only to users with the `RESTRICTED SESSION` privilege in the PDB.
`MOUNTED`	When a PDB is mounted, it does not allow changes to any objects, and it is accessible only to database administrators. It cannot read from or write to data files. Information about the PDB is removed from memory caches. Consistent backups of the PDB are supported.	Database administrators can create, modify, or drop common users and roles in the CDB. The CDB applies these changes to the PDB when its open mode is changed to read/write mode. Before the changes are applied, descriptions of common users and roles in the PDB might be different from the descriptions in the rest of the CDB.

Automatic Compatibility Check

When a PDB is opened, Oracle Database checks the compatibility of the PDB with the CDB. Each compatibility violation is either of the following:

- Warning

  The database records the warning in the alert log, and then opens the PDB normally *without* displaying a warning message.

- Error

  The database displays a message when the PDB is opened stating that the PDB was altered with errors, and records the errors in the alert log. You must correct

the condition that caused each error. When there are errors, the PDB is opened, but access to the PDB is limited to users with `RESTRICTED SESSION` privilege so that the compatibility violations can be addressed. You can view descriptions of violations by querying `PDB_PLUG_IN_VIOLATIONS` view.

> **See Also:**
>
> - "Modifying the Open Mode of PDBs" for information about modifying the open mode of one or more PDBs when the current container is the root
> - "Modifying a PDB with the ALTER PLUGGABLE DATABASE Statement" for information about modifying the open mode of a PDB when the current container is the PDB
> - "Shutting Down a CDB Instance"
> - "Modifying a PDB at the Database Level" for information about modifying other attributes of a PDB

Clauses for Changing the Open State of PDBs

To change the open mode of a PDB when the current container is the CDB root, specify the *pdb_change_state* clause of `ALTER PLUGGABLE DATABASE`.

OPEN and CLOSE Clauses

`READ WRITE` is the default for `ALTER PLUGGABLE DATABASE OPEN` unless a PDB being opened belongs to a CDB used as a physical standby database, in which case `READ ONLY` is the default.

When you specify PDBs to open or close, you can do the following:

- List one or more PDBs.
- Specify `ALL` to modify all PDBs.
- Specify `ALL EXCEPT` to modify all PDBs, except for the PDBs listed.

The following table describes the clauses of the `ALTER PLUGGABLE DATABASE` statement that modify the mode of a PDB.

Table 15-3 ALTER PLUGGABLE DATABASE Clauses That Modify the Mode of a PDB

Clause	Description
`OPEN READ WRITE [RESTRICTED] [FORCE]`	Opens the PDB in read/write mode. When `RESTRICTED` is specified, the PDB is accessible only to users with `RESTRICTED SESSION` privilege in the PDB. All sessions connected to the PDB that do not have `RESTRICTED SESSION` privilege on it are terminated, and their transactions are rolled back. When `FORCE` is specified, the statement opens a PDB that is currently closed and changes the open mode of a PDB that is in open read-only mode.

Table 15-3 (Cont.) ALTER PLUGGABLE DATABASE Clauses That Modify the Mode of a PDB

Clause	Description
`OPEN READ ONLY[RESTRICTED] [FORCE]`	Opens the PDB in read-only mode. When `RESTRICTED` is specified, the PDB is accessible only to users with `RESTRICTED SESSION` privilege in the PDB. All sessions connected to the PDB that do not have `RESTRICTED SESSION` privilege on it are terminated. When `FORCE` is specified, the statement opens a PDB that is currently closed and changes the open mode of a PDB that is in open read/write mode.
`OPEN UPGRADE [RESTRICTED]`	Opens the PDB in migrate mode. When `RESTRICTED` is specified, the PDB is accessible only to users with `RESTRICTED SESSION` privilege in the PDB.
`CLOSE [IMMEDIATE|ABORT]`	Places the PDB in mounted mode. The `CLOSE` statement is the PDB equivalent of the SQL*Plus `SHUTDOWN` command. If you do not specify `IMMEDIATE` or `ABORT`, then the PDB is shut down with the normal mode. When `IMMEDIATE` is specified, this statement is the PDB equivalent of the SQL*Plus `SHUTDOWN IMMEDIATE` command. If the CDB is in `ARCHIVELOG` mode, and if `ABORT` is specified, then the PDB is forcefully closed. The PDB data files are not checkpointed or accessed during this process. If other instances have the PDB open, then an available instance performs instance recovery automatically. During this time, access to the PDB on other instances may observe a brown-out time. If no instance has the PDB open, then the next PDB open may cause automatic media recovery. If automatic media recovery fails (for example, because of inaccessible files), then you must manually recover the PDB before opening it. If the PDB keystore was in an open state, then `ALTER PLUGGABLE DATABASE CLOSE` does not close it. To close the keystore, run the `ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE IDENTIFIED BY "`*`pdb_ks_pwd`*`"` command.

SERVICES Clause

You can use the *services* clause to specify the services that are started when a single PDB is opened.

The clause has the following variations:

- List one or more services in the *services* clause in the following form:

  ```
  SERVICES = ('service_name' ['service_name'] … )
  ```

- Specify `ALL` in the *services* clause to start all PDB's services, as in the following example:

  ```
  SERVICES = ALL
  ```

- Specify `ALL EXCEPT` in the *services* clause to start all PDB's services, except for the services listed, in the following form:

  ```
  SERVICES = ALL EXCEPT('service_name' ['service_name'] … )
  ```

- Specify `NONE` in the *services* clause to start only the PDB's default service and none of the other PDB's services, as in the following example:

  ```
  SERVICES = NONE
  ```

  `NONE` is the default setting for the *services* clause. A PDB's default service is always started, regardless of the setting for the *services* clause.

INSTANCES Clause

In an Oracle RAC CDB, you can use the *instances* clause to specify the instances on which the PDB is modified.

You can close a PDB in some instances and leave it open in others. The *instances* clause has the following variations:

- List one or more instances in the *instances* clause in the following form:

  ```
  INSTANCES = ('instance_name' ['instance_name'] … )
  ```

- Specify `ALL` in the *instances* clause to modify the PDB in all running instances, as in the following example:

  ```
  INSTANCES = ALL
  ```

- Specify `ALL EXCEPT` in the *instances* clause to modify the PDB in all instances, except for the instances listed, in the following form:

  ```
  INSTANCES = ALL EXCEPT('instance_name' ['instance_name'] … )
  ```

The RELOCATE Clause

In an Oracle Real Application Clusters environment, use `RELOCATE` to instruct the database to reopen the PDB on a different Oracle RAC instance.

You can use the following options:

- Specify `NORELOCATE`, the default, to close the PDB in the current instance.
- Specify `RELOCATE TO` and specify an instance name to reopen the PDB in the specified instance.
- Specify `RELOCATE` to reopen the PDB on a different instance that is selected by Oracle Database.

> **Note:**
>
> If both the *services* clause and the *instances* clause are specified in the same `ALTER PLUGGABLE DATABASE` statement, then the specified services are started on the specified instances.

Modifying the Open Mode of PDBs with ALTER PLUGGABLE DATABASE

You can modify the open mode of PDBs with the `ALTER PLUGGABLE DATABASE` statement with a *pdb_change_state* clause.

Prerequisites

To change the open mode of PDBs with the `ALTER PLUGGABLE DATABASE` statement, you must meet the following prerequisites:

- The current user must have one of the following administrative privileges, which must be either commonly granted or locally granted in the PDB:
  - `SYSDBA`, exercised using `AS SYSDBA` at connect time
  - `SYSOPER`, exercised using `AS SYSOPER` at connect time
  - `SYSBACKUP`, exercised using `SYSBACKUP` at connect time
  - `SYSDG`, exercised using `AS SYSDG` at connect time

> **Note:**
>
> You can modify the open mode of a PDB when the current container is the PDB.

- When `RESTRICTED SESSION` is enabled, you must specify `RESTRICTED` when a PDB is opened.
- In an Oracle RAC CDB, if a PDB is open in one or more Oracle RAC instances, then it can be opened in additional instances. However, the PDB must be opened in the same mode as in the instances in which it is already open. A PDB can be closed in some instances and opened on others.

To place PDBs in a target mode with the `ALTER PLUGGABLE DATABASE` statement, you must meet the requirements described in the following table.

Table 15-4 Modifying the Open Mode of PDBs with ALTER PLUGGABLE DATABASE

Target Mode of PDBs	`ALL` Keyword Included	`FORCE` Keyword Included	Required Mode for the Root	Required Mode for Each PDB Being Modified
Read/write	Yes	Yes	Read/write	Mounted, read-only, or read/write
Read/write	Yes	No	Read/write	Mounted or read/write
Read/write	No	Yes	Read/write	Mounted, read-only, or read/write
Read/write	No	No	Read/write	Mounted
Read-only	Yes	Yes	Read-only or read/write	Mounted, read-only, or read/write
Read-only	Yes	No	Read-only or read/write	Mounted or read-only

Table 15-4 (Cont.) Modifying the Open Mode of PDBs with ALTER PLUGGABLE DATABASE

Target Mode of PDBs	ALL Keyword Included	FORCE Keyword Included	Required Mode for the Root	Required Mode for Each PDB Being Modified
Read-only	No	Yes	Read-only or read/write	Mounted, read-only, or read/write
Read-only	No	No	Read-only or read/write	Mounted
Migrate	Yes	Not applicable	Read-only or read/write	Mounted
Migrate	No	Not applicable	Read-only or read/write	Mounted
Mounted	Yes	Not applicable	Read-only or read/write	Mounted, read-only, migrate, or read/write
Mounted	No	Not applicable	Read-only or read/write	Read-only, migrate, or read/write

To modify the open mode:

1. In SQL*Plus, ensure that the current container is the root.

   See "About Container Access in a CDB".

2. Run an `ALTER PLUGGABLE DATABASE` statement with a *pdb_change_state* clause.

Example 15-23 Changing the Open Mode of Listed PDBs

This statement changes the open mode of PDBs `salespdb` and `hrpdb` to open in read/write mode.

```
ALTER PLUGGABLE DATABASE salespdb, hrpdb
   OPEN READ WRITE;
```

This statement changes the open mode of PDB `salespdb` to open in read-only mode. `RESTRICTED` specifies that the PDB is accessible only to users with `RESTRICTED SESSION` privilege in the PDB.

```
ALTER PLUGGABLE DATABASE salespdb
   OPEN READ ONLY RESTRICTED;
```

This statement changes the open mode of PDB `salespdb` to open in migrate mode:

```
ALTER PLUGGABLE DATABASE salespdb
   OPEN UPGRADE;
```

Example 15-24 Changing the Open Mode of All PDBs

Run the following query to display the open mode of each PDB associated with a CDB:

```
SELECT NAME, OPEN_MODE FROM V$PDBS WHERE CON_ID > 2;

NAME                           OPEN_MODE
------------------------------ ----------
```

```
HRPDB                          READ WRITE
SALESPDB                       MOUNTED
DWPDB                          MOUNTED
```

Notice that `hrpdb` is already in read/write mode. To change the open mode of `salespdb` and `dwpdb` to open in read/write mode, use the following statement:

```
ALTER PLUGGABLE DATABASE ALL
   OPEN READ WRITE;
```

The `hrpdb` PDB is not modified because it is already in open read/write mode. The statement does not return an error because two PDBs are in mounted mode and one PDB (`hrpdb`) is in the specified mode (read/write). Similarly, the statement does not return an error if all PDBs are in mounted mode.

However, if any PDB is in read-only mode, then the statement returns an error. To avoid an error and open all PDBs in the CDB in read/write mode, specify the `FORCE` keyword:

```
ALTER PLUGGABLE DATABASE ALL
   OPEN READ WRITE FORCE;
```

With the `FORCE` keyword included, all PDBs are opened in read/write mode, including PDBs in read-only mode.

Example 15-25 Changing the Open Mode of All PDBs Except for Listed Ones

This statement changes the mode of all PDBs except for `salespdb` and `hrpdb` to mounted mode.

```
ALTER PLUGGABLE DATABASE ALL EXCEPT salespdb, hrpdb
   CLOSE IMMEDIATE;
```

> **Note:**
>
> An `ALTER PLUGGABLE DATABASE` statement modifying the open mode of a PDB is instance-specific. Therefore, if this statement is issued when connected to an Oracle RAC instance, then it affects the open mode of the PDB only in that instance.

> **See Also:**
>
> - "Clauses for Changing the Open State of PDBs"
> - "Modifying a PDB at the Database Level" for information about modifying the other attributes of a PDB
> - *Oracle Database Administrator's Guide* for information about database modes and their uses
> - *Oracle Database SQL Language Reference*
> - *Oracle Database Concepts* for more information about shutdown modes

Preserving or Discarding the Open Mode of PDBs When the CDB Restarts

You can preserve the open mode of one or more PDBs when the CDB restarts by using the `ALTER PLUGGABLE DATABASE` SQL statement with a *pdb_save_or_discard_state* clause.

You can do this in the following way:

- Specify `SAVE STATE` to preserve the PDBs' mode when the CDB is restarted.

  For example, if a PDB is in open read/write mode before the CDB is restarted, then the PDB is in open read/write mode after the CDB is restarted; if a PDB is in mounted mode before the CDB is restarted, then the PDB is in mounted mode after the CDB is restarted.
- Specify `DISCARD STATE` to ignore the PDBs' open mode when the CDB is restarted.

  When `DISCARD STATE` is specified for a PDB, the PDB is always mounted after the CDB is restarted.

You can specify which PDBs to modify in the following ways:

- List one or more PDBs.
- Specify `ALL` to modify all PDBs.
- Specify `ALL EXCEPT` to modify all PDBs, except for the PDBs listed.

For an Oracle RAC CDB, you can use the *instances* clause in the *pdb_save_or_discard_state* clause to specify the instances on which a PDB's open mode is preserved in the following ways:

- List one or more instances in the *instances* clause in the following form:

  ```
  INSTANCES = ('instance_name' ['instance_name'] … )
  ```

- Specify `ALL` in the *instances* clause to modify the PDB in all running instances, as in the following example:

  ```
  INSTANCES = ALL
  ```

- Specify `ALL EXCEPT` in the *instances* clause to modify the PDB in all instances, except for the instances listed, in the following form:

  ```
  INSTANCES = ALL EXCEPT('instance_name' ['instance_name'] … )
  ```

For a PDB in an Oracle RAC CDB, `SAVE STATE` and `DISCARD STATE` only affect the mode of the current instance. They do not affect the mode of other instances, even if more than one instance is specified in the *instances* clause.

To issue an `ALTER PLUGGABLE DATABASE` SQL statement with a *pdb_save_or_discard_state* clause, the current user must have the `ALTER DATABASE` privilege in the root.

You can check the saved states for the PDBs in a CDB by querying the `DBA_PDB_SAVED_STATES` view.

To preserve or discard a PDB's open mode when the CDB restarts:

1. In SQL*Plus, ensure that the current container is the root.

   See "About Container Access in a CDB".

2. Run an `ALTER PLUGGABLE DATABASE` statement with a *pdb_save_or_discard_state* clause.

The following examples either preserve or discard the open mode of one or more PDBs when the CDB restarts.

Example 15-26 Preserving the Open Mode of a PDB When the CDB Restarts

This statement preserves the open mode of the `salespdb` when the CDB restarts.

```
ALTER PLUGGABLE DATABASE salespdb SAVE STATE;
```

Example 15-27 Discarding the Open Mode of a PDB When the CDB Restarts

This statement discards the open mode of the `salespdb` when the CDB restarts.

```
ALTER PLUGGABLE DATABASE salespdb DISCARD STATE;
```

Example 15-28 Preserving the Open Mode of All PDBs When the CDB Restarts

This statement preserves the open mode of all PDBs when the CDB restarts.

```
ALTER PLUGGABLE DATABASE ALL SAVE STATE;
```

Example 15-29 Preserving the Open Mode of Listed PDBs When the CDB Restarts

This statement preserves the open mode of the `salespdb` and `hrpdb` when the CDB restarts.

```
ALTER PLUGGABLE DATABASE salespdb, hrpdb SAVE STATE;
```

Example 15-30 Preserving the Open Mode of All PDBs Except for Listed Ones When the CDB Restarts

This statement preserves the open mode of all PDBs except for `salespdb` and `hrpdb`.

```
ALTER PLUGGABLE DATABASE ALL EXCEPT salespdb, hrpdb SAVE STATE;
```

Altering the Open Mode of a PDB Using STARTUP and SHUTDOWN

When the current container is a PDB, you can use the SQL*Plus `STARTUP` command to open the PDB and the SQL*Plus `SHUTDOWN` command to close the PDB.

About Modifying the Open Mode of PDBs with the SQL*Plus STARTUP Command

When the current container is the root, the `STARTUP PLUGGABLE DATABASE` command can open a single PDB.

Use the following options of the `STARTUP PLUGGABLE DATABASE` command to open a PDB:

- `FORCE`

  Closes an open PDB before re-opening it in read/write mode. When this option is specified, no other options are allowed.

- `RESTRICT`

  Enables only users with the `RESTRICTED SESSION` system privilege in the PDB to access the PDB.

  If neither `OPEN READ WRITE` nor `OPEN READ ONLY` is specified, then the PDB is opened in read-only mode when the CDB to which it belongs is a physical standby database. Otherwise, the PDB is opened in read/write mode.

- `OPEN` *open_pdb_options*

  Opens the PDB in either read/write mode or read-only mode. You can specify `OPEN READ WRITE` or `OPEN READ ONLY`. When you specify `OPEN` without any other options, `READ WRITE` is the default.

The following prerequisites must be met:

- The current user must have `SYSDBA`, `SYSOPER`, `SYSBACKUP`, or `SYSDG` administrative privilege, and the privilege must be either commonly granted or locally granted in the PDB. The user must exercise the privilege using `AS SYSDBA`, `AS SYSOPER`, `AS SYSBACKUP`, or `AS SYSDG`, respectively, at connect time.
- When `RESTRICTED SESSION` is enabled, `RESTRICT` must be specified when a PDB is opened.

In addition, to place PDBs in a target mode with the `STARTUP PLUGGABLE DATABASE` command, you must meet the requirements described in the following table.

Table 15-5 Modifying the Open Mode of a PDB with STARTUP PLUGGABLE DATABASE

Target Mode of the PDB	FORCE Option Included	Required Mode for the Root	Required Mode of the PDB Being Modified
Read/write	Yes	Read/write	Mounted, read-only, or read/write
Read/write	No	Read/write	Mounted
Read-only	No	Read-only or read/write	Mounted

> **Note:**
>
> You can also use the `STARTUP` command to modify the open mode of a PDB when the current container is the PDB.

> **See Also:**
>
> - "Starting Up a PDB Using the STARTUP Command"
> - "Modifying the Open Mode of PDBs with the SQL*Plus STARTUP Command"

Starting Up a PDB Using the STARTUP Command

When the current container is a PDB, the SQL*Plus `STARTUP` command opens the PDB.

Use the following options of the `STARTUP` command to open a PDB:

- `FORCE`

  Closes an open PDB before re-opening it in read/write mode. When this option is specified, no other options are allowed.

- `RESTRICT`

  Enables only users with the `RESTRICTED SESSION` system privilege in the PDB to access the PDB.

  If neither `OPEN READ WRITE` nor `OPEN READ ONLY` is specified and `RESTRICT` is specified, then the PDB is opened in read-only mode when the CDB to which it belongs is a physical standby database. Otherwise, the PDB is opened in read/write mode.

- `OPEN` *`open_pdb_options`*

  Opens the PDB in either read/write mode or read-only mode. Specify `OPEN READ WRITE` or `OPEN READ ONLY`. When `RESTRICT` is not specified, `READ WRITE` is always the default.

To issue the `STARTUP` command when the current container is a PDB, the following prerequisites must be met:

- The current user must have SYSDBA, SYSOPER, SYSBACKUP, or SYSDG administrative privilege, and the privilege must be either commonly granted or locally granted in the PDB. The user must exercise the privilege using AS SYSDBA, AS SYSOPER, AS SYSBACKUP, or AS SYSDG, respectively, at connect time.
- Excluding the use of the FORCE option, the PDB must be in mounted mode to open it.
- To place a PDB in mounted mode, the PDB must be in open read-only or open read/write mode.

To modify a PDB with the STARTUP command:

1. In SQL*Plus, ensure that the current container is a PDB.
2. Run the STARTUP command.

Example 15-31 Opening a PDB in Read/Write Mode with the STARTUP Command

```
STARTUP OPEN
```

Example 15-32 Opening a PDB in Read-Only Mode with the STARTUP Command

```
STARTUP OPEN READ ONLY
```

Example 15-33 Opening a PDB in Read-Only Restricted Mode with the STARTUP Command

```
STARTUP RESTRICT OPEN READ ONLY
```

Example 15-34 Opening a PDB in Read/Write Mode with the STARTUP Command and the FORCE Option

This example assumes that the PDB is currently open. The FORCE option closes the PDB and then opens it in the read/write mode.

```
STARTUP FORCE
```

See Also:

- "About the Current Container"
- "Connecting to a PDB".
- *Oracle Database Administrator's Guide* for information about starting up a database
- *SQL*Plus User's Guide and Reference*

Modifying the Open Mode of PDBs with the SQL*Plus STARTUP Command

You can use the `STARTUP PLUGGABLE DATABASE` command to open a single PDB.

To modify a PDB with the STARTUP PLUGGABLE DATABASE command:

1. In SQL*Plus, ensure that the current container is the root.

   See "About Container Access in a CDB".

2. Run the `STARTUP PLUGGABLE DATABASE` command.

> **Note:**
>
> When the current container is the root, the SQL*Plus `SHUTDOWN` command always shuts down the CDB instance. It cannot be used to close individual PDBs.

Example 15-35 Opening a PDB in Read/Write Mode with the STARTUP Command

```
STARTUP PLUGGABLE DATABASE hrpdb OPEN
```

Example 15-36 Opening a PDB in Read/Write Restricted Mode with the STARTUP Command

```
STARTUP PLUGGABLE DATABASE hrpdb RESTRICT
```

Example 15-37 Opening a PDB in Read-Only Restricted Mode with the STARTUP Command

```
STARTUP PLUGGABLE DATABASE hrpdb OPEN READ ONLY RESTRICT
```

Example 15-38 Opening a PDB in Read-Only Mode with the STARTUP Command

```
STARTUP PLUGGABLE DATABASE hrpdb OPEN READ ONLY
```

Example 15-39 Opening a PDB in Read/Write Mode with the STARTUP Command and the FORCE Option

This example assumes that the `hrpdb` PDB is currently open. The `FORCE` option closes the PDB and then opens it in the read/write mode.

```
STARTUP PLUGGABLE DATABASE hrpdb FORCE
```

See Also:

- "About Modifying the Open Mode of PDBs with the SQL*Plus STARTUP Command"
- "Altering the Open Mode of a PDB Using STARTUP and SHUTDOWN" for information about using the `STARTUP` or `SHUTDOWN` command when the current container is a PDB
- *Oracle Database Administrator's Guide*
- *SQL*Plus User's Guide and Reference*

Shutting Down a PDB Using the SHUTDOWN Command

When the current container is a PDB, the SQL*Plus `SHUTDOWN` command closes the PDB.

After the `SHUTDOWN` command is issued on a PDB successfully, it is in mounted mode.

The following `SHUTDOWN` modes are possible:

- When you specify `SHUTDOWN` only, then the PDB is shut down with the normal mode.
- When you specify `SHUTDOWN IMMEDIATE`, the PDB is shut down with the immediate mode.
- When you specify `SHUTDOWN ABORT`, the PDB is forcefully closed.

  For a single-instance CDB, PDB media recovery is required when you specify `SHUTDOWN ABORT`. For an Oracle Real Application Clusters (Oracle RAC) CDB, PDB media recovery is required if the `SHUTDOWN ABORT` command closes the last open instance.

Note that if the PDB keystore was in an open state, then issuing `SHUTDOWN` at the PDB level does not close it. To close the keystore, run the `ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE IDENTIFIED BY "pdb_ks_pwd"` command.

Prerequisites

To issue the `SHUTDOWN` command when the current container is a PDB, the following prerequisites must be met:

- The current user must have `SYSDBA`, `SYSOPER`, `SYSBACKUP`, or `SYSDG` administrative privilege, and the privilege must be either commonly granted or locally granted in the PDB. The user must exercise the privilege using `AS SYSDBA`, `AS SYSOPER`, `AS SYSBACKUP`, or `AS SYSDG`, respectively, at connect time.
- To close a PDB, the PDB must be open.

To modify a PDB with the `SHUTDOWN` command:

1. In SQL*Plus, ensure that the current container is a PDB.
2. Run the `SHUTDOWN` command.

Note:

- When the current container is a PDB, the `SHUTDOWN` command only closes the PDB, not the CDB instance.
- There is no `SHUTDOWN` command for a PDB that is equivalent to `SHUTDOWN TRANSACTIONAL` for a CDB.

Example 15-40 Closing a PDB with the SHUTDOWN IMMEDIATE Command

```
SHUTDOWN IMMEDIATE
```

See Also:

- "About the Current Container"
- "Connecting to a PDB"
- "Modifying the Open Mode of PDBs with ALTER PLUGGABLE DATABASE"
- *Oracle Database Administrator's Guide* for more information about shutdown modes
- *SQL*Plus User's Guide and Reference*

Starting and Stopping PDBs in Oracle RAC

Administering a pluggable database (PDB) involves a small subset of the tasks required to administer a non-CDB.

Administering an Oracle RAC-based multitenant container database (CDB) is similar to administering a non-CDB. The differences are that some administrative tasks apply to the entire CDB, some to the CDB root, and some to specific PDBs. In this subset of tasks, most are the same for a PDB and a non-CDB. There are some differences, however, such as when you modify the open mode of a PDB. Also, a PDB administrator is limited to managing a single PDB and is not affected by other PDBs in the CDB.

You manage PDBs in an Oracle RAC CDB by managing services. This is true regardless of whether the PDBs are policy managed or administrator managed. Assign one dynamic database service to each PDB to coordinate start, stop, and placement of PDBs across instances in a clustered container database.

For example, if you have a CDB called `raccont` with a policy-managed PDB called `spark` in a server pool called `prod`, then assign a service called `plug` to this database using the following command:

```
srvctl add service –db raccont –pdb spark –service plug –serverpool prod
```

The service `plug` is uniformly managed across all nodes in the server pool. If you want to have this service running as a singleton service in the same server pool, then use the `-cardinality singleton` parameter with the preceding command.

To open the PDB `spark`, you must start the service `plug` as follows:

```
srvctl start service -db raccont -service plug
```

To stop the service `plug`:

```
srvctl stop service -db raccont -service plug
```

The PDB `spark` remains open until you close the PDB using the SQL command `ALTER PLUGGABLE DATABASE` *`PDB_NAME`* `CLOSE IMMEDIATE`. You can check the status of the database using the `srvctl status service` command.

Because PDBs are managed using dynamic database services, typical Oracle RAC-based management practices apply. For this reason, if the service `plug` is in the online state when Oracle Clusterware is shut down on a server hosting this service, then the service is restored to its original state after the restart of Oracle Clusterware on this server. Thus, starting PDBs is automated as with any other Oracle RAC database.

> **Note:**
>
> Unlike SQL*Plus, SRVCTL operates on an entire cluster database. Starting a PDB using services therefore applies to multiple instances of the clustered CDB at the same time when the service is defined to run on multiple servers simultaneously and the current status of the cluster allows for this placement.

Related Topics

- *Oracle Real Application Clusters Administration and Deployment Guide*

16

Administering a PDB Snapshot Carousel

You can configure a PDB snapshot carousel for a specified PDB, create snapshots manually or automatically, and set the maximum number of snapshots.

About PDB Snapshot Carousel

A PDB snapshot carousel is a library of PDB snapshots.

A **PDB snapshot** is a point-in-time copy of a PDB. The source PDB can be open read-only or read/write while the snapshot is created. You can create snapshots manually using the `SNAPSHOT` clause of `CREATE PLUGGABLE DATABASE` (or `ALTER PLUGGABLE DATABASE`), or automatically using the `EVERY` *`interval`* clause. If the storage system supports sparse clones, then the preceding command creates a sparse copy. Otherwise, the command creates a full copy.

> **See Also:**
>
> *Oracle Database Licensing Information User Manual* for details on which features are supported for different editions and services

Purpose of PDB Snapshot Carousel

A PDB snapshot carousel is useful for maintaining a library of recent PDB copies for point-in-time recovery and cloning.

Cloning PDBs for Development and Testing

In a typical development use case, you clone a production PDB for testing using a command of the form `CREATE PLUGGABLE DATABASE` *`newpdb`* `FROM` *`srcpdb`*. When the CDB is in `ARCHIVELOG` mode and local undo mode, the source production PDB can be opened in read/write mode and fully functional when you clone it, a technique known as **hot cloning**. The hot clone PDB is transactionally consistent with the source PDB as of the SCN at the completion of the `ALTER PLUGGABLE DATABASE ... OPEN` statement.

The following steps illustrate a typical development scenario:

1. While the production PDB named `pdb1_prod` is open and in use, create a refreshable clone PDB named `pdb1_test_master`.

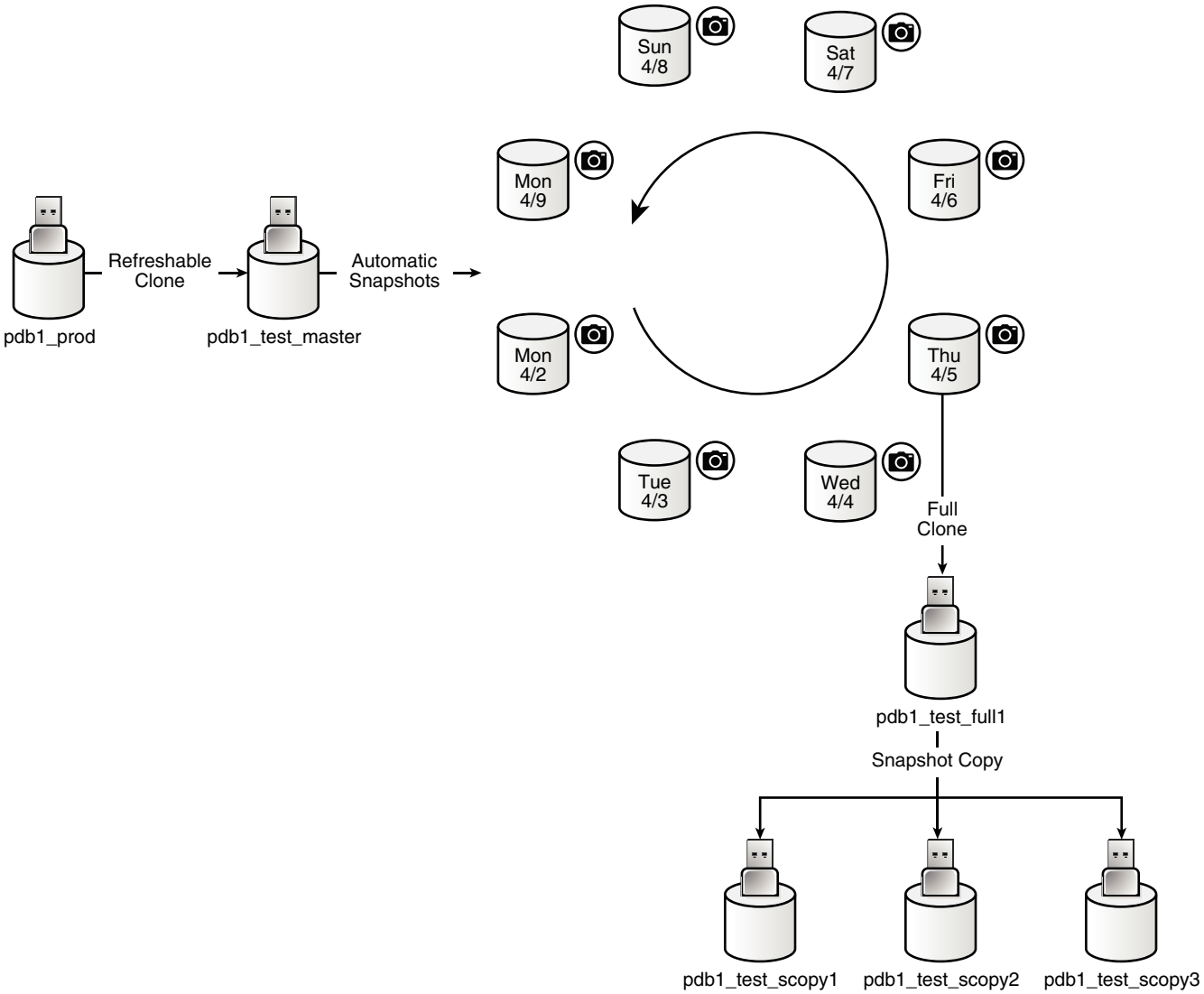
   A refreshable clone PDB can only be opened in read/only mode. To refresh the clone PDB from `pdb1_prod`, you must close it.

2. Run `ALTER PLUGGABLE DATABASE pdb1_test_master SNAPSHOT MODE EVERY 24 HOURS`, which configures the PDB to generate automatic snapshots of `pdb1_test_master` every day.

3. When you need new PDBs for testing, create a full clone PDB by using the `CREATE PLUGGABLE DATABASE … USING SNAPSHOT` command.
4. Create sparse snapshot copy PDBs of the full clone PDB using `CREATE PLUGGABLE DATABASE ... SNAPSHOT COPY`.

The following figure shows the creation of the clone `pdb1_test_full1` from the PDB snapshot taken on April 5. The figure shows three snapshot copy PDBs created from `pdb1_test_full1`.

Figure 16-1 Automatic Snapshots of a Refreshable Clone PDB



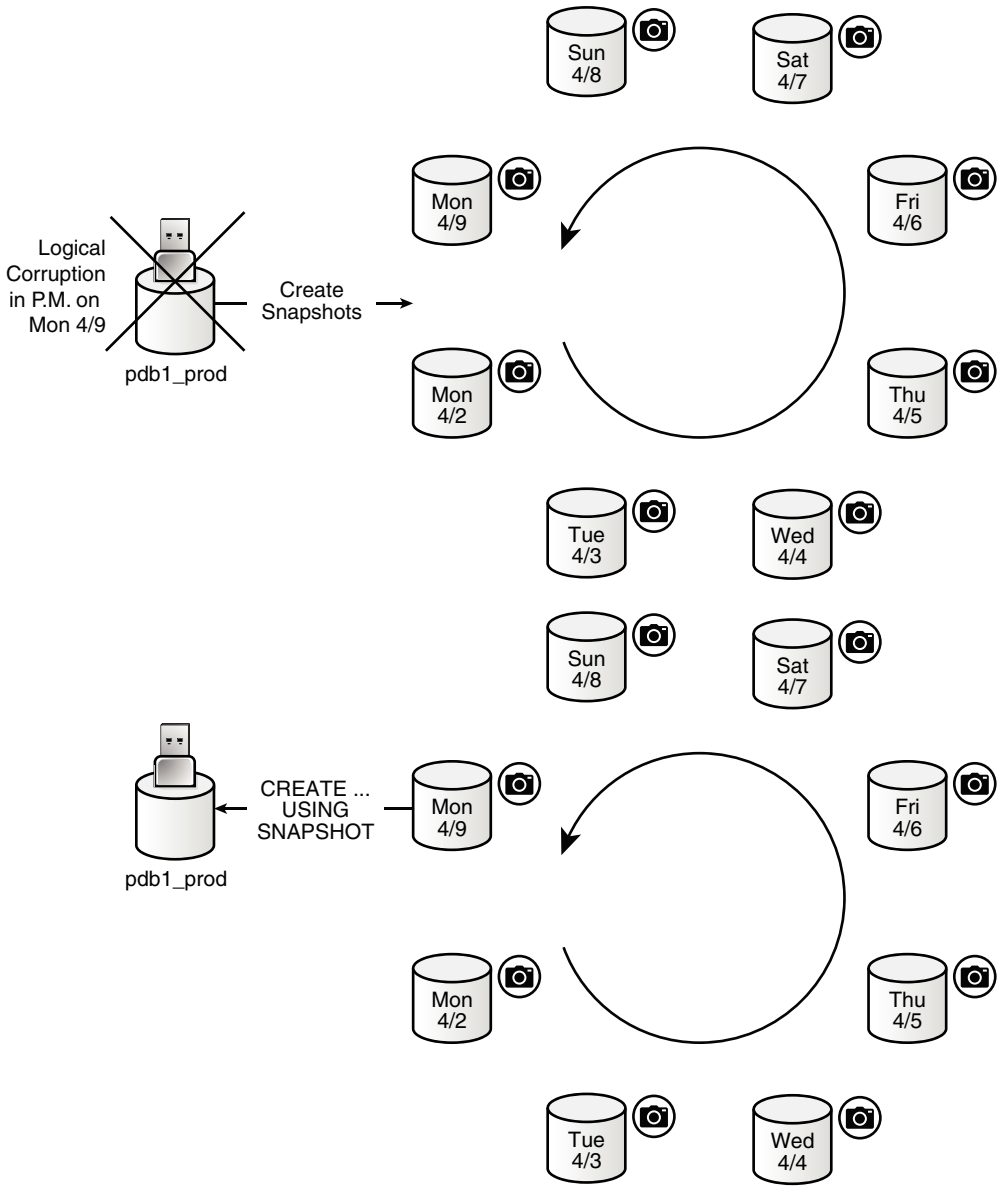
Point-in-Time Restore with PDB Snapshot Carousel

One strategy is to take a snapshot of a PDB every day at the same time. Another strategy is to take a PDB snapshot manually before data loads. In either case, a PDB snapshot carousel enables you to restore a PDB using any available PDB snapshot.

For example, a sales history PDB named `pdb1_prod` generates an automatic snapshot every day at 12:01 a.m. On the daily data load on the afternoon of Monday 4/9, you

accidentally load the wrong data, corrupting the PDB. You can create a new production PDB based on the Monday 4/9 snapshot, drop the corrupted PDB, and then retry the data load.

Figure 16-2 Restore a Production PDB Using a Snapshot



See Also:

- "About Cloning a PDB or Non-CDB"
- *Oracle Database SQL Language Reference* for `CREATE PLUGGABLE DATABASE` syntax and semantics
- *Oracle Database Licensing Information User Manual* for details on which features are supported for different editions and services

How PDB Snapshot Carousel Works

The carousel for a specific PDB is a circular library of copies for this PDB.

The database creates successive copies in the carousel either on demand or automatically. The database overwrites the oldest snapshot when the snapshot limit is reached.

Contents of a PDB Snapshot

The contents of a PDB snapshot depend on whether the underlying file system supports sparse files.

Snapshot Names

The name of a database-managed PDB snapshot is either user-specified or system-generated. For system-generated snapshot names, `SNAP_` is prefixed to a unique identifier, which contains the snapshot SCN. For example, the following query shows three snapshots with system-generated names and the SCNs at which they were taken:

```
SET LINESIZE 200
SET PAGESIZE 50000

COL CON_ID FORMAT 999999
COL CON_NAME FORMAT a15
COL SNAPSHOT_NAME FORMAT a27

SELECT CON_ID, CON_NAME, SNAPSHOT_NAME, SNAPSHOT_SCN FROM
DBA_PDB_SNAPSHOTS;

 CON_ID CON_NAME        SNAPSHOT_NAME               SNAPSHOT_SCN
------- --------------- --------------------------- ------------
      5 HRPDB           SNAP_1389467754_993556301        2925293
      5 HRPDB           SNAP_1389467754_993556306        2925679
      5 HRPDB           SNAP_1389467754_993556309        2925698
```

> **Note:**
>
> See *Oracle Database Licensing Information User Manual* for details on which features are supported for different editions and services.

Full and Sparse Snapshots

The content of snapshots generated by `ALTER PLUGGABLE DATABASE ... SNAPSHOT` depends on the underlying file system. If the underlying file system supports sparse copies, then the PDB-level snapshots are sparse. Only the first PDB-managed PDB snapshot is full. Otherwise, the PDB snapshots contain full copies of the data files. The snapshot includes other files necessary to create a PDB from the snapshot.

Snapshot Directories

Every PDB has its own snapshot directory. Within this directory, each snapshot has its own subdirectory named after the SCN at which it was taken. The following query shows the sparse PDB snapshots for `hrpdb`, which has a DBID of `1389467754`:

```
SET LINESIZE 200
SET PAGESIZE 50000

COL SNAPSHOT_NAME FORMAT a27
COL FULL_SNAPSHOT_PATH FORMAT a65

SELECT SNAPSHOT_NAME, SNAPSHOT_SCN, FULL_SNAPSHOT_PATH FROM DBA_PDB_SNAPSHOTS;

SNAPSHOT_NAME               SNAPSHOT_SCN FULL_SNAPSHOT_PATH
--------------------------- ------------ ---------------------------------------
SNAP_1389467754_993556301        2925293 /d1/snapshots/pdb_1389467754/2925293/
SNAP_1389467754_993556306        2925679 /d1/snapshots/pdb_1389467754/2925679/
SNAP_1389467754_993556309        2925698 /d1/snapshots/pdb_1389467754/2925698/
```

> **Note:**
>
> If the snapshot were full instead of sparse, then the full snapshot path would specify an archive with the `.pdb` suffix.

The directory for `/d1/snapshots/pdb_1389467754/2925698/` contains the following files:

```
archparlog_1_63_52d1986a_993552590.arc
o1_mf_salestbs_g03341t2_.dbf
o1_mf_sysext_g0333vqw_.dbf
o1_mf_undo_1_g033gd2j_.dbf
o1_mf_sysaux_g0333vqv_.dbf
o1_mf_system_g0333vqt_.dbf
HRPDB.xml
```

The set includes the data files, archived redo log files, and an XML file that contains metadata about the PDB snapshot. The following `du` command shows that the size of the snapshot data files, which are sparse, is small relative to the size of the data files:

```
% du -h *dbf
16K     o1_mf_salestbs_g03341t2_.dbf
16K     o1_mf_sysaux_g0333vqv_.dbf
16K     o1_mf_sysext_g0333vqw_.dbf
16K     o1_mf_system_g0333vqt_.dbf
16K     o1_mf_undo_1_g033gd2j_.dbf
```

The following data dictionary join shows the snapshot file names and types for snapshot `2925698`:

```
SELECT f.SNAPSHOT_FILENAME, f.SNAPSHOT_FILETYPE
FROM   DBA_PDB_SNAPSHOTS s, DBA_PDB_SNAPSHOTFILE f
WHERE  s.SNAPSHOT_SCN=f.SNAPSHOT_SCN
AND    s.CON_ID=f.CON_ID
ORDER BY s.SNAPSHOT_SCN DESC;

SNAPSHOT_FILENAME                                                    SNAPSHOT
-------------------------------------------------------------------- --------
/d1/snapshots/pdb_1389467754/2925698/o1_mf_sysaux_g0333vqv_.dbf          DATA
/d1/snapshots/pdb_1389467754/2925698/o1_mf_system_g0333vqt_.dbf          DATA
/d1/snapshots/pdb_1389467754/2925698/HRPDB.xml                           XML
/d1/snapshots/pdb_1389467754/2925698/o1_mf_sysext_g0333vqw_.dbf          DATA
/d1/snapshots/pdb_1389467754/2925698/o1_mf_salestbs_g03341t2_.dbf        DATA
/d1/snapshots/pdb_1389467754/2925698/o1_mf_undo_1_g033gd2j_.dbf          DATA
/d1/snapshots/pdb_1389467754/2925698/archparlog_1_63_52d1986a_993552590.arc  ARCH
/d1/snapshots/pdb_1389467754/2925679/o1_mf_sysext_g0333vqw_.dbf          DATA
/d1/snapshots/pdb_1389467754/2925679/o1_mf_salestbs_g03341t2_.dbf        DATA
/d1/snapshots/pdb_1389467754/2925679/o1_mf_undo_1_g033gd2j_.dbf          DATA
/d1/snapshots/pdb_1389467754/2925679/o1_mf_sysaux_g0333vqv_.dbf          DATA
/d1/snapshots/pdb_1389467754/2925679/archparlog_1_63_52d1986a_993552590.arc  ARCH
/d1/snapshots/pdb_1389467754/2925679/HRPDB.xml                           XML
/d1/snapshots/pdb_1389467754/2925679/o1_mf_system_g0333vqt_.dbf          DATA
/d1/snapshots/pdb_1389467754/2925293/HRPDB.xml                           XML
/d1/snapshots/pdb_1389467754/2925293/o1_mf_system_g0333vqt_.dbf          DATA
/d1/snapshots/pdb_1389467754/2925293/o1_mf_sysaux_g0333vqv_.dbf          DATA
/d1/snapshots/pdb_1389467754/2925293/o1_mf_undo_1_g033gd2j_.dbf          DATA
/d1/snapshots/pdb_1389467754/2925293/o1_mf_salestbs_g03341t2_.dbf        DATA
/d1/snapshots/pdb_1389467754/2925293/o1_mf_sysext_g0333vqw_.dbf          DATA
/d1/snapshots/pdb_1389467754/2925293/archparlog_1_63_52d1986a_993552590.arc  ARCH
```

Contents of a PDB Snapshot Carousel

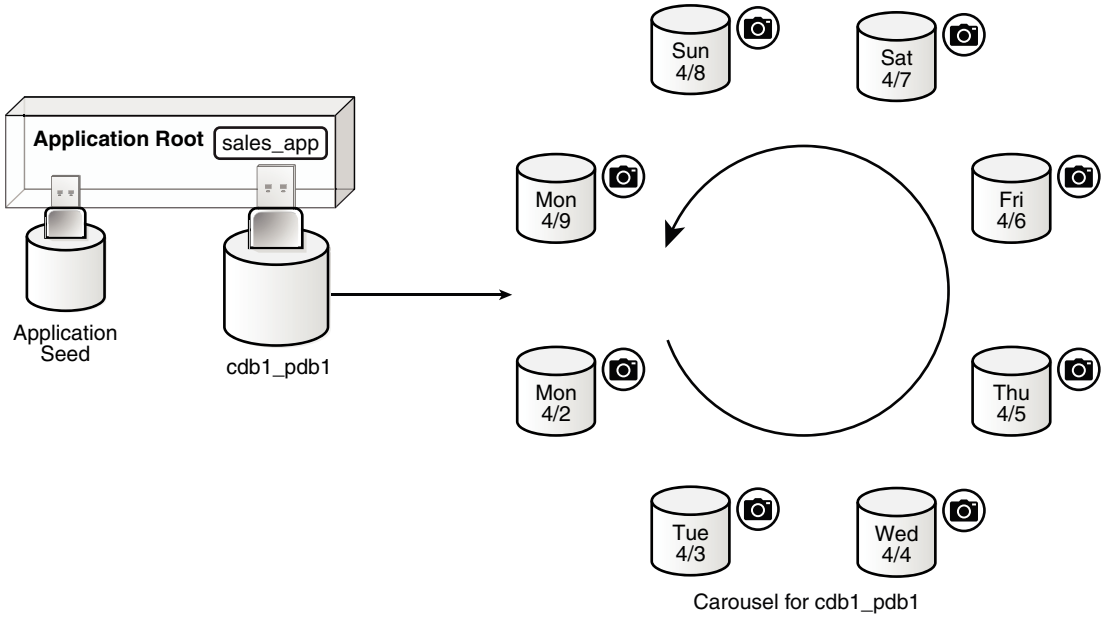
The PDB snapshot carousel is the set of all existing snapshots for a PDB.

The `MAX_PDB_SNAPSHOTS` property specifies the maximum number of snapshots permitted in the carousel. The current setting is visible in the `CDB_PROPERTIES` view.

The following figure shows a carousel for `cdb1_pdb1`. In this example, the database takes a PDB snapshot automatically every day, maintaining a set of 8. After the first 8 snapshots have been created, every new snapshot replaces the oldest snapshot.

For example, the Tuesday 4/10 snapshot replaces the Monday 4/2 snapshot; the Wednesday 4/11 snapshot replaces the Tuesday 4/3 snapshot; and so on.

Figure 16-3 PDB Snapshot Carousel



If the file system supports sparse files, then all PDB snapshots in the carousel except the first one are sparse. The source PDB can remain in read/write mode. Sparse files significantly reduce the carousel storage space.

> **See Also:**
>
> *Oracle Database Licensing Information User Manual* for details on which features are supported for different editions and services

User Interface for PDB Snapshot Carousel

The `SNAPSHOT MODE` clause controls creation of snapshots, and determines whether creation is manual, automatic, or disabled.

ALTER PLUGGABLE DATABASE ... SNAPSHOT Statement

To set the snapshot mode for a PDB, use one of the following values in the `SNAPSHOT MODE` clause of `ALTER PLUGGABLE DATABASE` or `CREATE PLUGGABLE DATABASE`:

- `MANUAL`

  This clause, which is the default, enables the creation of manual snapshots of the PDB. To create a snapshot on demand, specify the `SNAPSHOT` *`snapshot_name`* clause in an `ALTER PLUGGABLE DATABASE` or `CREATE PLUGGABLE DATABASE` statement.

- `EVERY snapshot_interval [MINUTES|HOURS]`

  This clause enables the automatic creation of snapshots after an interval of time. The following restrictions apply to the interval specified:

  – The minutes value must be less than `3000`.

  – The hours value must be less than `2000`.

  The database assigns each automatic snapshot a system-generated name. Note that manual snapshots are also supported for the PDB when `EVERY` is specified.

- `NONE`

  This clause disables snapshot creation for the PDB.

> **See Also:**
>
> - "About Cloning PDBs from PDB Snapshots"
> - *Oracle Database SQL Language Reference* for the syntax and semantics of the `SNAPSHOT` clause

MAX_PDB_SNAPSHOTS Database Property

To set the maximum number of snapshots for a PDB, specify the `MAX_PDB_SNAPSHOTS` property in `ALTER PLUGGABLE DATABASE` or `CREATE PLUGGABLE DATABASE`. The default is for the property is `8`, which is also the maximum value. When the maximum allowed number of snapshots has been created, the database purges the oldest snapshot. The `CDB_PROPERTIES` view shows the setting of `MAX_PDB_SNAPSHOTS`.

> **See Also:**
>
> *Oracle Database SQL Language Reference* for the syntax of the `ALTER PLUGGABLE DATABASE` statement

Snapshot-Related Data Dictionary Views

The following data dictionary views provide snapshot information:

- The `DBA_PDB_SNAPSHOTS` view records metadata about PDB snapshots, including snapshot name, creation SCN, creation time, and file name.
- The `DBA_PDB_SNAPSHOTFILE` view lists the names and types of the files in a PDB snapshot. This view is only populated when the snapshots are sparse.
- The `DBA_PDBS` view has a `SNAPSHOT_MODE` and `SNAPSHOT_INTERVAL` column.

> **See Also:**
>
> *Oracle Database Reference* to learn about `DBA_PDB_SNAPSHOTS`, `DBA_PDB_SNAPSHOTFILE`, and `DBA_PDBS`

Setting the Maximum Number of Snapshots in a PDB Snapshot Carousel

You can set the maximum number of PDB snapshots for a PDB.

The `MAX_PDB_SNAPSHOTS` database property sets the maximum number of snapshots for every PDB in a PDB snapshot carousel. The default maximum is `8`. You cannot set the property to a number greater than `8`.

Prerequisites

The PDB must be open in read/write mode.

To set the maximum number of PDB snapshots for a PDB:

1. In SQL*Plus, ensure that the current container is the PDB for which you want to set the limit.
2. Optionally, query `CDB_PROPERTIES` for the current setting of the `SET MAX_PDB_SNAPSHOTS` property.
3. Run an `ALTER PLUGGABLE DATABASE` or `ALTER DATABASE` statement with the `SET MAX_PDB_SNAPSHOTS` clause.

Example 16-1 Setting the Maximum Number of PDB Snapshots for a PDB

The following query shows the maximum in the carousel for `cdb1_pdb1` (sample output included):

```
SET LINESIZE 150
COL ID FORMAT 99
COL PROPERTY_NAME FORMAT a17
COL PDB_NAME FORMAT a9
COL VALUE FORMAT a3
COL DESCRIPTION FORMAT a43

SELECT r.CON_ID AS id, p.PDB_NAME, PROPERTY_NAME,
       PROPERTY_VALUE AS value, DESCRIPTION
FROM   CDB_PROPERTIES r, CDB_PDBS p
WHERE  r.CON_ID = p.CON_ID
AND    PROPERTY_NAME LIKE 'MAX_PDB%'
ORDER BY PROPERTY_NAME;

ID PDB_NAME  PROPERTY_NAME     VAL DESCRIPTION
-- --------- ----------------- --- -----------------------------------
 3 CDB1_PDB1 MAX_PDB_SNAPSHOTS 8   maximum number of snapshots for a PDB
```

The following SQL statement sets the maximum number of PDB snapshots for the current PDB to 7:

```
ALTER PLUGGABLE DATABASE SET MAX_PDB_SNAPSHOTS=7;
```

Example 16-2 Dropping All Snapshots in a PDB Snapshot Carousel

To drop all snapshots in a PDB snapshot carousel, set the `MAX_PDB_SNAPSHOTS` database property to `0` (zero), as shown in the following statement:

```
ALTER PLUGGABLE DATABASE SET MAX_PDB_SNAPSHOTS=0;
```

This technique is faster than executing `ALTER PLUGGABLE DATABASE ... DROP SNAPSHOT` *`snapshot_name`* for every snapshot.

> **See Also:**
>
> "About Container Access in a CDB"

Configuring Automatic PDB Snapshots

Configure a PDB for automatic snapshots by using the `SNAPSHOT MODE EVERY` clause when creating or altering a PDB.

By default, a PDB is configured for manual snapshots.

Prerequisites

Note the following prerequisites for the `ALTER PLUGGABLE DATABASE SNAPSHOT` statement:

- The CDB must be in local undo mode.
- The administrator must have the privileges to create a PDB and drop a PDB.

To configure automatic snapshots when altering a PDB:

1. In SQL*Plus, log in as an administrator to the PDB whose snapshot mode you intend to configure.
2. Optionally, query `DBA_PDBS` to determine the current snapshot mode.
3. Run `ALTER PLUGGABLE DATABASE` with the `SNAPSHOT MODE EVERY` *`interval`* clause, specifying either `MINUTES` or `HOURS`.

To configure automatic snapshots when creating a PDB:

1. In SQL*Plus, log in as an administrator to the CDB root or application root.
2. Optionally, query `DBA_PDBS` to determine the current snapshot mode.
3. Run `CREATE PLUGGABLE DATABASE` with the `SNAPSHOT MODE EVERY` *`interval`* clause, specifying either `MINUTES` or `HOURS`.

Example 16-3 Configuring an Automatic Snapshot Every Day for an Existing PDB

This example assumes that you are logged in to the PDB whose snapshot mode you intend to change. Query the data dictionary to confirm that the PDB is currently in `MANUAL` mode (sample output included):

```
SELECT SNAPSHOT_MODE "S_MODE", SNAPSHOT_INTERVAL/60 "SNAP_INT_HRS"
FROM   DBA_PDBS;

S_MODE SNAP_INT_HRS
------ ------------
MANUAL
```

Change the snapshot mode to every 24 hours:

```
ALTER PLUGGABLE DATABASE SNAPSHOT MODE EVERY 24 HOURS;
```

Confirm the change to automatic mode:

```
SELECT SNAPSHOT_MODE "S_MODE", SNAPSHOT_INTERVAL/60 "SNAP_INT_HRS"
FROM   DBA_PDBS;

S_MODE SNAP_INT_HRS
------ ------------
AUTO             24
```

Example 16-4 Creating a PDB That Takes Snapshots Every Two Hours

This example assumes that you are logged in to the CDB root. The following statement creates `cdb1_pdb3` from an existing PDB named `cdb1_pdb1`, and configures it to take snapshots automatically every 2 hours:

```
CREATE PLUGGABLE DATABASE cdb1_pdb3 FROM cdb1_pdb1
  FILE_NAME_CONVERT=('cdb1_pdb1','cdb1_pdb3')
  SNAPSHOT MODE EVERY 120 MINUTES;
```

See Also:

- "Cloning a PDB from a PDB Snapshot: Scenario"
- "Configuring a CDB to Use Local Undo Mode"

Creating PDB Snapshots Manually

To create a PDB snapshot manually, specify the `SNAPSHOT` *`snapshot_name`* clause in `ALTER PLUGGABLE DATABASE` or `CREATE PLUGGABLE DATABASE`.

Prerequisites

Note the following prerequisites for the `ALTER PLUGGABLE DATABASE SNAPSHOT` statement:

- The CDB must be in local undo mode. You can check the mode by using the following query, which returns `TRUE` when local undo is enabled:

  ```
  SELECT * FROM DATABASE_PROPERTIES WHERE
  PROPERTY_NAME='LOCAL_UNDO_ENABLED';
  ```

- The DBA must have the privileges to create and drop a PDB.
- If you want the snapshots to be sparse, then the underlying storage system must support sparse files. In this case, only the first snapshot will be full.

To create a PDB snapshot:

1. In SQL*Plus, log in as an administrator to the PDB whose snapshot you intend to create.
2. Optionally, query `DBA_PDBS.SNAPSHOT_MODE` to confirm that the snapshot mode is not set to `NONE`.
3. Run an `ALTER PLUGGABLE DATABASE` statement with the `SNAPSHOT` clause.

Example 16-5 Creating a PDB Snapshot with a User-Specified Name

The following SQL statements create two PDB snapshots of `cdb1_pdb1`, one before and one after a Wednesday data load:

```
ALTER PLUGGABLE DATABASE SNAPSHOT cdb1_pdb1_b4WLOAD;
-- data load
ALTER PLUGGABLE DATABASE SNAPSHOT cdb1_pdb1_afWLOAD;
```

The following query of `DBA_PDB_SNAPSHOTS` shows the locations of two snapshots of the PDB named `cdb1_pdb1` (sample output included):

```
SET LINESIZE 150
COL CON_NAME FORMAT a9
COL ID FORMAT 99
COL SNAPSHOT_NAME FORMAT a17
COL SNAP_SCN FORMAT 9999999
COL FULL_SNAPSHOT_PATH FORMAT a61

SELECT CON_ID AS ID, CON_NAME, SNAPSHOT_NAME,
       SNAPSHOT_SCN AS snap_scn, FULL_SNAPSHOT_PATH
FROM   DBA_PDB_SNAPSHOTS
ORDER BY SNAP_SCN;
```

```
 ID SNAPSHOT_NAME     SNAP_SCN FULL_SNAPSHOT_PATH
--- ---------------- --------
------------------------------------------------------------
  4 CDB1_PDB1_B4WLOAD  5056465 /ade/b/813544604/oracle/dbs/snapshots/
pdb_2935056285/5056465/
  4 CDB1_PDB1_AFWLOAD  5056501 /ade/b/813544604/oracle/dbs/snapshots/
pdb_2935056285/5056501/
```

If you do not specify a PDB snapshot name, then the database generates a unique name.

Example 16-6 Creating a PDB Snapshot with a System-Specified Name

The following SQL statement creates a snapshot, but does not specify a name:

```
ALTER PLUGGABLE DATABASE SNAPSHOT;
```

The following sample query shows that the database assigned the snapshot a name prefixed with `SNAP_`:

```
SET LINESIZE 150
COL CON_NAME FORMAT a9
COL ID FORMAT 99
COL SNAPSHOT_NAME FORMAT a26
COL SNAP_SCN FORMAT 9999999
COL FULL_SNAPSHOT_PATH FORMAT a61

SELECT CON_ID AS id, CON_NAME, SNAPSHOT_NAME,
       SNAPSHOT_SCN AS snap_scn, FULL_SNAPSHOT_PATH
FROM   DBA_PDB_SNAPSHOTS
ORDER BY SNAP_SCN;

 ID SNAPSHOT_NAME              SNAP_SCN FULL_SNAPSHOT_PATH
--- ------------------------- --------
------------------------------------------------------------
  4 CDB1_PDB1_B4WLOAD           5056465 /ade/b/813544604/oracle/dbs/snapshots/
pdb_2935056285/5056465/
  4 CDB1_PDB1_AFWLOAD           5056501 /ade/b/813544604/oracle/dbs/snapshots/
pdb_2935056285/5056501/
  4 SNAP_2935056285_1031574118  5057389 /ade/b/813544604/oracle/dbs/snapshots/
pdb_2935056285/5057389/
```

See Also:

- "About Container Access in a CDB"
- "Configuring a CDB to Use Local Undo Mode"

Dropping a PDB Snapshot

You can drop a PDB snapshot by running an `ALTER PLUGGABLE DATABASE` statement with the `DROP SNAPSHOT` clause.

To drop all PDB snapshots based on a PDB, set the `MAX_PDB_SNAPSHOTS` property in the PDB to `0` (zero).

To drop a PDB snapshot:

1. In SQL*Plus, ensure that the current container is the PDB from which you created the PDB snapshot.
2. Run an `ALTER PLUGGABLE DATABASE` statement with the `DROP SNAPSHOT` clause.

Example 16-7 Dropping a PDB Snapshot

The following SQL statement drops a PDB snapshot named `sales_snap`:

```
ALTER PLUGGABLE DATABASE DROP SNAPSHOT sales_snap;
```

> **See Also:**
>
> "About Container Access in a CDB"

Viewing Metadata for PDB Snapshots

The data dictionary views `DBA_PDB_SNAPSHOTS` and `DBA_PDB_SNAPSHOTFILE` show the metadata for PDB snapshots.

`DBA_PDB_SNAPSHOTS` contains general information about the snapshot, including name, SCN, time, and path. `DBA_PDB_SNAPSHOTFILE` shows the path and file type of every file in a snapshot: data files, archived redo log files, and XML files.

> **Note:**
>
> `DBA_PDB_SNAPSHOTFILE` only shows sparse clone PDBs. To create sparse clones, the `CLONEDB` initialization parameter must be set to `TRUE`.

To view metadata for PDB snapshots:

1. In SQL*Plus, log in to the database as an administrative user.
2. Query `DBA_PDB_SNAPSHOTS`.

   For example, run the following query (sample output included):

   ```
   COL SNAPSHOT_NAME FORMAT a30
   SELECT SNAPSHOT_NAME, SNAPSHOT_SCN, SNAPSHOT_TIME FROM
   DBA_PDB_SNAPSHOTS;
   ```

```
SNAPSHOT_NAME                  SNAPSHOT_SCN SNAPSHOT_TIME
------------------------------ ------------ -------------
HRPDB_SNAP_F                        3678939    1536262569
HRPDB_SNAP_S                        4954803     986473745
```

3. Query `DBA_PDB_SNAPSHOTFILE`.

   For example, run the following join query (sample output included):

```
SET LINESIZE 120
COL SNAPSHOT_NAME FORMAT a12
COL SNAPSHOT_FILENAME FORMAT a54

SELECT SNAPSHOT_NAME, SNAPSHOT_FILENAME, SNAPSHOT_FILETYPE AS TYPE
FROM   DBA_PDB_SNAPSHOTS s, DBA_PDB_SNAPSHOTFILE f
WHERE  s.SNAPSHOT_SCN=f.SNAPSHOT_SCN;

SNAPSHOT_NAM SNAPSHOT_FILENAME
TYPE
------------ ------------------------------------------------------
----
HRPDB_SNAP_S /d1/snapshots/4954803/o1_mf_undo_1_fry1l5bq_.dbf
DATA
HRPDB_SNAP_S /d1/snapshots/4954803/o1_mf_salestbs_fry19m6h_.dbf
DATA
HRPDB_SNAP_S /d1/snapshots/4954803/o1_mf_sysext_fry19d1n_.dbf
DATA
HRPDB_SNAP_S /d1/snapshots/4954803/o1_mf_sysaux_fry19d1m_.dbf
DATA
HRPDB_SNAP_S /d1/snapshots/4954803/o1_mf_system_fry19d1k_.dbf
DATA
HRPDB_SNAP_S /d1/snapshots/4954803/
HRPDB.xml                           XML
HRPDB_SNAP_S /d1/snapshots/4954803/archparlog_1_274_b87ca51e_985963
             814.arc
ARCH
```

Example 16-8 Querying Metadata for Full PDB Snapshots

The following query shows two PDB snapshots. The snapshots are full, not sparse, as indicated by the `.pdb` extension.

```
SET LINESIZE 200
SET PAGESIZE 50000

COL ID FORMAT 99
COL CON_NAME FORMAT a7
COL SNAPSHOT_NAME FORMAT a25
COL SNAPSHOT_SCN FORMAT a7
COL FULL_SNAPSHOT_PATH FORMAT a65

SELECT CON_ID AS ID, CON_NAME, SNAPSHOT_NAME,
       SNAPSHOT_SCN, FULL_SNAPSHOT_PATH
FROM   DBA_PDB_SNAPSHOTS;
```

```
ID CON_NAM SNAPSHOT_NAME             SNAPSHO FULL_SNAPSHOT_PATH
-- ------- ------------------------- -------
------------------------------
 5 HRPDB   SNAP_3286480866_994766895 3160319 /d1/
snap_3286480866_3160319.pdb
 5 HRPDB   SNAP_3286480866_994767095 3165758 /d1/
snap_3286480866_3165758.pdb
```

The following query of `DBA_PDB_SNAPSHOTFILE` returns no rows because this view is only populated when PDB snapshots are sparse:

```
SQL> SELECT COUNT(*) FROM DBA_PDB_SNAPSHOTFILE;

  COUNT(*)
----------
         0
```

17

Administering Application Containers

You can administer application containers, including application roots and application PDBs. You can also administer the applications installed in application containers.

> **Note:**
>
> You can complete the tasks in this chapter using SQL*Plus or Oracle SQL Developer.

Related Topics

- Creating and Removing Application Containers and Seeds
  You can create application containers and application seeds in several different ways. You can also remove application containers from a CDB, and you can remove application seeds from application containers.
- Tools for a Multitenant Environment
  You can use various tools to configure and administer a multitenant environment.

About Application Container Administration

Some aspects of administering an application container are similar to administering the CDB root and the CDB as a whole, while other aspects are similar to administering a PDB.

Administering an application container is similar to administering a CDB because you can manage both the application root and the application PDBs that are plugged into the application root. However, administering an application container is also similar to managing a PDB because changes to the application container do not affect other application containers or PDBs in the CDB.

The following table describes administrative tasks for application containers that are similar to administrative tasks that manage a CDB or CDB root.

Table 17-1 Application Container Administrative Tasks Similar to Those of a CDB

Administrative Task	Description	More Information
Configuring application common users and commonly granted privileges	Application common users and privileges are similar to common users and commonly granted privileges in a CDB root, but in an application container, common users and commonly granted privileges only exist within the containers of the application container, including the application root, application PDBs that belong to the application root, and an optional application seed that belongs to the application root.	"Overview of Common and Local Users in a CDB"
Creating application containers	A common user whose current container is the CDB root can create application containers that are plugged into the CDB root by specifying the `AS APPLICATION CONTAINER` clause in the `CREATE PLUGGABLE DATABASE` statement. The database files must be Oracle Managed Files.	"Creating Application Containers"
Creating application PDBs	A common user whose current container is the application root can create application PDBs that are plugged into the application root.	"Creating and Removing PDBs and Application Containers"
Switching to containers	A common user with the proper privileges can switch between containers in an application container, including the application root, application PDBs that belong to the application root, and an optional application seed that belongs to the application root.	"Switching to a Container Using the ALTER SESSION Statement"
Issuing `ALTER SYSTEM SET` statements	The `ALTER SYSTEM SET` statement can dynamically set an initialization parameter in one or more containers in an application container.	"Modifying a CDB with ALTER SYSTEM"
Issuing data definition language (DDL) statements	In an application container, some DDL statements can apply to all containers in the application container or to the current container only.	"Modifying Application Common Objects with DDL Statements"

The following table describes administrative tasks for application containers that are similar to administrative tasks that manage a PDB.

Table 17-2 Application Container Administrative Tasks Similar to Those of a PDB

Administrative Task	Description	More Information
Connecting to the application root	The application root has its own service name, and users can connect to the application root in the same way that they connect to a PDB. Similarly, each application PDB has its own service name, and the application seed has its own service name.	"Accessing a Container in a CDB"
Issuing the `ALTER PLUGGABLE DATABASE` statement	An `ALTER PLUGGABLE DATABASE` statement can modify an application root, application PDB, and application seed in the same way it modifies a PDB. For example, an administrator can open or close an application root with an `ALTER PLUGGABLE DATABASE` statement.	"Modifying Containers When Connected to the CDB Root" "Modifying a PDB at the Database Level"
Issuing the SQL*Plus `STARTUP` and `SHUTDOWN` commands	SQL*Plus `STARTUP` and `SHUTDOWN` commands operate on an application root, application PDB, and application seed in the same way that they operate on a PDB.	"Modifying the Open Mode of PDBs"
Issuing the `ALTER SYSTEM` statements	An `ALTER SYSTEM` statement operates on an application root, application PDB, and application seed in the same way that it operates on a PDB.	"Modifying a CDB with ALTER SYSTEM" "Modifying a PDB at the System Level"
Managing tablespaces	Administrators can create, modify, and drop tablespaces for an application root and for application PDBs. Each container has its own tablespaces.	"About Managing Tablespaces in a CDB"
Managing data files and temp files	Administrators can create, modify, and drop data files and temp files for an application root and for application PDBs. Each container has its own files.	*Oracle Database Administrator's Guide* for information about managing data files and temp files

Table 17-2 (Cont.) Application Container Administrative Tasks Similar to Those of a PDB

Administrative Task	Description	More Information
Managing schema objects	You can create, modify, and drop schema objects in an application root and in each application PDB in the same way that you would in a PDB. You can also create triggers that fire for a specific application root or application PDB. However, application containers support application common objects, which can be shared between the containers in an application container. Application common objects cannot be created in PDBs.	"Managing Application Common Objects"

About Modifying an Application Root

The `ALTER DATABASE` statement can modify an application root. The `ALTER PLUGGABLE DATABASE` statement can modify the open mode of application PDBs.

The following table lists which containers are modified by clauses in `ALTER DATABASE` and `ALTER PLUGGABLE DATABASE` statements issued in an application root. The table also lists statements that are not allowed in an application root.

> **Note:**
>
> Statements issued when the current container is the application root never affect the CDB root or PDBs that do not belong to the current application root.

Table 17-3 Statements That Modify Containers in an Application Root

Modify Application Root Only	Modify One or More Application PDBs	Cannot Be Issued in an Application Root
When connected as an application common user whose current container is the application root, `ALTER DATABASE` statements with the following clauses modify the application root only: • *database_file_clauses* • `DEFAULT EDITION` clause • `DEFAULT TABLESPACE` clause • `DEFAULT TEMPORARY TABLESPACE` clause `ALTER DATABASE` statements with the following clauses modify the application root and set default values for application PDBs: • *flashback_mode_clause* • `SET DEFAULT {BIGFILE | SMALLFILE} TABLESPACE` clause • *set_time_zone_clause* You can use these clauses to set nondefault values for specific application PDBs.	When connected as an application common user whose current container is the application root, `ALTER PLUGGABLE DATABASE` statements with the following clause can modify the open mode of one or more application PDBs: • *pdb_change_state* When the current container is an application PDB, `ALTER PLUGGABLE DATABASE` statements with this clause can modify the open mode of the current application PDB. When connected as an application common user whose current container is the application root, `ALTER PLUGGABLE DATABASE` statements with the following clause can preserve or discard the open mode an application PDB when the CDB restarts: • *pdb_save_or_discard_state*	When connected as an application common user whose current container is the application root, `ALTER DATABASE` statements with the following clauses are not allowed: • *startup_clauses* • *recovery_clauses* • *logfile_clauses* • *controlfile_clauses* • *standby_database_clauses* • *instance_clauses* • *security_clause* • `RENAME GLOBAL_NAME` clause • `ENABLE BLOCK CHANGE TRACKING` clause • `DISABLE BLOCK CHANGE TRACKING` clause

See Also:

- "About the Current Container"
- "Modifying a PDB at the Database Level"
- *Oracle Database SQL Language Reference*

Managing Applications in an Application Container

You install, upgrade, or patch an application in an application container.

You can also uninstall an application from an application container. You perform these operations in the application root. The application container propagates the application changes to the application PDBs when the application PDBs synchronize with the application in the application root.

Related Topics

- Overview of Applications in an Application Container
  Within an application container, an **application** is the named, versioned set of common data and metadata stored in the application root.

About Application Management

In an application container, an **application** is a named, versioned set of application metadata and common data. The application is stored in the application root.

In this context, the term “application” means “application back-end.” Application common objects include user accounts, tables, PL/SQL packages, and so on. You can share an application with the application PDBs that belong to the application root. When you perform application changes, application PDBs can synchronize with the application in the application root.

Basic Steps of Application Maintenance

You can install, upgrade, and patch an application in an application root.

You must issue an `ALTER PLUGGABLE DATABASE ... BEGIN` statement to start the operation and an `ALTER PLUGGABLE DATABASE ... END` statement to end the operation. You can issue these statements in the same user session or in different user sessions.

The following is the typical process for creating and maintaining an application in an application container:

1. Create the application container.
2. Install the application in the application root using `ALTER PLUGGABLE DATABASE ... BEGIN INSTALL`.

   This step includes creating the application data model and configuring the application common users and application common objects.

   > **Note:**
   >
   > SQL*Loader is the only supported utility for bulk inserts into tables during application install, upgrade, and patch operations.

3. Create the application PDBs in the application root.
4. Synchronize each application PDB that should install the application with the application root. The statement is `ALTER PLUGGABLE DATABASE APPLICATION ... SYNC`.
5. Load the data for each application PDB.
6. Maintain the application. Upgrade using `ALTER PLUGGABLE DATABASE ... BEGIN UPGRADE`, and patch using `ALTER PLUGGABLE DATABASE ... BEGIN PATCH`.
7. Synchronize application PDBs that should apply changes from upgrades and patches.
8. Add new application PDBs whenever necessary.
9. If necessary, uninstall the application using `ALTER PLUGGABLE DATABASE ... BEGIN UNINSTALL`.

> **See Also:**
>
> - "Creating Application Containers"
> - *Oracle Database Security Guide* to learn how to audit application maintenance operations

Application Versions

The application container also manages the versions of the application and the patches to the application.

The application container manages versions as follows:

- When you install an application, you must specify the application version number.
- When you upgrade an application, you must specify the old application version number and the new application version number.
- When you patch an application, you must specify the minimum application version number for the patch and the patch number.

As the application evolves, the application container maintains all of the versions and patch changes that you apply.

You can also configure the application container so that different application PDBs use different application versions. For example, if you provide an application to various customers, and each customer has its own application PDB, some customers might wait longer to upgrade the application. In this case, some application PDBs can use the latest version of the application, whereas other application PDBs can use an older version of the application.

Application Module Names and Service Names

The application module name is set by the `DBMS_APPLICATION_INFO.SET_MODULE` procedure or the equivalent OCI attribute setting.

The module name is necessary during application maintenance because of other activity that might be occurring in the database. For example, statements issued by background processes should not be captured in the application capture tables. Also, other users might execute statements that are unrelated to the application. A module name check distinguishes what should be captured from what should not be captured. Only sessions whose module name matches the module name of the session where `APPLICATION BEGIN` was issued are considered for capture.

Query `DBA_APPLICATIONS` to determine the module name of the session in which `APPLICATION BEGIN` was executed:

```
SELECT app_capture_module FROM dba_applications WHERE app_name='APEX';
```

Some clauses, such as the `SHARING` clause, are valid only when issued between an `ALTER PLUGGABLE DATABASE ... BEGIN` statement and an `ALTER PLUGGABLE DATABASE ... END` statement. For these clauses, if the module name for a session does not match, then this session is not included in between the `BEGIN` and `END`

statements, causing statements that include the clause to fail with ORA-65021 or other errors.

The most common cause for a module name mismatch is the default module name. For example, SQL*Plus sets a default module name when a connection is made to the database. A connection as a SYSDBA user results in one default module name (for example, sqlplus@host1 (TNS V1-V3)), whereas a connection as a non-SYSDBA user results in a different default module name (for example, SQL*Plus). When SYSDBA and non-SYSDBA users are both performing maintenance, you must explicitly set the module name in each session to the same value, and not rely the default settings in SQL*Plus.

Additionally, for the statement to be captured the service name of the session executing a statement should match the service name of the session where APPLICATION BEGIN was executed. Query DBA_APPLICATIONS to determine the service name of the session in which APPLICATION BEGIN was executed:

```
SELECT app_capture_service FROM dba_applications WHERE app_name='APEX';
```

Example 17-1 Checking the Session's Module Name

This example shows how the default module name changes depending on whether the connected user has SYSDBA privileges.

```
SQL> CONNECT / AS SYSDBA
Connected.

SQL> select module from v$session where audsid =
SYS_CONTEXT('USERENV','sessionid');

MODULE
----------------------------------------------------------------
sqlplus@host1 (TNS V1-V3)

SQL> CONNECT dba1
Password: *************
Connected.

SQL> select module from v$session where audsid =
SYS_CONTEXT('USERENV','sessionid');

MODULE
----------------------------------------------------------------
SQL*Plus
```

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference* to learn how to set the application module name

Installing Applications in an Application Container

You can install an application in an application container.

About Installing Applications in an Application Container

You issue `ALTER PLUGGABLE DATABASE APPLICATION` statements to install an application in the application root.

You install the application in the application root only. Application PDBs that synchronize with the application install the application automatically. With the automated method, you can perform the installation using one or more of the following techniques: scripts, SQL statements, and graphical user interface tools.

Start of the installation with an `ALTER PLUGGABLE DATABASE APPLICATION BEGIN INSTALL` statement and the end of the install with an `ALTER PLUGGABLE DATABASE APPLICATION END INSTALL` statement. Each installation must be associated with an application name and version number, which are specified in the `ALTER PLUGGABLE DATABASE APPLICATION` statements.

Related Topics

- Running Oracle-Supplied SQL Scripts in a CDB
  You can use the `catcon.pl` script to run Oracle-supplied SQL or SQL scripts within a CDB. You can run the script against any specified containers.

Installing an Application in an Application Container with Automated Propagation

In automated propagation, the application is installed in the application PDBs that synchronize with the application in the application root.

Prerequisites

You must meet the following prerequisites:

- The current user must have the `ALTER PLUGGABLE DATABASE` system privilege, and the privilege must be commonly granted in the application root.
- The application root must be in open read/write.

To install an application using automated propagation:

1. In SQL*Plus or SQL Developer, ensure that the current container is a PDB.
2. Run the `ALTER PLUGGABLE DATABASE APPLICATION BEGIN INSTALL` statement in the following form:

```
ALTER PLUGGABLE DATABASE APPLICATION application_name BEGIN INSTALL
'application_version_number';
```

For example, run the following statement if the *application_name* is `salesapp` and the *application_version_number* is 4.2:

```
ALTER PLUGGABLE DATABASE APPLICATION salesapp BEGIN INSTALL '4.2';
```

3. Install the application using scripts, SQL statements, or graphical user interface tools.
4. Run the `ALTER PLUGGABLE DATABASE APPLICATION END INSTALL` statement in the following form:

```
ALTER PLUGGABLE DATABASE APPLICATION application_name END INSTALL
'application_version_number';
```

For example, run the following statement if the *application_name* is `salesapp` and the *application_version_number* is 4.2:

```
ALTER PLUGGABLE DATABASE APPLICATION salesapp END INSTALL '4.2';
```

> **Note:**
>
> Ensure that the *application_name* and *application_version_number* match in the `ALTER PLUGGABLE DATABASE APPLICATION BEGIN INSTALL` statement and the`ALTER PLUGGABLE DATABASE APPLICATION END INSTALL` statement.

5. Synchronize all of the application PDBs that must install the application by issuing an `ALTER PLUGGABLE DATABASE APPLICATION` statement with the `SYNC` clause.

Related Topics

- Accessing a Container in a CDB
  Access a container in a CDB with SQL*Plus by issuing a `CONNECT` or `ALTER SESSION` command.
- Synchronizing Applications in an Application PDB
  Synchronizing an application updates the application in the application PDB to the latest version and patch in the application root.

Upgrading Applications in an Application Container

Major changes to an application constitute application upgrades. You can upgrade an application in an application container.

About Upgrading Applications in an Application Container

You issue `ALTER PLUGGABLE DATABASE APPLICATION` statements to upgrade an application in the application root.

Related Topics

- Running Oracle-Supplied SQL Scripts in a CDB
  You can use the `catcon.pl` script to run Oracle-supplied SQL or SQL scripts within a CDB. You can run the script against any specified containers.

Purpose of Application Upgrade

You can upgrade the application definition once in the application root so that other application PDBs can synchronize with the upgraded definition.

Application PDBs do not automatically inherit the upgraded application definition in the application root. Application PDBs synchronize with an application in the root when you manually run an `ALTER PLUGGABLE DATABASE` statement with the `SYNC` clause. You can upgrade using one or more of the following techniques: scripts, SQL statements, and graphical user interface tools.

How an Application Upgrade Works

When you upgrade an application, Oracle Database automatically clones the application root.

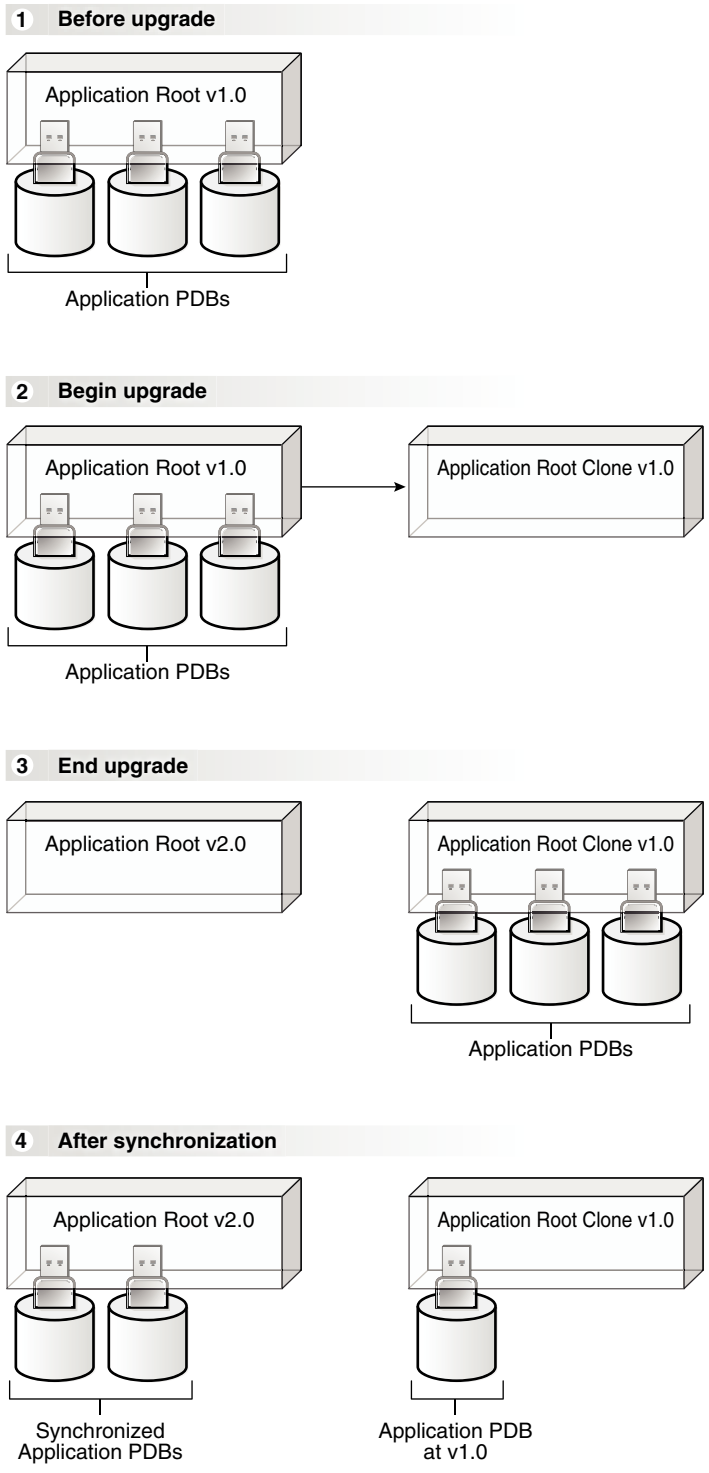
During the upgrade, application PDBs point to the root clone. Applications continue to run during the upgrade. Application PDBs can perform DML on metadata-linked and extended data-linked tables and views. Application PDBs can query metadata-linked objects, extended data-linked objects, and data-linked objects.

After the upgrade, the application root clone remains and continues to support any application PDB that still use the preupgrade version of the application in the root clone. Application PDBs that upgrade are pointed to the upgraded application root. Application PDBs that do not upgrade might continue to use the clone, and application PDBs that are plugged into the application root might also use the same application version as the root clone.

> **Note:**
>
> Unlike an application upgrade, a patch does not create an application root clone. If an application PDB is not synchronized after a patch, then queries are directed to the application root, which has already been patched.

The following figure illustrates the application upgrade process.

Figure 17-1 Upgrading Applications in an Application Container



> **Note:**
>
> When the application root is in any open mode, the application root clone is in read-only mode. When the application root is closed, the application root clone is also closed.

User Interface for Application Upgrade

To upgrade an application definition in the application root, use the `ALTER PLUGGABLE DATABASE APPLICATION ... UPGRADE` command.

Start the upgrade with an `ALTER PLUGGABLE DATABASE APPLICATION BEGIN UPGRADE` statement and end with an `ALTER PLUGGABLE DATABASE APPLICATION END UPGRADE` statement. Each upgrade must be associated with an application name, starting version number, and ending version number, which are specified in the `ALTER PLUGGABLE DATABASE APPLICATION` statements.

> **Note:**
>
> If Transparent Data Encryption is enabled in the application root, then an external password store must be configured.

Upgrading an Application in an Application Container

After an upgrade, application changes caused by the upgrade propagate to the application PDBs that synchronize with the application root.

Prerequisites

- The CDB must be in local undo mode.
- The current user must have the `ALTER PLUGGABLE DATABASE` system privilege, and the privilege must be commonly granted in the application root.
- The application root must be in open read/write.
- If Transparent Data Encryption is enabled in the application root, then an external password store must be configured.

To upgrade an application in an application container:

1. In SQL*Plus or SQL Developer, ensure that the current container is the application root.
2. Run the `ALTER PLUGGABLE DATABASE APPLICATION BEGIN UPGRADE` statement in the following form:

```
ALTER PLUGGABLE DATABASE APPLICATION application_name
BEGIN UPGRADE 'application_start_version_number' TO
'application_end_version_number';
```

For example, run the following statement if the *application_name* is `salesapp`, the *application_start_version_number* is 4.2, and the *application_end_version_number* is 4.3:

```
ALTER PLUGGABLE DATABASE APPLICATION salesapp BEGIN UPGRADE '4.2'
TO '4.3';
```

3. Upgrade the application using scripts, SQL statements, or graphical user interface tools.
4. Run the `ALTER PLUGGABLE DATABASE APPLICATION END UPGRADE` statement in the following form:

```
ALTER PLUGGABLE DATABASE APPLICATION application_name END UPGRADE
TO 'application_end_version_number';
```

For example, run the following statement if the *application_name* is `salesapp` and the *application_end_version_number* is 4.3:

```
ALTER PLUGGABLE DATABASE APPLICATION salesapp END UPGRADE TO '4.3';
```

> **Note:**
>
> Ensure that the *application_name* and *application_end_version_number* match in the `ALTER PLUGGABLE DATABASE APPLICATION BEGIN UPGRADE` statement and the `ALTER PLUGGABLE DATABASE APPLICATION END UPGRADE` statement.

5. Synchronize all of the application PDBs that must upgrade the application by issuing an `ALTER PLUGGABLE DATABASE APPLICATION` statement with the `SYNC` clause.

Related Topics

- Accessing a Container in a CDB
  Access a container in a CDB with SQL*Plus by issuing a `CONNECT` or `ALTER SESSION` command.
- Synchronizing Applications in an Application PDB
  Synchronizing an application updates the application in the application PDB to the latest version and patch in the application root.
- Setting the Undo Mode in a CDB Using ALTER DATABASE
  When local undo is enabled, each container has its own undo tablespace for every instance in which it is open. When local undo is disabled, there is one undo tablespace for the entire CDB.

Patching Applications in an Application Container

Minor changes to an application constitute application patches.

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

About Patching Applications in an Application Container

To patch an application in the application root, issue `ALTER PLUGGABLE DATABASE APPLICATION` statements.

You patch the application in the application root only. The application PDBs that synchronize with the application apply the changes. You can perform the patch using one or more of the following techniques: scripts, SQL statements, and graphical user interface tools.

The patch is restricted to a small set of operations. In general, destructive operations, such as dropping a table, are not allowed in a patch. If you attempt to patch an application, and the operation raises an "operation not supported in an application patch" error, then upgrade the application instead of patching it to make the necessary changes.

> **Note:**
>
> Unlike an application upgrade, a patch does not create an application root clone. If an application PDB is not synchronized after a patch, then queries are directed to the application root, which has already been patched.

Indicate the start of the patch with an `ALTER PLUGGABLE DATABASE APPLICATION BEGIN PATCH` statement and the end of the patch with an `ALTER PLUGGABLE DATABASE APPLICATION END PATCH` statement. Each patch must be associated with an application name, starting version number, and ending version number. Specify these values in the `ALTER PLUGGABLE DATABASE APPLICATION` statements.

Related Topics

- Running Oracle-Supplied SQL Scripts in a CDB
  You can use the `catcon.pl` script to run Oracle-supplied SQL or SQL scripts within a CDB. You can run the script against any specified containers.
- Upgrading Applications in an Application Container
  Major changes to an application constitute application upgrades. You can upgrade an application in an application container.

Patching an Application in an Application Container with Automated Propagation

Application changes for the patch are propagated to the application PDBs that synchronize with the application in the application root.

Prerequisites

The following prerequisites must be met:

- The current user must have the `ALTER PLUGGABLE DATABASE` system privilege, and the privilege must be commonly granted in the application root.
- The application root must be in open read/write mode.

1. In SQL*Plus, ensure that the current container is the application root.

2. Run the `ALTER PLUGGABLE DATABASE APPLICATION BEGIN PATCH` statement in the following form:

```
ALTER PLUGGABLE DATABASE APPLICATION application_name
  BEGIN PATCH patch_number
  MINIMUM VERSION 'minimum_application_version_number';
```

   For example, run the following statement if the *application_name* is `salesapp`, the *patch_number* is 987654, and the *minimum_application_version_number* is 4.2:

```
ALTER PLUGGABLE DATABASE APPLICATION salesapp
  BEGIN PATCH 987654 MINIMUM VERSION '4.2';
```

   The *minimum_application_version_number* indicates the minimum application version at which an application installation must be before the patch can be applied to it.

3. Patch the application using scripts, SQL statements, and graphical user interface tools.

4. Run the `ALTER PLUGGABLE DATABASE APPLICATION END PATCH` statement in the following form:

```
ALTER PLUGGABLE DATABASE APPLICATION application_name
  END PATCH patch_number;
```

   For example, run the following statement if the *application_name* is `salesapp` and the *patch_number* is 987654:

```
ALTER PLUGGABLE DATABASE APPLICATION salesapp END PATCH 987654;
```

   > **Note:**
   >
   > Ensure that the *application_name* and *patch_number* match in the `ALTER PLUGGABLE DATABASE APPLICATION BEGIN PATCH` statement and the `ALTER PLUGGABLE DATABASE APPLICATION END PATCH` statement.

5. Synchronize all of the application PDBs that must patch the application by issuing an `ALTER PLUGGABLE DATABASE APPLICATION` statement with the `SYNC` clause.

Related Topics

- Accessing a Container in a CDB
  Access a container in a CDB with SQL*Plus by issuing a `CONNECT` or `ALTER SESSION` command.
- Synchronizing Applications in an Application PDB
  Synchronizing an application updates the application in the application PDB to the latest version and patch in the application root.

Migrating an Existing Application to an Application Container

You can migrate an application that is installed in a PDB to an application container.

You can migrate the application to the application root or to an application PDB. For example, you might migrate an application installed in a PDB plugged into an Oracle Database 12c Release 2 (12.2) CDB to an application container in an Oracle Database 18c CDB.

About Migrating an Existing Application to an Application Container

You can migrate an application to an application root by creating an application root using an existing PDB.

If the application is installed in more than one PDB, then you can use one of the PDBs to create the application root. You can use one of the methods available for copying a PDB to an application root, such as cloning the PDB or plugging in the PDB as an application root.

When common users, roles, or profiles exist in the PDB used to create the application root, you must run procedures in the `DBMS_PDB` package to associate them with the application. When an application root created from a PDB is first opened, each local user, role, and profile is marked as common. The procedures in the `DBMS_PDB` package associate the user, role, or profile with the application. Therefore, all DDL operations on the user, role, or profile must subsequently be done within an application `BEGIN...END` block of this application.

When shared database objects exist in the application root, you must run procedures in the `DBMS_PDB` package to associate the database objects with the application as application common objects. Therefore, all DDL operations on the application common objects must subsequently be done within an application `BEGIN...END` block of this application.

After the application root is in place, you can create application PDBs in the new application container using the existing PDBs. The application PDBs that you create must contain the application objects, including their data. Additional steps are necessary to synchronize the application version and patch number and to establish shared database objects in the application PDBs.

Scenario with One Hundred PDBs Running the Same Application

Assume that you currently have one hundred PDBs that are running the same application, and you want to migrate these PDBs to an application container. These PDBs have the application common objects and common users, roles, and profiles required by the application. To migrate the PDBs to an application container, follow these steps:

1. Choose one of the PDBs, and use the instructions in "Creating an Application Root Using an Existing PDB" to create the application root with this PDB.

   As part of this step, you associate the database objects, users, roles, and profiles with the application by running procedures in the `DBMS_PDB` package.

2. Use the instructions in "Creating an Application PDB Using an Existing PDB" to create one hundred application PDBs using the PDBs that are running the application.

> **See Also:**
>
> - "Creating an Application Container"
> - "Installing an Application in an Application Container with Automated Propagation"
> - *Oracle Database PL/SQL Packages and Types Reference* to learn more about `DBMS_PDB`

Creating an Application Root Using an Existing PDB

Migrate an application that is installed in a PDB by copying the PDB to an application container.

Prerequisites

An Oracle Database 12c Release 2 (12.2) or later CDB must exist.

1. In the CDB, create the application root by cloning the existing PDB, relocating the existing PDB, or by unplugging and plugging in the existing PDB.

   The new application root must contain all database objects used by the application.
2. With the application root as the current container, start an application installation operation by issuing an `ALTER PLUGGABLE DATABASE ... BEGIN INSTALL` statement.
3. Optional: Query the `COMMON` column in the `DBA_USERS`, `DBA_ROLES`, and `DBA_PROFILES` views to determine which users, roles, and profiles are common.
4. Run the following procedures in the `DBMS_PDB` package to associate users, roles. and profiles with the application:
   - Run the `SET_USER_EXPLICIT` procedure to set application common users.
   - Run the `SET_ROLE_EXPLICIT` procedure to set application common roles.
   - Run the `SET_PROFILE_EXPLICIT` procedure to set application common profiles.

   If you do not have `EXECUTE` privilege on the `DBMS_PDB` package, then you can run these procedures in the `DBMS_PDB_ALTER_SHARING` package.
5. Optional: With the application root as the current container, query the `SHARING` column in the `DBA_OBJECTS` view to determine which database objects are shared.
6. Run the following procedures in the `DBMS_PDB` package to associate database objects with the application:
   - Run the `SET_DATA_LINKED` procedure to set data-linked application common objects.
   - Run the `SET_METADATA_LINKED` procedure to set metadata-linked application common objects.
   - Run the `SET_EXT_DATA_LINKED` procedure to set extended data-linked application common objects.

If you do not have EXECUTE privilege on the DBMS_PDB package, then you can run these procedures in the DBMS_PDB_ALTER_SHARING package.

7. End the application installation operation by issuing an ALTER PLUGGABLE DATABASE ... END INSTALL statement.
8. Optional: Rerun the queries that you ran previously to ensure that the sharing properties of the database objects are correct and that the common properties of the users, roles, and profiles are correct.
9. Optional: If existing PDBs use the application, then create application PDBs using these existing PDBs.

   See "Creating an Application PDB Using an Existing PDB".

Related Topics

- Creating an Application Container
  You can create an application container using the CREATE PLUGGABLE DATABASE statement with the AS APPLICATION CONTAINER clause.
- Managing Applications in an Application Container
  You install, upgrade, or patch an application in an application container.
- *Oracle Database PL/SQL Packages and Types Reference*

Creating an Application PDB Using an Existing PDB

After migrating an existing application to an application root, you can use an existing PDB that uses the application to create an application PDB.

Prerequisites

You must meet the following prerequisites:

- An Oracle Database 12c Release 2 (12.2) or later CDB must exist, and the application root to which the application PDB will belong must exist.
- The PDB must contain all application common objects used by the application.
- The application must be installed in the application root.

1. In the application root, create the application PDB by cloning the existing PDB or by unplugging and plugging in the existing PDB.

   Violations will be reported during PDB creation.
2. Connect to or switch to the new PDB as a user with the required privileges.
3. Run the pdb_to_apppdb.sql script in the *ORACLE_HOME*/rdbms/admin directory.

   The script automatically synchronizes the application PDB with the application root.
4. Optional: Query the SHARING column in the DBA_OBJECTS view to ensure that the sharing properties of the database objects are correct.
5. Optional: Query the COMMON column in the DBA_USERS, DBA_ROLES, and DBA_PROFILES views to ensure that the common properties of the users, roles, and profiles are correct.

Related Topics

- Creating and Removing PDBs and Application Containers
  You can create PDBs, application containers, and application seeds using a variety of techniques.

Synchronizing Applications in an Application PDB

Synchronizing an application updates the application in the application PDB to the latest version and patch in the application root.

Installing, upgrading, patching, or uninstalling an application in an application root does not change its application PDBs until they are synchronized. When the application PDB is the current container, synchronize manually using one of the following forms of `ALTER PLUGGABLE DATABASE APPLICATION ... SYNC`:

- Synchronize a single application as follows, where *app1* is the name of the application:

  ```
  ALTER PLUGGABLE DATABASE APPLICATION app1 SYNC;
  ```

  Optionally, specify `SYNC TO PATCH patchno` to synchronize *app1* to the specified patch, and `SYNC TO version` to synchronize *app1* to the specified version.
- Synchronize all applications as follows:

  ```
  ALTER PLUGGABLE DATABASE APPLICATION ALL SYNC;
  ```

Prerequisites and Restrictions

- The current user must have `ALTER PLUGGABLE DATABASE` system privilege.
- When specifying multiple applications using `ALL`, the `SYNC TO` clause is not supported.
- Specifying multiple applications using `ALL` replays application `BEGIN` and `END` blocks in the order in which they were captured. When applications depend on one another, synchronizing them in a single statement is necessary for functional correctness.

1. In SQL*Plus, ensure that the current container is the application PDB.
2. Run an `ALTER PLUGGABLE DATABASE APPLICATION` statement with the `SYNC` clause.

Example 17-2 Synchronizing a Specific Application in an Application PDB

This example synchronizes an application named `salesapp` in an application PDB with the latest application changes in the application root.

```
ALTER PLUGGABLE DATABASE APPLICATION salesapp SYNC;
```

Example 17-3 Synchronizing an Application to a Specified Patch

This example synchronizes an application named `salesapp` in an application PDB to patch 100.

```
ALTER PLUGGABLE DATABASE APPLICATION salesapp SYNC TO PATCH 100;
```

Example 17-4 Synchronizing an Application to a Specified Application Release

This example synchronizes an application named `salesapp` in an application PDB to release 2.0 of the application.

```
ALTER PLUGGABLE DATABASE APPLICATION salesapp SYNC TO '2.0';
```

Example 17-5 Synchronizing All of the Applications in an Application PDB

This example synchronizes all of the applications in an application PDB with the latest application changes in the application root.

```
ALTER PLUGGABLE DATABASE APPLICATION ALL SYNC;
```

Example 17-6 Synchronizing Implicitly Created Applications in an Application PDB

This example synchronizes all of the implicitly-created applications in an application PDB with the latest application changes to the implicitly created applications in the application root.

```
ALTER PLUGGABLE DATABASE APPLICATION APP$CON SYNC;
```

> **See Also:**
>
> "Application Synchronization"

Synchronizing an Application Root Replica with a Proxy PDB

When application containers in different CDBs have the same application, their application roots can be kept synchronized by creating a master application root, a replica application root, and a proxy PDB.

About Synchronizing an Application Root Replica with a Proxy PDB

A proxy PDB can synchronize an application root and a replica of the application root.

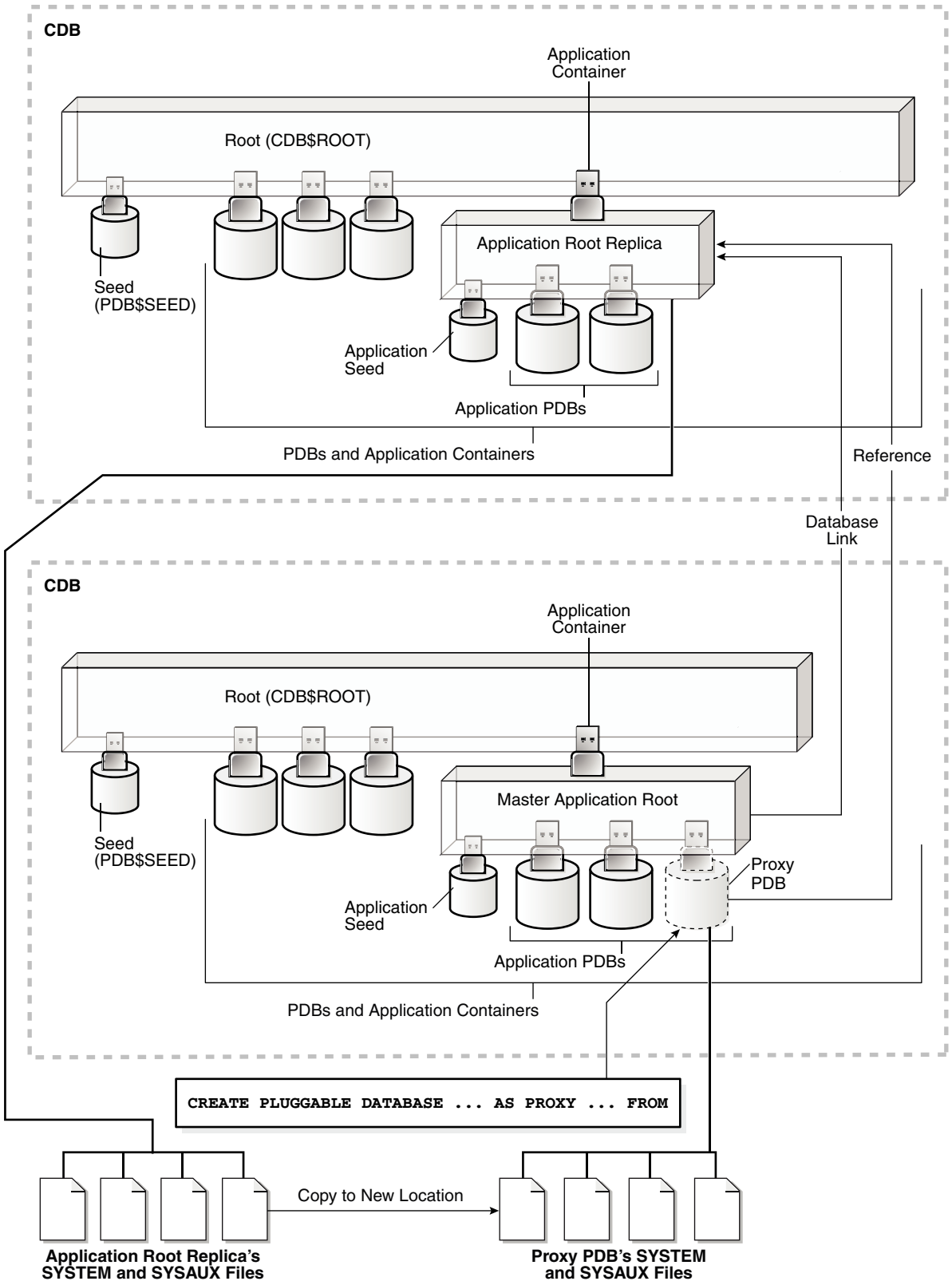
An application might be installed in several application containers. Installing, upgrading, and patching the application are more efficient when you use proxy PDBs.

In this configuration, one application container has the master application root. The master application root is where you install, upgrade, and patch the application. Application root replicas are exact copies of the master application root. Each application root replica is referenced by a proxy PDB in the master application root.

When a proxy PDB is synchronized with the application changes in the master application root, it propagates the changes to its referenced application root replica. After the application root replica is synchronized, application PDBs that are plugged into the application root replica can synchronize with the replica and in this way receive the changes.

The following figure shows a configuration that synchronizes an application root replica using a proxy PDB.

Figure 17-2 Synchronizing an Application Root Replica with a Proxy PDB



In addition, when an application root replica is configured and has its own application PDBs, a query that includes the `CONTAINERS` clause in the master application root can return data from the current application container and from the application container with the application root replica. The query can show results from the application root replica and from any open application PDBs plugged into the replica.

> **See Also:**
>
> "Querying Application Common Objects Across Application PDBs"

Creating a Proxy PDB That References an Application Root Replica

When multiple application containers run the same application, the application in the application containers can be kept synchronized using proxy PDBs.

1. Create the application container with the master application root by using a `CREATE PLUGGABLE DATABASE` statement.

   Install the application in the application container now or later.

2. Create the application container with the application root replica in one of the following ways:

   - Create an empty application container using any supported method.
   - Clone the master application root.

   If the port of the listener used by the application root replica is not 1521, then a `PORT` clause is required during creation. If the host of the application root replica is different from the host of the master application root, then a `HOST` clause is required during creation.

   This application root replica will be referenced by the proxy PDB.

3. In the master application root, create a proxy PDB that references the application root replica that you created in the previous step.

4. Open and synchronize the proxy PDB.

   When the proxy PDB is synchronized, it propagates the changes in the master application root to the application root replica.

5. Optional: In the master application root, modify the application by installing, upgrading, or patching it.

6. Optional: Synchronize the proxy PDB with the application changes in the master application root by running the `ALTER PLUGGABLE DATABASE APPLICATION` statement with the `SYNC` clause.

   When the proxy PDB is synchronized, it propagates the changes in the master application root to the application root replica.

Example 17-7 Synchronizing an Application Root Replica with a Proxy PDB

This example assumes that two CDBs exist: `hqdb` and `depdb`. The goal is to keep the same application synchronized in an application container in each CDB. To accomplish this goal, this example configures the following application containers:

- The hqdb CDB contains the application container with the master application root called msappcon.
  - An application called sampleapp is installed in the msappcon master application root.
  - The msappcon application root contains two application PDBs named mspdb1 and mspdb2.
  - The msappcon application root also contains a proxy PDB named prxypdb that references the application root replica in the other CDB.
- The depdb CDB contains the application container with the application root replica called depappcon.
  - An application called sampleapp is propagated from the proxy PDB prxypdb in the msappcon master application root and installed in the depappcon master application root.
  - The depappcon application root contains two application PDBs named deppdb1 and deppdb2.

This example shows how changes to the sampleapp application in the msappcon master application root are applied to the application PDBs in both CDBs when the application PDBs are synchronized.

1. Create the application container with the master application root in the hqdb CDB.
   a. In SQL*Plus, ensure that the current container is the hqdb CDB root.
   b. Create the application container from the PDB seed with the following statement:

```
CREATE PLUGGABLE DATABASE msappcon
  AS APPLICATION CONTAINER
  ADMIN USER msappconadm IDENTIFIED BY password
  STORAGE (MAXSIZE 2G)
  DEFAULT TABLESPACE appcontbs
    DATAFILE '/disk1/oracle/dbs/mssappcon/msappcon01.dbf' SIZE
250M
      AUTOEXTEND ON
      FILE_NAME_CONVERT = ('/disk1/oracle/dbs/pdbseed/',
                           '/disk1/oracle/dbs/msappcon/');
```

   c. Open the new master application root in read/write mode:

```
ALTER PLUGGABLE DATABASE msappcon OPEN;
```

2. Install an application in the master application root.
   a. Change container to the master application root:

```
ALTER SESSION SET CONTAINER=msappcon;
```

   b. Begin the application installation:

```
ALTER PLUGGABLE DATABASE APPLICATION sampleapp BEGIN INSTALL
'1.0';
```

c. Install the application.

For example, you can create database objects:

```
CREATE TABLE apptb SHARING=METADATA
   (id            NUMBER(6),
    widget_name   VARCHAR2(20));
```

d. End the application installation:

```
ALTER PLUGGABLE DATABASE APPLICATION sampleapp END INSTALL '1.0';
```

3. Create and synchronize one or more application PDBs in the master application root.

a. In SQL*Plus, ensure that the current container is the master application root.

b. Create application PDBs in the master application root.

For example, create two application PDBs from the PDB seed:

```
CREATE PLUGGABLE DATABASE mspdb1 ADMIN USER mspdb1admin
IDENTIFIED BY password
  STORAGE (MAXSIZE 2G)
  DEFAULT TABLESPACE mspdb1tbs
    DATAFILE '/disk1/oracle/dbs/mspdb1/mspdb101.dbf' SIZE 250M
    AUTOEXTEND ON
    FILE_NAME_CONVERT = ('/disk1/oracle/dbs/pdbseed/',
                         '/disk1/oracle/dbs/mspdb1/');

CREATE PLUGGABLE DATABASE mspdb2 ADMIN USER mspdb2admin
IDENTIFIED BY password
  STORAGE (MAXSIZE 2G)
  DEFAULT TABLESPACE mspdb2tbs
    DATAFILE '/disk1/oracle/dbs/mspdb2/mspdb201.dbf' SIZE 250M
    AUTOEXTEND ON
    FILE_NAME_CONVERT = ('/disk1/oracle/dbs/pdbseed/',
                         '/disk1/oracle/dbs/mspdb2/');
```

c. Open both application PDBs:

```
ALTER PLUGGABLE DATABASE mspdb1 OPEN;
ALTER PLUGGABLE DATABASE mspdb2 OPEN;
```

d. Synchronize the application PDBs with the master application root:

```
ALTER SESSION SET CONTAINER=mspdb1;
ALTER PLUGGABLE DATABASE APPLICATION sampleapp SYNC;

ALTER SESSION SET CONTAINER=mspdb2;
ALTER PLUGGABLE DATABASE APPLICATION sampleapp SYNC;
```

4. Create the application container with the application root replica in the `depdb` CDB.

a. In SQL*Plus, ensure that the current container is the `depdb` CDB root.

b. Create the application container from the PDB seed with the following statement:

```
CREATE PLUGGABLE DATABASE depappcon
  AS APPLICATION CONTAINER
  ADMIN USER depappconadm IDENTIFIED BY password
  STORAGE (MAXSIZE 2G)
  DEFAULT TABLESPACE appcontbs
    DATAFILE '/disk2/oracle/dbs/depsappcon/depappcon01.dbf' SIZE
250M
    AUTOEXTEND ON
    FILE_NAME_CONVERT = ('/disk2/oracle/dbs/pdbseed/',
                         '/disk2/oracle/dbs/depappcon/');
```

> **Note:**
>
> - If the port of the listener used by the application root replica is not 1521, then a `PORT` clause is required.
> - If the host of the application root replica is different from the host of the master application root, then a `HOST` clause is required.

c. Open the new application root replica in read/write mode:

```
ALTER PLUGGABLE DATABASE depappcon OPEN;
```

5. Create and synchronize the proxy PDB in the master application root.
   a. In SQL*Plus, ensure that the current container is the master application root.
   b. Create a database link to the application root replica:

```
CREATE PUBLIC DATABASE LINK depappcon
  CONNECT TO depappconadm IDENTIFIED BY password USING
'depappcon';
```

   c. Create the proxy PDB:

```
CREATE PLUGGABLE DATABASE prxypdb AS PROXY
   FROM depappcon@depappcon
   FILE_NAME_CONVERT = ('/disk2/oracle/dbs/depsappcon/',
                        '/disk1/oracle/dbs/prxypdb/');
```

   d. Open the proxy PDB:

```
ALTER PLUGGABLE DATABASE prxypdb OPEN;
```

   e. Synchronize the proxy PDB with the master application root:

```
ALTER SESSION SET CONTAINER=prxypdb;
ALTER PLUGGABLE DATABASE APPLICATION sampleapp SYNC;
```

6. Create and synchronize one or more application PDBs in the application root replica.

   a. Change container to the application root replica:

   ```
   ALTER SESSION SET CONTAINER=depappcon;
   ```

   b. Create application PDBs in the application root replica.

   For example, create two application PDBs from the PDB seed:

   ```
   CREATE PLUGGABLE DATABASE deppdb1
     ADMIN USER deppdb1admin IDENTIFIED BY password
     STORAGE (MAXSIZE 2G)
     DEFAULT TABLESPACE deppdb1tbs
       DATAFILE '/disk2/oracle/dbs/deppdb1/deppdb101.dbf' SIZE 250M
       AUTOEXTEND ON
       FILE_NAME_CONVERT = ('/disk2/oracle/dbs/pdbseed/',
                            '/disk2/oracle/dbs/deppdb1/');

   CREATE PLUGGABLE DATABASE deppdb2 ADMIN USER deppdb2admin
   IDENTIFIED BY password
     STORAGE (MAXSIZE 2G)
     DEFAULT TABLESPACE deppdb2tbs
       DATAFILE '/disk2/oracle/dbs/deppdb2/deppdb201.dbf' SIZE 250M
       AUTOEXTEND ON
       FILE_NAME_CONVERT = ('/disk2/oracle/dbs/pdbseed/',
                            '/disk2/oracle/dbs/deppdb2/');
   ```

   c. Open both application PDBs:

   ```
   ALTER PLUGGABLE DATABASE deppdb1 OPEN;
   ALTER PLUGGABLE DATABASE deppdb2 OPEN;
   ```

   d. Synchronize the application PDBs with the master application root:

   ```
   ALTER SESSION SET CONTAINER=deppdb1;
   ALTER PLUGGABLE DATABASE APPLICATION sampleapp SYNC;

   ALTER SESSION SET CONTAINER=deppdb2;
   ALTER PLUGGABLE DATABASE APPLICATION sampleapp SYNC;
   ```

7. Check the structure of the `apptb` table in an application PDB in the application root replica.

   a. From the application root replica, switch containers to the `deppdb1` application PDB:

   ```
   ALTER SESSION SET CONTAINER=deppdb1;
   ```

   b. Describe the `apptb` table:

   ```
   desc apptb
   ```

Your output is similar to the following:

```
Name                           Null?    Type
------------------------------ -------- ------------
ID                                      NUMBER(6)
WIDGET_NAME                             VARCHAR2(20)
```

8. In the master application root, upgrade the application.
   a. Change container to the master application root:

      ```
      ALTER SESSION SET CONTAINER=msappcon;
      ```

   b. Begin the application upgrade.

      ```
      ALTER PLUGGABLE DATABASE APPLICATION sampleapp
        BEGIN UPGRADE '1.0' TO '1.1';
      ```

   c. Modify the application.

      For example, add a row to the `apptb` table:

      ```
      ALTER TABLE apptb ADD (widget_type VARCHAR2(30));
      ```

   d. End the application upgrade:

      ```
      ALTER PLUGGABLE DATABASE APPLICATION sampleapp END UPGRADE TO
      '1.1';
      ```

9. Synchronize the proxy PDB with the master application root:

   ```
   ALTER SESSION SET CONTAINER=prxypdb;
   ALTER PLUGGABLE DATABASE APPLICATION sampleapp SYNC;
   ```

10. Synchronize the application PDBs in the application root replica and check for the application upgrade.
    a. Synchronize the application PDBs:

       ```
       ALTER SESSION SET CONTAINER=deppdb1;
       ALTER PLUGGABLE DATABASE APPLICATION sampleapp SYNC;

       ALTER SESSION SET CONTAINER=deppdb2;
       ALTER PLUGGABLE DATABASE APPLICATION sampleapp SYNC;
       ```

    b. From the application root replica, switch containers to the `deppdb1` application PDB:

       ```
       ALTER SESSION SET CONTAINER=deppdb1;
       ```

    c. Describe the `apptb` table:

       ```
       desc apptb
       ```

Your output is similar to the following:

```
Name                           Null?    Type
------------------------------ -------- ------------
ID                                      NUMBER(6)
WIDGET_NAME                             VARCHAR2(20)
WIDGET_TYPE                             VARCHAR2(30)
```

Notice that the change in the application upgrade is reflected in the output because the `widget_type` column has been added to the `apptb` table.

Related Topics

- Creating Application Containers
  You can use the `CREATE PLUGGABLE DATABASE` statement to create an application container in a CDB.
- Creating a PDB as a Proxy PDB
  You can create a PDB as a proxy PDB by referencing it in a remote CDB.
- Managing Applications in an Application Container
  You install, upgrade, or patch an application in an application container.
- Synchronizing Applications in an Application PDB
  Synchronizing an application updates the application in the application PDB to the latest version and patch in the application root.

Setting the Compatibility Version of an Application

The compatibility version of an application is the earliest version of the application possible for the application PDBs that belong to the application container.

The compatibility version is enforced when the compatibility version is set and when an application PDB is created. If there are application root clones that resulted from application upgrades, then all application root clones that correspond to versions earlier than the compatibility version are implicitly dropped.

You specify the compatibility version of an application by issuing one of the following SQL statements when the application root is the current container:

- `ALTER PLUGGABLE DATABASE APPLICATION` *`application_name`* `SET COMPATIBILITY VERSION '`*`application_version_number`*`';`

  application_name is the name of the application, and *application_version_number* is the earliest compatible version.

- `ALTER PLUGGABLE DATABASE APPLICATION` *`application_name`* `SET COMPATIBILITY VERSION CURRENT;`

  application_name is the name of the application. The current version is the version of the application in the application root.

> **Note:**
>
> You cannot plug in an application PDB that uses an application version earlier than the compatibility setting of the application container.

1. In SQL*Plus, ensure that the current container is the application root.
2. Run an `ALTER PLUGGABLE DATABASE APPLICATION SET COMPATIBILITY VERSION` statement.

Example 17-8 Setting the Compatibility Version to a Specific Version Number

This example sets the compatibility version for an application named `salesapp` to version 4.2.

```
ALTER PLUGGABLE DATABASE APPLICATION salesapp
  SET COMPATIBILITY VERSION '4.2';
```

Example 17-9 Setting the Compatibility Version to the Current Application Version

This example sets the compatibility version for an application named `salesapp` to the current application version.

```
ALTER PLUGGABLE DATABASE APPLICATION salesapp
  SET COMPATIBILITY VERSION CURRENT;
```

> **See Also:**
>
> "About Upgrading Applications in an Application Container" for information about application root clones

Performing Bulk Inserts During Application Install, Upgrade, and Patch Operations

SQL*Loader is the only supported utility for bulk inserts into tables during application install, upgrade, and patch operations. Only conventional path loads are supported for bulk inserts during application install, upgrade, and patch operations.

The correct SQL*Loader module name must be specified between the `ALTER PLUGGABLE DATABASE APPLICATION BEGIN` and the `ALTER PLUGGABLE DATABASE APPLICATION END` statements. The module name is `SQL Loader Conventional Path Load`.

1. In SQL*Plus, ensure that the current container is the application root.
2. Set the correct module by running the following procedure:

   ```
   BEGIN
     DBMS_APPLICATION_INFO.SET_MODULE(
       'SQL Loader Conventional Path Load', '');
   END;
   ```

   This module must remain set for the entire application install, upgrade, or patch operation.
3. Run the `ALTER PLUGGABLE DATABASE APPLICATION BEGIN` statement for beginning an application installation, upgrade, or patch.

For example, if you are performing the bulk insert as part of an application installation, then run the `ALTER PLUGGABLE DATABASE APPLICATION BEGIN INSTALL` statement in the following form:

```
ALTER PLUGGABLE DATABASE APPLICATION application_name
  BEGIN INSTALL 'application_version_number';
```

4. Perform the conventional path load with SQL*Loader.
5. Run the `ALTER PLUGGABLE DATABASE APPLICATION END` statement for ending an application installation, upgrade, or patch.

   For example, if you are performing the bulk insert as part of an application installation, then run the `ALTER PLUGGABLE DATABASE APPLICATION END INSTALL` statement in the following form:

   ```
   ALTER PLUGGABLE DATABASE APPLICATION application_name
     END INSTALL 'application_version_number';
   ```

   > **Note:**
   >
   > Ensure that the *application_name* and *application_version_number* match in the `ALTER PLUGGABLE DATABASE APPLICATION BEGIN INSTALL` statement and the `ALTER PLUGGABLE DATABASE APPLICATION END INSTALL` statement.

6. Synchronize all application PDBs that must include these application changes by issuing an `ALTER PLUGGABLE DATABASE APPLICATION` statement with the `SYNC` clause.

Example 17-10 Performing a Conventional Path Load During an Application Installation

In this example, the conventional path load is performed in an application root.

1. In SQL*Plus, switch to the application root.

   ```
   ALTER SESSION SET CONTAINER=cdb1_approot1;
   ```

2. Set the correct module.

   ```
   BEGIN
     DBMS_APPLICATION_INFO.SET_MODULE(
       'SQL Loader Conventional Path Load', '');
   END;
   ```

3. Start the application installation.

   ```
   ALTER PLUGGABLE DATABASE APPLICATION APP1 BEGIN INSTALL '1';
   ```

4. Use SQL*Loader to perform the conventional path load.

```
HOST sqlldr u1/u1@cdb1_approot1 control=my_bulk_load.ctl -
rows=3 log=my_bulk_load.log
```

5. End the application installation.

```
ALTER PLUGGABLE DATABASE APPLICATION APP1 END INSTALL '1';
```

> **See Also:**
>
> *Oracle Database Utilities* for information about SQL*Loader

Uninstalling Applications from an Application Container

You can uninstall an application in an application container.

About Uninstalling Applications from an Application Container

You issue `ALTER PLUGGABLE DATABASE APPLICATION` statements to uninstall an application from the application root.

You uninstall the application from the application root only, and application PDBs that synchronize with the application uninstall the application automatically. The uninstall operation can be done with one or more of the following: scripts, SQL statements, and graphical user interface tools.

You must indicate the start of the uninstallation with an `ALTER PLUGGABLE DATABASE APPLICATION BEGIN UNINSTALL` statement and the end of the uninstallation with an `ALTER PLUGGABLE DATABASE APPLICATION END UNINSTALL` statement. Each uninstallation must be associated with an application name and version number, which are specified in the `ALTER PLUGGABLE DATABASE APPLICATION` statements.

Uninstalling an application does not remove the application from the data dictionary. It marks the application as `UNINSTALLED` so that upgrade, patch, and uninstall of the application is disallowed.

Destructive changes to application objects are allowed during application uninstallation. Applications running in an application PDB continue to function during uninstallation and after the application is uninstalled from the application root. The application can continue to function in the application PDB because the `ALTER PLUGGABLE DATABASE APPLICATION BEGIN UNINSTALL` statement creates a clone of the application root called an application root clone. An application root clone serves as a metadata repository for old versions of application objects, so that application PDBs that have not been synchronized with latest version of the application can continue to function. Because the clone is created while the application PDB is open, local undo must be configured at the CDB level before an application can be uninstalled.

> **Note:**
>
> An application upgrade also creates an application root clone.

> **See Also:**
>
> - "About Upgrading Applications in an Application Container" for information about application root clones
> - "Running Oracle-Supplied SQL Scripts in a CDB"

Uninstalling an Application from an Application Container

To uninstall an application in from application container, run the `ALTER PLUGGABLE DATABASE APPLICATION BEGIN UNINSTALL` statement to begin the uninstallation and the `ALTER PLUGGABLE DATABASE APPLICATION END UNINSTALL` statement to end it. The application uninstalled from the application PDBs that synchronize with the application in the application root.

The following prerequisites must be met:

- The CDB must be in local undo mode.
- The current user must have the `ALTER PLUGGABLE DATABASE` system privilege, and the privilege must be commonly granted in the application root.
- The application root must be in open read/write mode.

1. In SQL*Plus, ensure that the current container is the application root.
2. Run the `ALTER PLUGGABLE DATABASE APPLICATION BEGIN UNINSTALL` statement in the following form:

   ```
   ALTER PLUGGABLE DATABASE APPLICATION application_name BEGIN UNINSTALL;
   ```

   For example, run the following statement if the *application_name* is `salesapp`:

   ```
   ALTER PLUGGABLE DATABASE APPLICATION salesapp BEGIN UNINSTALL;
   ```

3. Uninstall the application using scripts, SQL statements, or graphical user interface tools.
4. Run the `ALTER PLUGGABLE DATABASE APPLICATION END UNINSTALL` statement in the following form:

   ```
   ALTER PLUGGABLE DATABASE APPLICATION application_name END UNINSTALL;
   ```

   For example, run the following statement if the *application_name* is `salesapp`:

   ```
   ALTER PLUGGABLE DATABASE APPLICATION salesapp END UNINSTALL;
   ```

> **Note:**
>
> Ensure that the *application_name* matches in the `ALTER PLUGGABLE DATABASE APPLICATION BEGIN UNINSTALL` statement and the`ALTER PLUGGABLE DATABASE APPLICATION END UNINSTALL` statement.

5. Synchronize all of the application PDBs that must uninstall the application by issuing an `ALTER PLUGGABLE DATABASE APPLICATION` statement with the `SYNC` clause.

> **See Also:**
>
> - "Accessing a Container in a CDB"
> - "Synchronizing Applications in an Application PDB"
> - "Setting the Undo Mode in a CDB Using ALTER DATABASE"

Managing Application Common Objects

Application common objects are shared, user-created database objects in an application container. Application common objects are created in an application root.

About Application Common Objects

Application common objects are created in an application root and are shared with the application PDBs that belong to the application root.

There are three types of application common object: metadata-linked, data-linked, and extended data-linked. The following types of database objects can be application common objects:

- Analytic views
- Attribute dimensions
- Directories
- External procedure libraries
- Hierarchies
- Java classes, resources, and sources
- Object tables, types, and views
- Sequences
- Packages, stored functions, and stored procedures
- Synonyms
- Tables (including global temporary tables)
- Triggers
- Views

> **See Also:**
>
> "Application Common Objects"

Creation of Application Common Objects

Create application common objects by issuing a `CREATE` statement when the current container is the application root and specifying the `SHARING` clause.

You can specify the sharing attribute by including the `SHARING` clause in the `CREATE` statement or by setting the `DEFAULT_SHARING` initialization parameter in the application root. When you set the `DEFAULT_SHARING` initialization parameter, the setting is the default sharing attribute for all database objects of a supported type created in the application root. However, when a `SHARING` clause is included in a `CREATE` statement, its setting overrides the setting for the `DEFAULT_SHARING` initialization parameter.

You can specify one of the following for the sharing attribute:

- `METADATA`: A metadata link shares the database object's metadata, but its data is unique to each container. These database objects are referred to as metadata-linked application common objects. This setting is the default.
- `DATA`: A data link shares the database object, and its data is the same for all containers in the application container. Its data is stored only in the application root. These database objects are referred to as data-linked application common objects.
- `EXTENDED DATA`: An extended data link shares the database object, and its data in the application root is the same for all containers in the application container. However, each application PDB in the application container can store data that is unique to the application PDB. For this type of database object, data is stored in the application root and, optionally, in each application PDB. These database objects are referred to as extended data-linked application common objects.
- `NONE`: The database object is not shared.

For most types of application common objects, the only valid settings for the `SHARING` clause are `METADATA` and `NONE`. The following types of application common objects allow additional settings for the `SHARING` clause:

- For tables (excluding object tables), the `SHARING` clause can be set to `METADATA`, `DATA`, `EXTENDED DATA`, or `NONE`. For object tables, only `METADATA` or `NONE` is valid.
- For views (excluding object views), the `SHARING` clause can be set to `METADATA`, `DATA`, `EXTENDED DATA`, or `NONE`. For object views, only `METADATA` or `NONE` is valid.
- For sequences, the `SHARING` clause can be set to `METADATA`, `DATA`, or `NONE`.

  With a metadata-linked sequence, each application PDB has its own sequence. When the metadata-linked sequence is incremented using the `NEXTVAL` pseudocolumn in one application PDB, it does not affect the value of the sequence in the other application PDBs in the application container.

  With a data-linked sequence, each application PDB shares the same sequence in the application root. When the metadata-linked sequence is incremented using the `NEXTVAL` pseudocolumn in one application PDB, all other application PDBs in the same application container also see the change.

Application common objects can be created or changed only as part of an application installation, upgrade, or patch. An application PDB applies changes to application common objects when it synchronizes with the application that made the changes. If an application PDB is closed when an application common object is created, dropped, or modified, then the appropriate changes are applied in the application PDB when it is opened and synchronized with the application.

The names of application common objects must not conflict with those of local database objects in any of the application PDBs that belong to the application root or Oracle-supplied common objects in the CDB root. If a newly opened application PDB contains a local database object whose name conflicts with that of an application common object, then the application PDB is opened in `RESTRICTED` mode. In this case, you must resolve the naming conflict before the application PDB can be opened in normal mode.

About Metadata-Linked Application Common Objects

For metadata-linked application common objects, the metadata for the object is stored once in the application root.

A metadata link in each application PDB that belongs to the application root enables the application PDBs to share the metadata for the object, including the object name and structure. The data for the object is unique to each container, including the application root and each application PDB that belongs to the application root.

Data definition language (DDL) operations on a metadata-linked application common object can be run in the application root only as part of an application installation, upgrade, or patch. However, the data can be modified in an application PDB using normal data manipulation language (DML) operations.

For example, consider a company with several regional offices. The company wants the structure of the information about employees to be consistent, but each office has different employees. If this company has a human resources application in an application container, it can create a separate application PDB for each regional office and use a metadata-linked table to store employee information. The data structure of the table, such as the columns, is the same in the application PDB for each regional office, but the employee data is different.

Another example might involve a company that builds and maintains a sales application that is used by several different businesses. Each business uses the same sales application, but the data for each business is different. For example, each business has different customers and therefore different customer data. To ensure that each client uses the same data structure for its application, the company might create an application container with metadata-linked application common objects. Each business that uses the sales application has its own application PDB, and the data structure is the same in each application PDB, but the data is different.

About Extended Data-Linked Application Common Objects

For data-linked application common objects, both the metadata and the data for the object is stored once in the application root. A data link in each application PDB that belongs to the application root enables the application PDBs to share the metadata and data of the object.

DDL operations on a data-linked application common object can be run in the application root only as part of an application installation, upgrade, or patch. In

addition, the data can be modified using normal DML operations only in the application root. The data cannot be modified in application PDBs.

For example, consider a company with several regional offices. The company wants the information about the products they sell, such as the product names and descriptions, to be consistent at all of the regional offices. If this company has a sales application in an application container, then it can create a separate application PDB for each regional office and use a data-linked table to store product information. Each application PDB can query the product information, and the product information is consistent at each regional office.

Data-linked application common objects are also useful for data that is standard and does not change. For example, a table that stores the postal codes for a country might be a data-linked application common object in an application container. All of the application PDBs access the same postal code data in the application root.

> **Note:**
>
> If the data-linked application common object is part of a configuration that synchronizes an application root replica with a proxy PDB, then DML operations on a data-linked object in the application root can be done outside of an application action, but the DML operation is not automatically propagated to the application root replication through the proxy PDB. If you want the DML operation to be propagated to the application root replica, then the DML operation on a data-linked object in the application root must be done within an application installation, upgrade, or patch.

About Extended Data-Linked Application Common Objects

For an extended data-linked object, each application PDB can create its own data while sharing the common data in the application root. Only data stored in the application root is common for all application PDBs.

DDL operations on an extended data-linked application common object can be run in the application root only as part of an application installation, upgrade, or patch. However, the data can be modified in the application root or in an application PDB using normal DML operations.

For example, a sales application in an application container might support several application PDBs, and all of the application PDBs need the postal codes in the United States for shipping purposes. In this case the postal codes can be stored in the application root so that all of the application PDBs can access it. However, one application PDB also makes sales in Canada, and this application PDB requires the postal codes for the United States and Canada. This one application PDB can store the postal codes for Canada in an extended data-linked object in the application PDB instead of in the application root.

> **Note:**
>
> - Tables and views are the only types of database objects that can be extended data-linked objects.
> - If the extended data-linked application common object is part of a configuration that synchronizes an application root replica with a proxy PDB, then DML operations on an extended data-linked object in the application root can be done outside of an application action, but the DML operation is not automatically propagated to the application root replication through the proxy PDB. If you want the DML operation to be propagated to the application root replica, then the DML operation on an extended data-linked object in the application root must be done within an application installation, upgrade, or patch.

Restrictions for Application Common Objects

Some restrictions apply to application common objects.

Queries on application common objects can return data from a container that is not the current container. For example, when the current container is an application root, queries that include the `CONTAINERS` clause can return data from application PDBs for metadata-linked application common objects. Also, when the current container is an application PDB, queries on data-linked and extended data-linked application common objects return data that resides in the application root.

Columns of the following types return no data in queries that return data from a container other than the current container:

- The following user-defined types: object types, varrays, REFs, and nested tables
- The following Oracle-supplied types: `ANYTYPE`, `ANYDATASET`, URI types, `SDO_TOPO_GEOMETRY`, `SDO_GEORASTER`, and `Expression`

In addition, queries on object tables and object views return no data from containers other than the current container.

Related Topics

- Querying Application Common Objects Across Application PDBs
  The `CONTAINERS` clause enables you to query application common objects across all PDBs in an application container. Queries from the application root display data in objects that exist in all open PDBs in the container.

Creating Application Common Objects

You create an application common object in an application root either by ensuring that the `DEFAULT_SHARING` initialization parameter is set to the correct value or by including the `SHARING` clause in the `CREATE` SQL statement.

You can create a metadata-linked object, an extended data-linked, or a data-linked object in an application root as part of an application installation, upgrade, or patch. An application PDB applies changes to application common objects when it synchronizes with the application in the application root.

1. In SQL*Plus, ensure that the current container is the application root.

   The current user must have the privileges required to create the database object.

2. Run the `ALTER PLUGGABLE DATABASE APPLICATION BEGIN` statement for beginning an application installation, upgrade, or patch.

   For example, if you are creating the application common object as part of an application installation, then run the `ALTER PLUGGABLE DATABASE APPLICATION BEGIN INSTALL` statement in the following form:

   ```
   ALTER PLUGGABLE DATABASE APPLICATION application_name
     BEGIN INSTALL 'application_version_number';
   ```

3. Create the application common object and specify its sharing attribute in one of the following ways:

   - Ensure that the `DEFAULT_SHARING` initialization parameter is set to the desired sharing attribute in the application root, and issue the `CREATE` SQL statement to create the database object.
   - Issue the `CREATE` SQL statement, and include the `SHARING` clause set to `METADATA`, `DATA`, or `EXTENDED DATA`.

   When a `SHARING` clause is included in a SQL statement, it takes precedence over the value specified in the `DEFAULT_SHARING` initialization parameter. For example, if the `DEFAULT_SHARING` initialization parameter is set to `METADATA` in the application root, and a database object is created with `SHARING` set to `DATA`, then the database object is created as a data-linked database object.

   > **Note:**
   >
   > Once a database object is created, its sharing attribute cannot be changed.

4. Run the `ALTER PLUGGABLE DATABASE APPLICATION END` statement for ending an application installation, upgrade, or patch.

   For example, if you are creating the application common object as part of an application installation, then run the `ALTER PLUGGABLE DATABASE APPLICATION END INSTALL` statement in the following form:

   ```
   ALTER PLUGGABLE DATABASE APPLICATION application_name
     END INSTALL 'application_version_number';
   ```

   > **Note:**
   >
   > Ensure that the *application_name* and *application_version_number* match in the `ALTER PLUGGABLE DATABASE APPLICATION BEGIN INSTALL` statement and the `ALTER PLUGGABLE DATABASE APPLICATION END INSTALL` statement.

5. Synchronize all of the application PDBs that must apply these changes by issuing an `ALTER PLUGGABLE DATABASE APPLICATION` statement with the `SYNC` clause with the application PDB as the current container.

Example 17-11 Setting the `DEFAULT_SHARING` Initialization Parameter

This example sets the `DEFAULT_SHARING` initialization parameter to `DATA` both in memory and in the SPFILE. When a database object that supports sharing is created in the application root, and no `SHARING` clause is included in the `CREATE` SQL statement, the database object uses the sharing attribute specified in the `DEFAULT_SHARING` initialization parameter.

```
ALTER SYSTEM SET DEFAULT_SHARING=DATA SCOPE=BOTH;
```

Example 17-12 Creating a Metadata-Linked Object

This example creates the `employees_md` metadata-linked table by including the `SHARING=METADATA` clause. The *application_name* is `salesapp` and the *application_version_number* is 4.2, and the object is created during application installation.

```
ALTER PLUGGABLE DATABASE APPLICATION salesapp BEGIN INSTALL '4.2';
CREATE TABLE employees_md SHARING=METADATA
  (employee_id    NUMBER(6),
   first_name     VARCHAR2(20),
   last_name      VARCHAR2(25) CONSTRAINT emp_last_name_nn_demo NOT
NULL,
   email          VARCHAR2(25) CONSTRAINT emp_email_nn_demo     NOT
NULL,
   phone_number   VARCHAR2(20),
   hire_date      DATE DEFAULT SYSDATE
                    CONSTRAINT emp_hire_date_nn_demo  NOT NULL,
   job_id         VARCHAR2(10) CONSTRAINT emp_job_nn_demo  NOT NULL,
   salary         NUMBER(8,2)  CONSTRAINT emp_salary_nn_demo  NOT NULL,
   commission_pct NUMBER(2,2),
   manager_id     NUMBER(6),
   department_id  NUMBER(4),
   dn             VARCHAR2(300),
   CONSTRAINT emp_salary_min_demo CHECK (salary > 0),
   CONSTRAINT emp_email_uk_demo UNIQUE (email));
ALTER PLUGGABLE DATABASE APPLICATION salesapp END INSTALL '4.2';
```

Example 17-13 Creating a Data-Linked Object

This example creates the `product_descriptions_ob` data-linked table by including the `SHARING=DATA` clause. The *application_name* is `salesapp` and the *application_version_number* is 4.2, and the object is created during application installation.

```
ALTER PLUGGABLE DATABASE APPLICATION salesapp BEGIN INSTALL '4.2';
CREATE TABLE product_descriptions_ob SHARING=DATA (
  product_id             NUMBER(6),
  language_id            VARCHAR2(3),
  translated_name        NVARCHAR2(50)
    CONSTRAINT translated_name_nn NOT NULL,
```

```
  translated_description NVARCHAR2(2000)
    CONSTRAINT translated_desc_nn NOT NULL);
ALTER PLUGGABLE DATABASE APPLICATION salesapp END INSTALL '4.2';
```

Example 17-14 Creating an Extended Data-Linked Object

This example creates the `postalcodes` extended data-linked table by including the `EXTENDED` keyword and the `SHARING` clause. The *application_name* is `salesapp` and the *application_version_number* is 4.2, and the object is created during application installation.

```
ALTER PLUGGABLE DATABASE APPLICATION salesapp BEGIN INSTALL '4.2';
CREATE TABLE postalcodes SHARING=EXTENDED DATA
    (code        VARCHAR2(7),
     country_id  NUMBER,
     place_name  VARCHAR2(20));
ALTER PLUGGABLE DATABASE APPLICATION salesapp END INSTALL '4.2';
```

Example 17-15 Creating an Object That Is Not Shared in an Application Root

This example creates the `departments_ns` table and specifies that it is not a shared common application object by including the `SHARING=NONE` clause. After creation, this database object can be accessed only in the application root.

```
CREATE TABLE departments_ns SHARING=NONE
    (department_id    NUMBER(4),
     department_name  VARCHAR2(30) CONSTRAINT  dept_name_nn  NOT NULL,
     manager_id       NUMBER(6),
     location_id      NUMBER(4),
     dn               VARCHAR2(300));
```

Note:

The `ALTER PLUGGABLE DATABASE APPLICATION BEGIN` and `END` statements are not required when you create an object that is not a shared common object. However, if you create an object that is not shared in between `ALTER PLUGGABLE DATABASE APPLICATION BEGIN` and `END` statements, then the object is created in application PDBs that synchronize with the application.

Related Topics

- Managing Applications in an Application Container
  You install, upgrade, or patch an application in an application container.
- Synchronizing Applications in an Application PDB
  Synchronizing an application updates the application in the application PDB to the latest version and patch in the application root.

Issuing DML Statements on Application Common Objects

The rules are different for issuing DML statements on metadata-linked, data-linked, and extended data-linked application common objects.

Related Topics

- Managing Applications in an Application Container
  You install, upgrade, or patch an application in an application container.
- Synchronizing Applications in an Application PDB
  Synchronizing an application updates the application in the application PDB to the latest version and patch in the application root.
- Synchronizing an Application Root Replica with a Proxy PDB
  When application containers in different CDBs have the same application, their application roots can be kept synchronized by creating a master application root, a replica application root, and a proxy PDB.

Issuing DML on Metadata-Linked Common Objects

You can issue DML on metadata-linked application objects as normal.

For metadata-linked application common objects, the object definitions are the same in all application PDBs, but the data is different. Users and applications can issue DML statements on these objects in the same way as for ordinary database objects. The DML only affects the current container.

Querying Using the CONTAINERS Clause

For metadata-linked objects, the `CONTAINERS` clause enables you to query a table or view across all PDBs in an application container.

For metadata-linked objects, the `CONTAINERS` clause is useful when DML is run in the application root. The query performs a `UNION ALL`, returning all rows from the object in the root and all open application PDBs (except those in `RESTRICTED` mode).

To query a subset of the PDBs, specify the `CON_ID` or `CON$NAME` in predicate. If the queried table or view does not already contain a `CON_ID` column, then the query adds a `CON_ID` column to the query result, which identifies the container whose data a given row represents.

Prerequisites

Note the following prerequisites:

- To query data in an application container, you must be a common user connected to the application root.
- The table or view must exist in the application root and all PDBs in the application container.
- The table or view must be in your own schema. It is not necessary to specify `schema`, but if you do, then you must specify your own schema.

To query a metadata-linked object in an application container:

1. Log in to the application root as an application common user.
2. Specify the `CONTAINERS` clause in a `SELECT` statement.

For example, the following statement counts the number of rows in the `sh.customers` table in the root and every application PDB (sample output included):

```
SELECT c.CON_ID, COUNT(*)
FROM   CONTAINERS(sh.customers) c
GROUP BY c.CON_ID
ORDER BY 1;

    CON_ID   COUNT(*)
---------- ----------
         3      20002
         6        426
         8       7232
```

Setting the Default Container or DML

You can set the `CONTAINERS_DEFAULT` attribute on any metadata-linked object so that DML issued in the application root is wrapped in the `CONTAINERS` clause by default.

Set `ENABLE CONTAINERS_DEFAULT` in either an `ALTER TABLE` or `ALTER VIEW` statement. The `CONTAINERS_DEFAULT` column in the `DBA_TABLES` and `DBA_VIEWS` views shows whether the database object is enabled for the `CONTAINERS` clause by default.

To set the default container for DML involving a metadata-linked table or view:

1. Log in to the application root as an application common user.
2. Issue an `ALTER TABLE` or `ALTER VIEW` statement with the `ENABLE CONTAINERS_DEFAULT` clause in the application root.

   The following statement sets the default container for `sh.customers`:

   ```
   ALTER TABLE sh.customers ENABLE CONTAINERS_DEFAULT;
   ```

   After setting this attribute, queries and DML statements issued in the application root use the `CONTAINERS` clause by default for `sh.customers`.

Issuing DML on Data-Linked Common Objects

For data-linked application objects, issue DML as normal in the application root. For extended data-linked application objects, issue DML as normal in the application root and in application PDBs.

For data-linked application objects, DML in the application root affects the data accessible by all PDBs in the application container. You cannot issue DML on data-linked application objects in application PDBs.

For extended data-linked application objects, DML in the application root affects the data accessible by all PDBs in the application container. DML in an application PDB only affects data that is unique to the application PDB.

Consider an application root that has data-linked or extended data-linked objects. Also, assume that this root is the master for application root replicas synchronized with proxy PDBs. In this case, DML only synchronizes with the replicas when DML occurs during an application installation, upgrade, or patch. Specifically, DML must

occur in the root between `ALTER PLUGGABLE DATABASE APPLICATION ... {BEGIN|END}` statements. Other DML applies only to the current root and is not synchronized with root replicas.

To issue DML for an application common object that is not part of an application root replica configuration:

1. Connect to the appropriate container in the application container as a user with the privileges required to issue DML statements on the database object.
2. Issue DML statements normally.

To issue DML for a data-linked or extended data-linked object that is part of an application root replica configuration:

1. In SQL*Plus, ensure that the current container is the master application root in the application root replica in the configuration.

   The current user must have the privileges required to issue the DML statements on the database object.
2. Run the `ALTER PLUGGABLE DATABASE APPLICATION ... BEGIN` statement for beginning an application installation, upgrade, or patch.

   If you are modifying the application common object as part of an application upgrade, then issue the upgrade statement in the following form:

   ```
   ALTER PLUGGABLE DATABASE APPLICATION application_name BEGIN UPGRADE
     'application_start_version_number' TO
     'application_end_version_number';
   ```

   For example, run the following statement if the *application_name* is `salesapp`, the *application_start_version_number* is 4.2, and the *application_end_version_number* is 4.3:

   ```
   ALTER PLUGGABLE DATABASE APPLICATION salesapp
     BEGIN UPGRADE '4.2' TO '4.3';
   ```
3. Issue the DML statements on the data-linked application common object.
4. Run the `ALTER PLUGGABLE DATABASE APPLICATION ... END` statement.

   For example, if you are modifying the application common object as part of an application upgrade, then run the statement in the following form:

   ```
   ALTER PLUGGABLE DATABASE APPLICATION application_name END UPGRADE
     TO 'application_end_version_number';
   ```

   For example, run the following statement if the *application_name* is `salesapp`, the *application_start_version_number* is 4.2, and the *application_end_version_number* is 4.3:

   ```
   ALTER PLUGGABLE DATABASE APPLICATION salesapp END UPGRADE TO '4.3';
   ```

> **Note:**
>
> Ensure that the *application_name* and *application_end_version_number* match in the `ALTER PLUGGABLE DATABASE APPLICATION BEGIN UPGRADE` statement and `ALTER PLUGGABLE DATABASE APPLICATION END UPGRADE` statements.

5. To synchronize all application PDBs that must apply these changes, issue an `ALTER PLUGGABLE DATABASE APPLICATION` statement with the `SYNC` clause when the application PDB is the current container.

Modifying Application Common Objects with DDL Statements

When you modify an application common object in an application root with certain DDL statements, you must modify the object between `ALTER PLUGGABLE DATABASE APPLICATION BEGIN` and `ALTER PLUGGABLE DATABASE APPLICATION END` statements, and application PDBs must synchronize with the application to apply the changes.

You can alter a metadata-linked object or a data-linked object in an application root. You run an `ALTER`, `RENAME`, or `DROP` SQL statement on the database object to perform a DDL change.

1. In SQL*Plus, ensure that the current container is the application root.

   The current user must have the privileges required to make the planned changes to the database object.

2. Run the `ALTER PLUGGABLE DATABASE APPLICATION BEGIN` statement for beginning an application installation, upgrade, or patch.

   For example, if you are modifying the application common object as part of an application upgrade, then run the `ALTER PLUGGABLE DATABASE APPLICATION BEGIN UPGRADE` statement in the following form:

   ```
   ALTER PLUGGABLE DATABASE APPLICATION application_name BEGIN UPGRADE
     'application_start_version_number' TO
   'application_end_version_number';
   ```

   For example, run the following statement if the *application_name* is `salesapp`, the *application_start_version_number* is 4.2, and the *application_end_version_number* is 4.3:

   ```
   ALTER PLUGGABLE DATABASE APPLICATION salesapp BEGIN UPGRADE
     '4.2' TO '4.3';
   ```

3. Modify the application common object with the DDL statement.

   For example, an `ALTER TABLE` statement might add a column to a table.

4. Run the `ALTER PLUGGABLE DATABASE APPLICATION END` statement for ending an application installation, upgrade, or patch.

For example, if you are modifying the application common object as part of an application upgrade, then run the `ALTER PLUGGABLE DATABASE APPLICATION END UPGRADE` statement in the following form:

```
ALTER PLUGGABLE DATABASE APPLICATION application_name END UPGRADE
  TO 'application_end_version_number';
```

For example, run the following statement if the *application_name* is `salesapp` and the *application_end_version_number* is 4.3:

```
ALTER PLUGGABLE DATABASE APPLICATION salesapp END UPGRADE TO '4.3';
```

> **Note:**
>
> Ensure that the *application_name* and *application_end_version_number* match in the `ALTER PLUGGABLE DATABASE APPLICATION BEGIN UPGRADE` statement and the `ALTER PLUGGABLE DATABASE APPLICATION END UPGRADE` statement.

5. Synchronize all of the application PDBs that must apply these changes by issuing an `ALTER PLUGGABLE DATABASE APPLICATION` statement with the `SYNC` clause with the application PDB as the current container.

Related Topics

- Managing Applications in an Application Container
  You install, upgrade, or patch an application in an application container.
- Synchronizing Applications in an Application PDB
  Synchronizing an application updates the application in the application PDB to the latest version and patch in the application root.

Issuing DML Statements on Containers in an Application Container

A DML statement issued in an application root can modify one or more containers in the application container. In addition, you can specify one or more default container targets for DML statements.

About Issuing DML Statements on Containers in an Application Container

DML statements can affect database objects in more than one container in an application container.

In an application root, a single DML statement that includes the `CONTAINERS` clause can modify a table or view in one or more containers in the application container. To use the `CONTAINERS` clause, specify the table or view being modified in the `CONTAINERS` clause and the containers in the `WHERE` clause. A target container can be specified in an `INSERT VALUES` statement by specifying a value for `CON_ID` in the `VALUES` clause. Also, a target container can be specified in an `UPDATE` or `DELETE` statement by specifying a `CON_ID` predicate in the `WHERE` clause.

For example, the following DML statement updates the `sales.customers` table in the containers with a `CON_ID` of `7` or `8`:

```
UPDATE CONTAINERS(sales.customers) ctab
   SET ctab.city_name='MIAMI'
   WHERE ctab.CON_ID IN(7,8) AND
   CUSTOMER_ID=3425;
```

The values specified for the `CON_ID` in the `WHERE` clause must be for containers in the current application container.

You can specify default target containers for DML operations. If a DML statement does not specify values for the `CON_ID` in the `WHERE` clause, then the target containers of the DML operation are those specified in the database property `CONTAINERS_DEFAULT_TARGET` in the application root. When issued in an application root, the following DML statement modifies the default target containers for the application container:

```
UPDATE CONTAINERS(sales.customers) ctab
   SET ctab.city_name='MIAMI'
   WHERE CUSTOMER_ID=3425;
```

By default, the default target containers in an application container include all of its application PDBs but not its application root or application seed. You can determine the default target containers for an application container by running the following query:

```
SELECT PROPERTY_VALUE
FROM   DATABASE_PROPERTIES
WHERE  PROPERTY_NAME='CONTAINERS_DEFAULT_TARGET';
```

In addition, you can enable the `CONTAINERS_DEFAULT` attribute for a table or view in an application root. When this attribute is enabled, the `CONTAINERS` clause is used for queries and DML statements on the database object by default, and the `CONTAINERS` clause does not need to be specified in the SQL statements. To enable the `CONTAINERS_DEFAULT` attribute for a table or view in an application root, run the an `ALTER TABLE` or `ALTER VIEW` statement with the `ENABLE CONTAINERS_DEFAULT` clause.

The following restrictions apply to the `CONTAINERS` clause:

- The `CONTAINERS DEFAULT TARGET` clause does not affect `SELECT` statements.
- `INSERT` as `SELECT` statements where the target of the `INSERT` is in `CONTAINERS()` is not supported.
- A multitable `INSERT` statement where the target of the `INSERT` is in `CONTAINERS()` is not supported.
- DML statements using the `CONTAINERS` clause require that the database listener is configured using TCP (instead of IPC) and that the `PORT` and `HOST` values are specified for each target PDB using the `PORT` and `HOST` clauses, respectively.

Related Topics

- About Application Common Objects
  Application common objects are created in an application root and are shared with the application PDBs that belong to the application root.

Specifying the Default Container for DML Statements in an Application Container

To specify the default container for DML statements in an application container, issue the `ALTER PLUGGABLE DATABASE` statement with the `CONTAINERS DEFAULT TARGET` clause.

When a DML statement is issued in an application root without specifying containers in the `WHERE` clause, the DML statement affects the default container for the application container. The default container can be any container in the application container, including the application root or an application PDB. Only one default container is allowed.

1. In SQL*Plus, ensure that the current container is the application root.

   The current user must have the commonly granted `ALTER PLUGGABLE DATABASE` privilege.

2. Run the `ALTER PLUGGABLE DATABASE` statement with the `CONTAINERS DEFAULT TARGET` clause.

Example 17-16 Specifying the Default Container for DML Statements in an Application Container

This example specifies that `APDB1` is the default container for DML statements in the application container.

```
ALTER PLUGGABLE DATABASE CONTAINERS DEFAULT TARGET = (APDB1);
```

Example 17-17 Clearing the Default Container

This example clears the default container setting. When it is not set, the default container is the application root.

```
ALTER PLUGGABLE DATABASE CONTAINERS DEFAULT TARGET = NONE;
```

Partitioning by PDB with Container Maps

Container maps enable the partitioning of data at the application PDB level when the data is not physically partitioned at the table level.

About Container Maps

A **container map** is a database property that specifies a partitioned map table defined in an application root.

Use a container map to partition the data in metadata-linked objects. Container maps partition data in application PDBs based on a commonly-used column.

For example, you might create a metadata-linked table named `countries_mlt` (with a column `cname`) that stores different data in each application PDB. The map table named `pdb_map_tbl` partitions by list on the `cname` column. The partitions `amer_pdb`, `euro_pdb`, and `asia_pdb` correspond to the names of the application PDBs.

A container map can define a logical partition key on a column for a common object. Because the container is resolved internally based on the container map, this mapping removes the requirement to define a query with a `CON_ID` predicate or use the `CONTAINERS` clause in the query.

Some types of row-based consolidation use a tenant ID with a single PDB that contains multiple tenants. Container maps are useful for migrating to a configuration that uses a different PDB for each tenant.

Map Objects

The **map object** is the partitioned table.

The names of the partitions in the map table match the names of the application PDBs in the application container. The metadata-linked object is not physically partitioned at the table level, but it can be queried using the partitioning strategy used by the container map.

To associate the map table with the metadata-linked table, specify the map table in `ALTER PLUGGABLE DATABASE ... CONTAINER_MAP` while connected to the application root. You can create no more than one container map in an application container. You cannot create container maps in the CDB root.

> **Note:**
>
> - Data must be loaded into the PDB tables in a manner that is consistent with the partitions defined in map object.
> - When there are changes to the application PDBs in an application container, the map object is not synchronized automatically to account for these changes. For example, an application PDB that is referenced in a map object can be unplugged, renamed, or dropped. The map object must be updated manually to account for such changes.

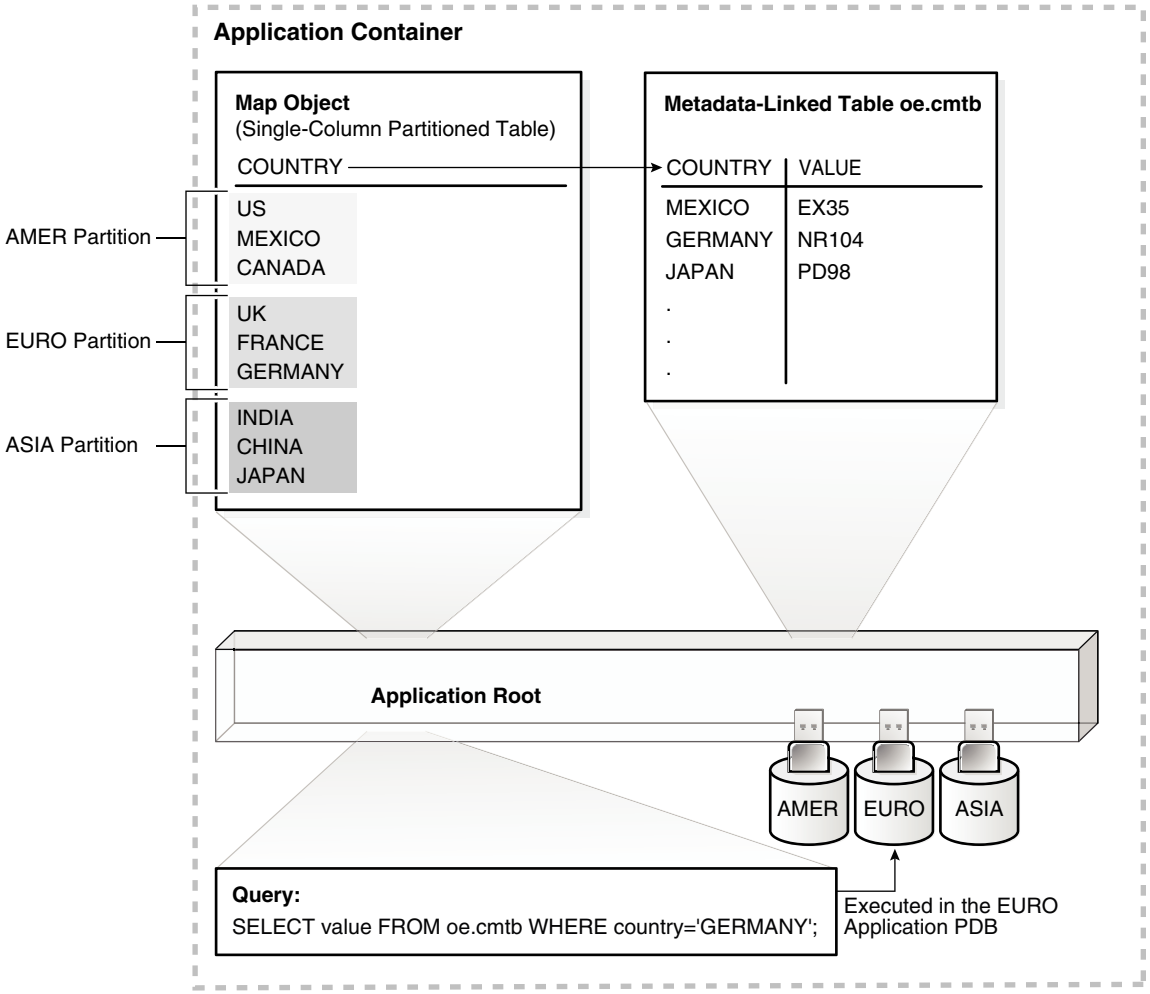
Starting in Oracle Database 18c, for a `CONTAINERS()` query to use a map, the partitioning column in the map table does not need to match a column in the metadata-linked table. Assume that the table `sh.sales` is enabled for the container map `pdb_map_tbl`, and `cname` is the partitioning column for the map table. Even though `sh.sales` does *not* include a `cname` column, the map table routes the following query to the appropriate PDB: `SELECT * FROM CONTAINERS(sh.sales) WHERE cname = 'US' ORDER BY time_id`.

List-Partitioned Container Map: Example

This example uses a container map to route queries to PDBs that store data for a geographical region.

The following illustration of an application root shows a map object, a metadata-linked table, and a query on the metadata-linked table. The query is executed in the appropriate application PDB.

Figure 17-3 Container Map



The illustration shows an application container with three application PDBs named `AMER`, `EURO`, and `ASIA`. The PDBs store data for the corresponding regions. A metadata-linked table named `oe.cmtb` stores information for an application. This table has a `COUNTRY` column. For this partitioning strategy, partition by list is used to create a map object that creates a partition for each region. The country value, which is `GERMANY` in the query shown in the illustration, determines the region, which is `EURO`.

> **See Also:**
>
> "Creating a Container Map" for a detailed description of this example

Range-Partitioned Container Map: Example

This example uses a container map to route queries to PDBs that store data for a particular department.

Consider another example that uses a range-partitioned table for the map object. The following SQL statement creates the map object in the application root:

```
CREATE TABLE app_con_admin.conmap (
   department_id NUMBER NOT NULL)
PARTITION BY RANGE (department_id) (
PARTITION apppdb1 VALUES LESS THAN (100),
PARTITION apppdb2 VALUES LESS THAN (200),
PARTITION apppdb3 VALUES LESS THAN (300));
```

This map object partitions data in the application PDBs `apppdb1`, `apppdb2`, and `apppdb3` based on the commonly-used column `department_id`. The following SQL statement sets the `CONTAINER_MAP` database property to the `app_con_admin.conmap` table in the application root:

```
ALTER PLUGGABLE DATABASE SET CONTAINER_MAP='app_con_admin.conmap';
```

Queries that use container maps produce similar results to queries that use the `CONTAINERS` clause. For example, the following queries return similar results:

```
SELECT employee_id
FROM   CONTAINERS(hr.employees)
WHERE  department_id = 10
AND    CON_ID IN (44);

SELECT employee_id
FROM   hr.employees
WHERE  department_id = 10;
```

As shown in the first query with the `CONTAINERS` clause, when the query only pertains to a single application PDB, the query must specify the container ID of this application PDB in the `WHERE` clause. This requirement might cause application changes.

The second query uses the container map, replacing the `CONTAINERS` clause. The second query does not specify the container because the container map directs the query to the correct application PDB. Queries that use container maps are generally more efficient than queries that use the `CONTAINERS` clause.

The container map must be created by a common user with `ALTER DATABASE` system privilege. Queries run against an object that is enabled for container map. Query privileges are determined by privileges granted on the object.

Creating a Container Map

Create a container map by creating a map object and setting the `CONTAINER_MAP` database property to the map object.

The map object is a partitioned table in which each partition name matches the name of an application PDB in an application container.

Prerequisites

To create a container map, you must meet the following prerequisites:

- Before creating a container map, an application container with application PDBs must exist in the CDB.
- The application container must have at least one application installed in it.

To create a container map:

1. In SQL*Plus, ensure that the current container is the application root.
2. Set the `CONTAINER_MAP` database property to the map object.

   In the following statement, replace *map_table_schema* with the owner of the table, and replace *map_table_name* with the name of the table:

   ```
   ALTER DATABASE SET CONTAINER_MAP =
   'map_table_schema.map_table_name';
   ```

3. Start an application installation, upgrade, or patch.
4. If the metadata-linked table that will be used by the container map does not exist, then create it.
5. Enable the container map for the table to be queried by issuing an `ALTER TABLE ... ENABLE CONTAINER_MAP` statement.
6. Ensure that the table to be queried is enabled for the `CONTAINERS` clause by issuing an `ALTER TABLE ... ENABLE CONTAINERS_DEFAULT` statement.
7. End the application installation, upgrade, or patch started previously.

Example 17-18 Creating and Using a Container Map

This example creates a simple application that uses a container map. Assume that an application container has three application PDBs named `AMER`, `EURO`, and `ASIA`. The application PDBs store data for the different regions (America, Europe, and Asia, respectively). A metadata-linked table stores information for an application and has a `COUNTRY` column. For this partitioning strategy, partition by list is used to create a map object that creates a partition for each region, and the country value is used to determine the region.

1. In SQL*Plus, ensure that the current container is the application root.
2. Create the map object.

   ```
   CREATE TABLE salesadm.conmap (country VARCHAR2(30) NOT NULL)
   PARTITION BY LIST (country) (
     PARTITION AMER VALUES ('US','MEXICO','CANADA'),
     PARTITION EURO VALUES ('UK','FRANCE','GERMANY'),
   ```

```
  PARTITION ASIA VALUES ('INDIA','CHINA','JAPAN')
);
```

3. Set the `CONTAINER_MAP` database property to the map object.

```
ALTER PLUGGABLE DATABASE SET CONTAINER_MAP='salesadm.conmap';
```

4. Begin an application installation.

```
ALTER PLUGGABLE DATABASE APPLICATION salesapp BEGIN INSTALL '1.0';
```

5. Create a metadata-linked table that will be queried using the container map.

```
CREATE TABLE oe.cmtb SHARING=METADATA (
  value   VARCHAR2(30),
  country VARCHAR2(30));
```

6. Enable the container map for the table to be queried.

```
ALTER TABLE oe.cmtb ENABLE CONTAINER_MAP;
```

7. Ensure that the table to be queried is enabled for the `CONTAINERS` clause.

```
ALTER TABLE oe.cmtb ENABLE CONTAINERS_DEFAULT;
```

8. End the application installation.

```
ALTER PLUGGABLE DATABASE APPLICATION salesapp END INSTALL '1.0';
```

9. Switch session into each application PDB and synchronize it.

```
ALTER SESSION SET CONTAINER=amer;
ALTER PLUGGABLE DATABASE APPLICATION salesapp SYNC;

ALTER SESSION SET CONTAINER=euro;
ALTER PLUGGABLE DATABASE APPLICATION salesapp SYNC;

ALTER SESSION SET CONTAINER=asia;
ALTER PLUGGABLE DATABASE APPLICATION salesapp SYNC;
```

10. Insert values into the `oe.cmtb` table in each application PDB based on the partitioning strategy.

```
ALTER SESSION SET CONTAINER=amer;
INSERT INTO oe.cmtb VALUES ('AMER VALUE','US');
INSERT INTO oe.cmtb VALUES ('AMER VALUE','MEXICO');
INSERT INTO oe.cmtb VALUES ('AMER VALUE','CANADA');
COMMIT;

ALTER SESSION SET CONTAINER=euro;
INSERT INTO oe.cmtb VALUES ('EURO VALUE','UK');
INSERT INTO oe.cmtb VALUES ('EURO VALUE','FRANCE');
INSERT INTO oe.cmtb VALUES ('EURO VALUE','GERMANY');
COMMIT;
```

```
ALTER SESSION SET CONTAINER=asia;
INSERT INTO oe.cmtb VALUES ('ASIA VALUE','INDIA');
INSERT INTO oe.cmtb VALUES ('ASIA VALUE','CHINA');
INSERT INTO oe.cmtb VALUES ('ASIA VALUE','JAPAN');
COMMIT;
```

11. Switch session into the application root and query the data using the container map.

```
ALTER SESSION SET CONTAINER=sales;

SELECT value FROM oe.cmtb WHERE country='MEXICO';

SELECT value FROM oe.cmtb WHERE country='GERMANY';

SELECT value FROM oe.cmtb WHERE country='JAPAN';
```

The output for the first query should be `AMER VALUE`, the output for the second query should be `EURO VALUE`, and the output for the third query should be `ASIA VALUE`. These values illustrate that the container map is working correctly.

18

Managing Security for a Multitenant Environment

You can manage common and local users and roles for a multitenant environment by using SQL*Plus and Oracle Enterprise Manager.

Managing Commonly and Locally Granted Privileges

In a multitenant environment, privileges can be granted commonly for an entire CDB or application container, or granted locally to a specific PDB.

How the Oracle Multitenant Option Affects Privileges

In a multitenant environment, all users, including common users, can exercise their privileges only within the current container.

However, a user connected to the root can perform certain operations that affect other pluggable databases (PDBs). These operations include `ALTER PLUGGABLE DATABASE`, `CREATE USER`, `CREATE ROLE`, and `ALTER USER`. The common user must possess the commonly granted privileges that enable these operations. A common user connected to the root can see metadata pertaining to PDBs by way of the container data objects (for example, multitenant container database (CDB) views and `V$` views) in the root, provided that the common user has been granted privileges required to access these views and his `CONTAINER_DATA` attribute has been set to allow seeing data about various PDBs. The common user cannot query tables or views in a PDB.

Common users cannot exercise their privileges across other PDBs. They must first switch to the PDB that they want, and then exercise their privileges from there. To switch to a different container, the common user must have the `SET CONTAINER` privilege. The `SET CONTAINER` privilege must be granted either commonly or in the container to which the user is attempting to switch. Alternatively, the common user can start a new database session whose initial current container is the container this user wants, relying on the `CREATE SESSION` privilege in that PDB.

Be aware that commonly granted privileges may interfere with the security configured for individual PDBs. For example, suppose an application PDB database administrator wants to prevent any user in the PDB from modifying a particular application common object. A privilege (such as `UPDATE`) granted commonly to `PUBLIC` or to a common user or common role on the object would circumvent the PDB database administrator's intent.

Related Topics

- *Oracle Database Security Guide*

About Commonly and Locally Granted Privileges

In a multitenant environment, both common users and local users can grant privileges to one another.

Privileges by themselves are neither common nor local. How the privileges are applied depends on whether the privilege is granted commonly or granted locally.

For commonly granted privileges:

- A privilege that is granted commonly can be used in every existing and future container.
- Only common users can grant privileges commonly, and only if the grantee is common.
- A common user can grant privileges to another common user or to a common role.
- The grantor must be connected to the root and must specify `CONTAINER=ALL` in the `GRANT` statement.
- Both system and object privileges can be commonly granted. (Object privileges become actual only with regard to the specified object.)
- When a common user connects to or switches to a given container, this user's ability to perform various activities (such as creating a table) is controlled by privileges granted commonly as well as privileges granted locally in the given container.
- Do not grant privileges to `PUBLIC` commonly.

For locally granted privileges:

- A privilege granted locally can be used only in the container in which it was granted. When the privilege is granted in the root, it applies only to the root.
- Both common users and local users can grant privileges locally.
- A common user and a local user can grant privileges to other common or local roles.
- The grantor must be connected to the container and must specify `CONTAINER=CURRENT` in the `GRANT` statement.
- Any user can grant a privilege locally to any other user or role (both common and local) or to the `PUBLIC` role.

Related Topics

- Overview of Privilege and Role Grants in a CDB
  Just as in a non-CDB, users in a CDB can grant and be granted roles and privileges. Roles and privileges in a CDB, however, are either locally or commonly granted.
- How the PUBLIC Role Works in a Multitenant Environment
  All privileges that Oracle grants to the `PUBLIC` role are granted locally.

How Commonly Granted System Privileges Work

Users can exercise system privileges only within the PDB in which they were granted.

For example, if a system privilege is locally granted to a common user A in a PDB B, user A can exercise that privilege only while connected to PDB B.

System privileges can apply in the root and in all existing and future PDBs if the following requirements are met:

- The system privilege grantor is a common user and the grantee is a common user, a common role, or the PUBLIC role. Do not commonly grant system privileges to the PUBLIC role, because this in effect makes the system privilege available to all users.
- The system privilege grantor possesses the ADMIN OPTION for the commonly granted privilege
- The GRANT statement must contain the CONTAINER=ALL clause.

The following example shows how to commonly grant a privilege to the common user c##hr_admin.

```
CONNECT SYSTEM
Enter password: password
Connected.

GRANT CREATE ANY TABLE TO c##hr_admin CONTAINER=ALL;
```

Related Topics

- *Oracle Database Security Guide*

How Commonly Granted Object Privileges Work

Object privileges on common objects applies to the object as well as all associated links on this common object.

These links include all metadata links, data links (previously called object links), or extended data links that are associated with it in the root and in all PDBs belonging to the container (including future PDBs) if certain requirements are met.

These requirements are as follows:

- The object privilege grantor is a common user and the grantee is a common user, a common role, or the PUBLIC role.
- The object privilege grantor possesses the commonly granted GRANT OPTION for the privilege
- The GRANT statement contains the CONTAINER=ALL clause.

The following example shows how to grant an object privilege to the common user c##hr_admin so that he can select from the DBA_PDBS view in the CDB root or in any of the associated PDBs that he can access.

```
CONNECT SYSTEM
Enter password: password
Connected.

GRANT SELECT ON DBA_OBJECTS TO c##hr_admin
CONTAINER=ALL;
```

Related Topics

- *Oracle Database Security Guide*
- Data Dictionary Architecture in a CDB
  From the user and application perspective, the data dictionary in each container in a CDB is separate, as it would be in a non-CDB.
- Namespaces in a CDB
  In a CDB, the namespace for every object is scoped to its container.
- How the PUBLIC Role Works in a Multitenant Environment
  All privileges that Oracle grants to the `PUBLIC` role are granted locally.

Granting or Revoking Privileges to Access a PDB

You can grant and revoke privileges for PDB access in a multitenant environment.

To grant a privilege in a multitenant environment:

- Include the `CONTAINER` clause in the `GRANT` or `REVOKE` statement.

Setting `CONTAINER` to `ALL` applies the privilege to all existing and future containers; setting it to `CURRENT` applies the privilege to the local container only. Omitting the `CONTAINER` clause applies the privilege to the local container. If you issue the `GRANT` statement from the root and omit the `CONTAINER` clause, then the privilege is applied locally.

Related Topics

- *Oracle Database SQL Language Reference*

Example: Granting a Privilege in a Multitenant Environment

You can use the `GRANT` statement to grant privileges in a multitenant environment.

Example 18-1 shows how to commonly grant the `CREATE TABLE` privilege to common user `c##hr_admin` so that this user can use this privilege in all existing and future containers.

Example 18-1 Granting a Privilege in a Multitenant Environment

```
CONNECT SYSTEM
Enter password: password
Connected.

GRANT CREATE TABLE TO c##hr_admin CONTAINER=ALL;
```

Enabling Common Users to View CONTAINER_DATA Object Information

Common users can view information about `CONTAINER_DATA` objects in the root or for data in specific PDBs.

Viewing Data About the Root, CDB, and PDBs While Connected to the Root

You can restrict view information for the X$ table and the V$, GV$ and CDB_* views when common users perform queries.

The X$ table and these views contain information about the application root and its associated application PDBs or, if you are connected to the CDB root, the entire CDB. Restricting this information is useful when you do not want to expose sensitive information about other PDBs. To enable this functionality, Oracle Database provides these tables and views as container data objects. You can find if a specific table or view is a container data object by querying the TABLE_NAME, VIEW_NAME, and CONTAINER_DATA columns of the USER_|DBA_|ALL_VIEWS|TABLES dictionary views.

To find information about the default (user-level) and object-specific CONTAINER_DATA attributes:

1. In SQL*Plus or SQL Developer, log in to the root.
2. Query the CDB_CONTAINER_DATA data dictionary view.

   For example:

```
COLUMN USERNAME FORMAT A13
COLUMN DEFAULT_ATTR FORMAT A7
COLUMN OWNER FORMAT A11
COLUMN OBJECT_NAME FORMAT A11
COLUMN ALL_CONTAINERS FORMAT A3
COLUMN CONTAINER_NAME FORMAT A10
COLUMN CON_ID FORMAT A6

SELECT USERNAME, DEFAULT_ATTR, OWNER, OBJECT_NAME,
       ALL_CONTAINERS, CONTAINER_NAME, CON_ID
FROM   CDB_CONTAINER_DATA
ORDER BY OBJECT_NAME;

USERNAME    DEFAULT OWNER OBJECT_NAME ALL CONTAINERS CON_ID
----------- ------- ----- ----------- --- ---------- ------
C##HR_ADMIN N       SYS   V$SESSION   N   CDB$ROOT        1
C##HR_ADMIN N       SYS   V$SESSION   N   SALESPDB        1
C##HR_ADMIN Y                         N   HRPDB           1
C##HR_ADMIN Y                         N   CDB$ROOT        1
DBSNMP      Y                         Y                   1
SYSTEM      Y                         Y                   1
```

Related Topics

- Container Data Objects in a CDB
  A **container data object** is a table or view containing data pertaining to multiple containers or the whole CDB.
- *Oracle Database Reference*

Enabling Common Users to Query Data in Specific PDBs

You can enable common users to access data pertaining to specific PDBs by adjusting the users' CONTAINER_DATA attribute.

To enable common users to access data about specific PDBs:

- Issue the ALTER USER statement in the root.

Example 18-2 Setting the CONTAINER_DATA Attribute

This example shows how to issue the ALTER USER statement to enable the common user c##hr_admin to view information pertaining to the CDB$ROOT, SALES_PDB, and HRPDB containers in the V$SESSION view (assuming this user can query that view).

```
CONNECT SYSTEM
Enter password: password
Connected.

ALTER USER c##hr_admin
SET CONTAINER_DATA = (CDB$ROOT, SALESPDB, HRPDB)
FOR V$SESSION CONTAINER=CURRENT;
```

In this specification:

- SET CONTAINER_DATA lists containers, data pertaining to which can be accessed by the user.
- FOR V$SESSION specifies the CONTAINER_DATA dynamic view, which common user c##hr_admin will query.
- CONTAINER = CURRENT must be specified because when you are connected to the root, CONTAINER=ALL is the default for the ALTER USER statement, but modification of the CONTAINER_DATA attribute must be restricted to the root.

If you want to enable user c##hr_admin to view information that pertains to the CDB$ROOT, SALES_PDB, HRPDB containers in all CONTAINER_DATA objects that this user can access, then omit FOR V$SESSION. For example:

```
ALTER USER c##hr_admin
SET CONTAINER_DATA = (CDB$ROOT, SALESPDB, HRPDB)
CONTAINER=CURRENT;
```

Related Topics

- *Oracle Database SQL Language Reference*

Managing Common Roles and Local Roles

A common role is a role that is created in the root; a local role is created in a PDB.

About Common Roles and Local Roles

In a multitenant environment, database roles can be specific to a PDB or used throughout the entire system container or application container.

A common role is a role whose identity and (optional) password are created in the root of a container and will be known in the root and in all existing and future PDBs belonging to that container.

A local role exists in only one PDB and can only be used within this PDB. It does not have any commonly granted privileges.

Note the following:

- Common users can both create and grant common roles to other common and local users.
- You can grant a role (local or common) to a local user or role only locally.
- If you grant a common role locally, then the privileges of that common role apply only in the container where the role is granted.
- Local users cannot create common roles, but they can grant them to common and other local users.
- The `CONTAINER = ALL` clause is the default when you create a common role in the CDB root or an application root.

Related Topics

- *Oracle Database Security Guide*

How Common Roles Work

Common roles are visible in the root and in every PDB of a container within which they are defined in a multitenant environment.

A privilege can be granted commonly to a common role if:

- The grantor is a common user.
- The grantor possesses the commonly granted `ADMIN OPTION` for the privilege that is being granted.
- The `GRANT` statement contains the `CONTAINER=ALL` clause.

If the common role contains locally granted privileges, then these privileges apply only within the PDB in which they were granted to the common role. A local role cannot be granted commonly.

For example, suppose the CDB common user `c##hr_mgr` has been commonly granted the `DBA` role. This means that user `c##hr_mgr` can use the privileges associated with the `DBA` role in the root and in every PDB in the multitenant environment. However, if the CDB common user `c##hr_mgr` has only been locally granted the `DBA` role for the `hr_pdb` PDB, then this user can only use the `DBA` role's privileges in the `hr_pdb` PDB.

How the PUBLIC Role Works in a Multitenant Environment

All privileges that Oracle grants to the `PUBLIC` role are granted locally.

This feature enables you to revoke privileges or roles that have been granted to the PUBLIC role individually in each PDB as needed. If you must grant any privileges to the PUBLIC role, then grant them locally. Never grant privileges to PUBLIC commonly.

Related Topics

- About Commonly and Locally Granted Privileges
  In a multitenant environment, both common users and local users can grant privileges to one another.

Privileges Required to Create, Modify, or Drop a Common Role

Only common users who have the commonly granted CREATE ROLE, ALTER ROLE, and DROP ROLE privileges can create, alter, or drop common roles.

Common users can also create local roles, but these roles are available only in the PDB in which they were created.

Rules for Creating Common Roles

When you create a common role, you must follow special rules.

The rules are as follows:

- **Ensure that you are in the correct root.** For the creation of common roles, you must be in the correct root, either the CDB root or the application root. You cannot create common roles from a PDB. To check if you are in the correct root, run one of the following:
  - To confirm that you are in the CDB root, you can issue the show_con_name command. The output should be CDB$ROOT.
  - To confirm that you are in an application root, verify that the following query returns YES:

    ```
    SELECT APPLICATION_ROOT FROM V$PDBS WHERE
    CON_ID=SYS_CONTEXT('USERENV', 'CON_ID');
    ```

  - **Ensure that the name that you give the common role starts with the value of the COMMON_USER_PREFIX parameter (which defaults to C##).** Note that this requirement does not apply to the names of existing Oracle-supplied roles, such as DBA or RESOURCE.
- **Optionally, set the CONTAINER clause to ALL.** As long as you are in the root, if you omit the CONTAINER = ALL clause, then by default the role is created as a common role for the CDB root or the application root.

Creating a Common Role

You can use the CREATE ROLE statement to create a common role.

1. Connect to the root of the CDB or the application container in which you want to create the common role.

For example:

```
CONNECT SYSTEM
Enter password: password
Connected.
```

2. Run the `CREATE ROLE` statement with the `CONTAINER` clause set to `ALL`.

   For example:

   ```
   CREATE ROLE c##sec_admin IDENTIFIED BY password CONTAINER=ALL;
   ```

Related Topics

- *Oracle Database Security Guide*

Rules for Creating Local Roles

To create a local role, you must follow special rules.

These rules are as follows:

- You must be connected to the PDB in which you want to create the role, and have the `CREATE ROLE` privilege.
- The name that you give the local role must not start with the value of the `COMMON_USER_PREFIX` parameter (which defaults to `C##`).
- You can include `CONTAINER=CURRENT` in the `CREATE ROLE` statement to specify the role as a local role. If you are connected to a PDB and omit this clause, then the `CONTAINER=CURRENT` clause is implied.
- You cannot have common roles and local roles with the same name. However, you can use the same name for local roles in different PDBs. To find the names of existing roles, query the `CDB_ROLES` and `DBA_ROLES` data dictionary views.

Creating a Local Role

You can use the `CREATE ROLE` statement to create a role.

1. Connect to the PDB in which you want to create the local role.

   For example:

   ```
   CONNECT SYSTEM@hrpdb
   Enter password: password
   Connected.
   ```

2. Run the `CREATE ROLE` statement with the `CONTAINER` clause set to `CURRENT`.

   For example:

   ```
   CREATE ROLE sec_admin CONTAINER=CURRENT;
   ```

Related Topics

- *Oracle Database Security Guide*

Role Grants and Revokes for Common Users and Local Users

Role grants and revokes apply only to the scope of access of the common user or the local user.

Common users can grant and revoke common roles to and from other common users. A local user can grant a common role to any user in a PDB, including common users, but this grant applies only within the PDB.

The following example shows how to grant the common user `c##sec_admin` the `AUDIT_ADMIN` common role for use in all containers.

```
CONNECT SYSTEM
Enter password: password
Connected.

GRANT AUDIT_ADMIN TO c##sec_admin CONTAINER=ALL;
```

Similarly, the next example shows how local user `aud_admin` can grant the common user `c##sec_admin` the `AUDIT_ADMIN` common role for use within the `hrpdb` PDB.

```
CONNECT aud_admin@hrpdb
Enter password: password
Connected.

GRANT AUDIT_ADMIN TO c##sec_admin CONTAINER=CURRENT;
```

This example shows how a local user `aud_admin` can revoke a role from another user in a PDB. If you omit the `CONTAINER` clause, then `CURRENT` is implied.

```
CONNECT aud_admin@hrpdb
Enter password: password
Connected.

REVOKE sec_admin FROM psmith CONTAINER=CURRENT;
```

Restricting Operations on PDBs Using PDB Lockdown Profiles

You can use PDB lockdown profiles in a multitenant environment to restrict sets of user operations in pluggable databases (PDBs).

About PDB Lockdown Profiles

A PDB lockdown profile is a named set of features that controls a group of operations.

In some cases, you can enable or disable operations individually. For example, a PDB lockdown profile can contain settings to disable specific clauses that come with the `ALTER SYSTEM` statement.

PDB lockdown profiles restrict user access to the functionality the features provided, similar to resource limits that are defined for users. As the name suggests, you use PDB lockdown profiles in a CDB, for an application container, or for a PDB or application PDB. You can create custom profiles to accommodate the requirements of your site. PDB profiles enable you to define custom security policies for an application. In addition, you can create a lockdown profile that is based on another profile, called a **base profile**. You can configure this profile to be dynamically updated when the base profile is modified, or configure it to be static (unchanging) when the base profile is updated. Lockdown profiles are designed for both Oracle Cloud and on-premises environments.

When identities are shared between PDBs, elevated privileges may exist. You can use lockdown profiles to prevent this elevation of privileges. Identities can be shared in the following situations:

- At the operating system level, when the database interacts with operating system resources such as files or processes
- At the network level, when the database communicates with other systems, and network identity is important
- Inside the database, as PDBs access or create common objects or they communicate across container boundaries using features such as database links

The features that use shared identifies and that benefit from PDB lockdown profiles are in the following categories:

- **Network access features.** These are operations that use the network to communicate outside the PDB. For example, the PL/SQL packages `UTL_TCP`, `UTL_HTTP`, `UTL_MAIL`, `UTL_SNMP`, `UTL_INADDR`, and `DBMS_DEBUG_JDWP` perform these kinds of operations. Currently, ACLs are used to control this kind of access to share network identity.
- **Common user or object access.** These are operations in which a local user in the PDB can proxy through common user accounts or access objects in a common schema. These kinds of operations include adding or replacing objects in a common schema, granting privileges to common objects, accessing common directory objects, granting the `INHERIT PRIVILEGES` role to a common user, and manipulating a user proxy to a common user.
- **Operating System access.** For example, you can restrict access to the `UTL_FILE` or `DBMS_FILE_TRANSFER` PL/SQL packages.
- **Connections.** For example, you can restrict common users from connecting to the PDB or you can restrict a local user who has the `SYSOPER` administrative privilege from connecting to a PDB that is open in restricted mode.

The general procedure for creating a PDB lockdown profile is to first create it in the CDB root or the application root using the `CREATE LOCKDOWN PROFILE` statement, and then use the `ALTER LOCKDOWN PROFILE` statement to add the restrictions.

To enable a PDB lockdown profile, you can use the `ALTER SYSTEM` statement to set the `PDB_LOCKDOWN` parameter. You can find information about existing PDB lockdown profiles by connecting to CDB or application root and querying the `DBA_LOCKDOWN_PROFILES` data dictionary view. A local user can find the contents of a PDB lockdown parameter by querying the `V$LOCKDOWN_RULES` dynamic data dictionary view.

Default PDB Lockdown Profiles

Oracle Database provides a set of default PDB lockdown profiles that you can customize for your site requirements.

By default, most of these profiles are empty. They are designed to be a placeholder or template for you to configure, depending on your deployment requirements.

Detailed information about these profiles is as follows:

- `PRIVATE_DBAAS` incorporates restrictions that are suitable for private Cloud Database-as-a-Service (DBaaS) deployments. These restrictions are:
  - Must have the same database administrator for each PDB
  - Different users permitted to connect to the database
  - Different applications permitted

  `PRIVATE_DBAAS` permits users to connect to the PDBs but prevents them from using Oracle Database administrative features.
- `SAAS` incorporates restrictions that are suitable for Software-as-a-Service (SaaS) deployments. These restrictions are:
  - Must have the same database administrator for each PDB
  - Different users permitted to connect to the database
  - Must use the same application

  The `SAAS` lockdown profile is more restrictive than the `PRIVATE_DBAAS` profile. Users can be different, but the application code is the same; users are prevented from directly connecting and must connect only through the application; and users are not granted the ability to perform any administrative features.
- `PUBLIC_DBAAS` incorporates restrictions that are suitable for public Cloud Database-as-a-Service (DBaaS) deployments. The restrictions are as follows:
  - Different DBAs in each PDB
  - Different users
  - Different applications

  The `PUBLIC_DBAAS` lockdown profile is the most restrictive of the lockdown profiles.

Creating a PDB Lockdown Profile

To create a PDB lockdown profile, you must have the `CREATE LOCKDOWN PROFILE` system privilege.

After you create the lockdown profile, you can add restrictions before enabling it.

1. Connect to the CDB root or the application root as a user who has the `CREATE LOCKDOWN PROFILE` system privilege.

   For example, to connect to the CDB root:

   ```
   CONNECT c##sec_admin
   Enter password: password
   ```

2. Run the `CREATE LOCKDOWN PROFILE` statement to create the profile by using the following syntax:

```
CREATE LOCKDOWN PROFILE profile_name
[FROM static_base_profile | INCLUDING dynamic_base_profile];
```

In this specification:

- *profile_name* is the name that you assign the lockdown profile. You can find existing names by querying the `PROFILE_NAMES` column of the `DBA_LOCKDOWN_PROFILES` data dictionary view.
- `FROM` *static_base_profile* creates a new lockdown profile by using the values from an existing profile. Any subsequent changes to the base profile will not affect the new profile.
- `INCLUDING` *dynamic_base_profile* also creates a new lockdown profile by using the values from an existing base profile, except that this new lockdown profile will inherit the `DISABLE STATEMENT` rules that comprise the base profile, as well as any subsequent changes to the base profile. If rules that are explicitly added to the new profile conflict with the rules in the base profile, then the rules in the base profile take precedence. For example, an `OPTION_VALUE` clause in the base profile takes precedence over the `OPTION_VALUE` clause in the new profile.

The following two PDB lockdown profile statements demonstrate how the inheritance works:

```
CREATE LOCKDOWN PROFILE hr_prof INCLUDING PRIVATE_DBAAS;
CREATE LOCKDOWN PROFILE hr_prof2 FROM hr_prof;
```

In the first statement, `hr_prof` inherits any changes made to the `PRIVATE_DBAAS` base profile. If a new statement is enabled for `PRIVATE_DBAAS`, then it is enabled for `hr_prof`. In the second statement, in contrast, when `hr_prof` changes, then `hr_prof2` does *not* change because it is independent of its base profile.

3. Run the `ALTER LOCKDOWN PROFILE` statement to provide restrictions for the profile.

For example:

```
ALTER LOCKDOWN PROFILE hr_prof DISABLE STATEMENT  = ('ALTER
SYSTEM');
ALTER LOCKDOWN PROFILE hr_prof ENABLE STATEMENT = ('ALTER SYSTEM')
clause = ('flush shared_pool');
ALTER LOCKDOWN PROFILE hr_prof DISABLE FEATURE = ('XDB_PROTOCOLS');
```

In the preceding example:

- `DISABLE STATEMENT = ('ALTER SYSTEM')` disables the use of all `ALTER SYSTEM` statements for the PDB.
- `ENABLE STATEMENT = ('ALTER SYSTEM') clause = ('flush shared_pool')` enables only the use of the `FLUSH_SHARED_POOL` clause for `ALTER SYSTEM`.
- `DISABLE FEATURE = ('XDB_PROTOCOLS')` prohibits the use of the XDB protocols (FTP, HTTP, HTTPS) by this PDB

After you create a PDB lockdown profile, you are ready to enable it by using the `ALTER SYSTEM SET PDB_LOCKDOWN` SQL statement.

Related Topics

- *Oracle Database SQL Language Reference*

Enabling or Disabling a PDB Lockdown Profile

To enable or disable a PDB lockdown profile, use the `PDB_LOCKDOWN` initialization parameter

You can use `ALTER SYSTEM SET PDB_LOCKDOWN` to enable a lockdown profile in any of the following contexts:

- CDB (affects all PDBs)
- Application root (affects all application PDBs in the container)
- Application PDB
- PDB

> **Note:**
>
> It is not necessary to restart the instance to enable the profile. When the `ALTER SYSTEM SET PDB_LOCKDOWN` statement completes, the profile rules take effect immediately.

When you set `PDB_LOCKDOWN` in the CDB root, every PDB and application root inherits this setting unless `PDB_LOCKDOWN` is set at the container level. To disable lockdown profiles, set `PDB_LOCKDOWN` to null. If you set this parameter to null in the CDB root, then lockdown profiles are disabled for all PDBs except those that explicitly set a profile within the PDB.

A CDB common user who has been commonly granted the `SYSDBA` administrative privilege or the `ALTER SYSTEM` system privilege can set `PDB_LOCKDOWN` only to a lockdown profile that was created in the CDB root. An application common user with the application common `SYSDBA` administrative privilege or the `ALTER SYSTEM` system privilege can set `PDB_LOCKDOWN` only to a lockdown profile created in an application root.

1. Log in to the desired container as a user who has the commonly granted `ALTER SYSTEM` or commonly granted `SYSDBA` privilege.

   For example, to enable the profile for all PDBs, log in to the CDB root:

   ```
   CONNECT c##sec_admin
   Enter password: password
   ```

2. Run the `ALTER SYSTEM SET PDB_LOCKDOWN` statement.

For example, the following statement enables the lockdown profile named `hr_prof` for all PDBs:

```
ALTER SYSTEM SET PDB_LOCKDOWN = hr_prof;
```

The following statement resets the `PDB_LOCKDOWN` parameter:

```
ALTER SYSTEM RESET PDB_LOCKDOWN;
```

This variation of the preceding statement includes the `SCOPE` clause::

```
ALTER SYSTEM RESET PDB_LOCKDOWN SCOPE = BOTH;
```

The following statement disables all lockdown profiles in the CDB except those that are explicitly set at the PDB level:

```
ALTER SYSTEM SET PDB_LOCKDOWN = '' SCOPE = BOTH;
```

To find the names of PDB lockdown profiles, query the `PROFILE_NAME` column of the `DBA_LOCKDOWN_PROFILES` data dictionary view.

3. Optionally, review information about the profiles by querying `DBA_LOCKDOWN_PROFILES`.

   For example, run the following query:

```
SET LINESIZE 150
COL PROFILE_NAME FORMAT a20
COL RULE FORMAT a20
COL CLAUSE FORMAT a25

SELECT PROFILE_NAME, RULE, CLAUSE, STATUS FROM
CDB_LOCKDOWN_PROFILES;
```

   Sample output appears below:

```
PROFILE_NAME      RULE                 CLAUSE
STATUS
----------------- -------------------- -------------------------
-------
HR_PROF           XDB_PROTOCOLS
DISABLE
HR_PROF           ALTER SYSTEM
DISABLE
HR_PROF           ALTER SYSTEM         FLUSH SHARED_POOL
ENABLE
HR_PROF2
EMPTY
PRIVATE_DBAAS
EMPTY
PUBLIC_DBAAS
EMPTY
```

```
SAAS
EMPTY
```

Dropping a PDB Lockdown Profile

To drop a PDB lockdown profile, you must have the `DROP LOCKDOWN PROFILE` system privilege and be logged into the CDB or application root.

You can find the names of existing PDB lockdown profiles by querying the `DBA_LOCKDOWN_PROFILES` data dictionary view.

1. Connect to the CDB root or the application root as a user who has the `DROP LOCKDOWN PROFILE` system privilege.

   For example, to connect to the CDB root:

   ```
   CONNECT c##sec_admin
   Enter password: password
   ```

2. Run the `DROP LOCKDOWN_PROFILE` statement.

   For example:

   ```
   DROP LOCKDOWN PROFILE hr_prof2;
   ```

3. Optionally, review the current list of profiles by querying `DBA_LOCKDOWN_PROFILES`.

   For example, run the following query:

   ```
   SET LINESIZE 150
   COL PROFILE_NAME FORMAT a20
   COL RULE FORMAT a20
   COL CLAUSE FORMAT a25

   SELECT PROFILE_NAME, RULE, CLAUSE, STATUS FROM
   CDB_LOCKDOWN_PROFILES;
   ```

   Sample output appears below:

   ```
   PROFILE_NAME      RULE                 CLAUSE
   STATUS
   ----------------- -------------------- -------------------------
   -------
   HR_PROF           XDB_PROTOCOLS
   DISABLE
   HR_PROF           ALTER SYSTEM
   DISABLE
   HR_PROF           ALTER SYSTEM         FLUSH SHARED_POOL
   ENABLE
   PRIVATE_DBAAS
   EMPTY
   PUBLIC_DBAAS
   EMPTY
   SAAS
   EMPTY
   ```

Configuring Operating System Users for a PDB

The DBMS_CREDENTIAL.CREATE_CREDENTIAL procedure configures user accounts to be operating system users for a pluggable database (PDB).

About Configuring Operating System Users for a PDB

Instead the oracle operating system user, you can set a specific user account to be the operating system user for that PDB.

If you do not set a specific user to be the operating system user for the PDB, then by default the PDB uses the oracle operating system user. For the root, you can use the oracle operating system user when you must interact with the operating system.

For better security, Oracle recommends that you set a unique operating system user for each PDB in a multitenant environment. Doing so helps to ensure that operating system interactions are performed as a less powerful user than the oracle operating system user, and helps to protect data that belongs to one PDB from being accessed by users who are connected to other PDBs.

Configuring an Operating System User for a PDB

The DBMS_CREDENTIAL.CREATE_CREDENTIAL procedure can set an operating system user for a PDB.

1. Log in to the database instance root as a user who has the EXECUTE privilege for the DBMS_CREDENTIAL PL/SQL package and the ALTER SYSTEM system privilege.

   For example:

   ```
   sqlplus c##sec_admin
   Enter password: password
   ```

2. Run the DBMS_CREDENTIAL.CREATE_CREDENTIAL procedure to create an Oracle credential for the operating system user.

   For example, to set the credential for a user named os_admin:

   ```
   BEGIN
    DBMS_CREDENTIAL.CREATE_CREDENTIAL (
       credential_name => 'PDB1_OS_USER',
       username        => 'os_admin',
       password        => 'password');
   END;
   /
   ```

3. Connect to the PDB for which the operating system user will be used.

   For example:

   ```
   CONNECT cc##sec_admin@hrpdb
   Enter password: password
   ```

To find the available PDBs, run the `show pdbs` command. To check the current PDB, run the show `con_name` command.

4. Set the `PDB_OS_CREDENTIAL` initialization parameter for the user whose credential was set in Step 2.

   For example:

   ```
   ALTER SYSTEM SET PDB_OS_CREDENTIAL = PDB1_OS_USER SCOPE = SPFILE;
   ```

   The `PDB_OS_CREDENTIAL` parameter is a static parameter, so you must set it using the `SCOPE = SPFILE` clause.

5. Restart the database instance.

   ```
   SHUTDOWN IMMEDIATE
   STARTUP
   ```

Related Topics

- *Oracle Database Security Guide*
- *Oracle Database Reference*
- *Oracle Database PL/SQL Packages and Types Reference*

Related Topics

- Setting the Default Credential in a PDB
  You can set the database property `DEFAULT_CREDENTIAL` for a specified PDB.

Setting the Default Credential in a PDB

You can set the database property `DEFAULT_CREDENTIAL` for a specified PDB.

A default credential is useful when importing files from an object store into a PDB. If you do not specify a credential name when using `impdp`, then Oracle Data Pump and the object store module can use the `DEFAULT_CREDENTIAL` object to retrieve the user name and password. When running `impdp` without specifying a credential, you must prefix the dump file name with `DEFAULT_CREDENTIAL:`.

To set the default credential:

1. Log in to a PDB with administrator privileges.
2. Use an `ALTER DATABASE` statement to set the default credential.

   For example, enter the following statement to set the credential to `SYSTEM.HR_CRED`.

   ```
   ALTER DATABASE PROPERTY SET DEFAULT_CREDENTIAL = 'SYSTEM.HR_CRED';
   ```

Example 18-3 Importing Data into a PDB Using the Default Credential

This example assumes that a default credential exists. The following command imports data from an object store , prefacing the URL with the string `DEFAULT_CREDENTIAL`:

```
impdp hr@pdb1 table_exists_action=replace \
  dumpfile=DEFAULT_CREDENTIAL:https://example.com/ostore/obucket/myt.dmp
```

See Also:

- "Using Data Pump to Move Data Into a CDB"
- *Oracle Database Utilities* to learn about Data Pump Import

Using Application Contexts in a Multitenant Environment

An application context stores user identification that can enable or prevent a user from accessing data in the database.

What Is an Application Context?

An **application context** is a set of name-value pairs that Oracle Database stores in memory.

The context has a label called a **namespace** (for example, `empno_ctx` for an application context that retrieves employee IDs). This context enables Oracle Database to find information about both database and nondatabase users during authentication.

Inside the context are the name-value pairs (an associative array): the name points to a location in memory that holds the value. An application can use the application context to access session information about a user, such as the user ID or other user-specific information, or a client ID, and then securely pass this data to the database.

You can then use this information to either permit or prevent the user from accessing data through the application. You can use application contexts to authenticate both database and non-database users.

Related Topics

- *Oracle Database Security Guide*

Application Contexts in a Multitenant Environment

Where you create an application in a multitenant environment determines where you must create the application context.

If an application is installed in the application root or CDB root, then it becomes accessible across the application container or system container and associated application PDBs. You will need to create a common application context in this root.

When you create a common application context for use with an application container, note the following:

- You can create application contexts in a multitenant environment by setting the `CONTAINER` clause in the `CREATE CONTEXT` SQL statement. For example, to create a common application context in the application root, you must execute `CREATE CONTEXT` with `CONTAINER` set to `ALL`. To create the application context in a PDB, set `CONTAINER` to `CURRENT`.
- You cannot use the same name for a local application context for a common application context. You can find the names of existing application contexts by running the following query:

  ```
  SELECT OBJECT_NAME FROM DBA_OBJECTS WHERE OBJECT_TYPE ='CONTEXT';
  ```

- The PL/SQL package that you create to manage a common application context must be a common PL/SQL package. That is, it must exist in the application root or CDB root. If you create the application context for a specific PDB, then you must store the associated PL/SQL package in that PDB.
- The name-value pairs that you set under a common session application context from an application container or a system container for a common application context are not accessible from other application containers or system containers when a common user accesses a different container.
- The name-value pairs that you set under a common global application context from an application container or a system container, are accessible only within the same container in the same user session.
- An application can retrieve the value of an application context whether it resides in the application root, the CDB root, or a PDB.
- During a plug-in operation of a PDB into a CDB or an application container, if the name of the common application context conflicts with a PDB's local application context, then the PDB must open in restricted mode. A database administrator would then need to correct the conflict before opening the PDB in normal mode.
- During an unplug operation, a common application context retains its common semantics, so that later on, if the PDB is plugged into another CDB where a common application context with the same name exists, it would continue to behave like a common object. If a PDB is plugged into an application container or a system container, where the same common application context does not exist, then it behaves like a local object.

To find if an application context is a local application context or an application common application context, query the `SCOPE` column of the `DBA_CONTEXT` or `ALL_CONTEXT` data dictionary view.

Related Topics

- *Oracle Database Security Guide*

Using Oracle Virtual Private Database in a Multitenant Environment

Oracle Virtual Private Database (VPD) enables you to filter users who access data.

What Is Oracle Virtual Private Database?

Oracle Virtual Private Database (VPD) creates security policies to control database access at the row and column level.

> **Note:**
>
> Oracle Database release 12c introduced Real Application Security (RAS) to supersede VPD. Oracle recommends that you use RAS for new projects that require row and column level access controls for their applications.

Essentially, Oracle Virtual Private Database adds a dynamic `WHERE` clause to a SQL statement that is issued against the table, view, or synonym to which an Oracle Virtual Private Database security policy was applied.

Oracle Virtual Private Database enforces security, to a fine level of granularity, directly on database tables, views, or synonyms. Because you attach security policies directly to these database objects, and the policies are automatically applied whenever a user accesses data, there is no way to bypass security.

When a user directly or indirectly accesses a table, view, or synonym that is protected with an Oracle Virtual Private Database policy, Oracle Database dynamically modifies the SQL statement of the user. This modification creates a `WHERE` condition (called a predicate) returned by a function implementing the security policy. Oracle Database modifies the statement dynamically, transparently to the user, using any condition that can be expressed in or returned by a function. You can apply Oracle Virtual Private Database policies to `SELECT`, `INSERT`, `UPDATE`, `INDEX`, and `DELETE` statements.

For example, suppose a user performs the following query:

```
SELECT * FROM OE.ORDERS;
```

The Oracle Virtual Private Database policy dynamically appends the statement with a `WHERE` clause. For example:

```
SELECT * FROM OE.ORDERS
 WHERE SALES_REP_ID = 159;
```

In this example, the user can only view orders by Sales Representative 159.

If you want to filter the user based on the session information of that user, such as the ID of the user, then you can create the `WHERE` clause to use an application context. For example:

```
SELECT * FROM OE.ORDERS
 WHERE SALES_REP_ID = SYS_CONTEXT('USERENV','SESSION_USER');
```

> **Note:**
>
> Oracle Virtual Private Database does not support filtering for DDLs, such as `TRUNCATE` or `ALTER TABLE` statements.

Related Topics

- *Oracle Database Real Application Security Administrator's and Developer's Guide*
- *Oracle Database Security Guide*

Oracle Virtual Private Database in a Multitenant Environment

You can create Virtual Private Database policies in an application root for use throughout any associated application PDBs.

The CDB restriction applies to shared context sensitive policies and views related to Virtual Private Database policies as well. You cannot create a Virtual Private Database policy for an entire multitenant environment.

With regard to application containers, you can create Virtual Private Database policies to protect application common objects by applying the common policy to all PDBs that belong to the application root. In other words, when you install an application in the application root, all the common Virtual Private Database policies that protect the common objects will be applied to and immediately enforced for all PDBs in the application container.

Note the following:

- You can only create the common Virtual Private Database policy and its associated PL/SQL function in the application root and only attach it to application common objects. If the function is not in the same location as the policy, then an error is raised at runtime.
- A Virtual Private Database policy that is applied to common objects is considered a common policy that will be automatically enforced in PDBs that belong to the application container when it accesses the application common objects from application PDBs.
- Application common Virtual Private Database policies can only protect application common objects.
- A Virtual Private Database policy that is applied to application common objects in the application root and is applied to all application PDBs is considered a common Virtual Private Database policy. A policy that is applied to a local database table and enforced in one PDB is considered a local Virtual Private Database policy.

  For example, if policy `VPD_P1` is applied to the application common table `T1` in the application root, then it is a considered to be a common policy. It will be enforced in each application PDB. If a policy named `VPD_P1` is applied to a local table called `T1` in `PDB1`, then it is considered a local policy, which means that it affects only `PDB1`. If a policy called `VPD_P1` is applied to a local table `T1` in the application root, then it is still considered a local policy because it affects only the application root. This concept applies to other operations, such as enabling, disabling, and removing Virtual Private Database policies.

- Application common Virtual Private Database policies only protect application common objects, while local Virtual Private Database policies only protect local objects.
- If you are using application contexts, then ensure common database session-based application contexts and common global application context objects are used in the common Virtual Private Database configuration.
- Application container Virtual Private Database policies are stored in the application root. PDBs store only local policies. If you plug a PDB into the application container, then the common policies are not converted to local policies. Instead, Oracle Database loads them from the application root and enforces them in the local PDB when the policies access common objects in the local PDB.

Related Topics

- *Oracle Database Security Guide*

Using Transport Layer Security in a Multitenant Environment

Transport Layer Security (TLS) can be used in a multitenant environment for application containers.

If you want to use Transport Layer Security (TLS) in a multitenant environment for an application container, then you must ensure that each PDB is able to use its own wallet with its own certificates for TLS authentication.

- Because there is no individual `sqlnet.ora` file for each PDB, place the wallet in a subdirectory of the `wallet` directory where the name of the subdirectory is the GUID of the PDB that uses the wallet.

  For example, suppose the `WALLET_LOCATION` parameter in `sqlnet.ora` is set as follows:

  ```
  (SOURCE=(METHOD=FILE)(METHOD_DATA=
     (DIRECTORY=/home/oracle/wallet)))
  ```

  Place each PDB's wallet in the `/home/oracle/wallet/PDB_GUID` directory. You can find the existing PDBs and their GUIDs by querying the `DBA_PDBS` data dictionary view.

  If the `WALLET_LOCATION` parameter is not specified, then you must place the PDB wallet in a leaf subdirectory of the default wallet path where the name of the subdirectory is the GUID of the PDB, and the name of the leaf subdirectory is `wallet`. For example:

  ```
  $ORACLE_BASE/admin/db_unique_name/PDB_GUID/wallet
  ```

  Or if the `ORACLE_BASE` environment variable is not set, then you can use the Oracle home:

  ```
  $ORACLE_HOME/admin/db_unique_name/PDB_GUID/wallet
  ```

  These default locations correspond to the default that is used by Oracle Enterprise User Security to locate wallets for authentication to LDAP.

Related Topics

- *Oracle Database Security Guide*

Oracle Data Redaction in a Multitenant Environment

In a multitenant environment, Oracle Data Redaction policies apply only to the objects within the current pluggable database (PDB).

You cannot create a Data Redaction policy for a multitenant container database (CDB). This is because the objects for which you create Data Redaction policies typically reside in a PDB. If you have the `SYSDBA` privilege, then you can list all the PDBs in a CDB by running the `SHOW PDBS` command.

As with the CDB root, you cannot create Data Redaction policies in an application root.

Overview of Auditing in a Multitenant Environment

Auditing is the monitoring and recording of configured database actions, from both database users and nondatabase users. Nondatabase users are application users who are recognized in the database using the `CLIENT_IDENTIFIER` attribute.

You can base auditing on individual actions, such as the type of SQL statement executed, or on combinations of data that can include the user name, application, time, and so on. You can configure auditing for both successful and failed activities, and include or exclude specific users from the audit. In a multitenant environment, you can audit individual actions of the pluggable database (PDB) or individual actions in the entire multitenant container database (CDB).

Auditing is enabled by default. Audit records are written to the unified audit trail in a uniform format and are made available through the `UNIFIED_AUDIT_TRAIL` view.

Unified Auditing in a Multitenant Environment

You can use unified auditing in a multitenant environment.

You can apply audit settings to individual PDBs or to the CDB, depending on the type of policy. In a multitenant environment, each PDB, including the root, has its own unified audit trail.

Audit settings in a multitenant environment affect the following areas:

- **Unified audit policies created with the CREATE AUDIT POLICY and AUDIT statements:** You can create policies for both the root and individual PDBs.
- **Audit records written to the syslog:** On UNIX platforms, you can set the `UNIFIED_AUDIT_COMMON_SYSTEMLOG` initialization parameter in the CDB root to enable certain unified audit trail columns to be written to SYSLOG. On both Windows and UNIX, you can set the `UNIFIED_AUDIT_SYSTEMLOG` parameter in both the root and PDB level.
- **Fine-grained audit policies:** You can create policies for individual PDBs only, not the root.
- **Purging the audit trail:** You can perform purge operations for both the root and individual PDBs.

Related Topics

- *Oracle Database Security Guide*

Example: Auditing the DBA Role in a Multitenant Environment

The `CREATE AUDIT POLICY` statement can audit roles in a multitenant environment.

The following example shows how to audit a predefined common role `DBA` in a multitenant environment.

Example 18-4 Auditing the DBA Role in a Multitenant Environment

```
CREATE AUDIT POLICY role_dba_audit_pol
 ROLES DBA
 CONTAINER = ALL;

AUDIT POLICY role_dba_audit_pol;
```

Unified Audit Policies or AUDIT Settings in a Multitenant Environment

In a multitenant environment, you can create unified audit policies for individual PDBs and in the root.

About Local, CDB Common, and Application Common Audit Policies

An audit policy can be either a local audit policy, a CDB common audit policy, or an application common audit policy.

This applies to both unified audit policies and policies that are created using the `AUDIT` SQL statement.

- **Local audit policy.** This type of policy can exist in either the root (CDB or application) or the PDB (CDB or application). A local audit policy that exists in the root can contain object audit options for both local and common objects. Both local and common users who have been granted the `AUDIT_ADMIN` role can enable local policies: local users from their PDBs and common users from the root or the PDB to which they have privileges. You can enable a local audit policy for both local and common users and roles.

  You can create local audit policies for application local objects and application local roles, as well as system action options and system privilege options. You cannot enforce a local audit policy for a common user across all containers, nor can you enforce a common audit policy for a local user.
- **CDB common audit policy.** This type of policy is available to all PDBs in the multitenant environment. Only common users who have been granted the `AUDIT_ADMIN` role can create and maintain common audit policies. You can enable common audit policies only for common users. You must create common audit policies only in the root. This type of policy can contain object audit options of only common objects, and be enabled only for common users. You can enable a common audit policy for common users and roles only.

  You cannot enforce a common audit policy for a local user across all containers.
- **Application common audit policy.** Similar to CDB common audit policies, this type of policy is available to all PDBs in the multitenant environment. You can

create common audit policies for application common objects and application common roles, as well as system action options and system privilege options. You can only create this type of policy in the application root container, but you can enable it on both application common users and CDB common users. If you want to audit objects, then ensure that these objects are application common objects. You can determine whether an object is an application common object by querying the `SHARING` column of the `DBA_OBJECTS` data dictionary view.

By default, audit policies are local to the current PDB, for both CDB and application scenarios.

The following table explains how audit policies apply in different multitenant environments.

Table 18-1 How Audit Policies Apply to the CDB Root, Application Root, and Individual PDBs

Audit Option Type	CDB Root	Application Root	Individual PDB
Common audit statement or audit policy	Applies to CDB common users	Applies to CDB common users	Applies to CDB common users
Application container common audit statement or audit policy	Not applicable	• Applies to CDB common users and are valid for the current application container only • Applies to application container common users	• Applies to CDB common users and are valid for this application container only • Applies to application common users
Local audit statement or audit policy	Local configurations not allowed	Local configurations not allowed	• Applies to CDB common users • Applies to application common users

Traditional Auditing in a Multitenant Environment

In traditional auditing (not unified auditing), the `AUDIT` and `NOAUDIT` statements can audit statements and privileges in a multitenant environment.

To configure the audit policy to be either a local audit policy or a common audit policy, you must include the `CONTAINER` clause, as you normally do for other SQL creation or modification statements. If you want to audit an application container, then you can audit SQL statement and system privileges performed by local and common users and roles. The audit record will be created in the container in which the action was performed.

- If you want to apply the `AUDIT` or `NOAUDIT` statement to the current CDB or application PDB, then in this PDB, you must set `CONTAINER` to `CURRENT`. For example:

```
AUDIT DROP ANY TABLE BY SYSTEM BY ACCESS CONTAINER = CURRENT;
```

- If you want to apply the AUDIT or NOAUDIT statement to the entire multitenant environment, then in the CDB root, then you must set CONTAINER to ALL. For an application container, you would set it in the application root. For example:

```
AUDIT DROP ANY TABLE BY SYSTEM BY ACCESS CONTAINER = ALL;
```

To find if a traditional audit option is designed for use in an application container, perform a join query with the DBA_OBJ_AUDIT_OPTS and DBA_OBJECTS data dictionary views, by using the OWNER and OBJECT_NAME columns in both views, and the APPLICATION column in DBA_OBJECTS.

Related Topics

- *Oracle Database SQL Language Reference*

> **See Also:**
>
> *Oracle Database SQL Language Reference* for more information about the traditional AUDIT and NOAUDIT SQL statements

Configuring a Local Unified Audit Policy or Common Unified Audit Policy

The CONTAINER clause is specific to multitenant environment use for the CREATE AUDIT POLICY statement.

To create a local or common (CDB or application) unified audit policy in either the CDB environment or an application container environment, include the CONTAINER clause in the CREATE AUDIT POLICY statement.

- Use the following syntax to create a local or common unified audit policy:

```
CREATE AUDIT POLICY policy_name
 action1 [,action2 ]
 [CONTAINER = {CURRENT | ALL}];
```

In this specification:

- CURRENT sets the audit policy to be local to the current PDB.
- ALL makes the audit policy a common audit policy, that is, available to the entire multitenant environment.

For example, for a common unified audit policy:

```
CREATE AUDIT POLICY dict_updates
 ACTIONS UPDATE ON SYS.USER$,
  DELETE ON SYS.USER$,
  UPDATE ON SYS.LINK$,
  DELETE ON SYS.LINK$
  CONTAINER = ALL;
```

Note the following:

- You can set the `CONTAINER` clause for the `CREATE AUDIT POLICY` statement but not for `ALTER AUDIT POLICY` or `DROP AUDIT POLICY`. If you want to change the scope of an existing unified audit policy to use this setting, then you must drop and re-create the policy.
- For `AUDIT` statements, you can set the `CONTAINER` clause for audit settings only if you have an Oracle database that has not been migrated to the Release 12.*x* and later audit features. You cannot use the `CONTAINER` clause in an `AUDIT` statement that is used to enable a unified audit policy.
- If you are in a PDB, then you can only set the `CONTAINER` clause to `CURRENT`, not `ALL`. If you omit the setting while in the PDB, then the default is `CONTAINER = CURRENT`.
- If you are in the root, then you can set the `CONTAINER` clause to either `CURRENT` if you want the policy to apply to the root only, or to `ALL` if you want the policy to apply to the entire CDB. If you omit the `CONTAINER` clause, then default is `CONTAINER = CURRENT`.
- For objects:
  - Common audit policies can have common objects only and local audit policies can have both local objects and common objects.
  - You cannot set `CONTAINER` to `ALL` if the objects involved are local. They must be common objects.
- For privileges:
  - You can set the `CONTAINER` to `CURRENT` (or omit the `CONTAINER` clause) if the user accounts involved are a mixture of local and common accounts. This creates a local audit configuration that applies only to the current PDB.
  - You cannot set `CONTAINER` to `ALL` if the users involved are local users. They must be common users.
  - If you set `CONTAINER` to `ALL` and do not specify a user list (using the `BY` clause in the `AUDIT` statement), then the configuration applies to all common users in each PDB.
- For application containers, you can run a common unified audit policy from the application container script that is used for application install, upgrade, patch, and uninstall operations. To do so:
  1. Create a common unified audit policy in the application container root, and set this policy to `CONTAINER = ALL`. Alternatively, you can include this policy in the script that is described in this next step.
  2. Create a custom version of the script you normally would use to install, upgrade, patch, or uninstall Oracle Database.
  3. Within this script, include the SQL statements that you want to audit within the following lines:

     ```
     ALTER PLUGGABLE DATABASE APPLICATION BEGIN INSTALL
     List SQL statements here. Separate each statement with a semi-
     colon.
     ALTER PLUGGABLE DATABASE APPLICATION END INSTALL
     ```

     If you include the unified audit policy in the script, then ensure that you include both the `CREATE AUDIT POLICY` and `AUDIT POLICY` statements.

After the audit policy is created and enabled, all user access to the application common objects is audited irrespective of whether the audit policy is defined in the database or from the script.

- To audit application install, upgrade, patch, and uninstall operations locally in an application root or an application PDB, follow a procedure similar to the preceding procedure for common unified audit policies, but synchronize the application PDB afterward. For example:

```
ALTER PLUGGABLE DATABASE APPLICATION application_name SYNC;
```

Related Topics

- Managing Applications in an Application Container
  You install, upgrade, or patch an application in an application container.

Example: Local Unified Audit Policy

The `CREATE AUDIT POLICY` statement can create a local unified audit policy in either the root or a PDB.

When you create a local unified audit policy in the root, it only applies to the root and not across the multitenant environment.

The following example shows a local unified audit policy that has been created by the common user `c##sec_admin` from a PDB and applied to common user `c##hr_admin`.

Example 18-5 Local Unified Audit Policy

```
CONNECT c##sec_admin@hrpdb
Enter password: password
Connected.

CREATE AUDIT POLICY table_privs
 PRIVILEGES CREATE ANY TABLE, DROP ANY TABLE
 CONTAINER = CURRENT;

AUDIT POLICY table_privs BY c##hr_admin;
```

Example: CDB Common Unified Audit Policy

The `CREATE AUDIT POLICY` statement can create a CDB common unified audit policy.

Example 18-6 shows a common unified audit policy that has been created by the common user `c##sec_admin` from the root and applied to common user `c##hr_admin`.

Example 18-6 Common Unified Audit Policy

```
CONNECT c##sec_admin
Enter password: password
Connected.

CREATE AUDIT POLICY admin_pol
 ACTIONS CREATE TABLE, ALTER TABLE, DROP TABLE
 ROLES c##hr_mgr, c##hr_sup
```

```
 CONTAINER = ALL;

AUDIT POLICY admin_pol BY c##hr_admin;
```

Example: Application Common Unified Audit Policy

For application container common unified audit policies, you can audit action options and system privilege options, and refer to common objects and roles.

You can create the application common audit policy only from the application root, and enable the policy for both application common users and CDB common users.

The following example shows how to create a policy that audits the application common user `SYSTEM` for the application container `app_pdb`. The audit policy audits `SELECT` actions on the `SYSTEM.utils_tab` table and on `DROP TABLE` actions on any of the PDBs in the container database, including the CDB root. The policy also audits the use of the `SELECT ANY TABLE` system privilege across all containers.

Example 18-7 Application Common Unified Audit Policy

```
CONNECT c##sec_admin@app_pdb
Enter password: password
Connected.

CREATE AUDIT POLICY app_pdb_admin_pol
 ACTIONS SELECT ON hr_app_cdb.utils_tab, DROP TABLE
 PRIVILEGES SELECT ANY TABLE
 CONTAINER = ALL;

AUDIT POLICY app_pdb_admin_pol by SYSTEM, c##hr_admin;
```

In the preceding example, setting `CONTAINER` to `ALL` applies the policy only to all the relevant object accesses in the application root and on all the application PDBs that belong to the application root. It does not apply the policy outside this scope.

How Local or Common Audit Policies or Settings Appear in the Audit Trail

You can query unified audit policy views from either the root or the PDB in which the action occurred.

You can perform the following types of queries:

- **Audit records from all PDBs.** The audit trail reflects audited actions that have been performed in the PDBs. For example, if user `lbrown` in `PDB1` performs an action that has been audited by either a common or a local audit policy, then the audit trail will capture this action. The `DBID` column in the `UNIFIED_AUDIT_TRAIL` data dictionary view indicates the PDB in which the audited action takes place and to which the policy applies. If you want to see audit records from all PDBs, you should query the `CDB_UNIFIED_AUDIT_TRAIL` data dictionary view from the root.
- **Audit records from common audit policies.** This location is where the common audit policy results in an audit record. The audit record can be generated anywhere in the multitenant environment—the root or the PDBs, depending on where the action really occurred. For example, the common audit policy `fga_pol` audits the `EXECUTE` privilege on the `DBMS_FGA` PL/SQL package, and if this action

occurs in PDB1, then the audit record is generated in PDB1 and not in the root. Hence, the audit record can be seen in PDB1.

You can query the UNIFIED_AUDIT_TRAIL data dictionary view for the policy from either the root or a PDB if you include a WHERE clause for the policy name (for example, WHERE UNIFIED_AUDIT_POLICIES = 'FGA_POL').

The following example shows how to find the results of a common unified audit policy:

```
CONNECT c##sec_admin
Enter password: password
Connected.

SELECT DBID, ACTION_NAME, OBJECT_SCHEMA, OBJECT_NAME FROM
CDB_UNIFIED_AUDIT_TRAIL WHERE DBUSERNAME = 'c##hr_admin';
46892-1
DBID        ACTION_NAME  OBJECT_SCHEMA  OBJECT_NAME
----------- -----------  -------------  -----------
653916017   UPDATE       HR             EMPLOYEES
653916018   UPDATE       HR             JOB_HISTORY
653916017   UPDATE       HR             JOBS
```

Fine-Grained Auditing in a Multitenant Environment

You can create fine-grained audit policies in the CDB root, application root, CDB PDBs, and application PDBs.

Note the following general rules about fine-grained audit policies in a multitenant environment:

- You cannot create fine-grained audit policies on SYS objects.
- You cannot create fine-grained audit policies, either local or application common, for extended data link objects.
- When you create a fine-grained audit policy in the CDB root, the policy cannot be applied to all PDBs. It only applies to objects within the CDB root. (In other words, there is no such thing as a common fine-grained audit policy for the CDB root.) If you want to create a fine-grained audit policy to audit a common object's access in all the PDBs, then you must explicitly create that policy in each PDB and then enable it on the common objects that is accessible in the PDB.
- When you create a fine-grained audit policy in a PDB, it applies only to objects within the PDB.
- You can create application common fine-grained audit policies only if you are connected to the application root and only within the BEGIN/END block. If you are connected to the application root and create the fine-grained audit policy outside the BEGIN/END block, then the fine-grained audit policy is created in the application root.
- You cannot create application common fine-grained audit policies on local PDB objects.
- If the application common fine-grained audit policy has a handler, then this handler must be owned by either an application common user or a CDB common user.
- You can create an application fine-grained audit policy on local (PDB) objects and CDB common objects. Because the policy is local to its container, the object on

which the policy is defined is audited only in the particular container where the policy is defined. For example, if you create a fine-grained audit policy in the `hr_pdb` PDB, the object for which you create this policy must exist in the `hr_pdb` PDB.

- You cannot create local fine-grained audit policies in an application PDB on object linked and extended data link objects. On metadata-linked objects are allowed in the fine-grained audit policy.
- Application root local policies are allowed for all application common objects.
- When you create a fine-grained audit policy as a common audit policy in an application root, it will be effective in each PDB that belongs to this application root. Therefore, any access to the application common object and CDB common object (on which the application common fine-grained audit policy is defined) from the application PDB is audited in the fine-grained audit trail in that application PDB.
- When you create scripts for application install, upgrade, patch, or uninstall operations, you can include SQL statements within the `ALTER PLUGGABLE DATABASE` *`app_name`* `BEGIN INSTALL` and `ALTER PLUGGABLE DATABASE` *`app_name`* `END INSTALL` blocks to perform various operations. You can include fine-grained audit policy statements only within these blocks.
- You can only enable, disable, or drop application common fine-grained audit policies from the application root, and from within a `ALTER PLUGGABLE DATABASE` *`app_name`* `BEGIN INSTALL` and `ALTER PLUGGABLE DATABASE` *`app_name`* `END INSTALL` block in a script.

Related Topics

- *Oracle Database Security Guide*

19
Monitoring CDBs and PDBs

You can view information about CDBs and PDBs using SQL*Plus or SQL Developer.

Related Topics

- Tools for a Multitenant Environment
  You can use various tools to configure and administer a multitenant environment.

About CDB and Container Information in Views

In a CDB, the metadata for data dictionary tables and view definitions is stored only in the root.

Each container, including each PDB, application root, and application PDB, has its own set of data dictionary tables and views for the objects contained in the container. Because each container can contain different data and schema objects, containers can display different metadata in data dictionary views, even when querying the same view in each container. For example, metadata about tables displayed in the `DBA_TABLES` view can be different in two different containers because the containers can contain different tables. An internal mechanism called a metadata link enables a container to access the metadata for these views in the root.

If a dictionary table stores information that pertains to the whole CDB, instead of for each container, then the metadata and the data displayed in a data dictionary view are stored in the root. For example, Automatic Workload Repository (AWR) data can be stored in the root, and this data is displayed in some data dictionary views, such as the `DBA_HIST_ACTIVE_SESS_HISTORY` view. An internal mechanism called a data link enables a container to access both the metadata and the data for these types of views in the root.

> **See Also:**
>
> "Data Dictionary Architecture in a CDB" for more information about dictionary access in containers, metadata links, and data links

About Viewing Information When the Current Container Is Not the CDB Root

When the current container is a PDB, an application root, or an application PDB, the data dictionary views show metadata for the current container only.

To an application connected to a PDB, application root, or application PDB, the data dictionary appears as it would for a non-CDB. The data dictionary only shows metadata related to the current container. Also, in a container that is not the CDB

root, CDB_ views only show information about database objects visible through the corresponding DBA_ view.

About Viewing Information When the Current Container Is the CDB Root

When the current container is the CDB root, a common user can view data dictionary information for the CDB root and for PDBs, application roots, and application PDBs by querying container data objects.

A container data object is a table or view that can contain data pertaining to the following:

- One or more containers
- The CDB as a whole
- One or more containers and the CDB as a whole

Container data objects include V$, GV$, CDB_, and some Automatic Workload Repository DBA_HIST* views. A common user's CONTAINER_DATA attribute determines which containers are visible in container data objects.

In a CDB, for every DBA_ view, there is a corresponding CDB_ view. All CDB_ views are container data objects, but most DBA_ views are not.

Each container data object contains a CON_ID column that identifies the container for each row returned. Table 19-1 describes the meanings of the values in the CON_ID column.

Table 19-1 CON_ID Column in Container Data Objects

Value in CON_ID Column	Description
0	The data pertains to the entire CDB
1	The data pertains to the CDB root
2	The data pertains to the PDB seed
3 - 4,098	The data pertains to a PDB, an application root, or an application PDB Each container has its own container ID.

The following views behave differently from other [G]V$ views:

- [G]V$SYSSTAT
- [G]V$SYS_TIME_MODEL
- [G]V$SYSTEM_EVENT
- [G]V$SYSTEM_WAIT_CLASS

When queried from the CDB root, these views return instance-wide data, with 0 in the CON_ID column for each row returned. However, you can query equivalent views that behave the same as other container data objects. The following views can return specific data for each container in a CDB: [G]V$CON_SYSSTAT, [G]V$CON_SYS_TIME_MODEL, [G]V$CON_SYSTEM_EVENT, and [G]V$CON_SYSTEM_WAIT_CLASS.

Note:

- When querying a container data object, the data returned depends on whether containers are open and on the privileges granted to the user running the query.
- In an Oracle Real Application Clusters (Oracle RAC) environment, the data returned by container data objects might vary based on the instance to which a session is connected.
- In a non-CDB, all `CON_ID` columns in container data objects are 0 (zero).
- When a container is opened in restricted mode, it is ignored in queries on `CDB_` views.

See Also:

- "About the Current Container"
- "Container Data Objects in a CDB"
- *Oracle Database Security Guide* for detailed information about container data objects

Views for a CDB

You can query a set of views for information about a CDB and its PDBs.

Table 19-2 describes data dictionary views that are useful for monitoring a CDB and its PDBs.

Table 19-2 Views for a CDB

View	Description	More Information
Container data objects, including: • `V$` views • `GV$` views • `CDB_` views • `DBA_HIST*` views	Container data objects can display information about multiple PDBs. Each container data object includes a `CON_ID` column to identify containers. There is a `CDB_` view for each corresponding `DBA_` view.	"Querying Container Data Objects" *Oracle Database Security Guide*
`{CDB|DBA}_PDBS`	Displays information about the PDBs associated with the CDB, including the status of each PDB.	"Viewing Information About PDBs" *Oracle Database Reference*
`CDB_PROPERTIES`	Displays the permanent properties of each container in a CDB.	*Oracle Database Reference*
`{CDB|DBA}_PDB_HISTORY`	Displays the history of each PDB.	*Oracle Database Reference*

Table 19-2 (Cont.) Views for a CDB

View	Description	More Information
{CDB|DBA}_CONTAINER_DATA	Displays information about the user-level and object-level CONTAINER_DATA attributes specified in the CDB.	*Oracle Database Reference*
{CDB|DBA}_HIST_PDB_INSTANCE	Displays the PDBs and instances in the Workload Repository.	*Oracle Database Reference*
{CDB|DBA}_PDB_SAVED_STATES	Displays information about the current saved PDB states in the CDB.	*Oracle Database Reference* "Preserving or Discarding the Open Mode of PDBs When the CDB Restarts"
{CDB|DBA}_APPLICATIONS	Describes all applications in an application container.	"Viewing Information About Applications"
{CDB|DBA}_APP_STATEMENTS	Describes all statements from application installation, upgrade, and patch operations in an application container.	"Viewing Information About Application Statements"
{CDB|DBA}_APP_PATCHES	Describes all application patches in an application container.	"Viewing Information About Application Patches"
{CDB|DBA}_APP_ERRORS	Describes all application error messages generated in an application container.	"Viewing Information About Application Errors"
{CDB|DBA}_CDB_RSRC_PLANS	Displays information about all the CDB resource plans.	*Oracle Database Reference* "Viewing CDB Resource Plans"
{CDB|DBA}_CDB_RSRC_PLAN_DIRECTIVES	Displays information about all the CDB resource plan directives.	*Oracle Database Reference* "Viewing CDB Resource Plan Directives"
PDB_ALERTS	Contains descriptions of reasons for PDB alerts.	*Oracle Database Reference*
PDB_PLUG_IN_VIOLATIONS	Displays information about incompatibilities between a PDB and the CDB to which it belongs. This view is also used to display information generated by executing DBMS_PDB.CHECK_PLUG_COMPATIBILITY.	*Oracle Database Reference* "Plugging In an Unplugged PDB"
{USER|ALL|DBA|CDB}_OBJECTS	Displays information about database objects, and the SHARING column shows whether a database object is a metadata-linked object, a data-linked object, an extended data-linked object, or a standalone object that is not linked to another object.	*Oracle Database Reference*
{ALL|DBA|CDB}_SERVICES	Displays information about database services, and the PDB column shows the name of the PDB associated with each service.	*Oracle Database Reference*

Table 19-2 (Cont.) Views for a CDB

View	Description	More Information
{USER|ALL|DBA|CDB}_VIEWS {USER|ALL|DBA|CDB}_TABLES	The CONTAINER_DATA column shows whether the view or table is a container data object.	*Oracle Database Reference*
{USER|ALL|DBA|CDB}_USERS	The COMMON column shows whether a user is a common user or a local user.	*Oracle Database Reference*
{USER|ALL|DBA|CDB}_ROLES {USER|ALL|DBA|CDB}_COL_PRIVS {USER|ALL}_COL_PRIVS_MADE {USER|ALL}_COL_PRIVS_RECD {USER|ALL}_TAB_PRIVS_MADE {USER|ALL}_TAB_PRIVS_RECD {USER|DBA|CDB}_SYS_PRIVS {USER|DBA|CDB}_ROLE_PRIVS ROLE_TAB_PRIVS ROLE_SYS_PRIVS	The COMMON column shows whether a role or privilege is commonly granted or locally granted.	*Oracle Database Reference*
{USER|ALL|DBA|CDB}_ARGUMENTS {USER|ALL|DBA|CDB}_CLUSTERS {USER|ALL|DBA| CDB}_CONSTRAINTS {ALL|DBA|CDB}_DIRECTORIES {USER|ALL|DBA| CDB}_IDENTIFIERS {USER|ALL|DBA|CDB}_LIBRARIES {USER|ALL|DBA|CDB}_PROCEDURES {USER|ALL|DBA|CDB}_SOURCE {USER|ALL|DBA|CDB}_SYNONYMS {USER|ALL|DBA|CDB}_VIEWS	The ORIGIN_CON_ID column shows the ID of the container from which the row originates.	*Oracle Database Reference*
[G]V$DATABASE	Displays information about the database from the control file. If the database is a CDB, then CDB-related information is included.	"Determining Whether a Database Is a CDB" *Oracle Database Reference*
[G]V$CONTAINERS	Displays information about the containers associated with the current CDB, including the root and all PDBs.	"Viewing Information About the Containers in a CDB" *Oracle Database Reference*
[G]V$PDBS	Displays information about the PDBs associated with the current CDB, including the open mode of each PDB.	"Viewing the Open Mode of Each PDB" *Oracle Database Reference*
[G]V$PDB_INCARNATION	Displays information about all PDB incarnations. Oracle creates a new PDB incarnation whenever a PDB is opened with the RESETLOGS option.	*Oracle Database Reference*

Table 19-2 (Cont.) Views for a CDB

View	Description	More Information
[G]V$SYSTEM_PARAMETER [G]V$PARAMETER	Displays information about initialization parameters, and the ISPDB_MODIFIABLE column shows whether a parameter can be modified for a PDB.	"Listing the Modifiable Initialization Parameters in PDBs" *Oracle Database Reference*
V$DIAG_ALERT_EXT [G]V$DIAG_APP_TRACE_FILE [G]V$DIAG_OPT_TRACE_RECORDS V$DIAG_SESS_OPT_TRACE_RECORDS V$DIAG_SESS_SQL_TRACE_RECORDS [G]V$DIAG_SQL_TRACE_RECORDS [G]V$DIAG_TRACE_FILE [G]V$DIAG_TRACE_FILE_CONTENTS	Displays trace file and alert file data for the current container in a CDB.	*Oracle Database SQL Tuning Guide*
V$DIAG_INCIDENT V$DIAG_PROBLEM	Displays information about problems and incidents for the current container in a CDB.	*Oracle Database Reference*

Determining Whether a Database Is a CDB

You can query the CDB column in the V$DATABASE view to determine whether a database is a CDB or a non-CDB. The CDB column returns YES if the current database is a CDB or NO if the current database is a non-CDB.

To determine whether a database is a CDB:

1. In SQL*Plus, connect to the database as an administrative user.
2. Query the V$DATABASE view.

Example 19-1 Determining Whether a Database is a CDB

```
SELECT CDB FROM V$DATABASE;
```

Sample output:

```
CDB
---
YES
```

> **See Also:**
>
> *Oracle Database Reference*

Viewing Information About the Containers in a CDB

The V$CONTAINERS view provides information about all containers in a CDB, including the root and all PDBs.

To view this information, the query must be run by a common user whose current container is the root. When the current container is a PDB, this view only shows information about the current PDB.

To view information about the containers in a CDB:

1. In SQL*Plus, ensure that the current container is the root.

   See "About Container Access in a CDB".

2. Query the V$CONTAINERS view.

Example 19-2 Viewing Identifying Information About Each Container in a CDB

```
COLUMN NAME FORMAT A8

SELECT NAME, CON_ID, DBID, CON_UID, GUID FROM V$CONTAINERS ORDER BY
CON_ID;
```

Sample output:

```
NAME         CON_ID       DBID    CON_UID GUID
-------- ---------- ---------- ----------
--------------------------------
CDB$ROOT          1  659189539          1
C091A6F89C7572A1E0436797E40AC78D
PDB$SEED          2 4026479912 4026479912
C091AE9C00377591E0436797E40AC138
HRPDB             3 3718888687 3718888687
C091B6B3B53E7834E0436797E40A9040
SALESPDB          4 2228741407 2228741407
C091FA64EF8F0577E0436797E40ABE9F
```

See Also:

- "About Configuring and Managing a Multitenant Environment"
- "About the Current Container"
- "Determining the Current Container ID or Name"
- *Oracle Database Reference*

Viewing Information About PDBs

The CDB_PDBS view and DBA_PDBS view provide information about the PDBs associated with a CDB, including the status of each PDB.

To view this information, the query must be run by a common user whose current container is the root. When the current container is a PDB, all queries on these views return no results.

To view information about PDBs:

1. In SQL*Plus, ensure that the current container is the root.

   See "Accessing a Container in a CDB with SQL*Plus".

2. Query the CDB_PDBS or DBA_PDBS view.

Example 19-3 Viewing Container ID, Name, and Status of Each PDB

```
COLUMN PDB_NAME FORMAT A15

SELECT PDB_ID, PDB_NAME, STATUS FROM DBA_PDBS ORDER BY PDB_ID;
```

Sample output:

```
    PDB_ID PDB_NAME        STATUS
---------- --------------- -------------
         2 PDB$SEED        NORMAL
         3 HRPDB           NORMAL
         4 SALESPDB        NORMAL
```

> **See Also:**
>
> "About the Current Container"

Viewing the Open Mode of Each PDB

The V$PDBS view provides information about the PDBs associated with the current database instance.

You can query this view to determine the open mode of each PDB. For each PDB that is open, this view can also show when the PDB was last opened. A common user can query this view when the current container is the root or a PDB. When the current container is a PDB, this view only shows information about the current PDB.

To view the open status of each PDB:

1. In SQL*Plus, access a container.

   See "Accessing a Container in a CDB with SQL*Plus".

2. Query the V$PDBS view.

Example 19-4 Viewing the Name and Open Mode of Each PDB

```
COLUMN NAME FORMAT A15
COLUMN RESTRICTED FORMAT A10
```

```
COLUMN OPEN_TIME FORMAT A30

SELECT NAME, OPEN_MODE, RESTRICTED, OPEN_TIME FROM V$PDBS;
```

Sample output:

```
NAME            OPEN_MODE  RESTRICTED OPEN_TIME
--------------- ---------- ---------- ------------------------------
PDB$SEED        READ ONLY  NO         21-MAY-12 12.19.54.465 PM
HRPDB           READ WRITE NO         21-MAY-12 12.34.05.078 PM
SALESPDB        MOUNTED    NO         22-MAY-12 10.37.20.534 AM
```

See Also:

- "Modifying the Open Mode of PDBs with ALTER PLUGGABLE DATABASE"
- "Modifying the Open Mode of PDBs"
- "Modifying a PDB with the ALTER PLUGGABLE DATABASE Statement"
- "About the Current Container"

Querying Container Data Objects

In the root, container data objects can show information about database objects (such as tables and users) contained in the root and in PDBs. Access to PDB information is controlled by the common user's `CONTAINER_DATA` attribute.

For example, `CDB_` views are container data objects. See "About Viewing Information When the Current Container Is the CDB Root" and *Oracle Database Security Guide* for more information about container data objects.

Each container data object contains a `CON_ID` column that shows the container ID of each PDB in the query results. You can view the PDB name for a container ID by querying the `DBA_PDBS` view.

To use container data objects to show information about multiple PDBs:

1. In SQL*Plus, ensure that the current container is the root.

   See "About Container Access in a CDB".

2. Query the container data object to show the desired information.

Note:

When a query contains a join of a container data object and a non-container data object, and the current container is the root, the query returns data for the entire CDB only (`CON_ID = 0`).

Example 19-5 Showing the Tables Owned by Specific Schemas in Multiple PDBs

This example queries the DBA_PDBS view and the CDB_TABLES view from the root to show the tables owned by hr user and oe user in the PDBs associated with the CDB. This query returns only rows where the PDB has an ID greater than 2 (p.PDB_ID > 2) to avoid showing the users in the CDB root and PDB seed.

```
COLUMN PDB_NAME FORMAT A15
COLUMN OWNER FORMAT A15
COLUMN TABLE_NAME FORMAT A30

SELECT p.PDB_ID, p.PDB_NAME, t.OWNER, t.TABLE_NAME
  FROM DBA_PDBS p, CDB_TABLES t
  WHERE p.PDB_ID > 2 AND
        t.OWNER IN('HR','OE') AND
        p.PDB_ID = t.CON_ID
   ORDER BY p.PDB_ID;
```

Sample output:

```
    PDB_ID PDB_NAME        OWNER           TABLE_NAME
---------- --------------- ---------------
------------------------------
         3 HRPDB           HR              COUNTRIES
         3 HRPDB           HR              JOB_HISTORY
         3 HRPDB           HR              EMPLOYEES
         3 HRPDB           HR              JOBS
         3 HRPDB           HR              DEPARTMENTS
         3 HRPDB           HR              LOCATIONS
         3 HRPDB           HR              REGIONS
         4 SALESPDB        OE              PRODUCT_INFORMATION
         4 SALESPDB        OE              INVENTORIES
         4 SALESPDB        OE              ORDERS
         4 SALESPDB        OE              ORDER_ITEMS
         4 SALESPDB        OE              WAREHOUSES
         4 SALESPDB        OE              CUSTOMERS
         4 SALESPDB        OE
SUBCATEGORY_REF_LIST_NESTEDTAB
         4 SALESPDB        OE              PRODUCT_REF_LIST_NESTEDTAB
         4 SALESPDB        OE              PROMOTIONS
         4 SALESPDB        OE              PRODUCT_DESCRIPTIONS
```

This sample output shows the PDB hrpdb has tables in the hr schema and the PDB salespdb has tables in the oe schema.

Example 19-6 Showing the Users in Multiple PDBs

This example queries the DBA_PDBS view and the CDB_USERS view from the root to show the users in each PDB. The query uses p.PDB_ID > 2 to avoid showing the users in the CDB root and the PDB seed.

```
COLUMN PDB_NAME FORMAT A15
COLUMN USERNAME FORMAT A30
```

```
SELECT p.PDB_ID, p.PDB_NAME, u.USERNAME
  FROM DBA_PDBS p, CDB_USERS u
  WHERE p.PDB_ID > 2 AND
        p.PDB_ID = u.CON_ID
  ORDER BY p.PDB_ID;
```

Sample output:

```
    PDB_ID PDB_NAME        USERNAME
---------- --------------- ------------------------------
         .
         .
         .
         3 HRPDB           HR
         3 HRPDB           OLAPSYS
         3 HRPDB           MDSYS
         3 HRPDB           ORDSYS
         .
         .
         .
         4 SALESPDB        OE
         4 SALESPDB        CTXSYS
         4 SALESPDB        MDSYS
         4 SALESPDB        EXFSYS
         4 SALESPDB        OLAPSYS
         .
         .
         .
```

Example 19-7 Showing the Data Files for Each PDB in a CDB

This example queries the `DBA_PDBS` and `CDB_DATA_FILES` views to show the name and location of each data file for all of the PDBs in a CDB, including the PDB seed.

```
COLUMN PID FORMAT 999
COLUMN PDB_NAME FORMAT A8
COLUMN FILE_ID FORMAT 9999
COLUMN TABLESPACE_NAME FORMAT A10
COLUMN FILE_NAME FORMAT A45

SELECT p.PDB_ID AS PID, p.PDB_NAME, d.FILE_ID, d.TABLESPACE_NAME,
d.FILE_NAME
  FROM DBA_PDBS p, CDB_DATA_FILES d
  WHERE p.PDB_ID = d.CON_ID
  ORDER BY p.PDB_ID;
```

Sample output:

```
PID PDB_NAME FILE_ID TABLESPACE FILE_NAME
--- -------- ------- ---------- ----------------------------------------
  2 PDB$SEED       6 SYSAUX     /disk1/oracle/dbs/pdbseed/cdb1_ax.f
  2 PDB$SEED       5 SYSTEM     /disk1/oracle/dbs/pdbseed/cdb1_db.f
```

```
  3 HRPDB          9 SYSAUX     /disk1/oracle/dbs/hrpdb/hrpdb_ax.f
  3 HRPDB          8 SYSTEM     /disk1/oracle/dbs/hrpdb/hrpdb_db.f
  3 HRPDB         13 USER       /disk1/oracle/dbs/hrpdb/hrpdb_usr.dbf
  4 SALESPDB      15 SYSTEM     /disk1/oracle/dbs/salespdb/salespdb_db.f
  4 SALESPDB      16 SYSAUX     /disk1/oracle/dbs/salespdb/salespdb_ax.f
  4 SALESPDB      18 USER       /disk1/oracle/dbs/salespdb/
salespdb_usr.dbf
```

Example 19-8 Showing the Temp Files in a CDB

This example queries the `CDB_TEMP_FILES` view to show the name and location of each temp file in a CDB, as well as the tablespace that uses the temp file.

```
COLUMN CON_ID FORMAT 999
COLUMN FILE_ID FORMAT 9999
COLUMN TABLESPACE_NAME FORMAT A15
COLUMN FILE_NAME FORMAT A45

SELECT CON_ID, FILE_ID, TABLESPACE_NAME, FILE_NAME
  FROM CDB_TEMP_FILES
  ORDER BY CON_ID;
```

Sample output:

```
CON_ID FILE_ID TABLESPACE_NAM FILE_NAME
------ ------- --------------
--------------------------------------------
     1       1 TEMP           /disk1/oracle/dbs/t_tmp1.f
     2       2 TEMP           /disk1/oracle/dbs/pdbseed/t_tmp1.f
     3       3 TEMP           /disk1/oracle/dbs/hrpdb/t_hrpdb_tmp1.f
     4       4 TEMP           /disk1/oracle/dbs/salespdb/
t_salespdb_tmp1.f
```

Example 19-9 Showing the Services Associated with PDBs

This example queries the `CDB_SERVICES` view to show the PDB name, network name, and container ID of each service associated with a PDB.

```
COLUMN NETWORK_NAME FORMAT A30
COLUMN PDB FORMAT A15
COLUMN CON_ID FORMAT 999

SELECT PDB, NETWORK_NAME, CON_ID FROM CDB_SERVICES
  WHERE PDB IS NOT NULL AND
        CON_ID > 2
  ORDER BY PDB;
```

Sample output:

```
PDB             NETWORK_NAME                   CON_ID
--------------- ------------------------------ ------
```

```
HRPDB           hrpdb.example.com               3
SALESPDB        salespdb.example.com            4
```

See Also:

- "About the Current Container"
- "Container Data Objects in a CDB"
- *Oracle Database Security Guide* for detailed information about container data objects
- *Oracle Database Reference*

Querying Across Containers with the CONTAINERS Clause

The `CONTAINERS` clause enables you to query tables and views across all containers in a CDB. It also enables you to query application common objects across all containers in an application container.

About Querying Across Containers with the CONTAINERS Clause

The `CONTAINERS` clause enables you to query across containers in a CDB.

The `CONTAINERS` clause enables you to query user-created tables and views across all containers in a CDB. This clause enables queries from the CDB root to display data in tables or views that exist in all open containers in a CDB.

The `CONTAINERS` clause also enables you to query application common objects, such as tables and views, across all application PDBs in an application container. This clause enables queries from the application root to display data in tables or views that exist in all open application PDBs in the application container.

The `CONTAINERS` clause exposes three implicitly generated columns:

- `CON_ID`: The ID of the container from which the row is retrieved.
- `CON$NAME`: The name of the container from which the row is retrieved. This is a hidden column.
- `CDB$NAME`: The name of the CDB from which the row is retrieved. In the absence of a proxy PDB or a CDB fleet, all rows will have the same value for `CDB$NAME`. This is a hidden column.

When the `CONTAINERS` clause is evaluated, each container is treated as a partition; therefore, the plan output for a query using the `CONTAINERS` clause includes a partition iterator. Partition pruning can be used to restrict the set of containers that is accessed during query execution. The pruning predicate may be specified either on the `CON_ID` column or the `CON$NAME` column, both of which are implicitly generated for a `CONTAINERS` clause.

Evaluation of the `CONTAINERS` clause makes use of parallel execution processes. Each container is assigned to a parallel execution process (P00*) and the process switches into the container to execute a recursive SQL statement on the base table or view.

The base table or view is the object whose name is passed as an argument to the CONTAINERS clause.

The CONTAINERS_PARALLEL_DEGREE initialization parameter can control the degree of parallelism of a query involving the CONTAINERS clause. If the value of CONTAINERS_PARALLEL_DEGREE is lower than 65535 (the default), then the specified value is used.

When the CONTAINERS_PARALLEL_DEGREE initialization parameter is set to the default value (65535), queries that use the CONTAINERS clause are parallel by default. The default degree of parallelism is calculated with the following formula:

```
max(min(cpu_count,number_of_open_containers),#instances)
```

In addition, you can pass a DEFAULT_PDB_HINT hint in the CONTAINERS clause. The hint is passed in the query that is run in each container.

The columns accessed by the recursive SQL statement are determined by the columns of the CONTAINERS clause accessed in the query. Predicates in the query using the CONTAINERS clause may be pushed down to the recursive SQL and evaluated within each container, significantly reducing the number of rows that need to be processed as a post filter on the CONTAINERS clause.

You can force the recursive SQL that results from a query that includes the CONTAINERS clause to be parallel by using the DEFAULT_PDB_HINT clause of a CONTAINERS hint or by using automatic degree of parallelism. However, parallel statement queuing is not possible for recursive SQL that results from a query that includes the CONTAINERS clause.

Columns of the following types are removed if they exist in a table specified in a CONTAINERS clause:

- The following user-defined types: object types, varrays, REFs, and nested tables
- The following Oracle-supplied types: ANYTYPE, ANYDATASET, URI types, SDO_TOPO_GEOMETRY, SDO_GEORASTER, and Expression

Note:

- When a container is opened in restricted mode, it is ignored by the CONTAINERS clause.
- When the CONTAINERS clause is used and an error is returned by a container, the query does not return results from the container that raised the error, and the error is not returned. For example, you cannot select a BFILE column from a remote table into a local variable. If a query that does this uses the CONTAINERS clause and includes local and remote containers, then the query returns results for the local containers, but not the remote containers, and no error is returned.

> **See Also:**
>
> - "About the Current Container"
> - *Oracle Database SQL Language Reference* for more information about the `CONTAINERS` clause and the `CONTAINERS` hint
> - *Oracle Database Security Guide* for detailed information about container data objects
> - *Oracle Database Reference* for more information about the `CONTAINERS_PARALLEL_DEGREE` initialization parameter
> - *Oracle Database Data Warehousing Guide* for more information about automatic degree of parallelism and parallel statement queuing

Querying User-Created Tables and Views Across All Containers

The `CONTAINERS` clause enables you to query user-created tables and views across all containers. This clause enables queries from the CDB root to display data in tables or views that exist in all open PDBs in a CDB.

Prerequisites

The tables and views, or synonyms of them, specified in the `CONTAINERS` clause must exist in the CDB root and in all other containers.

To use the `CONTAINERS` clause to query tables and views across all containers:

1. In SQL*Plus, access a container.

   To view data in multiple containers, ensure that the current container is the CDB root.

   See "About Container Access in a CDB".

2. Run a query that includes the `CONTAINERS` clause.

Example 19-10 Querying a Table Owned by a Common User Across All Containers

This example makes the following assumptions:

- An organization has several PDBs, and each PDB is for a different department in the organization.
- Each PDB has an `employees` table that tracks the employees in the department, but the table in each PDB contains different employees.
- The CDB root also has an empty `employees` table.
- The `employees` table in each container is owned by the same common user.

With the CDB root as the current container and the common user that owns the table as the current user, run the following query with the `CONTAINERS` clause to return all employees in the `employees` table in all PDBs:

```
SELECT * FROM CONTAINERS(employees);
```

Example 19-11 Querying a Table Owned by Local Users Across All Containers

This example makes the following assumptions:

- An organization has several PDBs, and each PDB is for a different department in the organization.
- Each PDB has an `hr.employees` table that tracks the employees in the department, but the table in each PDB contains different employees.
- The CDB root also has an empty `employees` table owned by a common user.

To run a query that returns all employees in all PDBs, first connect to each PDB as a common user, and create a view with the following statement:

```
CREATE OR REPLACE VIEW employees AS SELECT * FROM hr.employees;
```

The common user that owns the view must be the same common user that owns the `employees` table in the CDB root. After you run this statement in each PDB, the common user has a view named `employees` in each PDB.

With the CDB root as the current container and the common user as the current user, run the following query with the `CONTAINERS` clause to return all employees in the `hr.employees` table in all PDBs:

```
SELECT * FROM CONTAINERS(employees);
```

You can also query the view in specific containers. For example, the following SQL statement queries the view in the containers with a `CON_ID` of 3 and 4:

```
SELECT * FROM CONTAINERS(employees) WHERE CON_ID IN(3,4);
```

Note:

You can also use the `CONTAINERS` clause to query Oracle-supplied tables and views. When running the query, ensure that the current user is the owner of the table or view, or create a view using the `CONTAINERS` clause and grant `SELECT` privilege on the view to the appropriate users.

See Also:

- "About the Current Container"
- "Container Data Objects in a CDB"
- *Oracle Database SQL Language Reference* for more information about the `CONTAINERS` clause
- *Oracle Database Security Guide* for detailed information about container data objects

Querying Application Common Objects Across Application PDBs

The CONTAINERS clause enables you to query application common objects across all PDBs in an application container. Queries from the application root display data in objects that exist in all open PDBs in the container.

The CONTAINERS clause is most useful for metadata-linked application common objects. With metadata-linked application common objects, the structure is the same in all containers in an application container, but the data is different. You can use the CONTAINERS clause to view the data in a metadata-linked application common object in multiple application PDBs. The benefits are similar for extended data-linked objects. The CONTAINERS clause uses parallel execution to execute the query across the distinct application PDBs hosted in the application root.

To use the CONTAINERS clause to query tables and views across all application PDBs:

1. In SQL*Plus, access the application root.

   See "About Container Access in a CDB".

2. Run a query that includes the CONTAINERS clause.

> **Note:**
>
> You can enable the CONTAINERS_DEFAULT attribute for a table or view in an application root. When this attribute is enabled, the CONTAINERS clause is used for queries and DML statements on the database object by default, and the CONTAINERS clause is not required in the SQL statements. To enable the CONTAINERS_DEFAULT attribute for a table or view in an application root, run the ALTER TABLE or CREATE OR REPLACE VIEW statement with the ENABLE CONTAINERS_DEFAULT clause.

Example 19-12 Querying an Application Common Object Across All Application PDBs

This example makes the following assumptions:

- An organization has several application PDBs, and each application PDB is for a different department in the organization.
- Each application PDB has an employees table that tracks the employees in the department, but the table in each application PDB contains different employees.
- The application root also has an empty employees table.
- The employees table in each container is owned by the same common user.
- A company has multiple tenants that use an application in an application container, and each tenant has its own application PDB.
- The company uses metadata-linked application common objects to keep the structure of the data the same in all application PDBs, but the data is different in each application PDB.

- Each application PDB has a metadata-linked `sales.customers` table that stores information about each tenant's customers.

With the application root as the current container and the application common user that owns the table as the current user, run the following query with the `CONTAINERS` clause to return all customers in the `sales.customers` table in all application PDBs:

```
SELECT * FROM CONTAINERS(sales.customers);
```

See Also:

- "About Application Common Objects"
- "About the Current Container"
- *Oracle Database SQL Language Reference* for more information about the `CONTAINERS` clause
- *Oracle Database Security Guide* for detailed information about container data objects

Determining the Current Container ID or Name

You can determine your current container ID or container name in a CDB.

To determine the current container ID:

- Run the following SQL*Plus command:

```
SHOW CON_ID
```

To determine the current container name:

- Run the following SQL*Plus command:

```
SHOW CON_NAME
```

In addition, you can use the functions listed in Table 19-3 to determine the container ID, container name, and DBID of a container.

Table 19-3 Functions That Return the Container Information

Function	Description
`CON_NAME_TO_ID('`*container_name*`')`	Returns the container ID based on the container's name.
`CON_DBID_TO_ID(`*container_dbid*`)`	Returns the container ID based on the container's DBID.
`CON_UID_TO_ID(`*container_uid*`)`	Returns the container ID based on the container's unique identifier (UID).
`CON_GUID_TO_ID(`*container_guid*`)`	Returns the container ID based on the container's globally unique identifier (GUID).
`CON_ID_TO_CON_NAME(`*container_id*`)`	Returns the container name based on the container ID.

Table 19-3 (Cont.) Functions That Return the Container Information

Function	Description
CON_ID_TO_DBID(*container_id*)	Returns the container's DBID based on the container ID.

The V$CONTAINERS view shows the name, DBID, UID, and GUID for each container in a CDB.

Example 19-13 Returning the Container ID Based on the Container Name

```
SELECT CON_NAME_TO_ID('HRPDB') FROM DUAL;
```

Example 19-14 Returning the Container ID Based on the Container DBID

```
SELECT CON_DBID_TO_ID(2226957846) FROM DUAL;
```

Example 19-15 Returning the Container Name Based on the Container ID

```
SELECT CON_ID_TO_CON_NAME(4) FROM DUAL;
```

See Also:

- "About a Multitenant Environment"
- "About the Current Container"
- "Viewing Information About the Containers in a CDB"
- *Oracle Database Reference* for more information about the V$CONTAINERS view

Listing the Modifiable Initialization Parameters in PDBs

In a CDB, some initialization parameters apply to the root and to all PDBs. When such an initialization parameter is changed, it affects the entire CDB. You can set other initialization parameters to different values in each container.

For example, you might have a parameter set to one value in the root, set to another value in one PDB, and set to yet another value in a second PDB.

The query in this section lists the initialization parameters that you can set independently in each PDB.

To list the initialization parameters that are modifiable in each container:

1. In SQL*Plus, access a container.

   See "About Container Access in a CDB".

2. Run the following query:

```
SELECT NAME FROM V$SYSTEM_PARAMETER
  WHERE ISPDB_MODIFIABLE = 'TRUE'
  ORDER BY NAME;
```

If an initialization parameter listed by this query is not set independently for a PDB, then the PDB inherits the parameter value of the root.

> **See Also:**
>
> - "Modifying a CDB with ALTER SYSTEM"
> - "Modifying a PDB at the System Level"

Viewing the History of PDBs

The `CDB_PDB_HISTORY` view shows the history of the PDBs in a CDB. It provides information about when and how each PDB was created and other information about each PDB's history.

To view the history of each PDB:

1. In SQL*Plus, ensure that the current container is the root.

   See "Accessing a Container in a CDB with SQL*Plus".

2. Query `CDB_PDB_HISTORY` view.

Example 19-16 Viewing the History of PDBs

This example shows the following information about each PDB's history:

- The `DB_NAME` field shows the CDB that contained the PDB.
- The `CON_ID` field shows the container ID of the PDB.
- The `PDB_NAME` field shows the name of the PDB in one of its incarnations.
- The `OPERATION` field shows the operation performed in the PDB's history.
- The `OP_TIMESTAMP` field shows the date on which the operation was performed.
- If the PDB was cloned in an operation, then the `CLONED_FROM_PDB` field shows the PDB from which the PDB was cloned.

```
COLUMN DB_NAME FORMAT A10
COLUMN CON_ID FORMAT 999
COLUMN PDB_NAME FORMAT A15
COLUMN OPERATION FORMAT A16
COLUMN OP_TIMESTAMP FORMAT A10
COLUMN CLONED_FROM_PDB_NAME FORMAT A15

SELECT DB_NAME, CON_ID, PDB_NAME, OPERATION, OP_TIMESTAMP, CLONED_FROM_PDB_NAME
  FROM CDB_PDB_HISTORY
  WHERE CON_ID > 2
  ORDER BY CON_ID;
```

Sample output:

```
DB_NAME    CON_ID PDB_NAME        OPERATION        OP_TIMESTA CLONED_FROM_PDB
---------- ------ --------------- ---------------- ---------- ---------------
NEWCDB          3 HRPDB           CREATE           10-APR-12  PDB$SEED
NEWCDB          4 SALESPDB        CREATE           17-APR-12  PDB$SEED
NEWCDB          5 TESTPDB         CLONE            30-APR-12  SALESPDB
```

> **Note:**
>
> When the current container is a PDB, the `CDB_PDB_HISTORY` view shows the history of the current PDB only. A local user whose current container is a PDB can query the `DBA_PDB_HISTORY` view and exclude the `CON_ID` column from the query to view the history of the current PDB.

> **See Also:**
>
> "About the Current Container"

Viewing Information About Applications in Application Containers

Several views provide information about the applications in application containers in a CDB.

Related Topics

- Creating and Removing Application Containers
  You can create application containers in several different ways, including using the PDB seed, cloning an existing PDB or non-CDB, and plugging in an unplugged PDB. You can also remove application containers from a CDB.
- Administering Application Containers
  You can administer application containers, including application roots and application PDBs. You can also administer the applications installed in application containers.

Viewing Information About Applications

The `DBA_APPLICATIONS` view provides information about the applications in an application container.

> **Note:**
>
> The `DBA_APPLICATIONS` view provides information about the application in the current container only. To view information about applications in all of the application PDBs in the current application container, query the `DBA_APP_PDB_STATUS` with the application root as the current container.

To view information about the applications in an application container:

1. In SQL*Plus, access the application root of the application container.
2. Query the `DBA_APPLICATIONS` view.

Example 19-17 Viewing Details About the Applications in an Application Container

This query shows the name, the latest version, and the status of each user-created application in the application container.

```
COLUMN APP_NAME FORMAT A15
COLUMN APP_VERSION FORMAT A15
COLUMN APP_STATUS FORMAT A15

SELECT APP_NAME, APP_VERSION, APP_STATUS
FROM   DBA_APPLICATIONS
WHERE  APP_IMPLICIT='N';
```

The following sample output shows the `salesapp` application:

```
APP_NAME        APP_VERSION     APP_STATUS
--------------- --------------- ---------------
SALESAPP        1.2             NORMAL
```

> **Note:**
>
> Oracle Database creates some applications implicitly when an application common user operation is issued with a `CONTAINER=ALL` clause outside of `ALTER PLUGGABLE DATABASE APPLICATION BEGIN/END` statements. The sample query excludes implicitly-created applications by specifying `APP_IMPLICIT='N'` in the `WHERE` clause.

Related Topics

- Administering Application Containers
  You can administer application containers, including application roots and application PDBs. You can also administer the applications installed in application containers.
- Synchronizing Applications in an Application PDB
  Synchronizing an application updates the application in the application PDB to the latest version and patch in the application root.

Viewing Information About Application Status

The `DBA_APP_PDB_STATUS` view provides information about the status of the applications in an application container. It can show the status of each application in each application PDB.

The view can show the status of an application in an application PDB even if the application PDB is closed.

> **Note:**
>
> When queried from the application root, the DBA_APP_PDB_STATUS view provides information about the applications in all application PDBs in the current application container. To view information about the application in the current container only, query the DBA_APPLICATIONS view.

To view information about the application status in an application container:

1. In SQL*Plus, access the application root of the application container.

   See "About Container Access in a CDB".

2. Query the DBA_APP_PDB_STATUS view.

Example 19-18 Viewing Information About Application Status

This query shows the name of the application PDB, the name of the application, the version number of the application, and the status of the application.

```
COLUMN PDB_NAME FORMAT A15
COLUMN APP_NAME FORMAT A15
COLUMN APP_VERSION FORMAT A20
COLUMN APP_STATUS FORMAT A12

SELECT p.PDB_NAME, s.APP_NAME, s.APP_VERSION, s.APP_STATUS
  FROM DBA_PDBS p, DBA_APP_PDB_STATUS s
  WHERE p.CON_UID = s.CON_UID;
```

Your output is similar to the following:

```
PDB_NAME        APP_NAME        APP_VERSION          APP_STATUS
--------------- --------------- -------------------- ------------
SALES1          SALESAPP        4.2                  NORMAL
```

> **Note:**
>
> The status of an application can be NORMAL in an application PDB even when the application has not been synchronized to the latest version. Other statuses might indicate that an operation is in progress or that an operation encountered a problem. For example, the status UPGRADING might indicate that an upgrade of the application is in progress in the application PDB, or it might indicate that an error was encountered when the application PDB tried to upgrade an application.

> **See Also:**
>
> "Administering Application Containers"

Viewing Information About Application Statements

The DBA_APP_STATEMENTS view provides information about SQL statements issued during application installation, upgrade, and patch operations

Oracle Database records all of the SQL statements issued during application installation, upgrade, and patch operations, and you can view the history of these statements by querying the DBA_APP_STATEMENTS view.

To view information about the SQL statements issued during application operations:

1. In SQL*Plus, access the application root of the application container.

   See "About Container Access in a CDB".

2. Query the DBA_APP_STATEMENTS view.

Example 19-19 Viewing Information About Application Statements

This query shows the statement ID, capture time, SQL statement, and application name for the SQL statements for applications in the application container.

```
SET LONG  8000
SET PAGES 8000
COLUMN STATEM_ID FORMAT NNNNN
COLUMN CAPTURE_TIME FORMAT A12
COLUMN APP_STATEMENT FORMAT A36
COLUMN APP_NAME FORMAT A15

SELECT STATEMENT_ID AS STATEM_ID, CAPTURE_TIME, APP_STATEMENT, APP_NAME
FROM   DBA_APP_STATEMENTS
ORDER BY STATEMENT_ID;
```

Your output is similar to the following:

```
STATEM_ID CAPTURE_TIME APP_STATEMENT                        APP_NAME
--------- ------------ -----------------------------------
---------------
        1 30-AUG-15    SYS
APP$1E87C094764

1142FE0534018F8
                                                            0AA6C5
        2 30-AUG-15    ALTER PLUGGABLE DATABASE APPLICATION
APP$1E87C094764
                        APP$CON BEGIN INSTALL '1.0'
1142FE0534018F8
                                                            0AA6C5
        3 30-AUG-15    ALTER PLUGGABLE DATABASE APPLICATION
APP$1E87C094764
                        APP$CON END INSTALL '1.0'
1142FE0534018F8
                                                            0AA6C5
        4 30-AUG-15    SYS                                  SALESAPP
```

```
         5 30-AUG-15    ALTER PLUGGABLE DATABASE APPLICATION SALESAPP
                         salesapp BEGIN INSTALL '1.0'
         6 30-AUG-15    CREATE TABLE oe.cmtb SHARING=METADAT SALESAPP
                        A (
                           value   VARCHAR2(30),
                           country VARCHAR2(30))
         7 30-AUG-15    CREATE TABLE conmap (                SALESAPP
                            country VARCHAR2(30) NOT NULL)
                        PARTITION BY LIST (country) (
                        PARTITION AMER VALUES ('US','MEXICO'
                        ,'CANADA'),
                        PARTITION EURO VALUES ('UK','FRANCE'
                        ,'GERMANY'),
                        PARTITION ASIA VALUES ('INDIA','CHIN
                        A','JAPAN'))
         8 30-AUG-15    ALTER TABLE oe.cmtb ENABLE CONTAINER SALESAPP
                        _MAP
         9 30-AUG-15    ALTER PLUGGABLE DATABASE APPLICATION SALESAPP
                         salesapp END INSTALL '1.0'
.
.
.
```

> **Note:**
>
> Oracle Database creates some applications implicitly when an application common user operation is issued with a `CONTAINER=ALL` clause outside of `ALTER PLUGGABLE DATABASE APPLICATION BEGIN/END` statements. The names of these applications begin with `APP$`, and the sample output shows these applications.

> **See Also:**
>
> - "Administering Application Containers"
> - "Synchronizing Applications in an Application PDB"

Viewing Information About Application Versions

The `DBA_APP_VERSIONS` view provides information about the versions for applications in an application container.

Oracle Database records the versions for each application in an application container.

To view information about the application versions in an application container:

1. In SQL*Plus, access the application root of the application container.

   See "About Container Access in a CDB".

2. Query the `DBA_APP_VERSIONS` view.

Example 19-20 Viewing Information About Application Versions

This query shows the name of the application that was versioned, the version number, and the comment for the version.

```
COLUMN APP_NAME FORMAT A15
COLUMN APP_VERSION FORMAT A20
COLUMN APP_VERSION_COMMENT FORMAT A25

SELECT APP_NAME, APP_VERSION, APP_VERSION_COMMENT
  FROM DBA_APP_VERSIONS;
```

Your output is similar to the following:

```
APP_NAME        APP_VERSION          APP_VERSION_COMMENT
--------------- -------------------- -------------------------
SALESAPP        1.0                  Sales Application
```

> **See Also:**
>
> "Administering Application Containers"

Viewing Information About Application Patches

The `DBA_APP_PATCHES` view provides information about the patches for applications in an application container.

Oracle Database records the patches for each application in an application container.

To view information about the application patches in an application container:

1. In SQL*Plus, access the application root of the application container.

   See "About Container Access in a CDB".

2. Query the `DBA_APP_PATCHES` view.

Example 19-21 Viewing Information About Application Patches

This query shows the name of the application that was patched, the patch number, the minimum application version for the patch, and the status of the patch for each patch in the application container.

```
COLUMN APP_NAME FORMAT A15
COLUMN PATCH_NUMBER FORMAT NNNNNNNN
COLUMN PATCH_MIN_VERSION FORMAT A10
COLUMN PATCH_STATUS FORMAT A15

SELECT APP_NAME, PATCH_NUMBER, PATCH_MIN_VERSION, PATCH_STATUS
  FROM DBA_APP_PATCHES;
```

Your output is similar to the following:

```
APP_NAME        PATCH_NUMBER PATCH_MIN_ PATCH_STATUS
--------------- ------------ ---------- ---------------
SALESAPP                   1 1.2        INSTALLED
```

> **See Also:**
>
> "Administering Application Containers"

Viewing Information About Application Errors

The DBA_APP_ERRORS view provides information errors raised when an application PDB synchronizes with an application in the application root.

An application PDB issues the ALTER PLUGGABLE DATABASE APPLICATION statement with the SYNC clause. You can view the history of application errors during application synchronization by querying the DBA_APP_ERRORS view.

To view information about errors raised during application synchronization:

1. In SQL*Plus, access the application root of the application container.

   See "About Container Access in a CDB".

2. Query the DBA_APP_ERRORS view.

Example 19-22 Viewing Details About Errors Raised During Application Synchronization

This query shows the application name, the SQL statement that raised the error, the error number, and the error message for errors raised during application synchronization.

```
SET LONG  8000
SET PAGES 8000
COLUMN APP_NAME FORMAT A15
COLUMN APP_STATEMENT FORMAT A36
COLUMN ERRORNUM FORMAT NNNNNNNN
COLUMN ERRORMSG FORMAT A20

SELECT APP_NAME, APP_STATEMENT, ERRORNUM, ERRORMSG
  FROM DBA_APP_ERRORS;
```

> **See Also:**
>
> "Administering Application Containers"

Listing the Shared Database Objects in an Application Container

The DBA_OBJECTS view can list the shared database objects in an application container.

Shared database objects are metadata-linked application common objects, data-linked application common objects, and extended data-linked application common objects.

To list the shared database objects in an application container:

1. In SQL*Plus, access the application root of the application container.

   See "About Container Access in a CDB".

2. Query the DBA_OBJECTS view and specify the SHARING column.

Example 19-23 Listing the User-Created Shared Database Objects in an Application Container

This query shows the owner and name of the user-created shared database objects in the application container. It also shows whether each shared database object is a metadata-linked application common object or a data-linked application common object. The query excludes Oracle-supplied shared database objects.

```
COLUMN OWNER FORMAT A15
COLUMN OBJECT_NAME FORMAT A25
COLUMN SHARING FORMAT A13

SELECT OWNER, OBJECT_NAME, SHARING
   FROM DBA_OBJECTS WHERE SHARING != 'NONE'
   AND ORACLE_MAINTAINED = 'N';
```

Your output is similar to the following:

```
OWNER           OBJECT_NAME               SHARING
--------------- ------------------------- -------------
SALESADM        CONMAP                    METADATA LINK
OE              PRODUCT_DESCRIPTIONS_OB   DATA LINK
OE              CMTB                      METADATA LINK
```

> **See Also:**
>
> "Managing Application Common Objects"

Listing the Extended Data-Linked Objects in an Application Container

The DBA_TABLES and DBA_VIEWS views can list the extended data-linked objects in an application container.

An extended data-linked object is a special type of data-linked object for which each application PDB can create its own specific data while sharing the common data in the application root. Only the data stored in the application root is common for all application PDBs.

To list the extended data-linked objects in an application container:

1. In SQL*Plus, access the application root of the application container.

   See "About Container Access in a CDB".

2. Query the `DBA_TABLES` or `DBA_VIEWS` view and specify the `EXTENDED_DATA_LINK='YES'` in the `WHERE` clause.

Example 19-24 Listing the Extended Data-Linked Tables in an Application Container

This query shows the owner and name of the extended data-linked tables in the application container.

```
COLUMN OWNER FORMAT A20
COLUMN TABLE_NAME FORMAT A30

SELECT OWNER, TABLE_NAME FROM DBA_TABLES WHERE EXTENDED_DATA_LINK='YES';
```

Your output is similar to the following:

```
OWNER                TABLE_NAME
-------------------- ------------------------------
SALESADM             ZIPCODES
```

> **See Also:**
>
> "Managing Application Common Objects"

Part V

Using Oracle Features in a Multitenant Environment

You can use Oracle Database features in a multitenant environment.

20

Backing Up and Recovering CDBs and PDBs

You can back up and recover multitenant container databases (CDBs) and pluggable databases (PDBs).

Overview of Backing Up and Recovering CDBs and PDBs

When using the multitenant architecture, you can perform backup and recovery operations on a whole multitenant container database (CDB), the root, or one or more pluggable databases (PDBs).

The Oracle Recovery Manager (RMAN) commands used to backup and recover CDBs and PDBs are the same as those used for non-CDBs, with minor variations in the syntax. The backup and recovery operations performed on non-CDBs can also be performed on CDBs and PDBs. This includes the following:

- Full and incremental backups
- Complete and point-in-time recovery (PITR)
- Flashback Database
- Reporting operations (such as listing backups and cross-checking backups)

About Connecting to CDBs and PDBs

You can connect to the root in one of the following ways*:*

- Connect using operating system authentication

  You are connected to the root as the `SYS` user with the `SYSDBA` privilege.
- Connect locally as a common user
- Connect as a common user through Oracle Net Services

To connect as `TARGET` to a PDB, use one of the following techniques:

- Connect with a net service name that resolves to a database service for that PDB
- Connect locally as a common user or local user with the `SYSDBA` or `SYSBACKUP` privilege

> **Note:**
>
> Certain operations are not available when you connect directly to a PDB. See *Oracle Database Backup and Recovery User's Guide* for a list of these operations.

> **See Also:**
>
> The following sections in the *Oracle Database Backup and Recovery User's Guide* provide detailed information about backing up and recovering PDBs:
>
> - Connecting as Target to the Root
> - Connecting as Target to a PDB

Backup and Complete Recovery of CDBs

To perform backup and complete recovery operations on a whole multitenant container database (CDB), you connect as `TARGET` to the root.

The connection must be established as a common user with the `SYSDBA` or `SYSBACKUP` privilege.

After you connect to the root, the same commands that are used to perform operations on non-CDBs are used to perform backup and complete recovery on the entire CDB.

> **See Also:**
>
> The following sections in the *Oracle Database Backup and Recovery User's Guide* provide detailed information about performing backup and complete recovery of CDBs:
>
> - "Backing Up a Whole CDB"
> - "Performing Complete Recovery of a Whole CDB"
> - "Validating a Whole CDB"
> - "Reporting in CDBs"

Backup and Complete Recovery of PDBs

You can perform backup and complete recovery operations on a single pluggable database (PDB) or on multiple PDBs.

Backups of PDBs

When relocating a PDB or cloning a non-CDB as a PDB, you may want to retain the use of preplugin backups. For preplugin backups to be usable in the destination CDB, metadata about the preplugin backups must be exported to the RMAN repository of the destination CDB.

The technique for making the backups usable depends on the type of operation:

- Creating a PDB by cloning a non-CDB

  When the non-CDB is opened in read/write mode, you must execute the `DBMS_PDB.EXPORTRMANBACKUP` procedure as the last step before cloning. When plugging in the non-CDB as a PDB to a destination CDB, the operation copies the

backup metadata of the source non-CDB into the data dictionary of the destination CDB.

- Relocating a PDB to another CDB

  When you unplug the source PDB, the backup metadata is automatically exported. Therefore, you do not need to execute `DBMS_PDB.EXPORTRMANBACKUP`.

Preplugin backups are usable only on the destination CDB into which you plug in the source non-CDB or PDB.

> **Note:**
>
> - *Oracle Database Backup and Recovery User's Guide* to learn about preplugin backups
> - *Oracle Database Backup and Recovery User's Guide* to learn how to create a preplugin backup of the whole database
> - *Oracle Database PL/SQL Packages and Types Reference* to learn more about the `DBMS_PDB.EXPORTRMANBACKUP` procedure

Syntax for Backup Commands

Although the Oracle Recovery Manager (RMAN) commands are the same, the syntax used to perform operations on multiple PDBs contains some modifications.

To perform backup and complete recovery operations on a single PDB, you can connect as `TARGET` to either of the following containers:

- PDB

  In this case, use the same commands that you would use to backup or recover non-CDBs. For example, to back up a PDB, use the `BACKUP DATABASE` command.

- `CDB$ROOT`

  In this case, use the `PLUGGABLE DATABASE` clause in your RMAN commands. The following command backs up the PDB `hrpdb` when connected to the root:

  ```
  BACKUP PLUGGABLE DATABASE hrpdb;
  ```

  To perform backup and complete recovery operations on multiple PDBs using a single command, you must connect to the root. Use the `PLUGGABLE DATABASE` clause followed by the list of PDBs on which you want to perform the operation. The following example backs up the PDBs `hrpdb`, `salespdb`, and `invpdb` when connected to the root:

  ```
  BACKUP PLUGGABLE DATABASE hrpdb, salespdb, invpdb;
  ```

See Also:

The following sections in the *Oracle Database Backup and Recovery User's Guide* provide detailed information about backing up and recovering PDBs:

- "Connecting as Target to a PDB"
- "Connecting as Target to the Root"
- "Backing Up PDBs with RMAN"
- "Performing Complete Recovery of PDBs with RMAN"
- "Validating PDBs"
- "Reporting in PDBs"

Point-in-Time Recovery in a Multitenant Environment

You can perform point-in-time recovery of the whole multitenant container database (CDB) or a particular pluggable database (PDB).

Point-in-Time Recovery of a CDB

To perform point-in-time recovery of a CDB, you must meet the following prerequisites:

- You must be logged in to the root container as a common user with the `SYSDBA` or `SYSBACKUP` privilege.
- The CDB must be mounted.

When performing the recovery operation, use the same commands that you use for non-CDBs.

Point-in-Time Recovery of a PDB

When a PDB is closed in an open or closed CDB, you can recover the PDB to a past point in time. The technique depends on the undo mode of the CDB. The following table describes the differences.

Table 20-1 Differences in Point-in-Time Recovery Techniques

Undo Mode	Auxiliary Instance Used?	Connect as TARGET to ...	RMAN Commands to Use for Recovery
Shared	Yes	CDB root	Include the `PLUGGABLE DATABASE` clause to specify the PDB that must be recovered. RMAN uses an auxiliary destination to store temporary files created during recovery. If a fast recovery area has been configured, then it is used as the auxiliary destination. You can explicitly specify an auxiliary destination using the `AUXILIARY DESTINATION` clause in the `RECOVER` command.

Table 20-1 (Cont.) Differences in Point-in-Time Recovery Techniques

Undo Mode	Auxiliary Instance Used?	Connect as TARGET to ...	RMAN Commands to Use for Recovery
Local	No	CDB root or PDB	When connected to the PDB, use the same commands that you use for non-CDBs. When connected to the root, include the `PLUGGABLE DATABASE` clause to specify the PDB that must be recovered.

See Also:

The following sections in the *Oracle Database Backup and Recovery User's Guide* for more information about point-in-time recovery.

- "Overview of Restore Points in a Multitenant Environment"
- "Creating CDB Restore Points"
- "Creating PDB Restore Points"
- "Performing Point-in-Time Recovery of CDBs and PDBs"

Flashback Database in a Multitenant Environment

You can perform a Flashback Database operation for a whole multitenant container database (CDB) or for a particular pluggable (PDB).

RMAN uses an auxiliary destination to store temporary files created during point-in-time recovery. By default, the fast recovery area is used as the auxiliary destination. You can explicitly specify an auxiliary destination using the `AUXILIARY DESTINATION` clause in the `RECOVER` command.

Flashback of CDBs

To perform Flashback Database for a CDB, you must meet the following prerequisites:

- You must be connected to the root as a common user with the `SYSDBA` or `SYSBACKUP` privilege.
- The CDB must be mounted.

Specify the target point in time for the flashback operation using a CDB restore point, time expression, or SCN. A CDB restore point is accessible to every PDB within the CDB. However, the restore point does not reflect the PDB sub-incarnation of any of its PDBs.

Flashback of PDBs

When a PDB is closed and the CDB is open, you can perform a flashback database operation for this PDB using the `FLASHBACK DATABASE` command. Performing a Flashback Database operation on a particular PDB modifies only data files related to that PDB. The other PDBs in the CDB are not impacted and are available for use.

Note that a PDB restore point is accessible only to the PDB in which it is defined and can be used for operations only on this PDB.

Table 20-2 Differences in Flashback Techniques

CDB Undo Mode	Auxiliary Instance Used?	Connect as TARGET to ...	Commands
Shared	Yes	CDB root	Use the `FLASHBACK PLUGGABLE DATABASE` command. You can only flash back to a clean PDB restore point. RMAN uses an auxiliary destination to store temporary files created during flashback. If a fast recovery area has been configured, then it is used as the auxiliary destination. You can explicitly specify an auxiliary destination using the `AUXILIARY DESTINATION` clause in the `FLASHBACK PLUGGABLE DATABASE` command.
Local	No	CDB root or PDB	Use the `FLASHBACK PLUGGABLE DATABASE` command. You can specify the target point in time for the flashback operation using a CDB restore point, PDB restore point, time expression, or target SCN.

See Also:

The following sections in the *Oracle Database Backup and Recovery User's Guide* for more information about flashback of CDBs and PDBs:

- "Overview of Restore Points in a Multitenant Environment"
- "Creating CDB Restore Points"
- "Creating PDB Restore Points"
- "Performing a Flashback Database Operation for a Whole CDB"
- "Performing a Flashback Database Operation for PDBs"

21

Using Database Utilities in a Multitenant Environment

You can use utilities such as Oracle Data Pump, DBNEWID, and Oracle LogMiner in a multitenant environment.

Importing and Exporting Data in a CDB

Oracle Data Pump technology enables very high-speed movement of data and metadata from one database to another.

About Using Data Pump in a Multitenant Environment

In general, using Data Pump with PDBs is identical to using Data Pump with a non-CDB.

A multitenant container database (CDB) is an Oracle database that includes zero, one, or many user-created pluggable databases (PDBs). A PDB is a portable set of schemas, schema objects, and nonschema objects that appear to an Oracle Net client as a non-CDB. A non-CDB is an Oracle database that is not a CDB.

You can use Data Pump to migrate all or some of a database in the following scenarios:

- From a non-CDB into a PDB
- Between PDBs within the same or different CDBs
- From a PDB into a non-CDB

> **Note:**
>
> Data Pump does not support any CDB-wide operations. If you are connected to the root or seed database of a CDB, then Data Pump issues the following warning:
>
> ```
> ORA-39357: Warning: Oracle Data Pump operations are not
> typically needed when connected to the root or seed of a
> container database.
> ```

Using Data Pump to Move Data Into a CDB

After you create an empty PDB, you can use an Oracle Data Pump full-mode export and import operation to move data into the PDB.

You can import data with or without the transportable option. If you use the transportable option on a full mode export or import, then it is referred to as a full transportable export/import.

When the transportable option is used, export and import use both transportable tablespace data movement and conventional data movement; the latter for those tables that reside in non-transportable tablespaces such as `SYSTEM` and `SYSAUX`. Using the transportable option can reduce the export time and especially, the import time, because table data does not need to be unloaded and reloaded and index structures in user tablespaces do not need to be recreated.

Note the following requirements when using Data Pump to move data into a CDB:

- To administer a multitenant environment, you must have the `CDB_DBA` role.
- Full database exports from Oracle Database 11.2.0.2 and earlier can be imported into Oracle Database 12c (CDB or non-CDB). However, Oracle recommends that you first upgrade the source database to Oracle Database 11g release 2 (11.2.0.3 or later), so that information about registered options and components is included in the export.
- When migrating Oracle Database 11g release 2 (11.2.0.3 or later) to a CDB (or to a non-CDB) using either full database export or full transportable database export, you must set the Data Pump Export parameter `VERSION=12` in order to generate a dump file that is ready for import into Oracle Database 12c. If you do not set `VERSION=12`, then the export file that is generated does not contain complete information about registered database options and components.
- Network-based full transportable imports require use of the `FULL=YES`, `TRANSPORTABLE=ALWAYS`, and `TRANSPORT_DATAFILES=`*`datafile_name`* parameters. When the source database is Oracle Database 11g release 11.2.0.3 or later, but earlier than Oracle Database 12c Release 1 (12.1), the `VERSION=12` parameter is also required.
- File-based full transportable imports only require use of the `TRANSPORT_DATAFILES=`*`datafile_name`* parameter. Data Pump Import infers the presence of the `TRANSPORTABLE=ALWAYS` and `FULL=YES` parameters.
- As of Oracle Database 12c release 2 (12.2), in a multitenant container database (CDB) environment, the default Data Pump directory object, `DATA_PUMP_DIR`, is defined as a unique path for each PDB in the CDB. This unique path is defined whether the `PATH_PREFIX` clause of the `CREATE PLUGGABLE DATABASE` statement is defined or is not defined for relative paths.
- Starting in Oracle Database 19c, the `credential` parameter of `impdp` specifies the name of the credential object that contains the user name and password required to access an object store bucket. You can also specify a default credential using the PDB property named `DEFAULT_CREDENTIAL`. When you run `impdb` with then default credential, you prefix the dump file name with `DEFAULT_CREDENTIAL:` and you do not specify the `credential` parameter.

Example 21-1 Importing a Table into a PDB

To specify a particular PDB for the export/import operation, supply a connect identifier in the connect string when you start Data Pump. For example, to import data to a PDB named `pdb1`, you could enter the following on the Data Pump command line:

```
impdp hr@pdb1 DIRECTORY=dpump_dir1 DUMPFILE=hr.dmp TABLES=employees
```

Example 21-2 Specifying a Credential When Importing Data

This example assumes that you created a credential named `HR_CRED` using `DBMS_CREDENTIAL.CREATE_CREDENTIAL` as follows:

```
BEGIN
  DBMS_CLOUD.CREATE_CREDENTIAL(
    credential_name => 'HR_CRED',
    username => 'atpc_user@oracle.com',
    password => 'password'
  );
END;
/
```

The following command specifies credential `HR_CRED`, and specifies the a file stored in an object store. The URL of the file is `https://example.com/ostore/dnfs/myt.dmp`.

```
impdp hr@pdb1 \
      table_exists_action=replace \
      credential=HR_CRED \
      parallel=16 \
      dumpfile=https://example.com/ostore/dnfs/myt.dmp
```

Example 21-3 Importing Data Using a Default Credential

1. You create a credential named `HR_CRED` using `DBMS_CREDENTIAL.CREATE_CREDENTIAL` as follows:

   ```
   BEGIN
     DBMS_CLOUD.CREATE_CREDENTIAL(
       credential_name => 'HR_CRED',
       username => 'atpc_user@oracle.com',
       password => 'password'
     );
   END;
   /
   ```

2. You set the PDB property `DEFAULT_CREDENTIAL` as follows:

   ```
   ALTER DATABASE PROPERTY SET DEFAULT_CREDENTIAL = 'ADMIN.HR_CRED'
   ```

3. The following command specifies the default credential as a prefix to the dump file location `https://example.com/ostore/dnfs/myt.dmp`:

   ```
   impdp hr@pdb1 \
         table_exists_action=replace \
         parallel=16 \
         dumpfile=default_credential:https://example.com/ostore/dnfs/
   myt.dmp
   ```

   Note that the `credential` parameter is not specified.

See Also:

- *Oracle Database Security Guide* to learn how to configure SSL authentication, which is necessary for object store access
- *Oracle Database Utilities* to learn about using Data Pump Import to load files to the object store

Using Data Pump to Move PDBs Within Or Between CDBs

Data Pump export and import operations on PDBs are identical to those on non-CDBs, with the exception of how common users are handled.

If you have created a common user in a CDB, then a full database or privileged schema export of that user from within any PDB in the CDB results in a standard `CREATE USER C##common name` DDL statement being performed upon import. The statement will fail because of the common user prefix `C##` on the user name. The following error message will be returned:

```
ORA-65094:invalid local user or role name
```

In the PDB being exported, if you have created local objects in that user's schema and you want to import them, then either make sure a common user of the same name already exists in the target CDB instance or use the Data Pump Import `REMAP_SCHEMA` parameter on the `impdp` command, as follows:

```
REMAP_SCHEMA=C##common name:local user name
```

Related Topics

- *Oracle Database Utilities*
- *Oracle Database Utilities*
- *Oracle Database Utilities*

Using LogMiner in a CDB

You can use LogMiner in a multitenant container database (CDB).

The following sections discuss some differences to be aware of when using LogMiner in a CDB versus a non-CDB:

LogMiner supports CDBs that have PDBs of different character sets provided the root container has a character set that is a superset of all the PDBs.

To administer a multitenant environment you must have the `CDB_DBA` role.

LogMiner V$ Views and DBA Views in a CDB

In a CDB, views used by LogMiner to show information about LogMiner sessions running in the system contain an additional column named `CON_ID`.

The CON_ID column identifies the container ID associated with the session for which information is being displayed. When you query the view from a pluggable database (PDB), only information associated with the database is displayed. The following views are affected by this new behavior:

- V$LOGMNR_DICTIONARY_LOAD
- V$LOGMNR_LATCH
- V$LOGMNR_PROCESS
- V$LOGMNR_SESSION
- V$LOGMNR_STATS

> **Note:**
>
> To support CDBs, the V$LOGMNR_CONTENTS view has several other new columns in addition to CON_ID.

The following DBA views have analogous CDB views whose names begin with CDB.

Type of Log View	DBA View	CDB View
LogMiner Log Views	DBA_LOGMNR_LOG	CDB_LOGMNR_LOG
LogMiner Purged Log Views	DBA_LOGMNR_PURGED_LOG	CDB_LOGMNR_PURGED_LOG
LogMiner Session Log Views	DBA_LOGMNR_SESSION	CDB_LOGMNR_SESSION

The DBA views show only information related to sessions defined in the container in which they are queried.

The CDB views contain an additional CON_ID column, which identifies the container whose data a given row represents. When CDB views are queried from the root, they can be used to see information about all containers.

The V$LOGMNR_CONTENTS View in a CDB

In a CDB, the V$LOGMNR_CONTENTS view and its associated functions are restricted to the root database. Several new columns exist in V$LOGMNR_CONTENTS in support of CDBs.

- CON_ID — contains the ID associated with the container from which the query is executed. Because V$LOGMNR_CONTENTS is restricted to the root database, this column returns a value of 1 when a query is done on a CDB.
- SRC_CON_NAME — the PDB name. This information is available only when mining is performed with a LogMiner dictionary.
- SRC_CON_ID — the container ID of the PDB that generated the redo record. This information is available only when mining is performed with a LogMiner dictionary.
- SRC_CON_DBID — the PDB identifier. This information is available only when mining is performed with a current LogMiner dictionary.

- SRC_CON_GUID — contains the GUID associated with the PDB. This information is available only when mining is performed with a current LogMiner dictionary.

Enabling Supplemental Logging in a CDB

In a CDB, the syntax for enabling and disabling database-wide supplemental logging using the ALTER DATABASE command.

For example, use the following syntax when adding or dropping supplemental log data:

```
ALTER DATABASE [ADD|DROP] SUPPLEMENTAL LOG DATA ...
```

Note the following:

- In a CDB, supplemental logging levels that are enabled from CDB$ROOT are enabled across the CDB.
- If at least minimal supplemental logging is enabled in CDB$ROOT, then additional supplemental logging levels can be enabled at the PDB level.
- Supplemental logging levels enabled at the CDB level from CDB$ROOT cannot be disabled at the PDB level.
- Dropping all supplemental logging from CDB$ROOT disables all supplemental logging across the CDB regardless of previous PDB level settings.

Supplemental logging operations started with CREATE TABLE and ALTER TABLE statements can be executed from either the CDB root or a PDB. They affect only the table to which they are applied.

Using a Flat File Dictionary in a CDB

You cannot take a dictionary snapshot for an entire CDB in a single flat file. You must be connected to a distinct PDB, and can take a snapshot of only that PDB in a flat file.

Thus, when using a flat file dictionary, you can only mine the redo logs for the changes associated with the PDB whose data dictionary is contained within the flat file.

DBNEWID Considerations for CDBs and PDBs

The DBNEWID parameter PDB allows you to change the DBID on pluggable databases (PDBs).

By default, when you run the DBNEWID utility on a container database (CDB) it changes the DBID of only the CDB; the DBIDs of the pluggable databases (PDBs) comprising the CDB remain the same. This could cause problems with duplicate DBIDs for PDBs in some cases, such as when a CDB is cloned.

As of Oracle Database 12*c* Release 2 (12.2), you can change the DBID on the PDBs by using the new DBNEWID PDB parameter. You cannot specify a particular PDB; either all of them or none of them will have new DBIDs. The PDB parameter is applicable only in a CDB environment. It has the following format:

```
PDB=[ALL | NONE]
```

- If you specify `ALL`, then in addition to the DBID for the CDB changing, the DBIDs for all PDBs comprising the CDB are also changed.
- Specifying `NONE` (the default) leaves the PDB DBIDs the same, even if the CDB DBID is changed.

Oracle recommends that you use `PDB=ALL`, but `PDB=NONE` is the default for backward compatibility reasons.

22

Using Oracle Resource Manager for PDBs

Use PL/SQL package procedures to administer Oracle Resource Manager (Resource Manager) to allocate resources to pluggable databases (PDBs) in a multitenant container database (CDB).

This chapter assumes that you meet the following prerequisites:

- You understand how to configure and manage a CDB.
- You understand how to use Resource Manager to allocate resources in a non-CDB.

> **Note:**
>
> - You can complete the tasks in this chapter using SQL*Plus or Oracle SQL Developer.
> - You can also administer the Resource Manager with the graphical user interface of Oracle Enterprise Manager Cloud Control (Cloud Control).
> - For simplicity, this chapter refers to PDBs, application roots, and application PDBs as “PDBs.”

> **See Also:**
>
> - "Administering a Multitenant Environment"
> - *Oracle Database Administrator’s Guide* to learn more about Resource Manager

Overview of Oracle Resource Manager in a Multitenant Environment

In a CDB, workloads within multiple PDBs can compete for system and CDB resources. Resource plans solve this problem.

In a multitenant environment, Resource Manager operates on two levels:

- CDB level

  Resource Manager can manage the 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. The principal tool is a CDB resource plan.

- PDB level

  Resource Manager can manage the workloads within each PDB. The principal tool is a PDB resource plan.

Resource Manager allocates the 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 system resources obtained in the preceding step to each session connected to the PDB.

> **Note:**
>
> Resource Manager manages activity in the root automatically.

To use Resource Manager in a multitenant environment, you must meet the following prerequisites:

- The CDB must exist and must contain PDBs.
- To complete a task that uses the `DBMS_RESOURCE_MANAGER` package, you must have `ADMINISTER_RESOURCE_MANAGER` system privilege.

> **See Also:**
>
> - " Creating and Configuring a CDB" and "Creating and Removing PDBs and Application Containers"
> - *Oracle Database Administrator's Guide* to learn more about the `DBMS_RESOURCE_MANAGER` package

Purpose of Resource Management in a Multitenant Environment

Resource Manager can provide more efficient use of resources for a CDB.

When resource allocation decisions for a CDB are left to the operating system, you may encounter the following problems with workload management:

- Inappropriate allocation of resources among PDBs

  The operating system distributes resources equally among all active processes and cannot prioritize one task over another. Therefore, one or more PDBs might use an inordinate amount of the system resources, leaving the other PDBs starved for resources.
- Inappropriate allocation of resources within a single PDB

  One or more sessions connected to a single PDB might use an inordinate amount of the system resources, leaving other sessions connected to the same PDB starved for resources.
- Inconsistent performance of PDBs

A single PDB might perform inconsistently when other PDBs are competing for more system resources or less system resources at various times.

- Lack of resource usage data for PDBs

  Resource usage data is critical for monitoring and tuning PDBs. It might be possible to use operating system monitoring tools to gather the resource usage data for a non-CDB if it is the only database running on the system. However, in a CDB, operating system monitoring tools are no longer as useful because there are multiple PDBs running on the system.

Resource Manager helps to overcome these problems by allowing the CDB more control over how hardware resources are allocated among the PDBs and within PDBs.

In a CDB with multiple PDBs, some PDBs typically are more important than others. The Resource Manager enables you to prioritize and limit the resource usage of specific PDBs. With the Resource Manager, you can:

- Specify that different PDBs should receive different shares of the system resources so that more resources are allocated to the more important PDBs
- Limit the CPU usage of a particular PDB
- Limit the number of parallel execution servers that a particular PDB can use
- Limit the memory usage of a particular PDB
- Specify the amount of memory guaranteed for a particular PDB
- Specify the maximum amount of memory a particular PDB can use
- Use PDB performance profiles for different sets of PDB

  A performance profile for a set of PDBs can specify shares of system resources, CPU usage, and number of parallel execution servers. PDB performance profiles enable you to manage resources for large numbers of PDBs by specifying Resource Manager directives for profiles instead of individual PDBs.
- Limit the resource usage of different sessions connected to a single PDB
- Limit the I/O generated by specific PDBs
- Monitor the resource usage of PDBs

Overview of Resource Plan Directives

A **CDB resource plan** allocates resources to its PDBs according to its set of resource plan directives (directives).

A parent-child relationship exists between a CDB resource plan and its resource plan directives. Each directive references either a set of PDBs in a performance profile, or a single PDB.

You can specify directives for both individual PDBs and for PDB performance profiles in the same CDB. No two directives for the currently active plan can reference the same PDB or the same PDB performance profile.

PDB Performance Profiles

A **PDB performance profile** configures resource plan directives for a set of PDBs that have the same priorities and resource controls.

For example, you might create a performance profiles called Gold, Silver, and Bronze. Each profile specifies a different set of directives depending on the importance of the type of PDB. Gold PDBs are more mission critical than Silver PDBs, which are more mission critical than Bronze PDBs. A PDB specifies its performance profile with the `DB_PERFORMANCE_PROFILE` initialization parameter.

You can use PDB lockdown profiles to specify PDB initialization parameters that control resources, such as `SGA_TARGET` and `PGA_AGGREGATE_LIMIT`. A lockdown profile prevents the PDB administrator from modifying the settings.

Oracle recommends using matching names for performance profiles and lockdown profiles. To prevent PDB owners from switching profiles, Oracle recommends putting the PDB performance profile in the PDB lockdown profile.

Resource Plan Directives

Directives control allocation of CPU and parallel execution servers.

A directive can control the allocation of resources to PDBs based on the share value that you specify for each PDB or PDB performance profile. A higher share value results in more resources. For example, you can specify that one PDB is allocated double the resources allocated to a second PDB by setting the share value for the first PDB twice as high as the share value for the second PDB. Similarly, you can specify that one PDB performance profile is allocated double the resources allocated to a second PDB performance profile by setting the share value for the first PDB performance profile twice as high as the share value for the second PDB performance profile. The settings apply to the set of PDBs that use each profile.

You can also specify utilization limits for PDBs and PDB performance profiles. The limit controls allocation to the PDB or performance profile. For example, the limit can control how much CPU a PDB gets as a percentage of the total CPU available to the CDB.

You can use both shares and utilization limits together for precise control over the resources allocated to each PDB and PDB performance profile in a CDB.

> **See Also:**
>
> "About Restricting PDB Users for Enhanced Security" for more information about PDB lockdown profiles

Background and Administrative Tasks and Consumer Groups

In a CDB, background and administrative tasks map to the Resource Manager consumer groups that run them optimally.

Resource Manager uses the following rules to map a task to a consumer group:

- A task is mapped to a consumer group in the container that starts the task.

  If a task starts in the CDB root, then the task maps to a consumer group in the CDB root. If the task starts in a PDB, then the task maps to a consumer group in the PDB.

- Many maintenance and administrative tasks automatically map to a consumer group.

  For example, automated maintenance tasks map to `ORA$AUTOTASK`. In certain cases, the tasks map to a consumer group, but the mapping is modifiable. Such tasks include RMAN backup, RMAN image copy, Oracle Data Pump, and In-Memory population.

> **Note:**
>
> *Oracle Database Administrator's Guide* to learn more about the mapping rules for predefined consumer groups

Initialization Parameters for PDB-Level Resources

Use initialization parameters to control CPU, memory, sessions, and I/O in a PDB.

Memory-Related Initialization Parameters for PDBs

Several initialization parameters control the memory usage of a PDB.

When the PDB is the current container, the initialization parameters in the following table control the memory usage of the current PDB. When one or more of these parameters is set for a PDB, ensure that the CDB and the other PDBs have sufficient memory for their operations. The initialization parameters control the memory usage of PDBs only if the following conditions are met:

- The `NONCDB_COMPATIBLE` initialization parameter is set to `false` in the CDB root.
- The `MEMORY_TARGET` initialization parameter is not set or is set to `0` (zero) in the CDB root.

Table 22-1 Initialization Parameters That Control the Memory Usage of PDBs

Initialization Parameter	Description
DB_CACHE_SIZE	Sets the minimum, guaranteed buffer cache space for the PDB. The following requirements must be met: • It must be less than or equal to 50% of the setting for the DB_CACHE_SIZE in the CDB root. • The sum of the DB_CACHE_SIZE settings for all PDBs must be less than or equal to 50% of the setting for the DB_CACHE_SIZE in the CDB root. These requirements do not apply if the SGA_TARGET initialization parameter is set to a nonzero value in the CDB root. When the SGA_TARGET initialization parameter is set to a nonzero, the following requirements must be met: • The values of DB_CACHE_SIZE plus SHARED_POOL_SIZE in a PDB must be less than or equal to 50% of the PDB's SGA_TARGET value. • The values of DB_CACHE_SIZE plus SHARED_POOL_SIZE in a PDB must be less than or equal to 50% of the SGA_TARGET value at the CDB level. • The sum of DB_CACHE_SIZE plus SHARED_POOL_SIZE across all the PDBs in a CDB must be less than or equal to 50% of the SGA_TARGET value at the CDB level.
SHARED_POOL_SIZE	Sets the minimum, guaranteed shared pool space for the PDB. The following requirements must be met: • It must be less than or equal to 50% of the setting for the SHARED_POOL_SIZE in the CDB root. • The sum of the SHARED_POOL_SIZE settings for all PDBs must be less than or equal to 50% of the setting for the SHARED_POOL_SIZE in the CDB root. These requirements do not apply if the SGA_TARGET initialization parameter is set to a nonzero value in the CDB root. When the SGA_TARGET initialization parameter is set to a nonzero, the following requirements must be met: • The values of DB_CACHE_SIZE plus SHARED_POOL_SIZE in a PDB must be less than or equal to 50% of the PDB's SGA_TARGET value. • The values of DB_CACHE_SIZE plus SHARED_POOL_SIZE in a PDB must be less than or equal to 50% of the SGA_TARGET value at the CDB level. • The sum of DB_CACHE_SIZE plus SHARED_POOL_SIZE across all the PDBs in a CDB must be less than or equal to 50% of the SGA_TARGET value at the CDB level.

Table 22-1 (Cont.) Initialization Parameters That Control the Memory Usage of PDBs

Initialization Parameter	Description
SGA_MIN_SIZE	Sets the minimum SGA size for the PDB. The following requirements must be met: • It must be less than or equal to 50% of the setting for the SGA_TARGET in the CDB root. • It must be less than or equal to 50% of the setting for the SGA_TARGET in the PDB. • The sum of the SGA_MIN_SIZE settings for all PDBs must be less than or equal to 50% of the setting for the SGA_TARGET in the CDB root. These requirements do not apply if the SGA_TARGET initialization parameter is not set or is set to 0 (zero) in the CDB root.
SGA_TARGET	Sets the maximum SGA size for the PDB. The SGA_TARGET setting in the PDB is enforced only if the SGA_TARGET initialization parameter is set to a nonzero value in the CDB root. The SGA_TARGET setting in the PDB must be less than or equal to the SGA_TARGET setting in the CDB root.
PGA_AGGREGATE_LIMIT	Sets the maximum PGA size for the PDB. The following requirements must be met: • It must be less than or equal to the setting for the PGA_AGGREGATE_LIMIT in the CDB root. • It must be greater than or equal to two times the setting for the PGA_AGGREGATE_TARGET in the PDB.
PGA_AGGREGATE_TARGET	Sets the target aggregate PGA size for the PDB. The following requirements must be met: • It must be less than or equal to the PGA_AGGREGATE_TARGET value set at the CDB level. • It must be less than or equal to 50% of the PGA_AGGREGATE_LIMIT initialization parameter value set at the CDB level. • It must be less than or equal to 50% of the PGA_AGGREGATE_LIMIT value set in the PDB.

Example 22-1 Setting the Maximum Aggregate PGA Memory Available for a PDB

With the PDB as the current container, run the following SQL statement to set the PGA_AGGREGATE_LIMIT initialization parameter both in memory and in the SPFILE to 90 MB:

```
ALTER SYSTEM SET PGA_AGGREGATE_LIMIT = 90M SCOPE = BOTH;
```

Example 22-2 Setting the Minimum SGA Size for a PDB

With the PDB as the current container, run the following SQL statement to set the `SGA_MIN_SIZE` initialization parameter both in memory and in the SPFILE to 500 MB:

```
ALTER SYSTEM SET SGA_MIN_SIZE = 500M SCOPE = BOTH;
```

I/O-Related Initialization Parameters for PDBs

The `MAX_IOPS` and `MAX_MBPS` initialization parameters limit the disk I/O generated by a PDB.

A large amount of disk I/O can cause poor performance. Several factors can result in excess disk I/O, such as poorly designed SQL or index and table scans in high-volume transactions. If one PDB is generating a large amount of disk I/O, then it can degrade the performance of other PDBs in the same CDB.

Use one or both of the following initialization parameters to limit the I/O generated by a specific PDB:

- The `MAX_IOPS` initialization parameter limits the number of I/O operations for each second.
- The `MAX_MBPS` initialization parameter limits the megabytes for I/O operations for each second.

If you set both preceding initialization parameters for a single PDB, then Oracle Database enforces both limits. Note that these limits are *not* enforced for Oracle Exadata, which uses I/O Resource Management (IORM) to manage I/Os between PDBs.

If these initialization parameters are set with the CDB root as the current container, then the values become the default values for all containers in the CDB. If they are set with an application root as the current container, then the values become the default values for all application PDBs in the application container. When they are set with a PDB or application PDB as the current container, then the settings take precedence over the default settings in the CDB root or the application root. These parameters cannot be set in a non-CDB.

The default for both initialization parameters is 0 (zero). If these initialization parameters are set to 0 (zero) in a PDB, and the CDB root is set to 0, then there is no I/O limit for the PDB. If these initialization parameters are set to 0 (zero) in an application PDB, and its application root is set to 0, then there is no I/O limit for the application PDB.

Critical I/O operations, such as ones for the control file and password file, are exempted from the limit and continue to run even if the limit is reached. However, all I/O operations, including critical I/O operations, are counted when the number of I/O operations and the megabytes for I/O operations are calculated.

You can use the `DBA_HIST_RSRC_PDB_METRIC` view to calculate a reasonable I/O limit for a PDB. Consider the values in the following columns when calculating a limit: `IOPS`, `IOMBPS`, `IOPS_THROTTLE_EXEMPT`, and `IOMBPS_THROTTLE_EXEMPT`. The `rsmgr:io rate limit` wait event indicates that a limit was reached.

Example 22-3 Limiting the I/O Generated by a PDB

With the PDB as the current container, run the following SQL statement to set the MAX_IOPS initialization parameter both in memory and in the SPFILE to a limit of 1,000 I/O operations for each second:

```
ALTER SYSTEM SET MAX_IOPS = 1000 SCOPE = BOTH;
```

Example 22-4 Limiting the Megabytes of I/O Generated by a PDB

With the PDB as the current container, run the following SQL statement to set the MAX_MBPS initialization parameter both in memory and in the SPFILE to a limit of 200 MB of I/O for each second:

```
ALTER SYSTEM SET MAX_MBPS = 200 SCOPE = BOTH;
```

See Also:

- "Modifying a PDB at the System Level"
- *Oracle Database Reference* for more information about the MAX_IOPS initialization parameter
- *Oracle Database Reference* for more information about the MAX_MBPS initialization parameter

CPU-Related Initialization Parameters for PDBs

The CPU_COUNT initialization parameter specifies the number of CPUs available for Oracle Database to use.

Instance caging is a technique that uses an initialization parameter to limit the number of CPUs that an instance can use simultaneously. You can set CPU_COUNT at the PDB level. If Resource Manager is enabled, then the PDB is “caged” (restricted) to the number of CPUs specified by CPU_COUNT.

CPU_COUNT works the same way as the utilization_limit directive in the CDB plan. However, the CPU_COUNT limit is expressed in terms of number of CPUs rather than utilization percentage. If both the utilization_limit and CPU_COUNT are specified, then the lower limit is enforced.

CPU_COUNT is advantageous because when the PDB is plugged into a new container, the CPU_COUNT setting remains with the plugged-in PDB. Also, Oracle Database uses the CPU_COUNT setting for a PDB to derive many other PDB parameters, such as those for parallel execution.

Managing CDB Resource Plans

In a CDB, PDBs might have different levels of priority. You can create CDB resource plans to distribute resources to different PDBs based on these priorities.

About CDB Resource Plans

Create CDB resource plans that allocate shares and resource limits for PDBs.

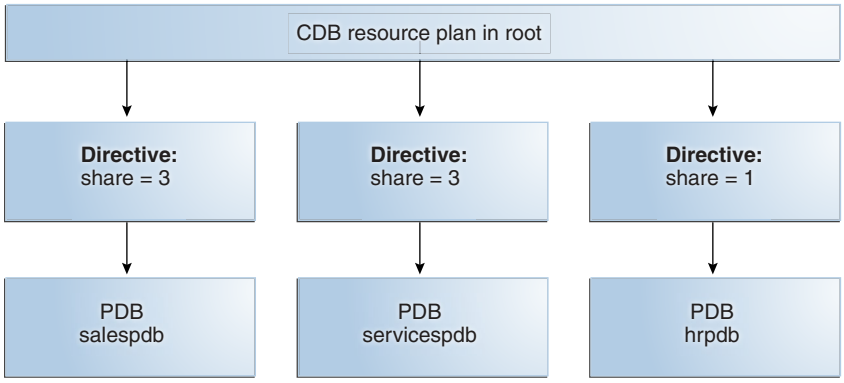
Shares for Allocating Resources to PDBs

To allocate resources among PDBs, assign a share value to each PDB or performance profile. A higher share value results in more guaranteed resources for a PDB or the PDBs that use the performance profile.

Specify a share value for a PDB using the `DBMS_RESOURCE_MANAGER.CREATE_CDB_PLAN_DIRECTIVE` procedure and for a PDB performance profile using the `DBMS_RESOURCE_MANAGER.CREATE_CDB_PROFILE_DIRECTIVE` procedure. In both cases, the `shares` parameter specifies the share value for the PDB. Multiple PDBs can use the same PDB performance profile.

The following figure shows an example of three PDBs with share values specified for them in a CDB resource plan.

Figure 22-1 Shares in a CDB Resource Plan



The preceding figure shows that the total number of shares is seven (3 plus 3 plus 1). The `salespdb` and the `servicespdb` PDB are each guaranteed 3/7 of the resources, while the `hrpdb` PDB is guaranteed 1/7 of the resources. However, any PDB can use more than the guaranteed amount of a resource when no resource contention exists.

The following table shows the resource allocation to the PDBs in the preceding figure based on the share values. The table assumes that loads of the PDBs consume all system resources allocated.

Table 22-2 Resource Allocation for Sample PDBs

Resource	Resource Allocation	See Also
CPU	The `salespdb` and `servicespdb` PDBs can consume the same amount of CPU resources. The `salespdb` and `servicespdb` PDBs are each guaranteed three times more CPU resource than the `hrpdb` PDB.	*Oracle Database Administrator's Guide* for more information about this resource
Parallel execution servers	Queued parallel queries from the `salespdb` and `servicespdb` PDBs are selected equally. Queued parallel queries from the `salespdb` and `servicespdb` PDBs are selected three times as often as queued parallel queries from the `hrpdb` PDB.	*Oracle Database Administrator's Guide* for more information about this resource

Utilization Limits for PDBs

A utilization limit restrains the system resource usage of a specific PDB or a specific PDB performance profile.

You can specify utilization limits for CPU and parallel execution servers. Utilization limits for a PDB are set by the CDB resource plan.

The following table describes utilization limits for PDBs and the Resource Manager action taken when a PDB reaches a utilization limit. For limits specified with a PDB performance profile, the limit applies to every PDB that uses the PDB performance profile. For example, if `pdb1` and `pdb20` have a performance profile `BRONZE`, and if `BRONZE` has a limit set to 10%, then `pdb1` has a 10% limit and `pdb20` has a 10% limit.

Table 22-3 Utilization Limits for PDBs

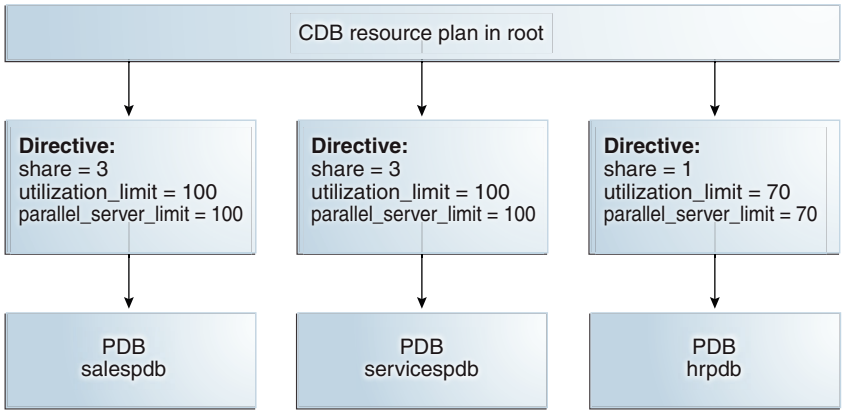
Resource	Resource Utilization Limit	Resource Manager Action When Limit Is Reached
CPU	The CPU utilization limit for sessions connected to a PDB is set by the `utilization_limit` parameter in subprograms of the `DBMS_RESOURCE_MANAGER` package. The `utilization_limit` parameter specifies the percentage of the system resources that a PDB can use. The value ranges from 0 to 100. You can also limit CPU for a PDB by setting the initialization parameter `CPU_COUNT`. For example, if you set the `CPU_COUNT` to 8, then the PDB cannot use more than 8 CPUs at any time. If both `utilization_limit` and `CPU_COUNT` are specified, then the more restrictive (lower) value is enforced.	Resource Manager throttles the PDB sessions so that the CPU utilization for the PDB does not exceed the utilization limit.

Table 22-3 (Cont.) Utilization Limits for PDBs

Resource	Resource Utilization Limit	Resource Manager Action When Limit Is Reached
Parallel execution servers	You can limit the number of parallel execution servers in a PDB by means of parallel statement queuing. The limit is a "queuing point" because the database queues parallel queries when the limit is reached. You can set the limit (queuing point) in either of the following ways: • The value of the PARALLEL_SERVERS_TARGET initialization parameter setting in the PDB • The value of the PARALLEL_SERVERS_TARGET initialization parameter setting in the CDB root multiplied by the value of the parallel_server_limit directive set for the PDB in the CDB resource manager plan For example, if the PARALLEL_SERVERS_TARGET initialization parameter is set to 200 in the CDB root, and if the parallel_server_limit directive for a PDB is set to 10%, then utilization limit for the PDB is 20 parallel execution servers (200 * .10). If the limit is set in both preceding ways, then the lower limit of the two is used. See *Oracle Database Reference* for the default value for PARALLEL_SERVERS_TARGET. **Note:** Oracle recommends using the PARALLEL_SERVERS_TARGET initialization parameter instead of the parallel_server_limit directive in a CDB plan.	Resource Manager queues parallel queries when the number of parallel execution servers used by the PDB would exceed the limit. **Note:** In a CDB, parallel statements are queued based on the PARALLEL_SERVERS_TARGET settings at both the PDB and CDB level. A statement is queued when the number of parallel servers used by the PDB exceeds the target for the PDB or when the number of parallel servers used by all PDBs exceeds the target for the CDB.

The following figure shows an example of three PDBs with shares and utilization limits specified for them in a CDB resource plan.

Figure 22-2 Shares and Utilization Limits in a CDB Resource Plan



The preceding figure shows that there are no utilization limits on the `salespdb` and `servicespdb` PDBs because `utilization_limit` and `parallel_server_limit` are both set to 100% for them. However, the `hrpdb` PDB is limited to 70% of the applicable system resources because `utilization_limit` and `parallel_server_limit` are both set to 70%.

Note:

This scenario assumes that the `PARALLEL_SERVERS_TARGET` initialization parameter does not specify a lower limit in a PDB. When the `PARALLEL_SERVERS_TARGET` initialization parameter specifies a lower limit for parallel execution servers in a PDB, the lower limit is used.

See Also:

- *Oracle Database Administrator's Guide*
- *Oracle Database Reference* to learn about `CPU_COUNT`

The Default Directive for PDBs

When you do not explicitly define directives for a PDB, the PDB uses the default directive for PDBs.

The following table shows the attributes of the initial default directive for PDBs.

Table 22-4 Initial Default Directive Attributes for PDBs

Directive Attribute	Value
`shares`	`1`

Table 22-4 (Cont.) Initial Default Directive Attributes for PDBs

Directive Attribute	Value
utilization_limit	100
parallel_server_limit	100

When a PDB is plugged into a CDB and no directive is defined for it, the PDB uses the default directive for PDBs.

You can create new directives for the new PDB. You can also change the default directive attribute values for PDBs by using the `UPDATE_CDB_DEFAULT_DIRECTIVE` procedure in the `DBMS_RESOURCE_MANAGER` package.

When a PDB is unplugged from a CDB, the directive for the PDB is retained. If the same PDB is plugged back into the CDB, then it uses the directive defined for it if the directive was not deleted manually.

Figure 22-3 shows an example of the default directive in a CDB resource plan.

Figure 22-3 Default Directive in a CDB Resource Plan

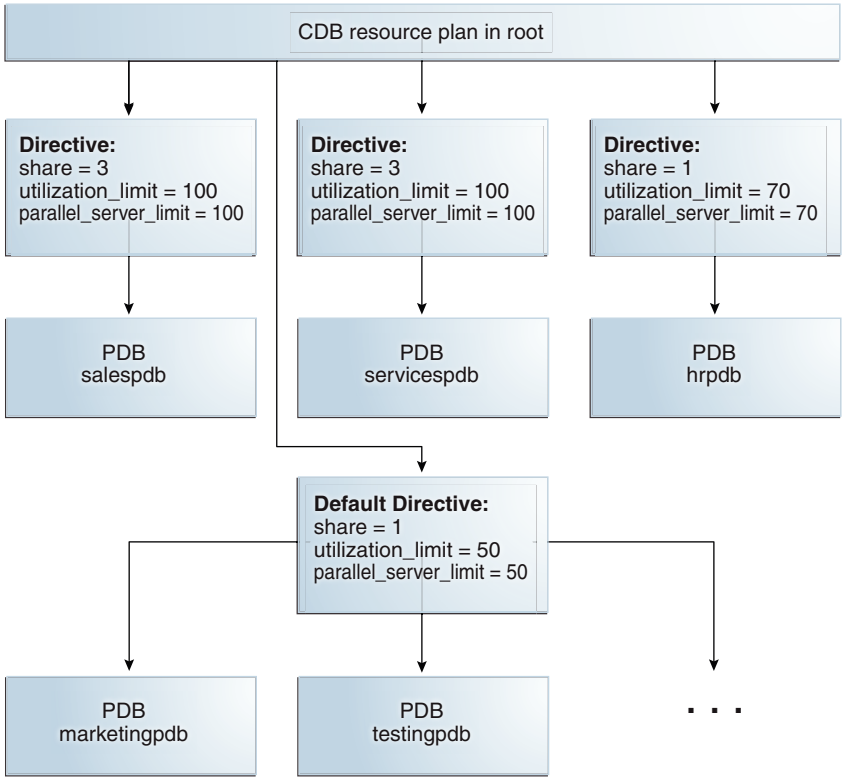


Figure 22-3 shows that the default PDB directive specifies that the `share` is 1, the `utilization_limit` is 50%, and the `parallel_server_limit` is 50%. Any PDB that is part of the CDB and does not have directives defined for it uses the default PDB directive. Figure 22-3 shows the PDBs `marketingpdb` and `testingpdb` using the

default PDB directive. Therefore, `marketingpdb` and `testingpdb` each get 1 share and utilization limits of 50.

See Also:

- "Creating New CDB Resource Plan Directives for a PDB"
- "Updating the Default Directive for PDBs in a CDB Resource Plan"
- "Creating and Removing PDBs and Application Containers"
- "Unplugging a PDB from a CDB"
- *Oracle Database Administrator's Guide* for information about the parallel server limit

Creating a CDB Resource Plan for Managing PDBs

To create a CDB resource plan for individual PDBs and define the directives for the plan, use the `DBMS_RESOURCE_MANAGER` package.

The general steps for creating a CDB resource plan for individual PDBs are the following:

1. Create the pending area using the `CREATE_PENDING_AREA` procedure.
2. Create the CDB resource plan using the `CREATE_CDB_PLAN` procedure.
3. Create directives for the PDBs using the `CREATE_CDB_PLAN_DIRECTIVE` procedure.
4. (Optional) Update the default PDB directive using the `UPDATE_CDB_DEFAULT_DIRECTIVE` procedure.
5. Validate the pending area using the `VALIDATE_PENDING_AREA` procedure.
6. Submit the pending area using the `SUBMIT_PENDING_AREA` procedure.

Creating a CDB Resource Plan for Managing PDBs: Scenario

This scenario illustrates each of the steps involved in creating a CDB resource plan for individual PDBs.

The scenario assumes that you want to create a CDB resource plan for a CDB named `newcdb`. The plan includes a directive for each PDB. In this scenario, you also update the default directive and the AutoTask directive.

The directives are defined using various procedures in the `DBMS_RESOURCE_MANAGER` package. The attributes of each directive are defined using parameters in these procedures. Table 22-5 describes the types of directives in the plan.

Table 22-5 Attributes for PDB Directives in a CDB Resource Plan

Directive Attribute	Description	See Also
`shares`	Resource allocation share for CPU and parallel execution server resources.	"Shares for Allocating Resources to PDBs"

Table 22-5 (Cont.) Attributes for PDB Directives in a CDB Resource Plan

Directive Attribute	Description	See Also
`utilization_limit`	Resource utilization limit for CPU.	"Utilization Limits for PDBs"
`parallel_server_limit`	Maximum percentage of parallel execution servers that a PDB can use before queuing parallel statements. When the `parallel_server_limit` directive is specified for a PDB, the limit is the `PARALLEL_SERVERS_TARGET` value of the CDB root multiplied by the value of the `parallel_server_limit` parameter in the `CREATE_CDB_PLAN_DIRECTIVE` procedure. **Note:** Oracle recommends using the `PARALLEL_SERVERS_TARGET` initialization parameter instead of the `parallel_server_limit` directive in a CDB plan.	"Utilization Limits for PDBs"

Table 22-6 describes how the CDB resource plan allocates resources to its PDBs using the directive attributes described in Table 22-5.

Table 22-6 Sample Directives for PDBs in a CDB Resource Plan

PDB	shares Directive	utilization_limit Directive	parallel_server_limit Directive
`salespdb`	3	Unlimited	Unlimited
`servicespdb`	3	Unlimited	Unlimited
`hrpdb`	1	70	70
Default	1	50	50
AutoTask	1	75	75

The `salespdb` and `servicespdb` PDBs are more important than the other PDBs in the CDB. Therefore, they get a higher share (3), unlimited CPU utilization resource, and unlimited parallel execution server resource.

The default directive applies to PDBs for which specific directives have not been defined. For this scenario, assume that the CDB has several PDBs that use the default directive. This scenario updates the default directive.

In addition, this scenario updates the AutoTask directive. The AutoTask directive applies to automatic maintenance tasks that are run in the root maintenance window.

To create a CDB resource plan:

1. Create a pending area using the `CREATE_PENDING_AREA` procedure:

```
exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
```

2. Create a CDB resource plan named `newcdb_plan` using the `CREATE_CDB_PLAN` procedure:

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_CDB_PLAN(
    plan    => 'newcdb_plan',
    comment => 'CDB resource plan for newcdb');
END;
/
```

3. Create the CDB resource plan directives for the PDBs using the `CREATE_CDB_PLAN_DIRECTIVE` procedure. Each directive specifies how resources are allocated to a specific PDB.

   Table 22-6 describes the directives for the `salespdb`, `servicespdb`, and `hrpdb` PDBs in this scenario. Run the following procedures to create these directives:

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_CDB_PLAN_DIRECTIVE(
    plan                 => 'newcdb_plan',
    pluggable_database   => 'salespdb',
    shares               => 3,
    utilization_limit    => 100,
    parallel_server_limit => 100);
END;
/

BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_CDB_PLAN_DIRECTIVE(
    plan                 => 'newcdb_plan',
    pluggable_database   => 'servicespdb',
    shares               => 3,
    utilization_limit    => 100,
    parallel_server_limit => 100);
END;
/

BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_CDB_PLAN_DIRECTIVE(
    plan                 => 'newcdb_plan',
    pluggable_database   => 'hrpdb',
    shares               => 1,
    utilization_limit    => 70,
    parallel_server_limit => 70);
END;
/
```

   All other PDBs in this CDB use the default PDB directive.

4. If the current default CDB resource plan directive for PDBs does not meet your requirements, then update the directive using the `UPDATE_CDB_DEFAULT_DIRECTIVE` procedure.

   The default directive applies to PDBs for which specific directives have not been defined. See "The Default Directive for PDBs" for more information.

   Table 22-6 describes the default directive that PDBs use in this scenario. Run the following procedure to update the default directive:

   ```
   BEGIN
     DBMS_RESOURCE_MANAGER.UPDATE_CDB_DEFAULT_DIRECTIVE(
       plan                      => 'newcdb_plan',
       new_shares                => 1,
       new_utilization_limit     => 50,
       new_parallel_server_limit => 50);
   END;
   /
   ```

5. Validate the pending area using the `VALIDATE_PENDING_AREA` procedure:

   ```
   exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
   ```

6. Submit the pending area using the `SUBMIT_PENDING_AREA` procedure:

   ```
   exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
   ```

Creating a CDB Resource Plan with PDB Performance Profiles

Use the `DBMS_RESOURCE_MANAGER` package to create a CDB resource plan for PDB performance profiles and define the directives for the plan. Each PDB that uses a profile adopts the CDB resource plan directive.

The general steps for creating a CDB resource plan with PDB performance profiles are the following:

1. Create the pending area using the `CREATE_PENDING_AREA` procedure.
2. Create the CDB resource plan using the `CREATE_CDB_PLAN` procedure.
3. Create directives for the PDB performance profiles using the `CREATE_CDB_PROFILE_DIRECTIVE` procedure.
4. (Optional) Update the default PDB directive using the `UPDATE_CDB_DEFAULT_DIRECTIVE` procedure.
5. Validate the pending area using the `VALIDATE_PENDING_AREA` procedure.
6. Submit the pending area using the `SUBMIT_PENDING_AREA` procedure.
7. For each PDB that will use a profile, set the `DB_PERFORMANCE_PROFILE` initialization parameter and specify the profile name.

Creating a CDB Resource Plan for PDB Performance Profiles: Scenario

This scenario illustrates the steps involved in creating a CDB resource plan for PDB performance profiles.

The scenario assumes that you want to create a CDB resource plan for a CDB named `newcdb`. The plan includes a directive for each PDB performance profile. In this scenario, you also update the default directive and the AutoTask directive.

In the CDB resource plan, you give each profile a name. In each PDB, you set the `DB_PERFORMANCE_PROFILE` initialization parameter to specify which PDB performance profile the PDB uses.

The directives are defined using various procedures in the `DBMS_RESOURCE_MANAGER` package. The attributes of each directive are defined using parameters in these procedures. The following table describes the types of directives in the plan.

Table 22-7 Attributes for PDB Performance Profile Directives in a CDB Resource Plan

Directive Attribute	Description	See Also
`shares`	Resource allocation share for CPU and parallel execution server resources.	"Shares for Allocating Resources to PDBs"
`utilization_limit`	Resource utilization limit for CPU.	"Utilization Limits for PDBs"
`parallel_server_limit`	Maximum percentage of parallel execution servers that a PDB can use. When the `parallel_server_limit` directive is specified for a PDB performance profile, the limit is the value of the `PARALLEL_SERVERS_TARGET` initialization parameter setting in the CDB root multiplied by the value of the `parallel_server_limit` parameter in the `CREATE_CDB_PROFILE_DIRECTIVE` procedure.	"Utilization Limits for PDBs"

The following table describes how the CDB resource plan allocates resources to its PDB performance profiles using the directive attributes described in Table 22-7.

Table 22-8 Sample Directives for PDB Performance Profiles in a CDB Resource Plan

PDB	shares Directive	utilization_limit Directive	parallel_server_limit Directive
`gold`	3	Unlimited	Unlimited
`silver`	2	40	40

Table 22-8 (Cont.) Sample Directives for PDB Performance Profiles in a CDB Resource Plan

PDB	shares Directive	utilization_limit Directive	parallel_server_limit Directive
bronze	1	20	20
Default	1	10	10
AutoTask	2	60	60

The default directive applies to PDBs for which specific directives have not been defined. For this scenario, assume that the CDB has several PDBs that use the default directive. This scenario updates the default directive.

In addition, this scenario updates the AutoTask directive. The AutoTask directive applies to automatic maintenance tasks that are run in the root maintenance window.

To create a CDB resource plan for PDB performance profiles:

1. For each PDB that will use a profile, set the `DB_PERFORMANCE_PROFILE` initialization parameter to the name of the profile that the PDB will use.
   a. Run an `ALTER SYSTEM` statement to set the parameter.

      For example, with the PDB as the current container, run the following SQL statement:

      ```
      ALTER SYSTEM SET DB_PERFORMANCE_PROFILE=gold SCOPE=spfile;
      ```

   b. Close the PDB:

      ```
      ALTER PLUGGABLE DATABASE CLOSE IMMEDIATE;
      ```

   c. Open the PDB:

      ```
      ALTER PLUGGABLE DATABASE OPEN;
      ```

2. Create a pending area using the `CREATE_PENDING_AREA` procedure:

   ```
   exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
   ```

3. Create a CDB resource plan named `newcdb_plan` using the `CREATE_CDB_PLAN` procedure:

   ```
   BEGIN
     DBMS_RESOURCE_MANAGER.CREATE_CDB_PLAN(
       plan    => 'newcdb_plan',
       comment => 'CDB resource plan for newcdb');
   END;
   /
   ```

4. Create the CDB resource plan directives for the PDBs using the `CREATE_CDB_PLAN_DIRECTIVE` procedure. Each directive specifies how resources are allocated to a specific PDB.

Table 22-6 describes the directives for the `gold`, `silver`, and `bronze` profiles in this scenario. Run the following procedures to create these directives:

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_CDB_PROFILE_DIRECTIVE(
    plan                  => 'newcdb_plan',
    profile               => 'gold',
    shares                => 3,
    utilization_limit     => 100,
    parallel_server_limit => 100);
END;
/

BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_CDB_PROFILE_DIRECTIVE(
    plan                  => 'newcdb_plan',
    profile               => 'silver',
    shares                => 2,
    utilization_limit     => 40,
    parallel_server_limit => 40);
END;
/

BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_CDB_PROFILE_DIRECTIVE(
    plan                  => 'newcdb_plan',
    profile               => 'bronze',
    shares                => 1,
    utilization_limit     => 20,
    parallel_server_limit => 20);
END;
/
```

All other PDBs in this CDB use the default PDB directive.

5. If the current default CDB resource plan directive for PDBs does not meet your requirements, then update the directive using the `UPDATE_CDB_DEFAULT_DIRECTIVE` procedure.

   The default directive applies to PDBs for which specific directives have not been defined.

   Table 22-6 describes the default directive that PDBs use in this scenario. Run the following procedure to update the default directive:

```
BEGIN
  DBMS_RESOURCE_MANAGER.UPDATE_CDB_DEFAULT_DIRECTIVE(
    plan                      => 'newcdb_plan',
    new_shares                => 1,
    new_utilization_limit     => 10,
    new_parallel_server_limit => 10);
END;
/
```

6. Validate the pending area using the `VALIDATE_PENDING_AREA` procedure:

```
exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
```

7. Submit the pending area using the `SUBMIT_PENDING_AREA` procedure:

```
exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
```

> **See Also:**
>
> "The Default Directive for PDBs"

Enabling a CDB Resource Plan

You enable the Resource Manager for a CDB by setting the `RESOURCE_MANAGER_PLAN` initialization parameter in the root.

This parameter specifies the top plan, which is the plan to be used for the current CDB instance. If no plan is specified with this parameter, then the Resource Manager is not enabled.

Prerequisites

Before enabling a CDB resource plan, complete the prerequisites described in "Overview of Oracle Resource Manager in a Multitenant Environment".

To enable a CDB resource plan:

1. In SQL*Plus, ensure that the current container is the root.
2. Perform one of the following actions:
   - Use an `ALTER SYSTEM` statement to set the `RESOURCE_MANAGER_PLAN` initialization parameter to the CDB resource plan.

     The following example sets the CDB resource plan to `newcdb_plan` using an `ALTER SYSTEM` statement:

     ```
     ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = 'newcdb_plan';
     ```

   - In a text initialization parameter file, set the `RESOURCE_MANAGER_PLAN` initialization parameter to the CDB resource plan, and restart the CDB.

     The following example sets the CDB resource plan to `newcdb_plan` in an initialization parameter file:

     ```
     RESOURCE_MANAGER_PLAN = 'newcdb_plan'
     ```

> **See Also:**
>
> - "About Container Access in a CDB"
> - *Oracle Database Administrator's Guide* to learn how to schedule a CDB resource plan change with Oracle Scheduler

Modifying a CDB Resource Plan

Modifying a CDB resource plan includes tasks such as updating the plan, creating, updating, or deleting plan directives for PDBs, and updating default directives.

Updating a CDB Resource Plan

You can update a CDB resource plan to change its comment using the `UPDATE_CDB_PLAN` procedure.

Prerequisites

Before updating a CDB resource plan, complete the prerequisites described in "Overview of Oracle Resource Manager in a Multitenant Environment".

To update a CDB resource plan:

1. In SQL*Plus, ensure that the current container is the root.
2. Create a pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
   ```

3. Run the `UPDATE_CDB_PLAN` procedure, and enter a new comment in the `new_comment` parameter.

   For example, the following procedure changes the comment for the `newcdb_plan` CDB resource plan:

   ```
   BEGIN
     DBMS_RESOURCE_MANAGER.UPDATE_CDB_PLAN(
       plan        => 'newcdb_plan',
       new_comment => 'CDB plan for PDBs in newcdb');
   END;
   /
   ```

4. Validate the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
   ```

5. Submit the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
   ```

> **See Also:**
>
> - "About CDB Resource Plans"
> - "About Container Access in a CDB"

Managing CDB Resource Plan Directives for a PDB

You can create, update, and delete CDB resource plan directives for a PDB.

Creating New CDB Resource Plan Directives for a PDB

When you create a PDB in a CDB, you can create a CDB resource plan directive for the PDB using the `CREATE_CDB_PLAN_DIRECTIVE` procedure. The directive specifies how resources are allocated to the new PDB.

Prerequisites

Before creating a new CDB resource plan directive for a PDB, complete the prerequisites described in "Overview of Oracle Resource Manager in a Multitenant Environment".

To create a new CDB resource plan directive for a PDB:

1. In SQL*Plus, ensure that the current container is the root.
2. Create a pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
   ```

3. Run the `CREATE_CDB_PLAN_DIRECTIVE` procedure, and specify the appropriate values for the new PDB.

   For example, the following procedure allocates resources to a PDB named `operpdb` in the `newcdb_plan` CDB resource plan:

   ```
   BEGIN
     DBMS_RESOURCE_MANAGER.CREATE_CDB_PLAN_DIRECTIVE(
       plan                  => 'newcdb_plan',
       pluggable_database    => 'operpdb',
       shares                => 1,
       utilization_limit     => 20,
       parallel_server_limit => 30);
   END;
   /
   ```

4. Validate the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
   ```

5. Submit the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
   ```

> **See Also:**
>
> - "About CDB Resource Plans"
> - "About Container Access in a CDB"

Updating CDB Resource Plan Directives for a PDB

You can update the CDB resource plan directive for a PDB using the `UPDATE_CDB_PLAN_DIRECTIVE` procedure. The directive specifies how resources are allocated to the PDB.

Prerequisites

Before updating a CDB resource plan directive for a PDB, ensure that you meet the prerequisites described in "Overview of Oracle Resource Manager in a Multitenant Environment".

To update a CDB resource plan directive for a PDB:

1. In SQL*Plus, ensure that the current container is the root.
2. Create a pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
   ```

3. Run the `UPDATE_CDB_PLAN_DIRECTIVE` procedure, and specify the new resource allocation values for the PDB.

   For example, the following procedure updates the resource allocation to a PDB named `operpdb` in the `newcdb_plan` CDB resource plan:

   ```
   BEGIN
     DBMS_RESOURCE_MANAGER.UPDATE_CDB_PLAN_DIRECTIVE(
       plan                       => 'newcdb_plan',
       pluggable_database         => 'operpdb',
       new_shares                 => 1,
       new_utilization_limit      => 10,
       new_parallel_server_limit => 20);
   END;
   /
   ```

4. Validate the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
   ```

5. Submit the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
   ```

> **See Also:**
>
> - "About Container Access in a CDB"
> - "About CDB Resource Plans"

Deleting CDB Resource Plan Directives for a PDB

You can delete the CDB resource plan directive for a PDB using the `DELETE_CDB_PLAN_DIRECTIVE` procedure.

You might delete the directive for a PDB if you unplug or drop the PDB. However, you can retain the directive, and if the PDB is plugged into the CDB in the future, the existing directive applies to the PDB.

Prerequisites

Before deleting a CDB resource plan directive for a PDB, complete the prerequisites described in "Overview of Oracle Resource Manager in a Multitenant Environment".

To delete a CDB resource plan directive for a PDB:

1. In SQL*Plus, ensure that the current container is the root.
2. Create a pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
   ```

3. Run the `DELETE_CDB_PLAN_DIRECTIVE` procedure, and specify the CDB resource plan and the PDB.

   For example, the following procedure deletes the directive for a PDB named `operpdb` in the `newcdb_plan` CDB resource plan:

   ```
   BEGIN
     DBMS_RESOURCE_MANAGER.DELETE_CDB_PLAN_DIRECTIVE(
       plan               => 'newcdb_plan',
       pluggable_database => 'operpdb');
   END;
   /
   ```

4. Validate the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
   ```

5. Submit the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
   ```

> **See Also:**
>
> - "About Container Access in a CDB"
> - "About CDB Resource Plans"

Managing CDB Resource Plan Directives for a PDB Performance Profile

You can create, update, and delete CDB resource plan directives for a PDB performance profile.

Creating New CDB Resource Plan Directives for a PDB Performance Profile

You can create a CDB resource plan directive for the a new PDB performance profile using the `CREATE_CDB_PROFILE_DIRECTIVE` procedure. The directive specifies how resources are allocated to the all PDBs that use the new profile.

Prerequisites

Before creating a new CDB resource plan directive for a PDB performance profile, complete the prerequisites described in "Overview of Oracle Resource Manager in a Multitenant Environment".

To create a new CDB resource plan directive for a PDB performance profile:

1. In SQL*Plus, ensure that the current container is the root.
2. Create a pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
   ```

3. Run the `CREATE_CDB_PROFILE_DIRECTIVE` procedure, and specify the appropriate values for the new PDB performance profile.

   For example, the following procedure allocates resources to a PDB performance profile named `copper` in the `newcdb_plan` CDB resource plan:

   ```
   BEGIN
     DBMS_RESOURCE_MANAGER.CREATE_CDB_PROFILE_DIRECTIVE(
       plan                  => 'newcdb_plan',
       profile               => 'copper',
       shares                => 1,
       utilization_limit     => 20,
       parallel_server_limit => 30);
   END;
   /
   ```

4. Validate the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
   ```

5. Submit the pending area:

```
exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
```

> **Note:**
>
> For a PDB to use the new profile, the PDB must have the `DB_PERFORMANCE_PROFILE` initialization parameter set to the profile name.

> **See Also:**
>
> - "About CDB Resource Plans"
> - "About Container Access in a CDB"

Updating CDB Resource Plan Directives for a PDB Performance Profile

Update the CDB resource plan directive for a PDB performance profile using the `UPDATE_CDB_PROFILE_DIRECTIVE` procedure. The directive specifies how resources are allocated to the PDBs that use the PDB performance profile.

Before updating a CDB resource plan directive for a PDB performance profile, complete the prerequisites described in "Overview of Oracle Resource Manager in a Multitenant Environment".

To update a CDB resource plan directive for a PDB performance profile:

1. In SQL*Plus, ensure that the current container is the root.
2. Create a pending area:

```
exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
```

3. Run the `UPDATE_CDB_PROFILE_DIRECTIVE` procedure, and specify the new resource allocation values for the PDB performance profile.

   For example, the following procedure updates the resource allocation for a PDB performance profile named `copper` in the `newcdb_plan` CDB resource plan:

```
BEGIN
  DBMS_RESOURCE_MANAGER.UPDATE_CDB_PROFILE_DIRECTIVE(
    plan                      => 'newcdb_plan',
    profile                   => 'copper',
    new_shares                => 1,
    new_utilization_limit     => 10,
    new_parallel_server_limit => 20);
END;
/
```

4. Validate the pending area:

```
exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
```

5. Submit the pending area:

```
exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
```

> **See Also:**
>
> - "About CDB Resource Plans"
> - "About Container Access in a CDB"

Deleting CDB Resource Plan Directives for a PDB Performance Profile

You can delete the CDB resource plan directive for a PDB performance profile using the `DELETE_CDB_PROFILE_DIRECTIVE` procedure.

If no PDBs use a performance profile, then you might delete the directive for the profile.

Prerequisites

Before deleting a CDB resource plan directive for a PDB performance profile, complete the prerequisites described in "Overview of Oracle Resource Manager in a Multitenant Environment".

To delete a CDB resource plan directive for a PDB performance profile:

1. In SQL*Plus, ensure that the current container is the root.
2. Create a pending area:

```
exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
```

3. Run the `DELETE_CDB_PROFILE_DIRECTIVE` procedure, and specify the CDB resource plan and the PDB performance profile.

   For example, the following procedure deletes the directive for a PDB named `operpdb` in the `newcdb_plan` CDB resource plan:

```
BEGIN
  DBMS_RESOURCE_MANAGER.DELETE_CDB_PLAN_DIRECTIVE(
    plan    => 'newcdb_plan',
    profile => 'operpdb');
END;
/
```

4. Validate the pending area:

```
exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
```

5. Submit the pending area:

```
exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
```

> **See Also:**
>
> - "About Container Access in a CDB".
> - "About CDB Resource Plans"

Updating the Default Directive for PDBs in a CDB Resource Plan

You can update the default directive for PDBs in a CDB resource plan using the `UPDATE_CDB_DEFAULT_DIRECTIVE` procedure. The default directive applies to PDBs for which specific directives have not been defined.

Prerequisites

Before updating the default directive for PDBs in a CDB resource plan, complete the prerequisites described in "Overview of Oracle Resource Manager in a Multitenant Environment".

To update the default directive for PDBs in a CDB resource plan:

1. In SQL*Plus, ensure that the current container is the root.
2. Create a pending area:

```
exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
```

3. Run the `UPDATE_CDB_DEFAULT_DIRECTIVE` procedure, and specify the appropriate default resource allocation values.

   For example, the following procedure updates the default directive for PDBs in the `newcdb_plan` CDB resource plan:

```
BEGIN
  DBMS_RESOURCE_MANAGER.UPDATE_CDB_DEFAULT_DIRECTIVE(
    plan                 => 'newcdb_plan',
    new_shares           => 2,
    new_utilization_limit => 40);
END;
/
```

4. Validate the pending area:

```
exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
```

5. Submit the pending area:

```
exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
```

> **See Also:**
>
> - See "The Default Directive for PDBs" for more information.
> - "About CDB Resource Plans"
> - See "About Container Access in a CDB".

Updating the Default Directive for Maintenance Tasks in a CDB Resource Plan

You can update the AutoTask directive in a CDB resource plan using the `UPDATE_CDB_AUTOTASK_DIRECTIVE` procedure. The AutoTask directive applies to automatic maintenance tasks that are run in the root maintenance window.

Prerequisites

Before updating the default directive for maintenance tasks in a CDB resource plan, complete the prerequisites described in "Overview of Oracle Resource Manager in a Multitenant Environment".

To update the AutoTask directive for maintenance tasks in a CDB resource plan:

1. In SQL*Plus, ensure that the current container is the root.
2. Create a pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
   ```

3. Run the `UPDATE_CDB_AUTOTASK_DIRECTIVE` procedure, and specify the appropriate AutoTask resource allocation values.

   For example, the following procedure updates the AutoTask directive for maintenance tasks in the `newcdb_plan` CDB resource plan:

   ```
   BEGIN
     DBMS_RESOURCE_MANAGER.UPDATE_CDB_AUTOTASK_DIRECTIVE(
       plan                  => 'newcdb_plan',
       new_shares            => 2,
       new_utilization_limit => 60);
   END;
   /
   ```

4. Validate the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
   ```

5. Submit the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
   ```

> **See Also:**
>
> - "About Container Access in a CDB"
> - "About CDB Resource Plans"

Deleting a CDB Resource Plan

You can delete a CDB resource plan using the `DELETE_CDB_PLAN` procedure.

The resource plan must be disabled. You might delete a CDB resource plan if the plan is no longer needed. You can enable a different CDB resource plan, or you can disable Resource Manager for the CDB.

Prerequisites

Before deleting a CDB resource plan, complete the prerequisites described in "Overview of Oracle Resource Manager in a Multitenant Environment".

To delete a CDB resource plan:

1. In SQL*Plus, ensure that the current container is the root.
2. Create a pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
   ```

3. Run the `DELETE_CDB_PLAN` procedure, and specify the CDB resource plan.

   For example, the following procedure deletes the `newcdb_plan` CDB resource plan:

   ```
   BEGIN
     DBMS_RESOURCE_MANAGER.DELETE_CDB_PLAN(
       plan => 'newcdb_plan');
   END;
   /
   ```

4. Validate the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
   ```

5. Submit the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
   ```

> **See Also:**
>
> - "About CDB Resource Plans"
> - See "About Container Access in a CDB".
> - "Enabling a CDB Resource Plan"
> - "Disabling a CDB Resource Plan"

Disabling a CDB Resource Plan

Disable the Resource Manager for a CDB by unsetting the `RESOURCE_MANAGER_PLAN` initialization parameter in the CDB root.

A CDB resource plan that specifies shares or utilization limits for PDBs is required to enable CPU management, both between PDBs and within a PDB. If a resource plan with shares or utilization limits is enabled for a PDB, and if the CDB resource plan is not specified, then the CDB resource plan is set to `DEFAULT_CDB_PLAN`. This setting gives equal shares to all PDBs and specifies no utilization limits. To disable CPU resource management throughout the CDB, set `RESOURCE_MANAGER_PLAN` to `ORA$INTERNAL_CDB_PLAN`.

Prerequisites

Before disabling a CDB resource plan, complete the prerequisites described in "Overview of Oracle Resource Manager in a Multitenant Environment".

To disable a CDB resource plan:

1. In SQL*Plus, ensure that the current container is the root.
2. Perform one of the following actions:
   - Use an `ALTER SYSTEM` statement to unset the `RESOURCE_MANAGER_PLAN` initialization parameter for the CDB.

     The following example unsets the `RESOURCE_MANAGER_PLAN` initialization parameter using an `ALTER SYSTEM` statement:

     ```
     ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = '';
     ```

   - In an initialization parameter file, unset the `RESOURCE_MANAGER_PLAN` initialization parameter, and restart the CDB.

     The following example unsets the `RESOURCE_MANAGER_PLAN` initialization parameter in an initialization parameter file:

     ```
     RESOURCE_MANAGER_PLAN =
     ```

> **See Also:**
>
> - "About Container Access in a CDB"
> - "Shutting Down a CDB Instance"
> - *Oracle Database Administrator's Guide* for information about starting up a database

Viewing Information About Plans and Directives in a CDB

You can view information about CDB resource plans, CDB resource plan directives, and predefined resource plans in a CDB.

> **See Also:**
>
> *Oracle Database Administrator's Guide* for information about monitoring Oracle Database Resource Manager

Viewing CDB Resource Plans

An example illustrates using the `DBA_CDB_RSRC_PLANS` view to display all CDB resource plans defined in the CDB.

The `DEFAULT_CDB_PLAN` is supplied with Oracle Database. You can use this default plan if it meets your requirements.

To view CDB resource plans:

1. Start SQL*Plus or SQL Developer, and log in to the CDB root.
2. Run the following query:

```
COLUMN PLAN FORMAT A30
COLUMN STATUS FORMAT A10
COLUMN COMMENTS FORMAT A35

SELECT PLAN, STATUS, COMMENTS
FROM   DBA_CDB_RSRC_PLANS
ORDER BY PLAN;
```

Your output looks similar to the following:

```
PLAN                           STATUS     COMMENTS
------------------------------ ---------- ----------------------------
DEFAULT_CDB_PLAN                          Default CDB plan
DEFAULT_MAINTENANCE_PLAN                  Default CDB maintenance plan
NEWCDB_PLAN                               CDB plan for PDBs in newcdb
ORA$INTERNAL_CDB_PLAN                     Internal CDB plan
```

> **Note:**
>
> Plans in the pending area have a status of `PENDING`. Plans in the pending area are being edited. Any plan that is not in the pending area has a `NULL` status.

> **See Also:**
>
> "About CDB Resource Plans"

Viewing CDB Resource Plan Directives

An example illustrates using the `DBA_CDB_RSRC_PLAN_DIRECTIVES` view to display all directives defined in all CDB resource plans in the CDB.

The `DEFAULT_CDB_PLAN` is a default CDB plan that is supplied with Oracle Database. With `DEFAULT_CDB_PLAN`, every PDB has 1 share and a utilization limit of 100. If the CDB resource plan has no CPU directives configured, that is, the `shares` and `utilization_limits` directives are unset, then CPU Resource Manager uses the PDB-level `CPU_MIN_COUNT` and `CPU_COUNT` parameters to manage CPU. Note that `ORA$DEFAULT_PDB_DIRECTIVE` is the default directive for PDBs.

To view CDB resource plan directives:

1. Start SQL*Plus or SQL Developer, and log in to the CDB root.
2. Run the following query:

```
COLUMN PLAN HEADING 'Plan' FORMAT A24
COLUMN PLUGGABLE_DATABASE HEADING 'Pluggable Database' FORMAT A25
COLUMN SHARES HEADING 'Shares' FORMAT 999
COLUMN UTILIZATION_LIMIT HEADING 'Utilization|Limit' FORMAT 999
COLUMN PARALLEL_SERVER_LIMIT HEADING 'Parallel|Server|Limit' FORMAT
999

SELECT PLAN,
       PLUGGABLE_DATABASE,
       SHARES,
       UTILIZATION_LIMIT,
       PARALLEL_SERVER_LIMIT
  FROM DBA_CDB_RSRC_PLAN_DIRECTIVES
  ORDER BY PLAN;
```

Your output looks similar to the following:

```
                                                                 Parallel
                                                      Utilization   Server
Plan                     Pluggable Database        Shares       Limit    Limit
------------------------ ------------------------- ------ ----------- --------
DEFAULT_CDB_PLAN         ORA$DEFAULT_PDB_DIRECTIVE      1         100      100
```

```
DEFAULT_CDB_PLAN          ORA$AUTOTASK                          90       100
DEFAULT_MAINTENANCE_PLAN  ORA$AUTOTASK                          90       100
DEFAULT_MAINTENANCE_PLAN  ORA$DEFAULT_PDB_DIRECTIVE      1     100       100
NEWCDB_PLAN               HRPDB                          1      70        70
NEWCDB_PLAN               SALESPDB                       3     100       100
NEWCDB_PLAN               ORA$DEFAULT_PDB_DIRECTIVE      1      50        50
NEWCDB_PLAN               ORA$AUTOTASK                   1      75        75
NEWCDB_PLAN               SERVICESPDB                    3     100       100
```

The preceding output shows the directives for the `newcdb_plan` created in "Creating a CDB Resource Plan for Managing PDBs: Scenario" and modified in "Modifying a CDB Resource Plan".

See Also:

- "About CDB Resource Plans"
- "The Default Directive for PDBs"

Managing PDB Resource Plans

You can create, enable, and modify resource plans for individual PDBs.

About PDB Resource Plans

A PDB resource plan determines how the resources for a specific PDB are allocated to consumer groups within this PDB.

A PDB resource plan is similar to a resource plan for a non-CDB. A PDB resource plan differs from a CDB resource plan, which determines the amount of resources allocated to each PDB.

The following restrictions apply to PDB resource plans:

- A PDB resource plan cannot have subplans.
- A PDB resource plan cannot have a multiple-level scheduling policy.

If you create a PDB using a non-CDB, and the non-CDB contains resource plans, then these resource plans might not conform to the preceding restrictions. In this case, Oracle Database automatically transforms these resource plans into equivalent PDB resource plans that meet these requirements. The original resource plans and directives are recorded in the `DBA_RSRC_PLANS` and `DBA_RSRC_PLAN_DIRECTIVES` views with the `LEGACY` status.

See Also:

- "About CDB Resource Plans"
- "Options for Creating a PDB from a Non-CDB"
- *Oracle Database Administrator's Guide* to learn more about resource plans

CDB Resource Plan Requirements When Creating PDB Resource Plans

When you create PDB resource plans, the CDB resource plan must meet certain requirements.

Create directives for a CDB resource plan by using the `DBMS_RESOURCE_MANAGER.CREATE_CDB_PLAN_DIRECTIVE` procedure. Create directives for a PDB resource plan using the `CREATE_PLAN_DIRECTIVE` procedure in the same package. When you create one or more PDB resource plans and there is no CDB resource plan, the CDB uses the `DEFAULT_CDB_PLAN` that is supplied with Oracle Database.

The following table describes the requirements for the CDB resource plan and the results when the requirements are not met. The parameter values described in the "CDB Resource Plan Requirements" column are for the `CREATE_CDB_PLAN_DIRECTIVE` procedure. The parameter values described in the "Results When Requirements Are Not Met" column are for the `CREATE_PLAN_DIRECTIVE` procedure.

Table 22-9 CDB Resource Plan Requirements for PDB Resource Plans

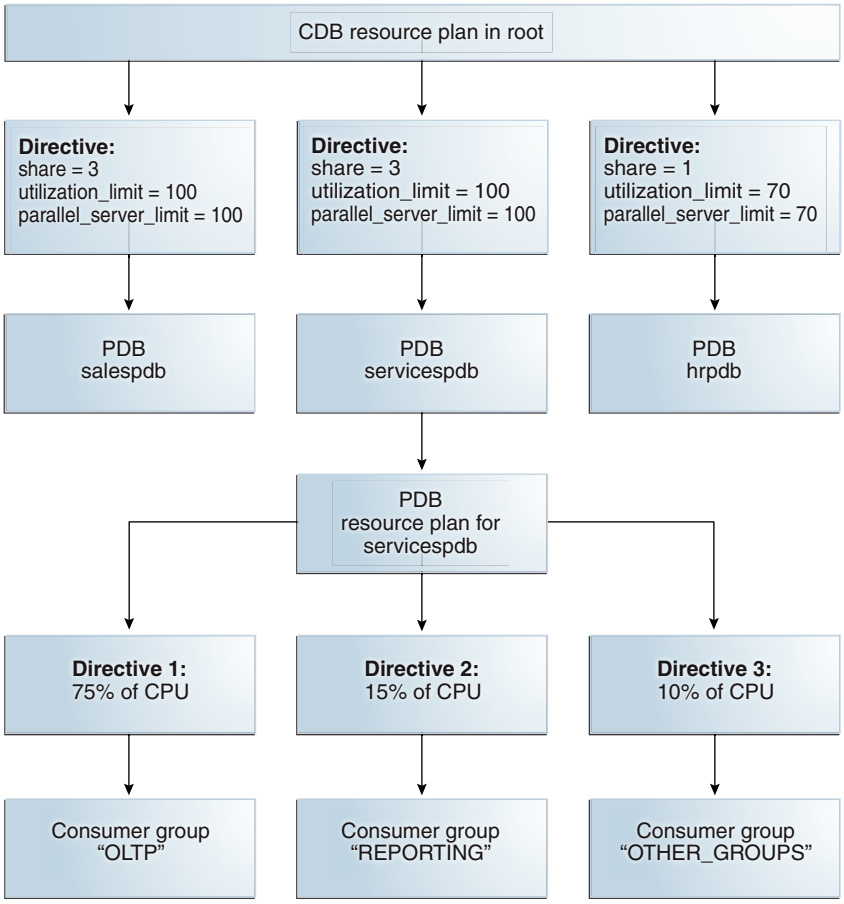
Resource	CDB Resource Plan Requirements	Results When Requirements Are Not Met
CPU	One of the following requirements must be met: • A share value must be specified for the PDB using the `shares` parameter. • A utilization limit for CPU below 100 must be specified for the PDB using the `utilization_limit` parameter. These values can be set in a directive for the specific PDB or in a default directive.	The CPU allocation policy of the PDB resource plan is not enforced. The CPU limit specified by the `utilization_limit` parameter in the PDB resource plan is not enforced.
Parallel execution servers	One of the following requirements must be met: • A share value must be specified for the PDB using the `shares` parameter. • A parallel server limit below 100 must be specified for the PDB using the `parallel_server_limit` parameter. These values can be set in a directive for the specific PDB or in a default directive.	The parallel execution server allocation policy of the PDB resource plan is not enforced. The parallel server limit specified by `parallel_server_limit` in the PDB resource plan is not enforced. However, you can set the `PARALLEL_SERVERS_TARGET` initialization parameter in a PDB to enforce the parallel limit.

PDB Resource Plan: Example

A one-to-many relationship exists between CDB resource plans and PDB resource plans.

The following figure shows an example of a CDB resource plan and a PDB resource plan.

Figure 22-4 A CDB Resource Plan and a PDB Resource Plan



The preceding figure shows some of the directives in a PDB resource plan for the `servicespdb` PDB. Other PDBs in the CDB can also have PDB resource plans.

Creating a PDB Resource Plan

You create a PDB resource plan in the same way that you create a resource plan for a non-CDB. You use procedures in the `DBMS_RESOURCE_MANAGER` PL/SQL package to create the plan.

A CDB resource plan allocates a portion of the system's resources to a PDB. A PDB resource plan determines how this portion is allocated within the PDB.

The following is a summary of the steps required to create a PDB resource plan:

1. In SQL*Plus, ensure that the current container is a PDB.
2. Create a pending area using the `CREATE_PENDING_AREA` procedure.
3. Create, modify, or delete consumer groups using the `CREATE_CONSUMER_GROUP` procedure.
4. Map sessions to consumer groups using the `SET_CONSUMER_GROUP_MAPPING` procedure.
5. Create the PDB resource plan using the `CREATE_PLAN` procedure.
6. Create PDB resource plan directives using the `CREATE_PLAN_DIRECTIVE` procedure.
7. Validate the pending area using the `VALIDATE_PENDING_AREA` procedure.
8. Submit the pending area using the `SUBMIT_PENDING_AREA` procedure.

Ensure that the current container is a PDB and that the user has the required privileges when you complete these steps. See *Oracle Database Administrator's Guide* for detailed information about completing these steps.

You also have the option of creating a simple resource plan that is adequate for many situations using the `CREATE_SIMPLE_PLAN` procedure. See *Oracle Database Administrator's Guide* for information about creating a simple resource plan.

> **Note:**
>
> Some restrictions apply to PDB resource plans. See "About PDB Resource Plans" for information.

Enabling a PDB Resource Plan

Enable a PDB resource plan by setting the `RESOURCE_MANAGER_PLAN` initialization parameter to the plan with an `ALTER SYSTEM` statement when the current container is the PDB.

If no plan is specified with this parameter, then no PDB resource plan is enabled for the PDB.

Prerequisites

Before enabling a PDB resource plan, complete the prerequisites described in "Overview of Oracle Resource Manager in a Multitenant Environment".

To enable a PDB resource plan:

1. In SQL*Plus, ensure that the current container is a PDB.
2. Use an `ALTER SYSTEM` statement to set the `RESOURCE_MANAGER_PLAN` initialization parameter to the PDB resource plan.

You can also schedule a PDB resource plan change with Oracle Scheduler.

Example 22-5 Enabling a PDB Resource Plan

The following example sets the PDB resource plan to `salespdb_plan`.

```
ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = 'salespdb_plan';
```

> **See Also:**
>
> - "About Container Access in a CDB"
> - "Modifying a PDB at the System Level"
> - *Oracle Database Administrator's Guide* to learn how to schedule a PDB resource plan change with Oracle Scheduler

Modifying a PDB Resource Plan

You can use the `DBMS_RESOURCE_MANAGER` package to modify a PDB resource plan in the same way you would modify the resource plan for a non-CDB.

Prerequisites

Before modifying a PDB resource plan, complete the prerequisites described in "Overview of Oracle Resource Manager in a Multitenant Environment".

To modify a PDB resource plan:

1. In SQL*Plus, ensure that the current container is a PDB.
2. Create a pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
   ```

3. Modify the PDB resource plan by completing one or more of the following tasks:
   - Update a consumer group using the `UPDATE_CONSUMER_GROUP` procedure.
   - Delete a consumer group using the `DELETE_CONSUMER_GROUP` procedure.
   - Update a resource plan using the `UPDATE_PLAN` procedure.
   - Delete a resource plan using the `DELETE_PLAN` procedure.
   - Update a resource plan directive using the `UPDATE_PLAN_DIRECTIVE` procedure.
   - Delete a resource plan directive using the `DELETE_PLAN_DIRECTIVE` procedure.
4. Validate the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
   ```

5. Submit the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
   ```

> **See Also:**
>
> - "About PDB Resource Plans"
> - "About Container Access in a CDB".
> - *Oracle Database Administrator's Guide* for instructions about completing the consumer group tasks

Disabling a PDB Resource Plan

You disable a PDB resource plan by unsetting the `RESOURCE_MANAGER_PLAN` initialization parameter in the PDB.

Prerequisites

Before disabling a PDB resource plan, complete the prerequisites described in "Overview of Oracle Resource Manager in a Multitenant Environment".

To disable a PDB resource plan:

1. In SQL*Plus, ensure that the current container is a PDB.
2. Use an `ALTER SYSTEM` statement to unset the `RESOURCE_MANAGER_PLAN` initialization parameter for the PDB.

Example 22-6 Disabling a PDB Resource Plan

The following example disables the PDB resource plan.

```
ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = '';
```

> **See Also:**
>
> - "Modifying a PDB at the System Level"
> - See "About Container Access in a CDB".

Monitoring PDBs Managed by Oracle Database Resource Manager

A set of dynamic performance views enables you to monitor the results of your Oracle Database Resource Manager settings for PDBs.

About Resource Manager Views for PDBs

You can monitor the results of your Oracle Database Resource Manager settings for PDBs using views.

The following views are available:

- `V$RSRCPDBMETRIC`

  The `V$RSRCPDBMETRIC` view provides current statistics on resource consumption for PDBs, including CPU usage, parallel execution, I/O generated, and memory usage.

- `V$RSRCPDBMETRIC_HISTORY`

  The columns in the `V$RSRCPDBMETRIC_HISTORY` view are the same as the columns in the `V$RSRCPDBMETRIC` view. The only difference between these views is that the `V$RSRCPDBMETRIC` view contains metrics for the past one minute only, whereas the `V$RSRCPDBMETRIC_HISTORY` view contains metrics for the last 60 minutes.

- `V$RSRC_PDB`

  The `V$RSRC_PDB` view provides cumulative statistics. The statistics are accumulated since the time that the CDB resource plan was set.

- `DBA_HIST_RSRC_PDB_METRIC`

  This view contains the historical statistics of `V$RSRCPDBMETRIC_HISTORY`, taken using Automatic Workload Repository (AWR) snapshots.

> **Note:**
>
> The `V$RSRCPDBMETRIC` and `V$RSRCPDBMETRIC_HISTORY` views record statistics for resources that are not currently being managed by Resource Manager when the `STATISTICS_LEVEL` initialization parameter is set to `ALL` or `TYPICAL`.

> **See Also:**
>
> - *Oracle Database SQL Tuning Guide* for more information about real-time SQL monitoring
> - *Oracle Database Reference* to learn about `V$RSRCPDBMETRIC`, `V$RSRCPDBMETRIC_HISTORY`, `V$RSRC_PDB`, and `DBA_HIST_RSRC_PDB_METRIC`

Monitoring CPU Usage for PDBs

The `V$RSRCPDBMETRIC` view enables you to track CPU metrics in milliseconds, in terms of number of sessions, or in terms of utilization for the past one minute.

The view provides real-time metrics for each PDB and is very useful in scenarios where you are running workloads and want to continuously monitor CPU resource utilization.

The active CDB resource plan manages CPU usage for a PDB. Use this view to compare the maximum and average CPU utilization for PDBs with other PDB settings such as the following:

- CPU time used

- Time waiting for CPU
- Average number of sessions that are consuming CPU
- Number of sessions that are waiting for CPU allocation

For example, you can view the amount of CPU resources a PDB used and how long it waited for resource allocation. Alternatively, you can view how many sessions from each PDB are executed against the total number of active sessions.

Tracking CPU Consumption in Terms of CPU Utilization for PDBs

To track CPU consumption in terms of CPU utilization, query the `CPU_UTILIZATION_LIMIT` and `AVG_CPU_UTILIZATION` columns. `AVG_CPU_UTILIZATION` lists the average percentage of the server's CPU that is consumed by a PDB. `CPU_UTILIZATION_LIMIT` represents the maximum percentage of the server's CPU that a PDB can use. This limit is set using the `UTILIZATION_LIMIT` directive attribute.

The following query displays this information by showing the container ID (`CON_ID`) and name of each PDB:

```
COLUMN PDB_NAME FORMAT A10

SELECT r.CON_ID,
       p.PDB_NAME,
       r.CPU_UTILIZATION_LIMIT,
       r.AVG_CPU_UTILIZATION
FROM   V$RSRCPDBMETRIC r,
       CDB_PDBS p
WHERE r.CON_ID = p.CON_ID;
```

Tracking CPU Consumption and Throttling for PDBs

Use the `CPU_CONSUMED_TIME` and `CPU_TIME_WAIT` columns to track CPU consumption and throttling in milliseconds for each PDB. The column `NUM_CPUS` represents the number of CPUs that Resource Manager is managing.

The following query displays this information by showing the container ID (`CON_ID`) and name of each PDB:

```
COLUMN PDB_NAME FORMAT A10

SELECT r.CON_ID,
       p.PDB_NAME,
       r.CPU_CONSUMED_TIME,
       r.CPU_WAIT_TIME,
       r.NUM_CPUS
FROM   V$RSRCPDBMETRIC r,
       CDB_PDBS p
WHERE r.CON_ID = p.CON_ID;
```

Tracking CPU Consumption and Throttling in Terms of Number of Sessions for PDBs

To track the CPU consumption and throttling in terms of number of sessions, use the `RUNNING_SESSIONS_LIMIT`, `AVG_RUNNING_SESSIONS`, and `AVG_WAITING_SESSIONS` columns. `RUNNING_SESSIONS_LIMIT` lists the maximum number of sessions from

a particular PDB that can be running at any time. This limit is defined by the UTILIZATION_LIMIT directive attribute that you set for the PDB. AVG_RUNNING_SESSIONS lists the average number of sessions that are consuming CPU, and AVG_WAITING_SESSIONS lists the average number of sessions that are waiting for CPU.

The following query displays this information by showing the container ID (CON_ID) and name of each PDB:

```
COLUMN PDB_NAME FORMAT A10

SELECT r.CON_ID,
       p.PDB_NAME,
       r.RUNNING_SESSIONS_LIMIT,
       r.AVG_RUNNING_SESSIONS,
       r.AVG_WAITING_SESSIONS
FROM   V$RSRCPDBMETRIC r,
       CDB_PDBS p
WHERE r.CON_ID = p.CON_ID;
```

Monitoring Parallel Execution for PDBs

The V$RSRCPDBMETRIC view enables you to track parallel statements and parallel server use for PDBs.

Parallel execution servers for a PDB are managed with the active CDB resource plan of the PDB's CDB. To track parallel statements and parallel server use for PDBs, use the AVG_ACTIVE_PARALLEL_STMTS, AVG_QUEUED_PARALLEL_STMTS, AVG_ACTIVE_PARALLEL_SERVERS, AVG_QUEUED_PARALLEL_SERVERS, and PARALLEL_SERVERS_LIMIT columns.

AVG_ACTIVE_PARALLEL_STMTS and AVG_ACTIVE_PARALLEL_SERVERS list the average number of parallel statements running and the average number of parallel servers used by the parallel statements. AVG_QUEUED_PARALLEL_STMTS and AVG_QUEUED_PARALLEL_SERVERS list the average number of parallel statements queued and average number of parallel servers that were requested by queued parallel statements. PARALLEL_SERVERS_LIMIT lists the number of parallel servers allowed to be used by the PDB.

The following query displays this information by showing the container ID (CON_ID) and name of each PDB:

```
COLUMN PDB_NAME FORMAT A10

SELECT r.CON_ID, p.PDB_NAME, r.AVG_ACTIVE_PARALLEL_STMTS,
r.AVG_QUEUED_PARALLEL_STMTS,
   r.AVG_ACTIVE_PARALLEL_SERVERS, r.AVG_QUEUED_PARALLEL_SERVERS,
r.PARALLEL_SERVERS_LIMIT
   FROM V$RSRCPDBMETRIC r, CDB_PDBS p
   WHERE r.CON_ID = p.CON_ID;
```

Monitoring the I/O Generated by PDBs

The V$RSRCPDBMETRIC view enables you to track the amount of I/O generated by PDBs.

I/O is limited for a PDB by setting the MAX_IOPS initialization parameter or the MAX_MBPS initialization parameter in the PDB. Use this view to compare the I/O generated by PDBs in terms of the number of operations each second and the number of megabytes each second.

Tracking the Number of I/O Operations Generated Each Second by PDBs

To track the I/O operations generated each second by PDBs during the previous minute, use the IOPS column.

The following query displays this information by showing the container ID (CON_ID) and name of each PDB:

```
COLUMN PDB_NAME FORMAT A10

SELECT r.CON_ID, p.PDB_NAME, r.IOPS
   FROM V$RSRCPDBMETRIC r, CDB_PDBS p
   WHERE r.CON_ID = p.CON_ID;
```

Tracking the Number Megabytes Generated for I/O Operations Each Second by PDBs

To track number of megabytes generated for I/O operations each second by PDBs during the previous minute, use the IOMBPS column.

The following query displays this information by showing the container ID (CON_ID) and name of each PDB:

```
COLUMN PDB_NAME FORMAT A10

SELECT r.CON_ID, p.PDB_NAME, r.IOMBPS
   FROM V$RSRCPDBMETRIC r, CDB_PDBS p
   WHERE r.CON_ID = p.CON_ID;
```

Monitoring Memory Usage for PDBs

The V$RSRCPDBMETRIC view enables you to track the amount memory used by PDBs.

Use this view to track the amount of SGA, PGA, buffer cache, and shared pool memory currently used by PDBs.

To track the current memory usage, in bytes, for specific PDBs, use the SGA_BYTES, PGA_BYTES, BUFFER_CACHE_BYTES, and SHARED_POOL_BYTES columns.

The following query displays this information by showing the container ID (CON_ID) and name of each PDB:

```
COLUMN PDB_NAME FORMAT A10
```

```
SELECT r.CON_ID, p.PDB_NAME, r.SGA_BYTES, r.PGA_BYTES,
r.BUFFER_CACHE_BYTES, r.SHARED_POOL_BYTES
   FROM V$RSRCPDBMETRIC r, CDB_PDBS p
   WHERE r.CON_ID = p.CON_ID;
```

23
Using Oracle Scheduler with a CDB

You can use Oracle Scheduler to schedule jobs in a multitenant container database (CDB).

Before using Oracle Scheduler with a CDB, meet the following requirements:

- You understand how to configure and manage a CDB.
- You understand how to use Oracle Scheduler to schedule jobs in a non-CDB.

> **See Also:**
>
> - "Administering a Multitenant Environment"
> - *Oracle Database Administrator's Guide*

DBMS_SCHEDULER Invocations in a CDB

Most scheduler calls work the same way as they do in non-CDBs, except for two scheduler global attributes.

The `JOB_QUEUE_PROCESSES` initialization parameter specifies the maximum number of job slaves per instance that can be created for the execution of `DBMS_JOB` jobs and Oracle Scheduler (`DBMS_SCHEDULER`) jobs. The range of values is `0` to `4000` (default).

To limit job slaves in a CDB environment, you can set `JOB_QUEUE_PROCESSES` in the following locations:

- CDB root

  Set `JOB_QUEUE_PROCESSES` to the maximum number of slave processes that Scheduler can use simultaneously in the entire database instance.

  If `JOB_QUEUE_PROCESSES` is `0` in the CDB root, then `DBMS_JOB` and Oracle Scheduler jobs cannot run in the root or any PDB, regardless of the `JOB_QUEUE_PROCESSES` setting at the PDB level.

- PDB

  Set `JOB_QUEUE_PROCESSES` to the maximum number of simultaneous jobs for this PDB. The actual number depends on the resources assigned by Resource Manager and the demand in other containers. When multiple PDBs request jobs, Oracle Scheduler attempts to give all PDBs a fair share of the processes.

  If `JOB_QUEUE_PROCESSES` is `0` in a PDB, then `DBMS_JOB` and Oracle Scheduler jobs cannot run in this PDB, regardless of the `JOB_QUEUE_PROCESSES` setting in the CDB root.

You must set all global Oracle Scheduler attributes at the PDB level. For example, if you set the `EMAIL_SENDER` attribute in the root database using

DBMS_SCHEDULER.SET_ATTRIBUTE, then it applies to the jobs that run in the root, not the jobs running in a specific PDB. If you choose a new EMAIL_SENDER for a PDB, then you must set the global attribute in this PDB.

> **See Also:**
>
> *Oracle Database Reference* to learn more about JOB_QUEUE_PROCESSES

Job Coordinator and Slave Processes in a CDB

The major CDB-related changes are to the job coordinator process.

In a non-CDB, the coordinator looks at all jobs that are ready to run, picks a subset of them to run, and assigns them to job slaves. It also opens and closes windows, which changes the resource plan in effect for the database.

That is essentially what happens inside a CDB except for the following:

- Jobs are selected from all PDBs

  The coordinator looks at the root database and all the child PDBs and selects jobs based on the job priority, the job scheduled start time, and the availability of resources to run the job. The latter criterion depends on the consumer group of the job and the resource plan currently in effect. The coordinator makes no attempt to be fair to every PDB. The only way to ensure that jobs from a PDB are not starved is to allocate enough resources to it.

- Windows are open in the PDB and root database levels

  In a non-CDB, only one window can be open at any given time. In a CDB, there are two levels of windows. At the PDB level, windows can be used to set resource plans that allocate resources among consumer groups belonging to that PDB. At the root database level, windows can allocate resources to different PDBs. Therefore, at any time, there can be a window open in the root database and one in each PDB.

- Job slave switches to the specific PDB it belongs to

  The job slaves are essentially the same as in a non-CDB, except that when a slave executes a job, it switches to the PDB that the job belongs to and then executes it. The rest of the code is essentially unchanged.

DBMS_JOB and DBMS_SCHEDULER

You can use DBMS_JOB within a PDB only if you grant the CREATE JOB privilege to the schemas that submit DBMS_JOB jobs.

The DBMS_JOB interface is implemented using DBMS_SCHEDULER. For this reason, Oracle recommends that you switch from DBMS_JOB to DBMS_SCHEDULER.

For the scheduler, the coordinator selects jobs to run from every PDB. Also, for the scheduler, the slave process switches into a PDB before executing a job; otherwise, the code is essentially unchanged.

> **See Also:**
>
> *Oracle Database Administrator's Guide* for information about support for `DBMS_JOB`

Processes to Close a PDB

If a PDB is closed with the immediate option, then the coordinator terminates jobs running in the PDB, and the jobs must be recovered before they can run again.

In an Oracle RAC database, the coordinator can, in most cases, recover the jobs on another instance where that PDB is open. So, if the coordinator on the first instance can find another instance where the PDB is still open, it moves the jobs there. In certain cases, moving the jobs to another instance may not be possible. For example, if the PDB in question is not open anywhere else, the jobs cannot be moved. Also, moving a job to another instance is not possible when the job has the `INSTANCE_ID` attribute set. In this case the job cannot run until the PDB on that instance is open again.

In a non-Oracle RAC case, the question of moving jobs does not arise. Terminated jobs can only be recovered after the PDB is opened again.

New and Changed CDB Views

Some CDB views are specific to CDBs, whereas others have a CDB-specific column.

- `V$` and `GV$` views have a `CON_ID` column that identifies a container whose data is represented by a `CDB_*` row. In non-CDBs, the `CON_ID` column is `NULL`.
- `CDB_*` views correspond to all Scheduler `DBA_*` views.

  In a PDB, these views only show objects visible through a corresponding `DBA_*` view, but all objects are visible in the root. The `CDB_*` view contains all columns found in a given `DBA_*` view and the column (`CON_ID`). In non-CDBs, this column is `NULL`.

24

Using Oracle Database Vault with a CDB

You can use Oracle Database Vault in a multitenant container database (CDB).

About Oracle Database Vault

The Oracle Database Vault security controls protect application data from unauthorized access, and comply with privacy and regulatory requirements.

You can deploy controls to block privileged account access to application data and control sensitive operations inside the database using trusted path authorization. Through the analysis of privileges and roles, you can increase the security of existing applications by using least privilege best practices. Oracle Database Vault secures existing database environments transparently, eliminating costly and time consuming application changes.

Related Topics

- *Oracle Database Vault Administrator's Guide*

How Oracle Database Vault Works in a Multitenant Environment

To provide increased security for consolidation, you can use Oracle Database Vault with Oracle Multitenant.

Oracle Database Vault can prevent privileged user access inside a pluggable database (PDB) and between the PDB and the common privileged user at the container database. Each PDB has its own Database Vault metadata, such as realms, rule sets, command rules, default policies (such as default realms), and so on. In addition, the objects within the `DVSYS` and `DVF` schemas are automatically available to any child PDBs. Both schemas are common user schemas.

You can configure common realms in the application root only, but you can create common rule sets and command rules in either the application root or the CDB root. A common command rule in the application root applies to its associated PDBs, and common command rules in the CDB root apply to all PDBs in the CDB environment. The ability to create common realms and command rules enables you to create policies that use a shared set of realms, rule sets, or command rules throughout the CDB environments, rather than having to create these same components for every PDB in the multitenant environment. The common protection applies for all PDBs associated with the application root that have Oracle Database Vault enabled.

You can create individual local policies for each PDB. When you use Database Vault to protect an object, Database Vault subjects common privileges for common objects to the same enforcement rules as local system privileges.

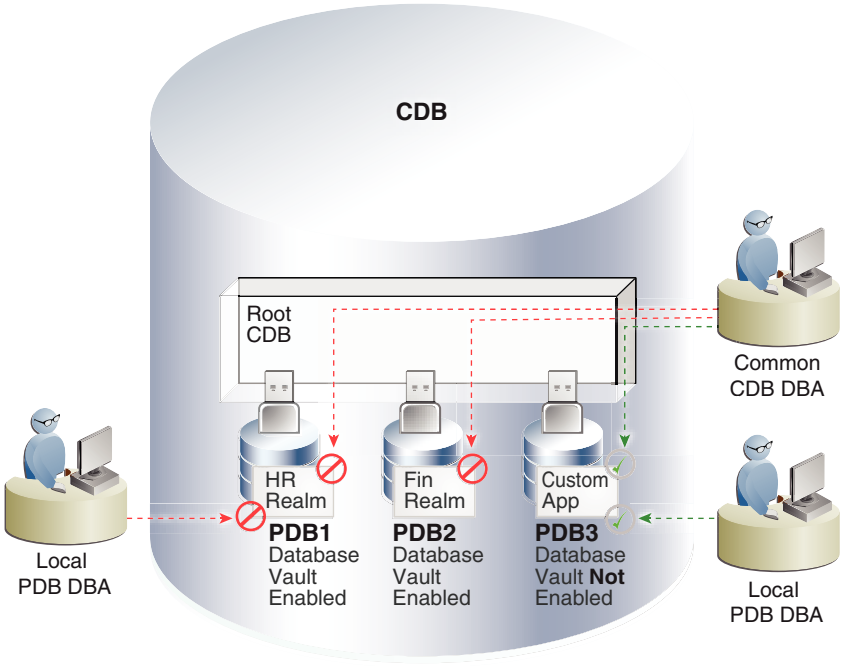
When you configure a PDB that has Database Vault enabled, the `DVSYS` schema is a common user schema that is stored in the root. This means that all the objects within

the DVSYS schema (tables, data dictionary views, user accounts, PL/SQL packages, default policies, and so on) are subject to the common privileges available for this schema. In other words, you can create realms, factors, and so on in the root to protect the schema in the root. Ensure that you configure Database Vault in the root first, before you configure it in the associated PDBs.

When you enable Oracle Database Vault in the CDB root, you can choose either regular mode or strict mode. The settings propagate throughout the CDB based on the setting you choose. For example, suppose a CDB contains both Database Vault-enabled PDBs and PDBs in which Database Vault is not enabled. If you enable Database Vault using regular mode, then both types of PDBs continue to function normally. If you enable Database Vault using strict mode, then the Database Vault-disabled PDBs operate in restricted mode.

Figure 24-1 illustrates how the database in regular mode allows different access for common and local database administrators depending if Database Vault is enabled. In this scenario, neither the common user nor the local users have access to the realms in PDB1 and PDB2. Both the common user and the PDB3 local user have access to the Custom App application in PDB3, where Database Vault is not enabled.

Figure 24-1 Oracle Database Vault in a Multitenant Environment with Regular Mode



Related Topics

- Realms in a Multitenant Environment
  In a multitenant environment, you can create a realm to protect common objects in the application root.
- Rule Sets and Rules in a Multitenant Environment
  In a multitenant environment, you can create a rule set and its associated rules in the application root.

- Command Rules in a Multitenant Environment
  In a multitenant environment, you can create common and local command rules in either the CDB root or the application root.
- Converting a Standalone Oracle Database to a PDB and Plugging It into a CDB
  You can convert a standalone Oracle Database Release 12c or later database to a PDB, and then plug this PDB into a CDB.

Verifying That Database Vault Is Configured and Enabled

The DBA_DV_STATUS, CDB_DV_STATUS, DBA_OLS_STATUS, and CDB_OLS_STATUS data dictionary views verify if Oracle Database is configured and enabled.

In addition to Oracle Database Vault administrators, the Oracle Database SYS user and users who have been granted the DBA role can query these views.

- For Database Vault:
  - If you want to find the Database Vault status for a non-multitenant database, or in a multitenant environment for the root only or an individual PDB, then query DBA_DV_STATUS. For example:

    ```
    SELECT * FROM DBA_DV_STATUS;
    ```

    Output similar to the following appears:

    ```
    NAME                 STATUS
    -------------------- -----------
    DV_CONFIGURE_STATUS  TRUE
    DV_ENABLE_STATUS     TRUE
    ```

  - If you want to find the Database Vault status of all PDBs in a multitenant environment, as a common user with administrative privileges, then query CDB_DV_STATUS, which provides the addition of a container ID (CON_ID) field.
- For Oracle Label Security, query the following data dictionary views, which are similar to their Database Vault equivalent views:
  - DBA_OLS_STATUS
  - CDB_OLS_STATUS

Registering Oracle Database Vault with an Oracle Database in a Multitenant Environment

You can register Oracle Database Vault in a multitenant environment based on several scenarios.

Registering Database Vault in the CDB Root

In a multitenant environment, you register Oracle Database Vault with common users who will use the Database Vault-enforced roles in the CDB root.

1. In a multitenant environment, log into the root of the database instance as a user who has privileges to create users and grant the `CREATE SESSION` and `SET CONTAINER` privileges.

   For example:

   ```
   sqlplus c##dba_debra
   Enter password: password
   ```

2. Select user accounts (or create new users) that will be used for the Database Vault Owner (`DV_OWNER` role) and Database Vault Account Manager (`DV_ACCTMGR` role) accounts.

   Oracle strongly recommends that you maintain two accounts for each role. One account, the primary named user account, will be used on a day-to-day basis and the other account will be used as a backup account in case the password of the primary account is lost and must be reset.

   Prepend the names of these accounts with `c##` or `C##`. For example:

   ```
   GRANT CREATE SESSION, SET CONTAINER TO c##sec_admin_owen
     IDENTIFIED BY password CONTAINER = ALL;
   GRANT CREATE SESSION, SET CONTAINER TO c##dbv_owner_root_backup
     IDENTIFIED BY password CONTAINER = ALL;
   GRANT CREATE SESSION, SET CONTAINER TO c##accts_admin_ace
     IDENTIFIED BY password CONTAINER = ALL;
   GRANT CREATE SESSION, SET CONTAINER TO c##dbv_acctmgr_root_backup
     IDENTIFIED BY password CONTAINER = ALL;
   ```

   In this specification:

   - Create the primary accounts (`c##sec_admin_owen` and `c##accts_admin_ace`) if these do not already exist for the new roles, `DV_ADMIN` and `DV_ACCTMGR`.
   - Replace *password* with a password that is secure.

3. Connect to the root as user `SYS` with the `SYSDBA` administrative privilege

   ```
   CONNECT SYS AS SYSDBA
   Enter password: password
   ```

4. Configure the two backup Database Vault user accounts.

   For example:

   ```
   BEGIN
    CONFIGURE_DV (
      dvowner_uname         => 'c##dbv_owner_root_backup',
      dvacctmgr_uname       => 'c##dbv_acctmgr_root_backup',
      force_local_dvowner   => FALSE);
    END;
   /
   ```

   In this example, setting `force_local_dvowner` to `FALSE` enables the common users to have `DV_OWNER` privileges for the PDBs that are associated with this CDB root. Setting it to `TRUE` restricts the common `DV_OWNER` user to have the `DV_OWNER` role

privileges for the CDB root only. If you grant DV_OWNER locally to the CDB root common user, then that user cannot grant the DV_OWNER role commonly to any other user.

5. Run the utlrp.sql script to recompile invalidated objects in the root.

```
@?/rdbms/admin/utlrp.sql
```

If the script provides instructions, follow them, and then run the script again. If the script terminates abnormally without giving any instructions, then run it again.

6. Connect to the root as the primary Database Vault Owner user that you just configured.

For example:

```
CONNECT c##dbv_owner_root_backup
Enter password: password
```

7. Enable Oracle Database Vault using one of the following commands:

- To enable Oracle Database Vault to use regular mode:

```
EXEC DBMS_MACADM.ENABLE_DV;
```

- If every associated PDB will need to have Database Vault enabled in this database, then use the following command. (You will need to enable each of these PDBs after you complete this procedure.) PDBs that do not have Database Vault enabled will be in restricted mode after the database is restarted and until Database Vault is enabled in the PDB:

```
EXEC DBMS_MACADM.ENABLE_DV (strict_mode => 'y');
```

8. Connect with the SYSDBA administrative privilege.

```
CONNECT / AS SYSDBA
```

9. Restart the database.

```
SHUTDOWN IMMEDIATE
STARTUP
```

10. Verify that Oracle Database Vault and Oracle Label Security are installed and enabled.

```
SELECT * FROM DBA_DV_STATUS;
SELECT * FROM DBA_OLS_STATUS;
```

11. Connect as the backup DV_OWNER user and then grant the DV_OWNER role to the primary DV_OWNER user that you created earlier.

For example:

```
CONNECT c##dbv_owner_root_backup
Enter password: password
```

```
GRANT DV_OWNER TO c##sec_admin_owen WITH ADMIN OPTION;
```

12. Connect as the backup `DV_ACCTMGR` user and then grant the `DV_ACCTMGR` role to the backup `DV_ACCTMGR` user.

    For example:

    ```
    CONNECT c##dbv_acctmgr_root_backup
    Enter password: password

    GRANT DV_ACCTMGR TO c##accts_admin_ace WITH ADMIN OPTION;
    ```

13. Store the two backup account passwords in a safe location such as a privileged account management (PAM) system in case they are needed in the future.

Related Topics

- Verifying That Database Vault Is Configured and Enabled
  The `DBA_DV_STATUS`, `CDB_DV_STATUS`, `DBA_OLS_STATUS`, and `CDB_OLS_STATUS` data dictionary views verify if Oracle Database is configured and enabled.
- *Oracle Database Vault Administrator's Guide*
- *Oracle Database Vault Administrator's Guide*

Registering Database Vault Common Users to Manage Specific PDBs

In a multitenant environment, you must register Oracle Database Vault in the root first, then in the PDBs afterward.

If you try to register in a PDB first, then an `ORA-47503: Database Vault is not enabled on CDB$ROOT` error appears.

1. If you have not already done so, then identify or create named common user accounts to be used as the Database Vault accounts along with associated backup accounts.
2. Ensure that you have registered Oracle Database Vault in the CDB root and that the `DV_OWNER` role was granted commonly to the common user.
3. Connect to the PDB as an administrator who is local to the PDB.

   For example:

   ```
   CONNECT dba_debra@pdb_name
   Enter password: password
   ```

   To find the available PDBs, query the `DBA_PDBS` data dictionary view. To check the current PDB, run the `show con_name` command.
4. Grant the `CREATE SESSION` and `SET CONTAINER` privileges to the users for this PDB.

   For example:

   ```
   GRANT CREATE SESSION, SET CONTAINER TO c##sec_admin_owen CONTAINER
   = CURRENT;
   ```

```
GRANT CREATE SESSION, SET CONTAINER TO c##accts_admin_ace CONTAINER
= CURRENT;
```

5. Connect as user SYS with the SYSDBA administrative privilege

```
CONNECT SYS@pdb_name AS SYSDBA
Enter password: password
```

6. While still in the PDB, configure the two backup Database Vault user accounts.

```
BEGIN
 CONFIGURE_DV (
   dvowner_uname         => 'c##dbv_owner_root_backup',
   dvacctmgr_uname       => 'c##dbv_acctmgr_root_backup');
 END;
/
```

   In this example, the force_local_dvowner parameter is omitted because it is unnecessary. All common users who are configured within a PDB are restricted to the scope of the PDB.

7. Run the utlrp.sql script to recompile invalidated objects in this PDB.

```
@?/rdbms/admin/utlrp.sql
```

   If the script provides instructions, follow them, and then run the script again. If the script terminates abnormally without giving any instructions, then run it again.

8. Connect to the PDB as the backup Database Vault Owner user that you just configured.

   For example:

```
CONNECT c##dbv_owner_root_backup@pdb_name
Enter password: password
```

9. Enable Oracle Database Vault in this PDB.

```
EXEC DBMS_MACADM.ENABLE_DV;
```

10. Connect to the CDB with the SYSDBA administrative privilege.

```
CONNECT / AS SYSDBA
```

11. Close and reopen the PDB.

    For example:

```
ALTER PLUGGABLE DATABASE pdb_name CLOSE IMMEDIATE;
ALTER PLUGGABLE DATABASE pdb_name OPEN;
```

12. Verify that the PDB is configured and enabled for Database Vault.

```
CONNECT SYS@pdb_name AS SYSDBA
Enter password: password
```

```
SELECT * FROM DBA_DV_STATUS;
```

13. Connect as the backup `DV_OWNER` user and then grant the `DV_OWNER` role to the primary `DV_OWNER` user that you created earlier.

    For example:

    ```
    CONNECT c##dbv_owner_root_backup@pdb_name
    Enter password: password

    GRANT DV_OWNER TO c##sec_admin_owen WITH ADMIN OPTION;
    ```

14. Connect as the backup `DV_ACCTMGR` user and then grant the `DV_ACCTMGR` role to the primary `DV_ACCTMGR` user.

    For example:

    ```
    CONNECT c##dbv_acctmgr_root_backup@pdb_name
    Enter password: password

    GRANT DV_ACCTMGR TO c##accts_admin_ace WITH ADMIN OPTION;
    ```

15. Store the two backup account passwords in a safe location such as a privileged account management (PAM) system in case they are needed in the future.

Related Topics

- Verifying That Database Vault Is Configured and Enabled
  The `DBA_DV_STATUS`, `CDB_DV_STATUS`, `DBA_OLS_STATUS`, and `CDB_OLS_STATUS` data dictionary views verify if Oracle Database is configured and enabled.
- *Oracle Database Vault Administrator's Guide*
- *Oracle Database Vault Administrator's Guide*

Registering Database Vault Local Users to Manage Specific PDBs

You must register Oracle Database Vault in the root first, and then in the PDBs afterward.

If you try to register in a PDB first, then an `ORA-47503: Database Vault is not enabled on CDB$ROOT` error appears.

1. Log in to the PDB as a user who has privileges to create users and to grant the `CREATE SESSION` and `SET CONTAINER` privileges.

   For example:

   ```
   sqlplus sec_admin@pdb_name
   Enter password: password
   ```

   To find the available PDBs, query the `DBA_PDBS` data dictionary view. To check the current PDB, run the `show con_name` command.

2. If you are not using existing local user named accounts for the new Database Vault roles, create new named local user accounts.

In both cases, you must create backup accounts to hold the Database Vault roles in case the named user loses or forgets their password.

```
GRANT CREATE SESSION, SET CONTAINER TO sec_admin_owen
  IDENTIFIED BY password;
GRANT CREATE SESSION, SET CONTAINER TO dbv_owner_backup
  IDENTIFIED BY password;
GRANT CREATE SESSION, SET CONTAINER TO accts_admin_ace
  IDENTIFIED BY password;
GRANT CREATE SESSION, SET CONTAINER TO dbv_acctmgr_backup
  IDENTIFIED BY password;
```

3. Ensure that you have registered Oracle Database Vault in the CDB root.

   Temporarily connect to the root and then query the `DBA_DV_STATUS` view.

   ```
   CONNECT SYS / AS SYSDBA
   Enter password: password

   SELECT * FROM DBA_DV_STATUS;
   ```

4. Connect to the PDB as user `SYS` with the `SYSDBA` administrative privilege.

   ```
   CONNECT SYS@pdb_name AS SYSDBA
   Enter password: password
   ```

5. While still in the PDB, configure the two backup Database Vault user accounts.

   ```
   BEGIN
    CONFIGURE_DV (
      dvowner_uname     => 'dbv_owner_backup',
      dvacctmgr_uname   => 'dbv_acctmgr_backup');
    END;
   /
   ```

   In this example, the `force_local_dvowner` parameter is omitted because it is unnecessary. Database Vault roles are granted locally when configured in a PDB.

6. Run the `utlrp.sql` script to recompile invalidated objects in this PDB.

   ```
   @?/rdbms/admin/utlrp.sql
   ```

   If the script provides instructions, follow them, and then run the script again. If the script terminates abnormally without giving any instructions, run it again.

7. Connect to the PDB as the backup Database Vault Owner user that you just configured.

   For example:

   ```
   CONNECT dbv_owner_backup@pdb_name
   Enter password: password
   ```

8. Enable Oracle Database Vault in this PDB.

```
EXEC DBMS_MACADM.ENABLE_DV;
```

9. Connect to the CDB with the SYSDBA administrative privilege.

```
CONNECT / AS SYSDBA
```

10. Close and reopen the PDB.

```
ALTER PLUGGABLE DATABASE pdb_name CLOSE IMMEDIATE;
ALTER PLUGGABLE DATABASE pdb_name OPEN;
```

11. Verify that the PDB is configured and enabled for Database Vault and Oracle Label Security.

```
CONNECT SYS@pdb_name AS SYSDBA
Enter password: password

SELECT * FROM DBA_DV_STATUS;
SELECT * FROM DBA_OLS_STATUS;
```

12. Connect as the backup DV_OWNER user and then grant the DV_OWNER role to the primary DV_OWNER user that you created earlier.

    For example:

```
CONNECT dbv_owner_backup@pdb_name
Enter password: password

GRANT DV_OWNER TO sec_admin_owen WITH ADMIN OPTION;
```

13. Connect as the backup DV_ACCTMGR user and then grant the DV_ACCTMGR role to the backup DV_ACCTMGR user.

    For example:

```
CONNECT dbv_acctmgr_backup@pdb_name
Enter password: password

GRANT DV_ACCTMGR TO c##accts_admin_ace WITH ADMIN OPTION;
```

14. Store the two backup account passwords in a safe location such as a privileged account management (PAM) system in case they are needed in the future.

Plugging in a Database Vault-Enabled PDB

From SQL*Plus, in a multitenant environment, you can plug in a database that already has Database Vault enabled.

In this scenario, the plugged in database has its own local Database Vault accounts. Be aware that if you plug a Database Vault-enabled database into a CDB that is not Database Vault enabled, then the PDB will remain in restricted mode until you enable Database Vault in the CDB and then restart the CDB. If you plug a non-Database Vault-enabled PDB into a CDB that is Database Vault enabled, then the PDB remains

in restricted mode until you enable Database Vault in the PDB and then restart the PDB. This plugged in non-Database Vault enabled PDB can still be used. However, if the CDB is Database Vault enabled with the strict option set, then the PDB must be Database Vault enabled.

Before you plug in a Database Vault-enabled PDB and if the Database Vault roles are granted to common users, ensure that you understand fully how plugging in PDBs affect common users.

Related Topics

- *Oracle Database Security Guide*

Manually Installing Oracle Database Vault in a Multitenant Environment

Under certain conditions, for a multitenant environment, you must manually install Oracle Database Vault.

For example, you must manually install Oracle Database Vault if a release 11g Oracle database without Database Vault is upgraded to release 12c, then converted to a PDB to be plugged into a 12c Database Vault-enabled database. In addition, you must manually install Oracle Database Vault (and Oracle Label Security) in a PDB if this PDB does not have these products when the PDB has been plugged into a CDB where Database Vault and Label Security are installed.

1. As user who has been granted the `SYSDBA` administrative privilege, log in to the PDB in which you want to install Oracle Database Vault.

   For example, to log in to a PDB named `hr_pdb`:

   ```
   sqlplus sec_admin@hr_pdb as sysdba
   Enter password: password
   ```

   To find the available PDBs, run the `show pdbs` command. To check the current PDB, run the `show con_name` command.

2. If necessary, check if Oracle Database Vault and Oracle Label Security are already installed on this PDB.

   If the `DVSYS` account (for Database Vault) and the `LBACSYS` account (for Label Security) exist, then Database Vault and Label Security exist on the PDB.

   ```
   SELECT USERNAME FROM DBA_USERS WHERE USERNAME IN ('DVSYS',
   'LBACSYS');
   ```

3. If neither Database Vault nor Label Security have been installed, then install Oracle Label Security by executing the `catols.sql` script.

   ```
   @$ORACLE_HOME/rdbms/admin/catols.sql
   ```

   Oracle Label Security must be installed before you can install Oracle Database Vault.

4. Install Oracle Database Vault by executing the `catmac.sql` script.

   ```
   @$ORACLE_HOME/rdbms/admin/catmac.sql
   ```

5. At the `Enter value for 1` prompt, enter `SYSTEM` as the tablespace to install `DVSYS`.
6. At the `Enter value for 2` prompt, enter the temporary tablespace for the PDB.

After the installation is complete, you can register Oracle Database Vault in the PDB. If Database Vault is not registered in the CDB already, you must close the PDB before you can register Database Vault in the CDB root. Database Vault must be registered in CDB root before it can be registered in the PDB. After Database Vault is registered in the CDB root and the database has been restarted, then you can open the PDB and register Database Vault.

Related Topics

- *Oracle Database Vault Administrator's Guide*

Configuring Realms

You can create a realm around database objects to protect them, and then set authorizations to control user access to this data.

What Are Realms?

Realms enable you to protect database objects, including specific object types.

About Realms

A realm is a grouping of database schemas, database objects, and database roles that must be secured for a given application.

Think of a realm as zone of protection for your database objects. A schema is a logical collection of database objects such as tables, views, and packages, and a role is a collection of privileges. By arranging schemas and roles into functional groups, you can control the ability of users to use system privileges against these groups and prevent unauthorized data access by the database administrator or other powerful users with system privileges. Oracle Database Vault does not replace the discretionary access control model in the existing Oracle database. It functions as a layer on top of this model for both realms and command rules.

Oracle Database Vault provides two types of realms: regular and mandatory. Both realm types can protect either an entire schema or crucial objects within a schema selectively, such as tables and indexes. With a regular realm, an object owner or users who has been granted object privileges can perform queries or DML operations without realm authorization but must have realm authorization to perform DDL operations. A mandatory realm provides stronger protection for objects within a realm. Mandatory realms block both object privilege-based and system privilege-based access and will not allow users with object privileges to perform queries, DML, or DDL operations without realm authorization. In other words, even an object owner cannot access his or her own objects without proper realm authorization if the objects are protected by mandatory realms.

For databases that use Oracle Flashback Technology, then both regular and mandatory realms will enforce the same behavior for a flashback table. Users can

execute a `FLASHBACK TABLE` SQL statement on a realm-protected table if the user is authorized to the realm.

For databases that use Information Lifecycle Management (ILM), a Database Vault administrator can use the `DBMS_MACADM.AUTHORIZE_MAINTENANCE_USER` and `DBMS_MACADM.UNAUTHORIZE_MAINTENANCE_USER` procedure to control who can perform ILM operations on realm-protected objects.

You can register schemas, all objects of a certain type in a schema, or individual objects within a schema into a realm. After you create a realm, you can register a set of schema objects or roles (secured objects) for realm protection and authorize a set of users or roles to access the secured objects. Objects that are protected by a regular realm allow DML access to users who have direct object grants.

For example, you can create a realm to protect all existing database schemas that are used in an accounting department. The realm prohibits any user who is not authorized to the realm to use system privileges to access the secured accounting data. When an entire schema is protected, all objects in the schema are protected, including tables, indexes, procedures and other objects.

You can run reports on realms that you create in Oracle Database Vault. You can use simulation mode during development, test, and even production phases to log only realm violations instead of blocking access. This enables you to quickly test applications using Database Vault realms.

You can configure realms by using the Oracle Database Vault Administrator pages in Oracle Enterprise Manager Cloud Control. Alternatively, you can configure realms by using the PL/SQL interfaces and packages provided by Oracle Database Vault.

Realms in a Multitenant Environment

In a multitenant environment, you can create a realm to protect common objects in the application root.

The advantage of creating a realm in the application root instead of creating a large number objects and realms around these objects within individual pluggable databases (PDBs) is that you can create them in one place, the application root. This way, you can manage them centrally.

You cannot create a common realm in the CDB root.

A Database Vault common realm can be either a regular realm or a mandatory realm. The realm protects only objects within the application root, not local objects in a PDB. The CDB root, application root, and any affected PDBs all must be Database Vault enabled.

To configure a common realm, you must be commonly granted the `DV_OWNER` or `DV_ADMIN` role. To grant common authorizations for a common realm, you must be in the application root. To propagate the realm to the PDBs that are associated with the application root, you must synchronize the application root. For example, to synchronize an application called `saas_sales_app`

```
ALTER PLUGGABLE DATABASE APPLICATION saas_sales_app SYNC;
```

Related Topics

- *Oracle Database Vault Administrator's Guide*

Realm Authorizations in a Multitenant Environment

In a multitenant environment, the rules and behavior for common realm authorizations are similar to the authorizations for other common objects.

Local Authorization for a Common Realm

The local authorization for a common realm refers to the authorization a user has for the PDB that this user is accessing.

The rules for the local authorization for a common realm are as follows:

- A user who has been commonly granted the `DV_OWNER` or `DV_ADMIN` role can grant local authorization to common users, common roles, local users, and local roles. The common `DV_OWNER` or `DV_ADMIN` user can also remove local authorization from a common realm in a PDB.
- A local Database Vault administrator can authorize locally (that is, grant local authorizations to both local and common users) within the PDB. A common Database Vault administrator can also grant authorizations in each PDB. A common realm authorization can only be granted by a common Database Vault administrator in the application root.
- The common Database Vault administrator can both add or remove local authorization to and from a common realm from within the PDB.
- If a common user has only local authorization for a common realm, then this user cannot access the common realm in any other PDB than this local authorization.
- A common user or a common role can have both the local authorization and the common authorization to a common realm at the same time. Removing a common user's local authorization from a common realm does not affect the common user's common authorization. Removing a common user's common authorization from a common realm does not affect the common user's local authorization.

Common Authorization for a Common Realm

The common authorization for a common realm refers to the authorization a common user or a common role has in the application root while the authorization takes effect in every container that is Database Vault enabled.

The rules for the local authorization for a common realm are as follows:

- A user who has been commonly granted the `DV_OWNER` or `DV_ADMIN` role can grant common realm authorization to common users or roles in the application root. This common Database Vault administrator can perform the removal of common authorizations while in the application root.
- This common authorization applies to the containers that have been Database Vault enabled in the CDB.
- If a common user is authorized to a common realm in the application root, then this user has access to the objects protected by the common realm in the application root and any application PDBs.
- Any rule sets that are associated with a common realm must be common rule sets. The rules that are added to a common rule set that is associated with common authorization cannot involve any local objects.

How the Authorization of a Realm Works in Both the Application Root and in an Individual PDB

During the Database Vault enforcement in a container, a common realm performs the same enforcement behaviors as the same realm when it is used locally in a PDB.

Rule Sets and Rules in a Multitenant Environment

In a multitenant environment, you can create a rule set and its associated rules in the application root.

A common realm must use a common rule set when the associated realm or command rule is evaluated by Database Vault. The common rule set and its rules can only be created in the application root. After the common rule set is created, it exists in every container that is associated with the root where the common rule set is created. The common rule set can only include common rules.

To configure a common rule set and its rules, you must be commonly granted the `DV_OWNER` or `DV_ADMIN` role.

Related Topics

- Command Rules in a Multitenant Environment
  In a multitenant environment, you can create common and local command rules in either the CDB root or the application root.

Command Rules in a Multitenant Environment

In a multitenant environment, you can create common and local command rules in either the CDB root or the application root.

Common command rules can be associated only with common realms, rule sets, and rules. Local command rules can be associated only with local realm, rule sets, and rules.

To apply these command rules to the entire multitenant environment, you must execute the command rule procedures from the CDB root or application root as a common user who has been granted the `DVADM` or `DVOWNER` role. A common command rule that is created in the CDB root will be applied to all PDBs in that CDB environment. A common command rule that is created in the application root will only be applied to the PDBs that are associated with this application root. To propagate the command rule to the PDBs that are associated with the CDB root or application root, you must synchronize the PDB. For example, to synchronize an application root called `saas_sales_app` to its application PDBs:

```
ALTER PLUGGABLE DATABASE APPLICATION saas_sales_app SYNC;
```

To synchronize a common command rule in the CDB root to a PDB:

```
ALTER PLUGGABLE DATABASE APPLICATION APP$CDB$SYSTEM SYNC;
```

You can check a user's roles by querying the `USER_ROLE_PRIVS` data dictionary view. To find information about command rules, query the `DBA_DV_COMMAND_RULE` data dictionary view.

Oracle Database Vault Policies in a Multitenant Environment

Oracle Database Vault policies are only local to the pluggable database (PDB) in which they were created.

That is, if you created the policy in a PDB, then only local realms and command rules can be added to it. You cannot create Database Vault policies that can have common realms or common command rules.

Using Database Vault Operations Control to Restrict Multitenant Common User Access to Local PDB Data

You can control PDB access by CDB root common users, such as infrastructure database administrators.

Related Topics

- DBA Operations in an Oracle Database Vault Environment*Oracle Database Vault Administrator's Guide*

About Using Database Vault Operations Control

You can automatically restrict common users from accessing pluggable database (PDB) local data in autonomous, regular Cloud, or on-premises environments.

To accomplish this, you can use Oracle Database Vault operations control, which applies to common users such as infrastructure database administrators and applications.

Database Vault operations control is useful for situations where a database administrator must log in to the CDB root as a highly privileged user, but still not be able to access PDB customer data. Database operations control does not block PDB database administrators. To block these users, enable Oracle Database Vault in the PDB and then use the Database Vault features such as realm control to block these users.

You can create an exception list for Database Vault operations control of common users and packages for situations where a common user or application must perform tasks that must access local data on a PDB. An example of the type of common user that you would specify for the exception list is the `CTXSYS` application account, which is responsible for Oracle Text. Specifying a package in an exception list enables you to apply more fine-grained control instead of providing full access to a user in an exception list.

The general process for using Database Vault operations control is as follows:

1. Enable Database Vault operations control and keep it enabled for the production environment.
2. At this stage Database Vault operations control applies to all PDBs in the environment, regardless of whether the PDB has enabled Database Vault or not.
3. To enable specific users and packages to have access to the local schemas of the PDBs, add them to an exception list. When the user or package no longer needs access, then you can remove them from the exception list. For example, if the

database is using Oracle Text, then you can add the CTXSYS administrative user account and the package to the exception list.

Enabling Database Vault Operations Control

To enable Database Vault operations control, use the DBMS_MACADM.ENABLE_APP_PROTECTION PL/SQL procedure.

Oracle recommends that if you elect to use Database Vault operations control for your multitenant production server, then you should keep Database Vault operations control enabled full time.

In most cases, you will enable Database Operations control for the entire CDB, not just a specific PDB. If you need to disable it for a specific PDB (for example, for troubleshooting purposes), then you can execute the DBMS_MACADM.DISABLE_APP_PROTECTION procedure on the PDB. When you are finished troubleshooting the PDB, re-enable it for Database Vault operations control, as shown in the example in this topic.

Before you enable Database Vault operations control, Database Vault must be enabled and configured in the CDB root. However, Database Vault does not need to be enabled in the PDBs.

1. Log in to the CDB root as a common user who has been granted the DV_OWNER role.

   For example:

   ```
   sqlplus c##sec_admin_owen_root
   Enter password: password
   ```

2. Execute the DBMS_MACADM.ENABLE_APP_PROTECTION procedure.

   - To enable Database Vault operations control for all PDBs in the CDB environment:

     ```
     EXEC DBMS_MACADM.ENABLE_APP_PROTECTION;
     ```

   - The operations control for a specific PDB may have been disabled for troubleshooting reasons. To re-enable Database Vault operations control for a specific PBB (for example, HRPDB):

     ```
     EXEC DBMS_MACADM.ENABLE_APP_PROTECTION ('HRPDB');
     ```

At this stage, one or all of the PDBs are enabled for Database Vault operations control. You can confirm by connecting as user SYS with the SYSDBA administrative privilege and then executing the SELECT * FROM DBA_DV_STATUS; query. If specific trusted common users or packages must have access to the local schemas of these PDBs to perform special operations, then you can use the DBMS_MACADM.ADD_APP_EXCEPTION procedure to add the user or package to an exception list for Database Vault operations control.

Related Topics

- Adding Common Users and Packages to an Exception List*Oracle Database Vault Administrator's Guide*

Adding Common Users and Packages to an Exception List

Common users and applications that must access PDB local data can be added to an exception list.

Add a user package to the exception list if the package is the only object in the user account that needs access to the PDB local data. This allows for fine grained control over what is put into the exception list. The kinds of common users and packages that you would add to the exception list are ones that are necessary for the functioning of the PDB. For example, if you are using Oracle Spatial, then you should add the `MDSYS` account to the exception list. `MDSYS` requires access to customer PDB data for Oracle Spatial functions. To add a common user and a package to the Database Vault operations control exception list, you can use the `DBMS_MACADM.ADD_APP_EXCEPTION` PL/SQL procedure. To find existing exceptions, you can query the `DBA_DV_APP_EXCEPTION` data dictionary view.

1. Log in to the CDB root as a common user who has been granted the `DV_OWNER` role.

   For example:

   ```
   sqlplus c##sec_admin_owen_root
   Enter password: password
   ```

2. Ensure that the package that you will specify for the common user meets the following requirements:

   - The package must be owned by the common user.
   - A user-created package must be created with definer's rights procedures.

   You can find more information about user-created packages by querying the `DBA_OBJECTS` data dictionary view.

3. Execute the `DBMS_MACADM.ADD_APP_EXCEPTION` procedure.

   For example:

   ```
   DBMS_MACADM.ADD_APP_EXCEPTION ('MDSYS', 'PATCH_APP');
   ```

Deleting Common Users and Packages from an Exception List

Users and applications that no longer need to access PDB local data can be removed from the exception list.

To remove a common user and a package from the Database Vault operations control exception list, you can use the `DBMS_MACADM.DELETE_APP_PROTECTION` PL/SQL procedure. To find existing exceptions, you can query the `DBA_DV_APP_EXCEPTION` data dictionary view.

1. Log in to the CDB root as a common user who has been granted the `DV_OWNER` role.

   For example:

   ```
   sqlplus c##sec_admin_owen_root
   Enter password: password
   ```

2. Execute the `DBMS_MACADM.DELETE_APP_EXCEPTION` procedure.

   For example:

   ```
   DBMS_MACADM.DELETE_APP_EXCEPTION ('MDSYS', 'PATCH_APP');
   ```

Disabling Database Vault Operations Control

To disable Database Vault operations control, use the `DBMS_MACADM.DISABLE_APP_PROTECTION` PL/SQL procedure.

In most cases, you should keep Database Vault operations control enabled. If troubleshooting requires that a PDB be dropped from Database Vault operations control, then Oracle recommends that you temporarily disable Database Vault operations control for the PDB (and maintain operations control for the rest of the PDBs). After the troubleshooting is complete, then you should re-enable Database Vault operations control.

1. Log in to the CDB root as a common user who has been granted the `DV_OWNER` role.

   For example:

   ```
   sqlplus c##sec_admin_owen_root
   Enter password: password
   ```

2. Execute the `DBMS_MACADM.DISABLE_APP_PROTECTION` procedure.

   - To disable Database Vault operations control for all PDBs in the CDB environment:

     ```
     EXEC DBMS_MACADM.DISABLE_APP_PROTECTION;
     ```

   - To disable Database Vault operations control for a specific PBB (for example, `HRPDB`):

     ```
     EXEC DBMS_MACADM.DISABLE_APP_PROTECTION ('HRPDB');
     ```

Converting a Standalone Oracle Database to a PDB and Plugging It into a CDB

You can convert a standalone Oracle Database Release 12c or later database to a PDB, and then plug this PDB into a CDB.

1. Connect to the root as a user who has been granted the `DV_OWNER` role.

   For example:

   ```
   sqlplus c##sec_admin
   Enter password: password
   ```

2. Grant the DV_PATCH_ADMIN role to user SYS with CONTAINER = CURRENT.

```
GRANT DV_PATCH_ADMIN TO SYS CONTAINER = CURRENT;
```

3. In the root, connect as user SYS with the SYSOPER system privilege.

   For example:

```
CONNECT SYS AS SYSOPER
Enter password: password
```

4. Restart the database in read-only mode.

   For example:

```
SHUTDOWN IMMEDIATE
STARTUP MOUNT
ALTER DATABASE OPEN READ ONLY
```

5. Connect to the Database Vault-enabled database as a user who has the DV_OWNER role.

   For example:

```
CONNECT sec_admin@pdb_name
```

6. Grant the DV_PATCH_ADMIN role to user SYS on this database.

```
GRANT DV_PATCH_ADMIN TO SYS;
```

7. Optionally, run the DBMS_PDB.CHECK_PLUG_COMPATIBILITY function to determine whether the unplugged PDB is compatible with the CDB.

   When you run the function, set the following parameters:

   - pdb_descr_file: Set this parameter to the full path to the XML file that will contain a description of the PDB.
   - store_report: Set this parameter to indicate whether you want to generate a report if the PDB is not compatible with the CDB. Set it to TRUE to generate a report or FALSE to not generate a report. A generated report is stored in the PDB_PLUG_IN_VIOLATIONS temporary table and is generated only if the PDB is not compatible with the CDB.

   For example, to determine whether a PDB described by the /disk1/usr/dv_db_pdb.xml file is compatible with the current CDB, run the following PL/SQL block:

```
SET SERVEROUTPUT ON
DECLARE
  compatible CONSTANT VARCHAR2(3) :=
    CASE DBMS_PDB.CHECK_PLUG_COMPATIBILITY(
           pdb_descr_file => '/disk1/usr/dv_db_pdb.xml',
           store_report   => TRUE)
    WHEN TRUE THEN 'YES'
    ELSE 'NO'
END;
```

```
BEGIN
  DBMS_OUTPUT.PUT_LINE(compatible);
END;
/
```

If the output is YES, then the PDB is compatible, and you can continue with the next step.

If the output is NO, then the PDB is not compatible. You can check the PDB_PLUG_IN_VIOLATIONS temporary table to see why it is not compatible.

8. Create an XML file that describes the PDB.

   For example:

   ```
   BEGIN
     DBMS_PDB.DESCRIBE(
       pdb_descr_file => '/disk1/oracle/dv_db.xml');
   END;
   /
   ```

9. Run the CREATE PLUGGABLE DATABASE statement, and specify the XML file in the USING clause. Specify other clauses when they are required.

   For example:

   ```
   CREATE PLUGGABLE DATABASE pdb_name AS CLONE USING 'dv_db.xml'
   NOCOPY;
   ```

10. Connect to the PDB that you just created as user SYS with the SYSDBA administrative privilege.

    ```
    CONNECT SYS@pdb_name AS SYSDBA
    ```

11. Execute the noncdb_to_pdb.sql script.

    ```
    @$ORACLE_HOME/rdbms/admin/noncdb_to_pdb.sql
    ```

12. Open this PDB in a read/write restricted mode.

    ```
    ALTER PLUGGABLE DATABASE pdb_name OPEN READ WRITE RESTRICTED;
    ```

13. Run the following procedure to synchronize the PDB:

    ```
    EXECUTE DBMS_PDB.SYNC_PDB;
    ```

14. Connect to the root as a user who has been granted the DV_OWNER role.

    ```
    sqlplus c##sec_admin
    Enter password: password
    ```

15. Revoke the DV_PATCH_ADMIN role from user SYS with CONTAINER = CURRENT.

    ```
    REVOKE DV_PATCH_ADMIN FROM SYS CONTAINER = CURRENT;
    ```

16. Connect to the legacy Database Vault-enabled database as user SYS with the SYSOPER system privilege.

    ```
    CONNECT SYS@pdb_name AS SYSOPER
    ```

17. Restart this database.

    For example:

    ```
    SHUTDOWN IMMMEDIATE
    STARUP
    ```

18. Revoke the DV_PATCH_ADMIN role from user SYS.

    ```
    REVOKE DV_PATCH_ADMIN FROM SYS;
    ```

Related Topics

- Creating and Removing PDBs and Application Containers
  You can create PDBs, application containers, and application seeds using a variety of techniques.

25
Using XStream with a CDB

You can use Oracle Database XStream in a multitenant container database (CDB).

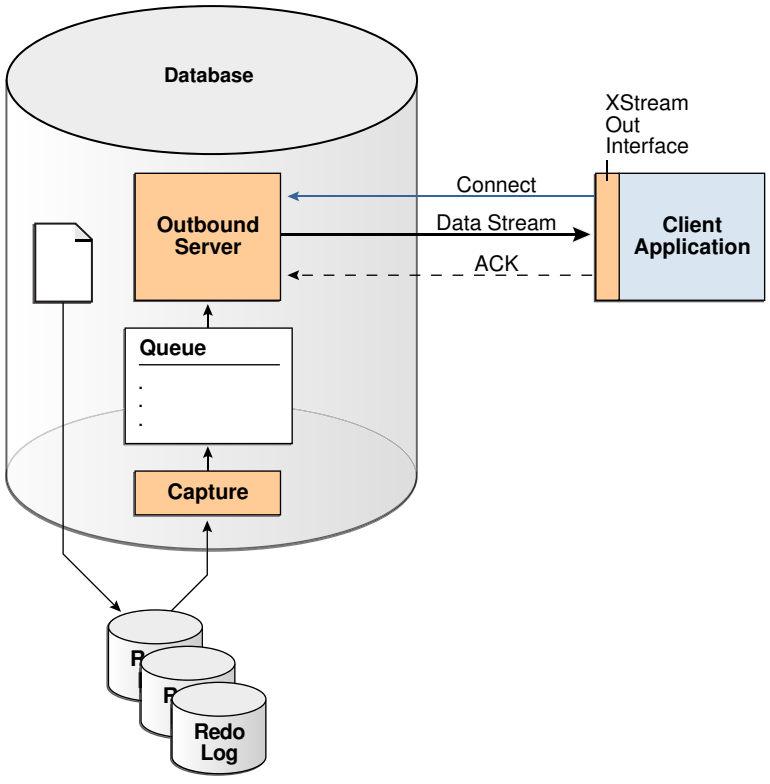
About XStream

XStream consists of Oracle Database components and application programming interfaces (APIs) that enable client applications to receive data changes from an Oracle database and send data changes to an Oracle database.

These data changes can be shared between Oracle databases and other systems. The other systems include non-Oracle databases, non-RDBMS Oracle products, file systems, third party software applications, and so on. A client application is designed by the user for specific purposes and use cases.

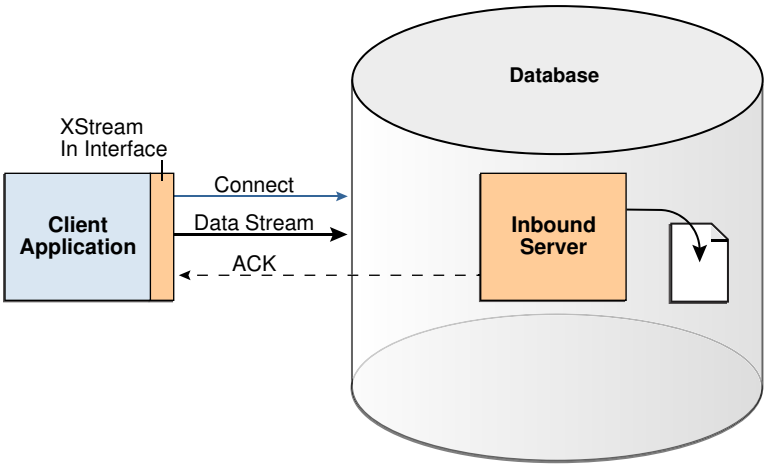
XStream consists of two major features: XStream Out and XStream In. **XStream Out** provides Oracle Database components and APIs that enable you to share data changes made to an Oracle database with other systems. XStream Out can retrieve both data manipulation language (DML) and data definition language (DDL) changes from the redo log and send these changes to a client application that uses the APIs, as shown in the following figure.

Figure 25-1 XStream Out



XStream In provides Oracle Database components and APIs that enable you to share data changes made to other systems with an Oracle database. XStream In can apply these changes to database objects in the Oracle database, as shown in the following figure.

Figure 25-2 XStream In



XStream uses the capture and apply features of the Oracle database. These features enable the following functionality for XStream:

- The logical change record (LCR) format for streaming database changes

  An **LCR** is a message with a specific format that describes a database change. If the change was a data manipulation language (DML) operation, then a **row LCR** encapsulates each row change resulting from the DML operation. One DML operation might result in multiple row changes, and so one DML operation might result in multiple row LCRs. If the change was a data definition language (DDL) operation, then a single **DDL LCR** encapsulates the DDL change.
- Rules and rule sets that control behavior, including inclusion and exclusion rules

  Rules enable the filtering of database changes at the database level, schema level, table level, and row/column level.
- Rule-based transformations that modify captured data changes
- Support for most data types in the database, including LOBs, `LONG`, `LONG RAW`, and `XMLType`
- Customized configurations, including multiple inbound streams to a single database instance, multiple outbound streams from a single database instance, multiple outbound streams from a single capture process, and so on
- Full-featured apply for XStream In, including apply parallelism for optimal performance, SQL generation, conflict detection and resolution, error handling, and customized apply with apply handlers

> **Note:**
>
> In both XStream Out and XStream In configurations, the client application must use a dedicated server connection.

Related Topics

- *Oracle Database XStream Guide*

System-Created Rules and a Multitenant Environment

A multitenant environment enables an Oracle database to contain a portable set of schemas, objects, and related structures that appears logically to an application as a separate database. This self-contained collection is called a pluggable database (PDB). A CDB contains PDBs.

It can also contain application containers. An application container is an optional component of a CDB that consists of an application root and the application PDBs associated with it. An application container stores data for one or more applications. An application container shares application metadata and common data. In a CDB, each of the following is a container: the CDB root, each PDB, each application root, and each application PDB.

In a CDB, LCRs can contain the global name of the container where the change originated in the `source_database_name` attribute and the global name of the CDB root in the `root_name` attribute. The rules for XStream components can consider these attributes.

Related Topics

- Multitenant Architecture
  The **multitenant architecture** enables an Oracle database to function as a multitenant container database (CDB).

System-Created Rules in a CDB and XStream Out

In a CDB, XStream Out must be configured in the CDB root. Therefore, the PL/SQL procedures in the `DBMS_XSTREAM_ADM` package that create system-created rules must be run in the CDB root while connected as a common user.

Excluding the procedures that create rules for propagations, the procedures that create system-created rules for XStream Out, such as the `ADD_GLOBAL_RULES` procedure, include the key parameters in the following table:

Table 25-1 Key Procedure Parameters for System-Created Rules in a CDB

Parameter	Description
`source_database`	The global name of the source database. In a CDB, specify the global name of the container to which the rules pertain. The container can be the CDB root, a PDB, an application root, or an application PDB. The following are examples: `mycdb.example.com` or `hrpdb.example.com`.

Table 25-1 (Cont.) Key Procedure Parameters for System-Created Rules in a CDB

Parameter	Description
source_root_name	The global name of the CDB root in the source CDB. The following are examples: mycdb.example.com.
source_container_name	The short name of the source container. The container can be the CDB root, a PDB, an application root, or an application PDB. The following are examples: CDB$ROOT or hrpdb.

If you do not include the domain name when you specify source_database or source_root_name, then the procedure appends it to the name automatically. For example, if you specify DBS1 and the domain is .EXAMPLE.COM, then the procedure specifies DBS1.EXAMPLE.COM automatically.

The combination of these key parameters determines which containers' changes XStream Out captures and streams to the client application, based on the rules generated by the procedures. Regardless of the settings for these parameters, system-generated rules can still limit the changes captured and streamed to specific schemas and tables.

Local capture means that a capture process runs on the source CDB. In a local capture configuration, the source_root_name parameter specifies the global name of the CDB root in the local CDB. If this parameter is NULL, then the global name of the CDB root in the local CDB is specified automatically. The resulting rules include a condition for the global name of the CDB root in the current CDB.

Downstream capture means that a capture process runs on a CDB other than the source CDB. In a downstream capture configuration, the source_root_name parameter must be non-NULL, and it must specify the global name of the CDB root in the remote source CDB. The resulting rules include a condition for the global name of the CDB root in the remote CDB. If this parameter is NULL, then local capture is assumed.

The following table describes the rule conditions for various source_database and source_container_name parameter settings in a local capture configuration.

Table 25-2 Local Capture and XStream Out Container Rule Conditions

source_database Parameter Setting	source_container_name Parameter Setting	Description
NULL	NULL	XStream Out captures and streams changes made in any container in the local CDB, including the CDB root, all PDBs, all application roots, and all application PDBs.
non-NULL	NULL	XStream Out captures and streams changes made in the specified source container of the local CDB. The source container can be the CDB root, a PDB, an application root, or an application PDB. The DBMS_XSTREAM_ADM procedure queries the CDB_PDBS view and CDB_PROPERTIES view to determine the source_container_name value.

Table 25-2 (Cont.) Local Capture and XStream Out Container Rule Conditions

source_database Parameter Setting	source_container_name Parameter Setting	Description
NULL	non-NULL	XStream Out captures and streams changes made in the specified source container of the local CDB. The source container can be the CDB root, a PDB, an application root, or an application PDB. The DBMS_XSTREAM_ADM procedure queries the CDB_PDBS view and CDB_PROPERTIES view to determine the source_database value.
non-NULL	non-NULL	XStream Out captures and streams changes made in the specified source container of the local CDB. The source container can be the CDB root, a PDB, an application root, or an application PDB. If the prefix of the source_database value is different from the source_container_name value, then the resulting rules include a condition for the source_database value, and an internal table maps the source_database value to the source_container_name value.

The following table describes the rule conditions for various source_database and source_container_name parameter settings in a downstream capture configuration.

Table 25-3 Downstream Capture and XStream Out Container Rule Conditions

source_database Parameter Setting	source_container_name Parameter Setting	Description
NULL	NULL	XStream Out captures and streams changes made in any container in the remote source CDB, including the CDB root, all PDBs, all application roots, and all application PDBs.
non-NULL	NULL	XStream Out captures and streams changes made in the specified source container of the remote source CDB. The source container can be the CDB root, a PDB, an application root, or an application PDB. The DBMS_XSTREAM_ADM procedure derives the source_container_name value from the prefix of source_database value.
NULL	non-NULL	The DBMS_XSTREAM_ADM procedure raises an error.

Table 25-3 (Cont.) Downstream Capture and XStream Out Container Rule Conditions

source_database Parameter Setting	source_container_name Parameter Setting	Description
non-NULL	non-NULL	XStream Out captures and streams changes made in the specified source container of the remote source CDB. The source container can be the CDB root, a PDB, an application root, or an application PDB. If the prefix of the `source_database` value is different from the `source_container_name` value, then the resulting rules include a condition for the `source_database` value, and an internal table maps the `source_database` value to the `source_container_name` value.

Related Topics

- *Oracle Database XStream Guide*
- *Oracle Database PL/SQL Packages and Types Reference*

System-Created Rules in a CDB and XStream In

You can configure XStream In in the root or in any container in a CDB.

Typically, an inbound server does not use rule sets or rules. Instead, it usually processes all LCRs that it receives from its client application. An inbound server can apply changes to the current container only. Therefore, if an inbound server is configured in the CDB root, then it can apply changes only to the CDB root. If an inbound server is configured in a PDB, then it can apply changes only to that PDB. If an inbound server is configured in an application root, then it can apply changes only to that application root, and if an inbound server is configured in an application PDB, then it can apply changes only to that application PDB.

Related Topics

- Administering a Multitenant Environment
  You can administer a multitenant environment using SQL*Plus or Enterprise Manager Cloud Control (Cloud Control).

XStream Out and a Multitenant Environment

A multitenant environment enables a database to contain a portable set of schemas, objects, and related structures that appears logically to an application as a separate database.

This self-contained collection is called a pluggable database (PDB). A multitenant container database (CDB) contains PDBs. In a CDB, XStream Out functions much the same as it does in a non-CDB.

A CDB can also contain application containers. An application container is an optional component of a CDB that consists of an application root and all application PDBs associated with it. An application container stores data for one or more applications. An application container shares application metadata and common data. In a CDB, each of the following is a container: the CDB root, each PDB, each application root, and each application PDB.

The main differences in the way XStream Out functions in a CDB and non-CDB are:

- XStream Out must be configured only in the CDB root.
- XStream Out can see changes made to any container within the CDB.
- XStream Out capture rules can limit the LCRs to those that are needed for the client application. The system-generated capture rules select the appropriate LCRs based on the parameters that were passed to the `ADD_OUTBOUND` and `CREATE_OUTBOUND` procedures in the `DBMS_XSTREAM_ADM` package. You can use the `ADD_*_RULES` procedures in the same package for more fine-grained control over the rules used by the XStream Out components.
- The user who performs XStream Out tasks must be a common user.

Unplug and Plug Operations in an XStream Environment

When a PDB, application root, or application PDB involved with XStream Out is unplugged from its CDB and plugged into another CDB, any capture process or outbound server is not considered part of the container. You must configure the capture process and outbound server again in the other CDB.

If an outbound server is configured in a different database than the capture process, then unplug and plug operations have additional considerations.

For this example, assume the following:

- A CDB named `CDB1` contains PDB `PDB1`.
- A capture process is configured in `CDB1`, and it sends LCRs from `PDB1` to an outbound server in a CDB named `CDB2`.
- You unplug `PDB1` from `CDB1`, and then plug it into a CDB named `CDB3`.

To continue delivering LCRs from `PDB1` to the outbound server in `CDB2`, you must configure a new capture process in `CDB3` to capture and send LCRs to `CDB2`.

The rules used by the outbound server in database B must be altered to change references to the root of `CDB1` to the root of `CDB3`. In addition, if `PDB1` was given a different name in `CDB3`, then the rules must be altered to reflect the new PDB name.

Application Containers in an XStream Environment

When a CDB has one or more application containers, XStream Out must be configured in the CDB root, and XStream Out can capture changes made in any container in the CDB, including the application roots and application PDBs. Changes captured in an application container can be sent to containers of any type, including PDBs, application roots, and application PDBs.

When replicating changes from one application root to another application root, XStream can replicate `ALTER PLUGGABLE DATABASE APPLICATION` statements. To avoid errors, the target application root that applies the statements must have the same application installed as the source application root, and the application name must be identical in both application roots.

To avoid errors when replicating changes from an application root to a container that is not an application root, you must ensure that `ALTER PLUGGABLE DATABASE APPLICATION` statements are not replicated.

With the XStream OCI API, you can control whether `ALTER PLUGGABLE DATABASE APPLICATION` statements are replicated using the `OCIXStreamOutAttach` function and the `OCILCRHeaderGet` function. With the XStream Java API, you can control this behavior using the `mode` parameter in the `XStreamOut.attach` method.

Related Topics

- System-Created Rules and a Multitenant Environment
  A multitenant environment enables an Oracle database to contain a portable set of schemas, objects, and related structures that appears logically to an application as a separate database. This self-contained collection is called a pluggable database (PDB). A CDB contains PDBs.
- Configuring XStream Out in a CDB
  When you configure XStream Out in a CDB, you must decide which database changes will be captured by XStream Out and sent to the client application.
- Multitenant Architecture
  The **multitenant architecture** enables an Oracle database to function as a multitenant container database (CDB).

Configuring XStream Out in a CDB

When you configure XStream Out in a CDB, you must decide which database changes will be captured by XStream Out and sent to the client application.

XStream Out can stream all database changes for all containers, including the CDB root and all PDBs, application roots, and application PDBs, or XStream Out can stream the changes from specific containers. In addition, you can configure XStream Out with local capture, or you can configure it with downstream capture to offload the work required to capture changes from the source database.

The following restrictions apply when you configure XStream Out in a CDB:

- The capture process and outbound server must be in the CDB root.
- The capture process and outbound server must be in the same CDB.
- Each container in the CDB must be open during XStream Out configuration.
- When changes made to an application root are captured, you must ensure that `ALTER PLUGGABLE DATABASE APPLICATION` statements are replicated only to other application roots.

In addition, ensure that you create the XStream administrator properly for a CDB.

> **Note:**
>
> When a container is created using a non-CDB, any XStream Out components from the non-CDB cannot be used in the container. You must drop and re-create the XStream Out components, including the capture process and outbound servers, in the CDB root.

Related Topics

- XStream Out and a Multitenant Environment
  A multitenant environment enables a database to contain a portable set of schemas, objects, and related structures that appears logically to an application as a separate database.
- System-Created Rules in a CDB and XStream Out
  In a CDB, XStream Out must be configured in the CDB root. Therefore, the PL/SQL procedures in the `DBMS_XSTREAM_ADM` package that create system-created rules must be run in the CDB root while connected as a common user.
- *Oracle Database XStream Guide*
- Options for Creating a PDB from a Non-CDB
  You have multiple options for moving a non-CDB into a PDB.

Configuring XStream Out with Local Capture in a CDB

An example illustrates configuring XStream Out with local capture in a CDB.

Prerequisites

Before configuring XStream Out, ensure that all containers in the CDB are in open read/write mode during XStream Out configuration.

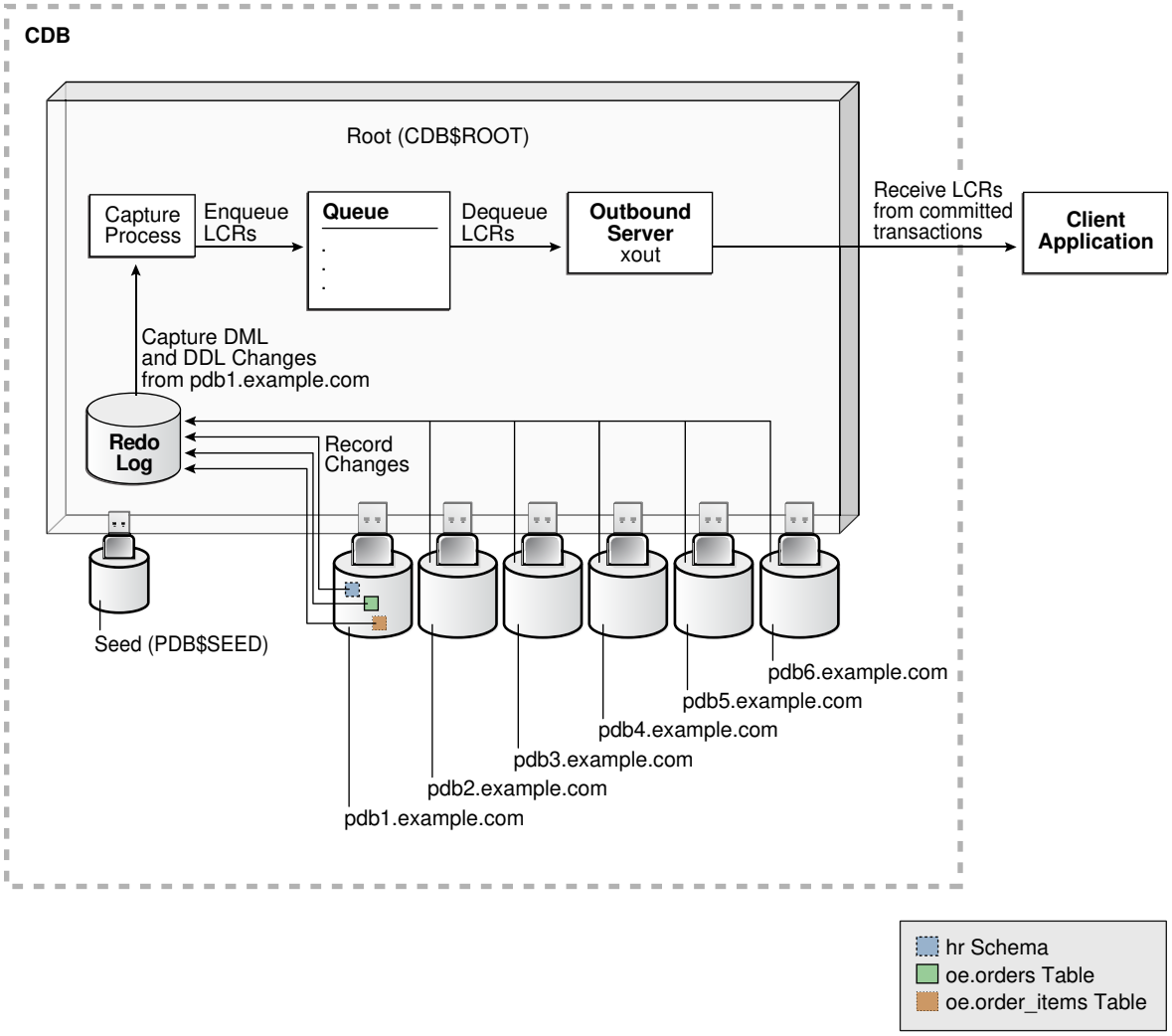
Assumptions

This section makes the following assumptions:

- The capture process will be a local capture process, and it will run on the same database as the outbound server.
- The name of the outbound server is `xout`.
- Data manipulation language (DML) and data definition language (DDL) changes made to the `oe.orders` and `oe.order_items` tables in PDB `pdb1.example.com` are sent to the outbound server.
- DML and DDL changes made to the `hr` schema in the PDB `pdb1.example.com` are sent to the outbound server.

Figure 25-3 provides an overview of this XStream Out configuration.

Figure 25-3 Sample XStream Out Configuration Created Using CREATE_OUTBOUND for a PDB



To create an outbound server using the `CREATE_OUTBOUND` procedure:

1. In SQL*Plus, connect to the root in the CDB (not to the PDB `pdb1.example.com`) as the XStream administrator.
2. Create the outbound server and other XStream components.
   a. Ensure that all containers in the source CDB are in open read/write mode.
   b. Run the `CREATE_OUTBOUND` procedure.

      Given the assumptions for this example, run the following `CREATE_OUTBOUND` procedure:

```
DECLARE
  tables  DBMS_UTILITY.UNCL_ARRAY;
  schemas DBMS_UTILITY.UNCL_ARRAY;
BEGIN
    tables(1)  := 'oe.orders';
    tables(2)  := 'oe.order_items';
```

```
    schemas(1) := 'hr';
  DBMS_XSTREAM_ADM.CREATE_OUTBOUND(
    server_name     =>  'xout',
    source_database =>  'pdb1.example.com',
    table_names     =>  tables,
    schema_names    =>  schemas);
END;
/
```

> **Note:**
>
> To capture changes in all containers in a CDB, including the CDB root, all PDBs, all application roots, and all application PDBs, and send those changes to the XStream client application, you can omit the `source_database` parameter when you run the `CREATE_OUTBOUND` procedure.

c. After the `CREATE_OUTBOUND` procedure completes successfully, optionally change the open mode of one or more containers if necessary.

Running the procedure in Step b performs the following actions:

- Configures supplemental logging for the `oe.orders` and `oe.order_items` tables and for all tables in the `hr` schema in the `pdb1.example.com` PDB.
- Creates a queue with a system-generated name that is used by the capture process and the outbound server.
- Creates and starts a capture process with a system-generated name with rule sets that instruct it to capture DML and DDL changes to the `oe.orders` table, the `oe.order_items` table, and the `hr` schema from the `pdb1.example.com` PDB.
- Creates and starts an outbound server named `xout` with rule sets that instruct it to send DML and DDL changes to the `oe.orders` table, the `oe.order_items` table, and the `hr` schema to the client application.
- Sets the current user as the connect user for the outbound server. In this example, the current user is the XStream administrator. The client application must connect to the database as the connect user to interact with the outbound server.

> **Note:**
>
> The `server_name` value cannot exceed 30 bytes.

> **Tip:**
>
> To capture and send all database changes from the `pdb1.example.com` database to the outbound server, specify `NULL` (the default) for the `table_names` and `schema_names` parameters.

3. Create and run the client application that will connect to the outbound server in the root of the CDB and receive the LCRs.

   When you run the client application, the outbound server is started automatically.

Related Topics

- *Oracle Database XStream Guide*
- *Oracle Database XStream Guide*

Configuring XStream Out with Downstream Capture in CDBs

Using downstream capture, the XStream Out components can reside in databases other than the source database.

When you have multiple CDBs, the source database can be in one CDB, and you can use downstream capture to capture the changes in another CDB.

Prerequisites

Before configuring XStream Out, the following prerequisites must be met:

- Ensure that all containers in the CDB are in open read/write mode during XStream Out configuration.
- This example uses downstream capture. Therefore, you must configure log file transfer from the source database to a downstream database.
- If you want to use real-time downstream capture, then you must also add the required standby redo logs.

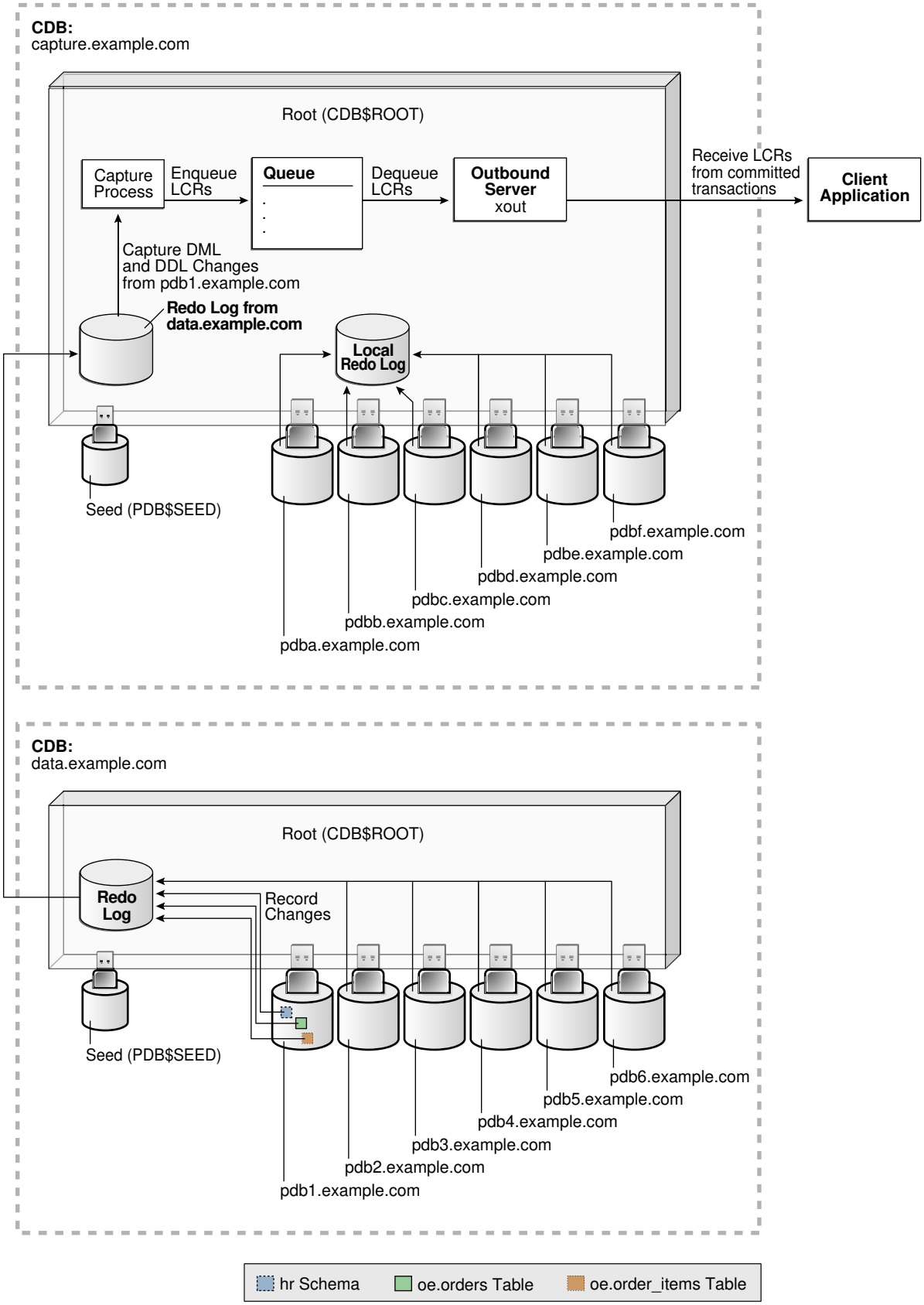
Assumptions

This section makes the following assumptions:

- The name of the outbound server is `xout`.
- The queue used by the outbound server is `c##xstrmadmin.xstream_queue`.
- The source database is the PDB `pdb1.example.com` in the CDB `data.example.com`.
- The capture process runs in the CDB `capture.example.com`.
- The outbound server runs in the CDB `capture.example.com`.
- DML and DDL changes made to the `oe.orders` and `oe.order_items` tables from the PDB `pdb1.example.com` are sent to the outbound server.
- DML and DDL changes made to the `hr` schema from the PDB `pdb1.example.com` are sent to the outbound server.

The following figure gives an overview of this XStream Out configuration.

Figure 25-4 Sample XStream Out Configuration Using Multiple CDBs and Downstream Capture



To configure XStream Out with downstream capture in CDBs:

1. In SQL*Plus, connect to the root of the downstream capture CDB as the XStream administrator.

   In this example. the downstream capture CDB is `capture.example.com`.

2. Create the queue that will be used by the capture process.

   For example, run the following procedure:

   ```
   BEGIN
     DBMS_XSTREAM_ADM.SET_UP_QUEUE(
       queue_table => 'c##xstrmadmin.xstream_queue_table',
       queue_name  => 'c##xstrmadmin.xstream_queue');
   END;
   /
   ```

3. Optionally, create the database link from the root in the downstream capture CDB to the root in the source CDB.

   In this example, create a database link from the root in `capture.example.com` to the root in `data.example.com`. For example, if the user `c##xstrmadmin` is the XStream administrator on both databases, then create the following database link:

   ```
   CREATE DATABASE LINK data.example.com CONNECT TO c##xstrmadmin
      IDENTIFIED BY password USING 'data.example.com';
   ```

4. Ensure that all containers in the source CDB are in open read/write mode.

5. If you did not create the database link in Step 3, then you must complete additional steps in the root of the source CDB.

   These steps are not required if you created the database link in Step 3.

   Run the `BUILD` procedure and ensure that required supplemental logging is specified for the database objects in the source CDB:

   a. Connect to the root in the source CDB as the XStream administrator.

   b. Run the `DBMS_CAPTURE_ADM.BUILD` procedure. For example:

      ```
      SET SERVEROUTPUT ON
      DECLARE
        scn  NUMBER;
      BEGIN
        DBMS_CAPTURE_ADM.BUILD(
          first_scn => scn);
        DBMS_OUTPUT.PUT_LINE('First SCN Value = ' || scn);
      END;
      /
      First SCN Value = 409391
      ```

      This procedure displays the valid first SCN value for the capture process that will be created in the root in the `capture.example.com` CDB. Make a note of the SCN value returned because you will use it when you create the capture process in Step 6.

c. Ensure that required supplemental logging is specified for the database objects in the source CDB.

   For this example, ensure that supplemental logging is configured for the `hr` schema, the `oe.orders` table, and the `oe.order_items` table in the `pdb1.example.com` PDB.

6. While connected to the root in the downstream capture CDB, create the capture process.

   For example, run the following procedure to create the capture process while connected as the XStream administrator to `capture.example.com`:

```
BEGIN
  DBMS_CAPTURE_ADM.CREATE_CAPTURE(
    queue_name         => 'c##xstrmadmin.xstream_queue',
    capture_name       => 'real_time_capture',
    rule_set_name      => NULL,
    start_scn          => NULL,
    source_database    => NULL,
    use_database_link  => TRUE,
    first_scn          => NULL,
    logfile_assignment => 'implicit',
    source_root_name   => 'data.example.com',
    capture_class      => 'xstream');
END;
/
```

   If you did not create a database link in Step 3, then specify the SCN value returned by the `DBMS_CAPTURE_ADM.BUILD` procedure for the `first_scn` parameter.

   Do not start the capture process.

7. After the capture process is created, optionally change the open mode of one or more PDBs if necessary.

8. Run the `ADD_OUTBOUND` procedure.

   Given the assumption for this section, run the following `ADD_OUTBOUND` procedure:

```
DECLARE
  tables  DBMS_UTILITY.UNCL_ARRAY;
  schemas DBMS_UTILITY.UNCL_ARRAY;
BEGIN
    tables(1)  := 'oe.orders';
    tables(2)  := 'oe.order_items';
    schemas(1) := 'hr';
  DBMS_XSTREAM_ADM.ADD_OUTBOUND(
    server_name           =>  'xout',
    queue_name            =>  'c##xstrmadmin.xstream_queue',
    source_database       =>  'pdb1.example.com',
    table_names           =>  tables,
    schema_names          =>  schemas,
    source_root_name      => 'data.example.com',
    source_container_name => 'pdb1');
END;
/
```

Running this procedure performs the following actions:

- Creates an outbound server named `xout`. The outbound server has rule sets that instruct it to send DML and DDL changes to the `oe.orders` table, the `oe.order_items` table, and the `hr` schema to the client application. The rules specify that these changes must have originated at the PDB `pdb1.example.com` in the CDB `data.example.com`. The outbound server dequeues LCRs from the queue `c##xstrmadmin.xstream_queue`.
- Sets the current user as the `connect_user` for the outbound server. In this example, the `current_user` is the XStream administrator. The client application must connect to the database as the `connect_user` to interact with the outbound server.

> **Note:**
>
> The `server_name` value cannot exceed 30 bytes.

9. Create and run the client application that will connect to the outbound server and receive the LCRs.

   When you run the client application, the outbound server is started automatically at the downstream capture CDB.

Related Topics

- *Oracle Database XStream Guide*

XStream In and a Multitenant Environment

A multitenant environment enables an Oracle database to contain a portable set of schemas, objects, and related structures that appears logically to an application as a separate database.

This self-contained collection is called a pluggable database (PDB). A multitenant container database (CDB) contains PDBs. It can also contain application containers. An application container is an optional component of a CDB that consists of an application root and all application PDBs associated with it. An application container stores data for one or more applications. An application container shares application metadata and common data. In a CDB, each of the following is a container: the CDB root, each PDB, each application root, and each application PDB.

In a CDB, the inbound server is restricted to receiving LCRs from one source database and only executing changes in the current container (one PDB, one application root, one application PDB, or the CDB root). A single inbound server cannot apply changes to more than one container in a CDB.

When the inbound server is in the CDB root, the apply user must be a common user. When the inbound server is in an application root, the apply user must be a common user or an application common user. When the inbound server is in a PDB or application PDB, the apply user can be a common user or a local user.

> **Note:**
>
> XStream does not synchronize changes done in the application root container. Do not use the XStream In replication to replicate operations done in the application root container. You can manually apply these changes in the application root containers in the target. Note that the operations done in the PDBs can still be replicated.

Related Topics

- System-Created Rules and a Multitenant Environment
  A multitenant environment enables an Oracle database to contain a portable set of schemas, objects, and related structures that appears logically to an application as a separate database. This self-contained collection is called a pluggable database (PDB). A CDB contains PDBs.
- Multitenant Architecture
  The **multitenant architecture** enables an Oracle database to function as a multitenant container database (CDB).

Glossary

application

Within an application root, an application is a named, versioned set of data and metadata created by a common user. An application might include an application common user, an application common object, or some multiple and combination of the preceding.

application common object

A shared database object created while connected to an application root. The metadata (for a metadata-linked object) or data (for a data-linked common object) is shared by application PDBs in the application container.

application common user

A common user created while connected to an application root. The metadata (for a metadata-linked common object) or data (for a data-linked common object) is shared by application PDBs in the application container.

application container

A named set of application PDBs plugged in to an application root. An application container may contain an application seed.

application patch

In an application container, a small change to an application. Typical examples of patching include bug fixes and security patches. An application upgrade begins and ends with an `ALTER PLUGGABLE DATABASE APPLICATION` statement.

application PDB

A PDB that is plugged in to an application container.

application root

The root container within an application container. Every application container has exactly one application root. An application root shares some characteristics with the

CDB root, because it can contain common objects, and some characteristics with a PDB, because it is created with the `CREATE PLUGGABLE DATABASE` statement.

application seed

An optional application PDB that serves as a template for creating other PDBs within an application container. An application container includes 0 or 1 application seed.

application upgrade

In an application container, a major change to the physical architecture of an application. An application upgrade begins and ends with an `ALTER PLUGGABLE DATABASE APPLICATION` statement.

CDB

An Oracle Database installation that contains at least one PDB. A PDB appears to an Oracle Net client as a traditional Oracle database. Every Oracle database is either a CDB or a non-CDB.

CDB administrator

A database administrator who manages a CDB. A PDB administrator manages individual PDBs within the CDB.

CDB fleet

A collection of different CDBs that can be managed as one logical CDB.

CDB restore point

In a CDB, a restore point that is created when connected to the root, and when the FOR PLUGGABLE DATABASE clause is not specified. Unlike a PDB restore point, a CDB restore point is usable by all PDBs.

CDB root

In a multitenant container database (CDB), a collection of schemas, schema objects, and nonschema objects to which all PDBs belong. Every CDB has exactly one root container, which stores the system metadata required to manage PDBs. All PDBs belong to the CDB root.

clean restore point

A PDB restore point that is created when the PDB is closed. A Flashback PDB to a clean restore point does not require restoring backups or creating a temporary instance.

common object

An object that resides either in the CDB root or an application root that shares either data (a data-linked common object) or metadata (a metadata-linked common object). All common objects in the CDB root are Oracle-supplied. A common object in an application root is called an application common object.

common user

In a multitenant container database (CDB), a database user that exists with the same identity in multiple containers. A common user created in the CDB root has the same identity in every existing and future PDB. A common user created in an application container has the same identity in every existing and future application PDB in this application container.

container

In a multitenant container database (CDB), either the root or a PDB.

container data object

In a CDB, a table or view containing data pertaining to multiple containers and possibly the CDB as a whole, along with mechanisms to restrict data visible to specific common users through such objects to one or more containers. Examples of container data objects are Oracle-supplied views whose names begin with `V$` and `CDB_`.

cross-container operation

In a CDB, a DDL statement that affects the CDB itself, multiple containers, multiple common users or roles, or a container other than the one to which the user is connected. Only a common user connected to the root can perform cross-container operations.

data link

In a PDB, an internal mechanism that points to data (not metadata) in the root. For example, AWR data resides in the root. Each PDB uses an object link to point to the AWR data in the root, thereby making views such as `DBA_HIST_ACTIVE_SESS_HISTORY` and `DBA_HIST_BASELINE` accessible in each separate container.

database consolidation

The general process of moving data from one or more non-CDBs into a multitenant container database (CDB).

data-linked common object

A common object that exists either in the CDB root or an application root. The data, rather than the metadata, is shared by any PDB that contains a data link that points to the common object.

extended data-linked common object

A hybrid of a data-linked common object and a metadata-linked common object. For an extended data-linked object, each application PDB can create its own PDB-specific data while sharing the common data in the application root.

Fast Application Notification (FAN)

Applications can use FAN to enable rapid failure detection, balancing of connection pools after failures, and re-balancing of connection pools when failed components are repaired. The FAN notification process uses system events that Oracle Database publishes when cluster servers become unreachable or if network interfaces fail.

hot cloning

Cloning a PDB while the source PDB is open in read/write mode.

lead CDB

In a CDB fleet, the central location for monitoring and managing several CDBs.

local undo mode

The use of a separate set of undo data files for each PDB in a CDB.

local user

In a multitenant container database (CDB), any user that is not a common user.

metadata link

In a PDB, an internal mechanism that points to a dictionary object definition stored in the root. For example, the `OBJ$` table in each PDB uses a metadata link to point to the definition of `OBJ$` stored in the root.

metadata-linked common object

A common object that exists either in the CDB root or an application root. The metadata, rather than the data, is shared by any PDB that contains a metadata link that points to the common object.

multitenant architecture

The architecture that enables an Oracle database to function as a multitenant container database (CDB), which means that it can contain multiple PDBs. A PDB is a portable collection of schemas, schema objects, and nonschema objects that appears to an Oracle Net client as a traditional Oracle database (non-CDB).

multitenant container database (CDB)

See CDB.

non-CDB

An Oracle database that is not a multitenant container database (CDB). Before Oracle Database 12c, all databases were non-CDBs. Starting in Oracle Database 12c, every database must be either a CDB or a non-CDB.

Oracle Multitenant

A database option that enables you to create multiple PDBs in a CDB.

PDB

In a multitenant container database (CDB), a portable collection of schemas, schema objects, and nonschema objects that appears to an Oracle Net client as a traditional Oracle database (non-CDB).

PDB administrator

A database administrator who manages one or more PDBs. A CDB administrator manages the whole CDB.

PDB archive file

A compressed file that contains both PDB data files and an XML metadata file. You can create a PDB by specifying the archive file, and thereby avoid copying the XML file and the data files separately.

PDB lockdown profile

A security mechanism to restrict operations that are available to local users connected to a specified PDB. A typical use is to limit the effect of a grant privilege. For example,

you limit the grant of `ALTER SYSTEM` to only those options whose names begin with `PLSQL`.

PDB performance profile

A specified share of system resources, CPU, parallel execution servers, and memory for a PDB or set of PDBs.

PDB restore point

Within a CDB, a restore point that usable only for a specific PDB. In contrast, a CDB restore point is usable by all PDBs.

PDB snapshot

A named, point-in-time copy of a PDB created using the `ALTER PLUGGABLE DATABASE SNAPSHOT` command. At the file level, a PDB snapshot is an archive file containing the contents of the PDB copy.

If the underlying file system supports sparse files, then the first snapshot is full, and every subsequent snapshot is sparse.

PDB synchronization

The user-initiated update of the application in an application PDB to the latest version and patch in the application root.

PDB/non-CDB compatibility guarantee

In the multitenant architecture, the guarantee that a PDB behaves the same as a non-CDB as seen from a client connecting with Oracle Net.

pluggable database (PDB)

See PDB.

proxy PDB

A PDB that references a PDB in a remote CDB using a database link. The remote PDB is called a referenced PDB.

referenced PDB

The PDB that is referenced by a proxy PDB. A local PDB is in the same CDB as its referenced PDB, whereas a remote PDB is in a different CDB.

refreshable clone PDB

A read-only clone that can periodically synchronize with its source PDB. Depending on the value in the `REFRESH MODE` clause, the synchronization occurs either automatically or manually.

resource plan

A container for resource plan directives that specify how resources are allocated to resource consumer groups.

resource plan directive

A set of limits and controls for CPU, physical I/O, or logical I/O consumption for sessions in a consumer group.

seed PDB

In a multitenant container database (CDB), a default pluggable database (PDB) that the system uses as a template for user-created PDBs. A PDB seed is either the system-supplied `PDB$SEED` or an application seed.

shared undo mode

In a single-instance CDB, only one active undo tablespace exists. For an Oracle RAC CDB, one active undo tablespace exists for every instance.

snapshot copy PDB

A PDB that is created by running the `CREATE PLUGGABLE DATABASE ... FROM ... SNAPSHOT COPY` command. A storage-managed snapshot is a copy of the underlying storage that is only supported on specific file systems.

> **Note:**
>
> A *storage*-managed snapshot, which is used to make a snapshot copy PDB, is different from a *PDB*-managed snapshot, which can be specified in a `CREATE PLUGGABLE DATABASE ... USING SNAPSHOT` command. Storage-managed snapshots are not involved in clones from PDB snapshots.

split mirror clone PDB

A PDB that is created by splitting a mirror in Oracle ASM.

Enter the your glossary term here.

Enter the description here

system container

The container that includes the CDB root and all PDBs in the CDB.

unplugged PDB

A self-contained set of PDB data files, and an XML metadata file that specifies the locations of the PDB files.

Index

A

B

C

D

E

F

P

R

S

T

U

V

X